Aplicación de los videojuegos a la medida de la inteligencia

Measuring intelligence through video games

Mingxiao Guo
Pablo Gutiérrez Sánchez
Alejandro Ortega Álvarez

Grado en Desarrollo de Videojuegos
Doble Grado en Ingeniería Informática y Matemáticas

UNIVERSIDAD
COMPLUTENSE
MADRID

Trabajo de fin de grado en
Desarrollo de Videojuegos e Ingeniería Informática

Facultad de Informática

Curso 2019/2020

Directores:
Pedro Pablo Gómez Martín
Pedro Antonio González Calero

Resumen

Diversos proyectos de investigación recientes han demostrado un marcado grado de correlación entre el factor g (o factor de inteligencia general) y el rendimiento global en los videojuegos. Los experimentos llevados a cabo en ellos incluyeron no sólo juegos "mentales" clásicos (o *brain games*), sino también diferentes títulos comerciales ya existentes. Esos juegos "cerrados" no permiten la extracción automática de datos de comportamiento en el juego. Como resultado, los investigadores a menudo recurrieron a métodos manuales externos para registrar las medidas de rendimiento e interacción con el juego. El objetivo de este trabajo es ayudar a mejorar el proceso de recopilación de datos utilizado en dichos estudios mediante (1) la reimplementación de un pequeño subconjunto que consta de tres de los juegos empleados en estos estudios anteriores (2) desarrollar un sistema de telemetría para automatizar y mejorar la recopilación de eventos y variables de usuario en el juego y (3) implementar una plataforma web para realizar un experimento en línea con el fin de reunir dichos datos. En tres semanas, un total de 53 participantes completaron los tres juegos disponibles en el sistema. A pesar del tamaño modesto del conjunto de datos, pudimos mostrar un factor de inteligencia latente subyacente a través de un análisis factorial exploratorio. Además, intentamos predecir el resultado final de un jugador en un juego dado a partir de registros de partidas truncadas (hasta cierto punto en el tiempo) a través de redes neuronales y bosques aleatorios. Este análisis posterior podría permitir a futuros estudios acortar sus tiempos de experimentación, aumentando así la viabilidad de la evaluación de inteligencia basada en juegos.

Palabras Clave— Videojuegos, Inteligencia, Telemetría, Analíticas de Juego, Knowledge Discovery, Análisis Factorial Exploratorio, Redes Neuronales, Random Forests

Abstract

Several recent research projects have shown a high correlation between the g factor (or general intelligence factor) and overall performance in video games. The experiments carried out in them included not only classical brain games, but also different existing commercial titles. These closed games do not allow the automatic extraction of in-game behaviour data. As a result, researchers often resorted to manual methods of registering measurements externally. The aim of our work is to help to improve the data collection process used in those studies by (1) reimplementing a small subset consisting of three of the games administered in these former studies (2) developing a telemetry system to automate and enhance the recording of in-game user events and variables and (3) deploying a web platform to conduct an online experiment to collect such data. Within three weeks, a total of 53 participants completed all three games available on the system. Despite the dataset's modest size, we were able to show an underlying latent intelligence factor through an exploratory factor analysis. Moreover, we attempted to predict a player's final score in a given game from truncated play logs (up to a certain point in time) using neural networks and random forests. This later analysis could potentially allow future studies to shorten experiment times, thus increasing the viability of game-based intelligence assessment.

Keywords— Videogames, Intelligence, Telemetry, Game Analytics, Knowledge Discovery, Exploratory Factor Analysis, Neural Networks, Random Forests

Agradecimientos

- A la doctora en psicología **María Ángeles Quiroga Estévez**, por todo el apoyo aportado en los ámbitos del trabajo relacionados con psicología, y por sus distintos artículos y estudios sobre la relación entre videojuegos e inteligencia, que sirvieron como punto de partida para este trabajo. Agradecer también su ayuda y asesoramiento en las distintas fases experimentales y de análisis de este TFG, de conseguir que alumnos sin ningún conocimiento previo sobre modelos estadísticos capaces de realizar todo el análisis incluido en el trabajo. Así como por su asistencia en el desarrollo y diseño de las distintas sesiones, experimentos y métricas a las que hacemos mención en la presente memoria.
- A **Pedro Pablo Gómez Martín**, por su constante esfuerzo en todas las distintas partes técnicas del proyecto, por todas las tardes pasadas y correos intercambiados discutiendo temas de programación de los juegos. Agradecemos enormemente toda su dedicatoria a nuestra memoria, todas las revisiones exhaustivas y correcciones aportadas para mejorar este trabajo lo máximo posible. Sin él, no hubiéramos podido entregar este texto con el grado de completitud actual.
- A nuestro tutor **Pedro Antonio González Calero**, por su apoyo constante y su esfuerzo en sacar este TFG adelante, tanto en situaciones de estancamiento o como en dificultades por razones externas, ayudando siempre a proponer ideas nuevas y enfoques distintos. Gracias a su asesoramiento en el análisis por medio de redes neuronales y random forests, que llevó a la obtención de resultados muy prometedores para el tamaño modesto de nuestras muestras.
- A todos los participantes que colaboraron con este trabajo, dedicando su tiempo a jugar a nuestros juegos en la plataforma web, sin los cuales no hubiera sido posible recoger los resultados necesarios para el análisis aquí conducido, y también por el feedback que nos proporcionaron para poder resolver problemas que pudieron no sufrir usuarios posteriores.

Índice general

Introducción

A principios de la década de los 80, los videojuegos comenzaron a alcanzar una gran popularidad, al presentarse por primera vez varios títulos y consolas arcade al público general. Desde entonces, los videojuegos se han convertido no solo en un elemento básico en la industria y la cultura del entretenimiento, sino también en una herramienta ampliamente utilizada para la investigación científica, la educación y diversos fines más allá de la recreación pura. La aparición de estas aplicaciones "serias" de los videojuegos vino acompañado de un creciente interés en su uso para la medida de capacidades cognitivas, cristalizado en diferentes estudios emergentes con índices de correlación bastante bajos [1, 2]. Tras estos proyectos pioneros, varios investigadores se centraron en profundizar sobre el potencial impacto de los videojuegos en la educación [3], así como en tareas o habilidades particulares como las matemáticas [4] o ciertas habilidades cognitivas espaciales [5]. En 2009, la profesora Quiroga y su equipo [6] emplearon videojuegos para el entrenamiento mental (o *brain games*) que involucraban cinco tipos de tareas: razonamiento, cálculo, visualización, memoria y velocidad perceptiva, para proponer la idea de una relación estrecha entre el rendimiento en juegos y la inteligencia general. Este mismo grupo de investigación desarrolló una primera batería de pruebas con 12 juegos de *Big Brain Academy* [7] para evaluar un subconjunto de habilidades cognitivas, alcanzando un alto valor de correlación de 0,93. Con estos resultados en mente, [8] tenía como objetivo ir más allá de los títulos clásicos de entrenamiento mental (*brain training* en inglés) al intentar establecer una dependencia entre una serie de métricas personalizadas y registradas manualmente en juegos comerciales (resumidas por un factor de "rendimiento general de videojuegos", gVG) y el factor de inteligencia general (g). Se logró un resultado considerablemente satisfactorio de 0,79.

Pese al prometedor resultado de este último estudio, se contemplaron varios puntos de posibles mejoras para eludir algunas de sus limitaciones técnicas. Los problemas percibidos pueden resumirse de la siguiente manera: primero, las medidas utilizadas fueron unidimensionales, incluyendo una única variable en la mayoría de los casos. En segundo lugar, estas se registraron manualmente, lo que requería un moderador para supervisar a cada participante durante un experimento presencial. Con estas dos últimas inconveniencias como motivación, se presenta el actual Trabajo de Fin de Grado. Intentamos reproducir los resultados obtenidos, aunque a escala pequeña, cubriendo al mismo tiempo sus puntos débiles. Se plantea implementar tres de los diez juegos incluidos inicialmente en la batería original, en concreto *Blek*, *Edge* y *Unpossible*. Se incorpora un sistema de telemetría para evitar la grabación manual, incluyendo adicionalmente un conjunto ampliado de datos a registrar durante los juegos. Estos son incluidos en un análisis posterior, con el fin de determinar su capacidad para evaluar las habilidades cognitivas ya mencionadas anteriormente.

Con el objetivo de tratar de reproducir los resultados del experimento original, así como de confirmar la validez de las nuevas métricas propuestas en este trabajo, recolectamos datos experimentales de participantes voluntarios. Las condiciones exactas de las sesiones difieren sin embargo del entorno descrito

en el artículo de referencia, como consecuencia de la situación del COVID-19. A partir de los datos obtenidos, conducimos un análisis factorial exploratorio con el fin de examinar las relaciones existentes entre los indicadores observados, en lugar del modelo confirmatorio empleado en dicho artículo.

A partir del hecho de que el sistema de telemetría permite recoger datos muy precisos durante las sesiones de juego, lo segundo que se plantea el trabajo es averiguar hasta qué punto es posible aprovechar esta mayor densidad de información. Proponemos un problema de regresión en el que se plantea predecir el resultado final de un participante, utilizando sesiones con menor duración a la originalmente establecida. La finalidad es acortar el tiempo del experimento, al mismo tiempo asegurar una medida fiable de la inteligencia del sujeto. Para ello, la información de las partidas se transforma en series temporales de datos y se aplican técnicas de aprendizaje profundo para tratar de generar un predictor adecuado a la tarea entre manos. Analizamos, asimismo, la viabilidad de otros métodos tradicionales de aprendizaje automático sobre el problema en cuestión.

Plan del trabajo

Tras haber presentado los objetivos principales del presente trabajo, pasamos ahora a describir brevemente el esquema de planificación general que se seguirá a lo largo del mismo. Algunos de los puntos aquí incluidos están afectados por la pandemia de COVID-19, como se explicará más adelante en esta sección.

- En primera instancia iniciamos el estudio y toma de contacto con el experimento original [8] sobre el que basamos el trabajo. A modo de prueba, seleccionamos tres juegos orientado a medir cada una de las habilidades cognitivas consideradas en el estudio de referencia, concretamente razonamiento fluido (Gf), habilidad visoespacial (Gv), y velocidad de procesamiento (Gs). En principio, el objetivo es desarrollar estos juegos de manera que sean lo más fieles posibles a sus versiones comerciales, con el fin de minimizar discrepancias entre nuestro estudio y el de referencia.
- La segunda fase del trabajo conlleva la implementación de dichos juegos, a realizar en tres etapas fundamentales:
    1. Se comenzará con el desarrollo de un primer prototipo de cada uno de los juegos, centrado en mostrar las mecánicas básicas por medio de un conjunto reducido de niveles; estos prototipos serán evaluados por los tutores del trabajo antes de avanzar a la siguiente fase.
    2. Tras recibir la aprobación por parte de los tutores, comenzará un proceso de desarrollo iterativo, compuesto por subfases de toma de nuevos requisitos o cambios en los requisitos de partida, implementación de niveles y mecánicas adicionales, y testeo por parte tanto de los desarrolladores como de los tutores.
    3. Tras obtener una versión satisfactoria del conjunto de juegos escogidos, se realizará una prueba final sobre un grupo reducido de estudiantes de la facultad de informática que actuarán como evaluadores externos, con el fin de terminar de detectar posibles fallos que hayan sido pasados por alto durante el proceso.
- De forma simultánea con las últimas iteraciones de la fase anterior, llevaremos a cabo un proceso de investigación y búsqueda de información sobre los distintos sistemas de telemetría ya existentes en diferentes contextos. Optaremos por utilizar un sistema ya existente o desarrollar uno propio en función a los resultados de dicha investigación. En el segundo caso, será necesario implementar un

servidor para el sistema de telemetría. Una vez determinada la modalidad a emplear, se procederá a la integración del sistema de telemetría en el conjunto de juegos desarrollados.

- Antes de iniciar un primer experimento piloto, se redactará un protocolo de organización y administración de los juegos a los participantes de las sesiones, tomando como punto de partida los protocolos seguidos por el equipo responsable del estudio de referencia. En esta fase también será necesario reunir información sobre los distintos tests de inteligencia disponibles, y buscar aquellos que se adapten mejor a nuestras necesidades (idealmente aquellos empleados por el estudio original).
- Una vez satisfechos todos los requisitos técnicos del trabajo, y se haya dado por válido el protocolo pertinente, se procederá a la realización del experimento con un grupo de estudiantes voluntarios en los laboratorios de la facultad de informática.
- Finalmente procederemos a analizar los datos obtenidos, replicando las técnicas estadísticas empleadas en el artículo, con ligeras adaptaciones para acomodar los cambios en el número de juegos y en los tipos de métricas o datos a procesar.

Sin embargo, la situación actual debida a la pandemia por el COVID-19 da lugar a una serie de modificaciones en el esquema de planificación anterior, especialmente en los aspectos relacionados con la realización del experimento en un entorno físico.

- En cuanto al método de administración de los juegos a los participantes del experimento, optamos por reemplazar las sesiones presenciales por un "experimento online". Esto supone modificar el tipo de plataforma de desarrollo del conjunto de juegos, siendo ahora necesario migrar los proyectos a versiones web.
- En la misma línea, se implementará una aplicación web para albergar los tres juegos y distribuirlos de forma online, incluyendo un sistema de gestión de participantes/usuarios. Esto debe reflejarse también en la infraestructura del servidor, teniendo que realizarse los cambios pertinentes.
- Del mismo modo, al no ser ya posible la administración de los tests de inteligencia físicamente, debemos proceder a realizar un estudio de viabilidad sobre dichos tests en línea. El resultado del mismo determinará si es posible mantener de algún modo el propósito inicial del trabajo o si por el contrario resultará más razonable realizar un análisis alternativo.

Estructura del trabajo

En el capítulo 1, se explica en detalle el concepto de inteligencia desde la perspectiva de la psicología, y cómo esta puede ser medida de manera sistemática, concluyendo con una descripción del experimento inicial al respecto y las habilidades cognitivas allí medidas. A continuación, el capítulo 2 introduce las ideas fundamentales de telemetría en videojuegos, así como una comparativa de tecnologías ya existentes en el mercado para el registro de datos en juegos. En el capítulo 3, se presenta la elección de los juegos a desarrollar en el proyecto y sus detalles técnicos más relevantes. Por su parte, el capítulo 4 desarrolla los aspectos más importantes en la implementación del sistema de telemetría. Por otra parte, el capítulo 5 se centra en describir el entorno online necesario para la realización de los experimentos, así como los puntos más importantes sobre el servidor y la infraestructura web subyacente a este. Se incluye, asimismo, una visión global de los resultados obtenidos. Finalmente el capítulo 6 está dedicado a los diferentes métodos

y estrategias adoptadas a la hora de abordar el análisis de dichos resultados, desde modelos estadísticos dirigidos a encontrar un factor de inteligencia latente, a técnicas de aprendizaje automático y aprendizaje profundo para el problema de regresión ya mencionado.

Introduction

At the beginning of the 1980s, video gaming started reaching mainstream popularity as several major arcade titles and consoles were first introduced to the general public. Ever since then, video games have become not only a staple in today's entertainment industry and culture, but also a widely used tool for scientific research, education and various purposes other than pure recreation. The rise of these "serious" applications of video games came accompanied by a sparked interest in using them for the measure of cognitive abilities, resulting in different studies showing rather low correlation indices [1, 2]. Following these pioneer projects, several researchers focused on delving into the potential impact of video games in education [3], as well as on particular tasks or abilities such as mathematical [4] and spacial cognitive skills [5]. In 2009, Quiroga and her team [6] put forward the idea of a strong relationship between in-game performance and general intelligence through the use of "brain" video games involving five types of tasks: reasoning, calculus, visualization, memory and perceptual speed. A first test battery using 12 games from *Big Brain Academy* was developed in [7] by the same research group in order to assess a subset of cognitive abilities, leading to a high correlation value of 0.93. With this new focus in mind, [8] aimed to go beyond classical brain training titles by attempting to establish a correlation between custom, manually recorded metrics in commercial games (summarized by a "general video game performance" factor, gVG) and the general intelligence factor (g). A considerably satisfactory result of 0.79 was achieved.

We will henceforth be referring to this later study. There, a number of further steps were contemplated as potential improvements in order to circumvent some of its technical limitations. The perceived issues were as follows: first, the measurements used were one-dimensional, including a single variable in most cases. Secondly, they were registered manually, requiring a moderator to supervise each participant during an on-site experiment. With these two inconveniences as motivation, three of the ten games included in the test battery are implemented in this work, namely *Blek*, *Edge* and *Unpossible*, incorporating an extended data set to be registered in-game. This data is then used for a later analysis with the aim of determining its capacity to assess the cognitive abilities considered in the original paper.

In order to reproduce the original experiment's results and confirm the validity of the new metrics proposed in this work, we collect experimental data from volunteer participants. As a consequence of the COVID-19 situation, the exact conditions of the sessions differ from the environment described in the reference article. A custom telemetry system is developed in order to automatically collect the required information from game sessions, along with a web application to administer the games remotely to the players. From the obtained data we conduct an exploratory factor analysis in order to examine the relationships between the observed indicators, rather than the originally used confirmatory model.

In addition to this analysis, we propose a regression problem to determine the extent to which it is possible to predict the final result of a given participant in sessions with shorter duration than originally established in the paper. For this purpose, the information recorded from the players is transformed into time sequences data and deep learning techniques are applied, trying to generate a suitable predictor for

the task at hand. We also analyze the feasibility of other traditional machine learning methods for the problem in question.

Planning

Having introduced the main goals of the present work, we will now briefly outline the general planning scheme to be followed. Some of the points herein included are affected by the COVID-19 pandemic, as will be explained in the next section.

- In the first instance we include the study and initial contact with the original experiment [8] on which we base this work. As a first approximation, we select three games aimed at measuring each of the cognitive abilities considered in the reference study, namely fluid reasoning (Gf), visuospatial ability (Gv), and processing speed (Gs). In principle, the objective is to develop these games so that they are as faithful as possible to their commercial versions, in order to minimize discrepancies between our study and the reference one.

- The second phase of the work involves the implementation of these games, to be carried out in three fundamental stages: (1) the development of a first prototype for each of the games will start, focused on showing their basic mechanics through a reduced set of levels; these prototypes will be evaluated by the tutors of the work before advancing to the next phase. (2) After receiving approval from the tutors, an iterative development process will begin, consisting of a series of sub-phases: recording new requirements or changes in the initial ones, implementing additional levels and mechanics, and testing the results of each iteration by both developers and tutors. (3) After reaching a satisfactory version of the chosen set of games, a final test will be carried out on a small group of students from the computer science faculty who will act as external evaluators, in order to finish detecting possible bugs and usability issues that might have gone unnoticed during the development process.

- Simultaneously with the last iterations of the previous phase, we will carry out a process of research and search for information on the different telemetry systems already existing in various contexts. We will choose to use an existing system or develop our own based on the results of this research. In the second case, a server will need to be implemented to support the telemetry system. Once the modality to be implemented has been determined and the development of the chosen games is completed, we will integrate the telemetry system accordingly.

- Before starting a first pilot experiment, a protocol for the organization and administration of the games to the participants will be written, using the protocols followed by the team responsible for the original study as a reference. In this phase it will also be necessary to gather information about the different intelligence tests available, and search for those that best suit our needs, ideally those used in the original study.

- Once all technical requirements of the work have been satisfied and the relevant protocols are validated, an experiment will be carried out with a group of student volunteers in the laboratories of the computer science faculty.

- Finally, we will proceed to analyze the data obtained in the experiment, replicating the statistical techniques used in the reference article, with slight adaptations to accommodate changes and additions to the number of games and the types of metrics/data to be processed.

However, the current situation due to the COVID-19 pandemic gives rise to a series of modifications in the previous planning scheme, especially in aspects related to conducting the experiment in a physical environment.

- Regarding the method to administer the games to the participants of the experiment, we opt to replace the face-to-face sessions with an "online experiment". This involves modifying the type of development platform for the set of games, as it is now necessary to migrate the projects to web versions.
- Along the same lines, a web frontend will be implemented to host the three games and distribute them online, including a user management system. This should also be reflected in the server infrastructure, with the relevant changes having to be added.
- Similarly, since the administration of intelligence tests is no longer possible physically, we should carry out a study to check the feasibility of online intelligence tests. Its results will determine whether it is possible to somehow maintain the initial goal of this work or if, on the contrary, it will be more reasonable to perform an alternative analysis.

Outline

In chapter 1, the concept of intelligence is explained in detail from the perspective of psychology, in addition to how it can be measured in a systematic way, concluding with a description of the original experiment and the cognitive skills measured therein. Next, chapter 2 presents the fundamental ideas of telemetry in video games, as well as a comparison between existing technologies for in-game data recording. In chapter 3, the selection of the games to be developed in the project and their most relevant technical details are presented. Similarly, chapter 4 develops the most important aspects in the implementation of our custom telemetry system. Additionally, chapter 5 focuses on describing the online environment necessary to carry out the experiments, as well as the most important points about its backend and the underlying web infrastructure. It also includes a global vision of the obtained results. Finally, chapter 6 is dedicated to the different methods and strategies adopted when dealing with the analysis of these results, from statistical models aimed at finding the latent intelligence factor, to machine learning and deep learning techniques for the already stated regression problem.

Capítulo 1

Medida de la Inteligencia

1.1. ¿Qué es la inteligencia?

La inteligencia es una construcción psicológica crucial para comprender las diferencias de comportamiento humano. La mayoría de los humanos pueden percibir el mundo, almacenar información a corto y largo plazo, recuperar la información relevante cuando sea necesario, comprender y producir lenguaje, orientarse en entornos conocidos y desconocidos, hacer cálculos de altos y bajos niveles de sofisticación, etc. Estas acciones cognitivas deben coordinarse de alguna manera y la inteligencia es el factor psicológico que toma la iniciativa cuando una persona persigue este objetivo [9].

Según [10], la inteligencia "es una muy general capacidad mental que implica posibilidad de razonar, planificar, resolver problemas, aprender rápidamente, pensar de manera abstracta, comprender ideas complejas y aprender de la experiencia". Cabe mencionar también la importancia de evaluar la capacidad cognitiva a la hora de referirse a estudios de inteligencia, lo que podríamos definir como: "toda capacidad relacionada con el procesamiento de información, lo que incluiría la atención, percepción, memoria, resolución de problemas, comprensión y establecimiento de analogías, entre otras cosas". La diferencia fundamental entre estos dos grandes conceptos es que la inteligencia está más enfocada a la elección de una mejor solución para un problema, mientras que la cognición va referida al procesamiento de la información percibida.

Pese a ser la inteligencia un campo de estudio muy amplio, las características de este concepto están bien documentadas en numerosos contextos de la vida cotidiana. Su estudio ha sido tratado desde la perspectiva de, al menos, tres modelos complementarios para la investigación científica: modelos **psicométricos**, modelos **cognitivos** o de procesamiento de información, y modelos **biológicos** [9].

1.1.1. Modelos Psicométricos

Los modelos psicométricos de la inteligencia han identificado las dimensiones de variación en capacidad cognitiva. El término psicometría deriva del griego y según su origen etimológico (*psyke*, que significa alma; *metron* que es equivalente a medida; y el sufijo *-ia* que indica cualidad), podemos traducir este concepto como la medición de distintos hechos psicológicos [11].

La psicometría es la rama de la psicología orientada a la medición de los procesos psíquicos. Para ello desarrolla estudios destinados a generar métricas concretas que posibiliten cierta comparación entre las

características psicológicas de diferentes personas de manera objetiva. Estas características o aspectos psicológicos pueden ser sus habilidades, su conocimiento, su estado de opinión, la actitud que presenta, los rasgos de su personalidad e incluso sus capacidades mentales.

Es importante tener en cuenta que registrar y medir cuestiones psíquicas no resulta a priori sencillo, ya que estas características no pueden ser observadas de forma directa. La psicometría, por tanto, debe desarrollar primero métodos de evaluación fiables para obtener resultados certeros y poder cuantificar el objeto de investigación.

A su vez, la psicometría se divide en tres bloques:

- **Escalamiento**. Es la parte que da forma y desarrolla los distintos métodos que se emplean para la construcción de las escalas de tipo psicofísico o psicológico.
- **Teoría de la medición**. Es el área de la psicometría que gira en torno a la fundamentación teórica de la medición.
- **Teoría de los tests**. Versa sobre los modelos matemáticos y la lógica que se emplean para confeccionar y usar los distintos tests de evaluación.

Es común emplear los estudios y tests psicométricos en el ámbito laboral o incluso en el escolar, para obtener datos fiables acerca de la inteligencia y personalidad a la hora de contratar trabajadores en una empresa que lo requiera, o tratar de identificar niños en una escuela con una inteligencia superior al resto, por ejemplo. Es común que algunos de estos test tengan como resultado la variable conocida como *Cociente o Coeficiente Intelectual* (CI, o por sus siglas en alemán IQ), que es un estimador de la inteligencia general. Aunque esta puntuación esté internacionalmente aceptada con media de 100 y desviación típica de 15, su significado depende exclusivamente del test mediante el que se haya obtenido.

1.1.2. Modelos Cognitivos

Hemos definido capacidad cognitiva como "toda capacidad relacionada con el procesamiento de información", por lo que los modelos cognitivos o de procesamiento de información descubren los procesos cognitivos básicos relevantes para las aptitudes abordadas por los ya mencionados modelos psicométricos. Los procesos cognitivos son los que permiten que procesemos la información que nos llega a través de los sentidos, que la almacenemos, manipulemos, recuperemos, aprendamos, y que interactuemos con el mundo. Comprenden la memoria, el lenguaje, la percepción, el pensamiento y la atención, entre otros [12]. La mayoría de las acciones cotidianas llevan un procesamiento cognitivo, de ahí que cuando estas funciones sufren algún daño o no se desarrollan adecuadamente, se deteriora la capacidad para realizar determinadas actividades. Por lo tanto, los modelos cognitivos tienen como propósito explicar estos procesos a la hora de realizar una tarea.

Según [13], los modelos factoriales (psicométricos) intentan cuantificar y medir la inteligencia, pero no explican los procesos cognitivos que realiza una persona para resolver una tarea. Debido a estos procesos, dos personas pueden resolver una misma tarea de formas distintas. Podemos dar una serie de enfoques variados aplicados en estudios de modelos cognitivos para poder llegar a diferenciar las distintas maneras de resolución de tareas o problemas:

- El **enfoque cronométrico** relaciona la rapidez y eficacia de la respuesta-velocidad con la inteligencia mediante la realización de pruebas de tiempo de reacción. Con este enfoque podemos dividir las tareas en *simples* (si implican detección o respuesta inmediata, como por ejemplo apretar un

botón), *de elección* (si implican detección y discriminación entre varias respuestas) y *disyuntivas* (si se presentan dos estímulos con respuestas diferentes).

- El **enfoque de correlatos cognitivos** se centra en correlacionar aptitudes con parámetros de procesamiento de información.
- El **modelo componencial**, analiza los componentes elementales de información (codificación, comparación, traslación,...) que se utilizan al resolver problemas de razonamiento inductivo.

1.1.3. Modelos Biológicos

Para concluir, los modelos biológicos identifican las diferencias individuales en la estructura y función del cerebro, junto con factores genéticos y no genéticos. Se consideran con respecto a la variabilidad conductual tanto a nivel psicométrico como a niveles cognitivos. La mayoría de los científicos reconocen que el cerebro es el órgano donde tienen lugar los procesos biológicos relevantes para apoyar cada expresión de comportamiento inteligente. Sin embargo, el cerebro también es importante para otros fenómenos psicológicos que erróneamente se consideran independientes de la inteligencia [9].

1.1.4. ¿Cómo se puede medir la inteligencia?

Ya hemos comentado que mediante test psicométricos, dando valores a distintos factores psicológicos referidos a la inteligencia, podemos cuantificar la inteligencia en el sentido ya descrito antes. Pero ¿cómo podemos llegar a medir algo tan abstracto como es la inteligencia? También hemos hablado del estimador de inteligencia general CI o IQ, pero ¿cómo se puede llegar a obtener este valor?

De hecho, es común en resultados de test de inteligencia que se califiquen en ocasiones como imperfectos o incompletos, dado que el concepto de inteligencia es tan amplio que parece insuficiente evaluarlo a través de una sola prueba. La mayoría de tests de medición empleados sólo tienen en cuenta las capacidades lógico-matemáticas y lingüísticas. Lo más fácil sería centrarse en un único aspecto a medir, en el contexto una tarea concreta que se pueda cuantificar de algún modo y realizar varias pruebas sobre este aspecto para llevar a cabo una estimación media a partir de los resultados de cada prueba. Por ejemplo, si queremos realizar una evaluación sobre la memoria, podríamos contabilizar el tiempo empleado para recordar una imagen o sonido. Ahora bien, si mediante un test realizamos varias pruebas referidas a diversos factores psicológicos (normalmente cada prueba sólo abarca un factor), podemos obtener resultados y mediciones de varios factores y calcular, mediante un método o algoritmo común, una métrica que englobe todos los resultados obtenidos. Esto sería lo más parecido a evaluar la inteligencia global mediante un test, pero como ya se ha mencionado, es muy probable que varios factores de la inteligencia no se hayan tenido en cuenta a la hora de realizar dicha evaluación.

Teniendo en cuenta todo lo anterior, la respuesta pues a la pregunta inicial es que, efectivamente, es posible medir la inteligencia mediante pruebas o tests, y estos serán más fiables y sofisticados cuantas más pruebas tengan y cuantos más aspectos o factores de la inteligencia abarquen. Para ser un test completo, debe evaluar los 8 factores del segundo estrato según el modelo Cattell-Horn-Carroll (CHC) [14], que son:

- Inteligencia fluida (Gf)
- Inteligencia cristalizada (Gc)
- Memoria general y aprendizaje (Gy)

- Amplia capacidad de recuperación (Gr)
- Tratamiento visual (Gv)
- Tratamiento auditivo (Gu)
- Velocidad cognitiva amplia(Gs)
- Velocidad de procesamiento (Gt)

De estos factores se deduce la capacidad mental general (factor g). La mejor estimación de este último es el razonamiento lógico abstracto (Gf).

1.2. Estudio de la inteligencia en videojuegos

De acuerdo con [15], los estudios en inteligencia en los videojuegos se llevan desarrollando e investigando desde el año 1986, todo ello entorno a la gran cantidad de títulos existentes que requieren razonamiento, planificación, aprendizaje y realización de pruebas lógicas o desafíos. Todas estas características se incluyen en la definición ya mencionada anteriormente de la inteligencia.

Según el enfoque científico, las características superficiales de las situaciones que llevan a estudiar la inteligencia y la capacidad cognitiva son relativamente irrelevantes, y por ello es posible utilizar videojuegos para obtener medidas de interés en este ámbito. Resulta interesante evaluar las distintas formas de percepción y resolución de problemas o puzles en videojuegos. Dicha evaluación puede ayudar no sólo en estudios posteriores al desarrollo del mismo, sino también en el proceso de diseño de nuevos niveles o acertijos, o de modificación de los mismos con el propósito de ajustar lo más posible la curva de dificultad a lo deseable para el videojuego en cuestión.

Definimos *curva de dificultad en un videojuego* según [16] como la evolución del progreso o el aprendizaje del jugador y la necesidad de nuevas habilidades dentro de un videojuego para poder enfrentarse a nuevos desafíos. Esta curva es muy importante en prácticamente todos los videojuegos de cualquier género, especialmente en los que nos encontramos en este estudio concreto. Una curva de aprendizaje mal ajustada (muy pronunciada) puede derivar en la frustración, y una curva muy plana genera aburrimiento en los jugadores, pudiendo influir esto significativamente en la evaluación de los resultados. Las nuevas técnicas o habilidades de un videojuego deben introducirse de forma progresiva de modo que, idóneamente, el jugador tenga tiempo para aprenderlas y manejarlas en problemas cuya resolución exija el dominio de las mismas. Por ello es muy importante el uso de tutoriales para que el usuario tenga un primer paso de familiarización con el juego. En ciertos escenarios, puede darse el caso de que un jugador no sepa manejarse correctamente en un juego si no comprende bien las primeras mecánicas que se explican en el tutorial, lo cual dificulta la evaluación del sujeto correspondiente en el juego. Por lo general, si un jugador no comprende las mecánicas simples, no será capaz de aprender otras posteriores, habitualmente más complicadas.

La forma de medir la inteligencia en los videojuegos puede adoptar diferentes formas. Del mismo modo, es posible utilizar una gran variedad de métodos y algoritmos para el análisis de dichas medidas. De hecho, existen juegos cuyo propósito principal es precisamente tratar de obtener una métrica de la inteligencia de sus jugadores, evaluando un conjunto limitado de sus procesos cognitivos por separado. Un buen ejemplo de este tipo de videojuegos es *Big Brain Academy* [17], en el que se usa una métrica interna propia para proporcionar una estimación aproximada de la inteligencia a partir de una serie de pruebas. Cada una de estas pruebas entra dentro de una categoría cognitiva distinta (Lógica, Memoria, Análisis,

Cálculo y Percepción), y al sumar las valoraciones de todas ellas se obtendría un pentágono que actúa como representación gráfica global de la inteligencia (insistiendo aquí en que esta es una representación interna, no estandarizada, de uso único en el videojuego).

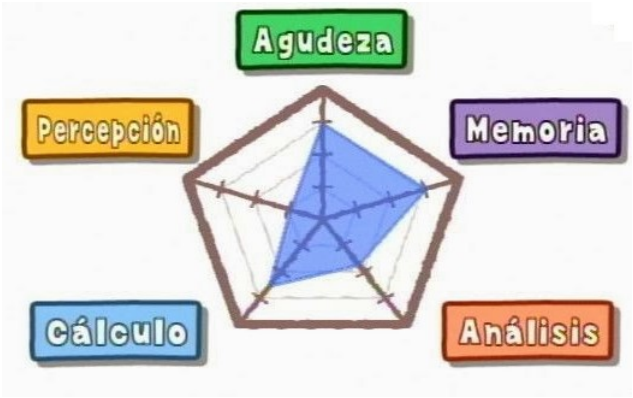


Figura 1.1: Ejemplo de medición de la inteligencia en Big Brain Academy

Además, en este ejemplo vemos clara y visualmente la importancia de tener que separar varios factores y evaluarlos por separado, aunque estén correlacionados entre sí, si queremos medir algo tan global y general como es la inteligencia.

La evaluación es relativamente sencilla cuando los juegos o desafíos están directamente relacionados con alguno de los procesos cognitivos, como ocurre en Big Brain Academy, pero ¿qué ocurre cuando no es así? Pero también puede interesar realizar estudios sobre videojuegos que no tienen este aspecto, como es nuestro caso. Además, pese a todo lo mencionado sobre la generalización de la curva de aprendizaje y la influencia en la evaluación de la inteligencia, los videojuegos engloban una gran variedad de géneros. Esto significa que, aunque en general los procesos cognitivos implican adquisición y comprensión de conocimiento, toma de decisiones y resolución de problemas, los requerimientos cognitivos son diferentes en cuanto a planificación, velocidad, capacidad psicomotriz, y muchos más aspectos a tener en cuenta. Incluso en el caso de este proyecto, los 3 videojuegos podrían entrar perfectamente en la misma categoría de Puzles o Plataformas, y aún así comprenden diferentes procesos cognitivos destacables, ya mencionados anteriormente:

- En Blek, el razonamiento fluido o la capacidad para resolver problemas nuevos.
- En Edge, destaca la capacidad visoespacial en 3 dimensiones.
- En Unpossible, interesa estudiar la velocidad de procesamiento.

Los resultados de este tipo de estudios se verán altamente influenciados por la habilidad del usuario en el videojuego en cuestión. Como ya hemos mencionado, en un juego en el que se van incluyendo mecánicas nuevas, el usuario que ya las conozca tendrá una ventaja sobre el resto (a lo que podemos llamar experiencia previa). Esta ventaja podrá ser mayor o menor, dependiendo del tipo de videojuego y de sus mecánicas. A esto se le añade que, la habilidad inicial en un juego dado no se ve únicamente influida por el conocimiento del mismo, sino también por la experiencia previa con juegos de géneros afines que conlleven el uso de patrones o mecánicas similares. Por ello es importante, a la hora de evaluar los resultados de un sujeto, tener en cuenta tanto si anteriormente el usuario jugó a ese mismo juego, como la cantidad de tiempo empleada en títulos de diferentes géneros, generalmente en horas por semana.

Este tipo de evaluaciones se realizan mediante cuestionarios de experiencia de juego, con lo que se puede clasificar los usuarios según la experiencia general que pueda influir para el estudio. En nuestro caso es importante conocer este tipo de datos.

Resulta relevante tener en cuenta que, si deseáramos medir el rendimiento de los usuarios dentro del videojuego (el progreso que realiza el jugador), no sería posible hacer uso del tiempo invertido en juegos digitales como factor de normalización de los datos. Esto se debe a que, en principio, la cantidad de tiempo dedicado a una actividad no supone ninguna garantía de éxito o rendimiento superiores. En términos más concretos, no se puede asumir una relación lineal entre tiempo de práctica y resultado obtenido para tareas de este tipo.

1.3. Estudio de referencia

Asumimos como punto de partida en el proyecto un estudio ya realizado por investigadores en psicología [8], que describe de manera sistemática una investigación basada en correlaciones a nivel latente entre inteligencia y videojuegos. Su objetivo principal fue analizar si el rendimiento en los videojuegos se correlaciona con el resultado obtenido de los tests de inteligencia. Para eso, se utilizaron los datos de 134 personas que se ofrecieron voluntarios para completar 10 juegos de diferentes géneros dentro de un entorno controlado.

De acuerdo con [15], estos juegos se centran en el estudio de las siguientes tres habilidades cognitivas dentro del segundo estrato del modelo CHC:

- **Gf (fluid reasoning o razonamiento fluido)**: Es la capacidad de pensar con flexibilidad (fluidez) a la hora de resolver problemas. Esta capacidad permite analizar nuevas tareas, razonarlas o identificar conceptos que permitan una extrapolación a problemas pasados para resolverla. La podemos observar en juegos que requieran técnicas o patrones para poder avanzar en la resolución de nuevos problemas, por ejemplo, juegos de escape o puzles.
- **Gv (visuospatial ability o habilidad visoespacial)**: Es la capacidad de representar, analizar y manipular objetos en la mente. En relación a las habilidades visoespaciales manejamos dos conceptos importantes, como son:
  - **Relación espacial**: capacidad de representar y manejar mentalmente objetos en dos dimensiones.
  - **Visualización espacial**: capacidad de representar y manipular mentalmente objetos en tres dimensiones.

  El Sokoban o juegos de plataformas tanto 2D como 3D serían un ejemplo de videojuegos donde se pone a prueba esta habilidad.
- **Gs (processing speed o velocidad de procesamiento)**: Referido a la capacidad de poder realizar acciones con mayor velocidad cuando son aprendidas por repetición. En todos aquellos juegos en los que la velocidad en incremento supone la mayor dificultad de juego se requiere esta habilidad. Unos ejemplos podrían ser el Tetris o el Flappy Bird.

El desarrollo del estudio consistió en registrar, para cada uno de los 10 juegos involucrados, las variables que a priori podrían influir en cada uno de estos factores cognitivos y realizar mediciones y cálculos sobre ellos. Después de completar los juegos, los participantes debían realizar 6 tests de aptitudes

(dos tests por cada factor aptitudinal considerado) y un cuestionario de hábito de videojuegos, esenciales para la extracción de resultados.

El rendimiento en los juegos se correlacionó con las medidas estándar de cada uno de estos factores. Los resultados revelaron un valor de correlación de 0.79 entre factores latentes que representan inteligencia general (g) y rendimiento general de videojuegos (gVG). Este hallazgo llevó a la conclusión de que las pruebas de inteligencia y los videojuegos estaban ambos respaldados por procesos cognitivos compartidos, y que los juegos mentales no son el único género capaz de producir medidas de rendimiento comparables a las pruebas estandarizadas de inteligencia. Desde una perspectiva teórica, el resultado observado respalda el principio de la indiferencia del indicador ([18] [19]) que se ha abordado en la investigación de inteligencia a lo largo de décadas.

Resumen

- En este capítulo hemos explicado la dificultad que supone medir algo tan complejo como es la inteligencia, tanto dentro como fuera del ámbito de videojuegos.
- Los factores cognitivos son las subdivisiones que podemos analizar de la inteligencia por separado para posteriormente obtener un valor global común a todos esos factores, que sería la medición de la inteligencia que buscamos.
- La forma más común de medir la inteligencia es mediante test y, en el caso de videojuegos, se le añade un cuestionario de hábito de juegos que ayuda en gran medida a interpretar los resultados.
- Para nuestro trabajo, tomamos como referencia un estudio ya realizado de psicología y videojuegos, con el objetivo principal de emplear las mismas técnicas, mejorarlas y comparar resultados.
- En el próximo capítulo, explicaremos el estudio sobre sistemas de telemetría en videojuegos necesario para el registro de datos que queremos analizar en nuestros juegos.

Capítulo 2

Telemetría y analíticas en videojuegos

Comenzaremos el capítulo realizando una breve introducción a algunas de las ideas básicas relacionadas con los conceptos de *telemetría*, métricas y análisis en juegos. A continuación, estableceremos el proceso general de analíticas en videojuegos y desarrollaremos un caso de estudio ilustrativo a modo de ejemplo. En última instancia, se expondrán algunos aspectos particulares de la telemetría en juegos serios, y exploraremos tres sistemas ya existentes para el análisis y gestión de eventos (Google Analytics [20], Firebase Analytics [21] y Unity Analytics [22]).

2.1. Telemetría, métricas y el proceso de analíticas

Comenzaremos esta sección definiendo el concepto que da nombre al capítulo. De acuerdo con [23], la telemetría puede definirse formalmente como "el conjunto de datos asociados a sucesos de juego específicos, el estado de un juego, u otros parámetros de interés". De este modo, la telemetría engloba todo mecanismo de especificación, captura o recopilación de cualquier tipo de suceso que pueda ocurrir en un juego, producido tanto por un usuario del mismo, como por el juego en sí. Resulta importante aquí distinguir entre telemetría y métricas de juego: la telemetría en sí sólo aporta datos en crudo, susceptibles a ser transformados en métricas, o medidas interpretables de cualquier elemento del juego. Cabe mencionar además que existen métricas sobre juegos no registradas por medio de telemetría, como puedan ser los tests de opinión con los usuarios.

Antes de pasar a hablar de los sistemas de telemetría, introduzcamos una clasificación común de los distintos tipos de métricas que pueden encontrarse en el contexto de videojuegos en general. La clasificación incluida aquí proviene de [24] y considera tres clases fundamentales de métricas:

1. **Métricas de usuario o jugador**. Son las métricas relacionadas con las personas o usuarios que toman el papel de jugadores de una aplicación, desde la perspectiva dual que los considera tanto *clientes* (fuentes de ingresos para una compañía) como *jugadores*, que se comportan de un modo específico al interactuar con los juegos. La primera perspectiva se utiliza por medio de métricas de finalidad económica o relacionadas con ingresos, mientras que la segunda tiene como objeto de estudio el comportamiento y la interacción de los usuarios con el sistema de juego (y con otros jugadores, en los casos correspondientes). Estas son las métricas tratadas en este proyecto.
2. **Métricas de rendimiento**. Son las métricas relacionadas con el rendimiento de la infraestructura técnica sobre la que se apoya un juego, tanto a nivel de hardware como de software. Algunos

ejemplos típicos son el *framerate* al que un juego se ejecuta en el dispositivo de un jugador, o la estabilidad de un servidor que ofrece un juego o un servicio en linea. Estas métricas prestan además especial atención al impacto de los cambios en un juego sobre el rendimiento del mismo, así como a los distintos *bugs* o fallos técnicos detectados en una determinada versión.

3. **Métricas de proceso**. Son aquellas asociadas al proceso de ingeniería de software asociado al desarrollo de un juego. Estas pueden incluir medidas para monitorizar el proceso de desarrollo desde la perspectiva por ejemplo del tamaño de las tareas asignadas a un equipo, o considerando variables como el tiempo de respuesta promedio a la hora de entregar nuevo contenido. Cabe mencionar que estas métricas difieren de las anteriores en tanto a que no están relacionadas de ningún modo con la recopilación de datos *in-game*, al ser métricas más bien de "gestión".

Dentro de las métricas de usuario o jugador podemos realizar una segunda clasificación fijándonos en los distintos niveles de aplicabilidad de las mismas:

- **Métricas genéricas**, válidas en principio para cualquier tipo de juego digital, por ejemplo el tiempo total de juego.
- **Métricas específicas de género**, aplicables dentro de un género específico; por ejemplo en un juego de tipo Puzle, número de items completados con éxito.
- **Métricas específicas de juego**, dirigidas a un juego específico, como número de cajas empujadas en Sokoban, o número de fantasmas comidos en Pacman.

Las métricas de juego pueden subdividirse de manera alternativa en otro conjunto adicional de tres categorías. En primer lugar tenemos las *métricas "en juego"*, que cubren todas las acciones internas realizadas por un usuario y sus comportamientos, incluyendo los económicos (compras) y la interacción con objetos del juego. En segundo lugar, las *métricas de interfaz* incluyen las acciones del usuario sobre los menús e interfaces del sistema, como por ejemplo modificar la configuración del teclado. Por último tenemos las llamadas *métricas de sistema*, que cubren todas las distintas acciones llevadas a cabo por los motores de juego y asociados, como una IA para un NPC, en respuesta a las propias acciones del usuario. El fin de estas clasificaciones es poner de manifiesto la gran variedad de posibles métricas a las que se puede destinar el procesamiento de datos recopilados por medio de telemetría.

Las métricas que consideraremos en este trabajo estarán construidas a partir de la información proporcionada por un *sistema de telemetría* o *tracker*. Este será el encargado de recopilar distintos datos del juego correspondiente a lo largo de cada partida de interés. Como ya se mencionó anteriormente, los datos generados por este sistema son "crudos", asociados a distintos atributos o configuraciones de objetos del juego, y necesitarán un cierto procesamiento previo antes de convertirse finalmente en métricas como tal. Llamaremos *instrumentalización* al proceso de incorporar una infraestructura de recogida de datos por código en un juego dado. Esta instrumentalización puede adoptar formas muy diversas, y generalmente requiere una serie de decisiones en cuanto al método de transmisión de la información relevante, o el tipo de datos a transmitir por medio del sistema de telemetría.

La transmisión de información a través de un tracker puede ocurrir en principio de dos formas fundamentales consideradas en [25]:

1. **Eventos**: Un evento en este contexto se puede definir como un objeto que contiene información sobre una acción (el *qué* sucede en el juego, por ejemplo, "muerte de un jugador"), el *cuándo* ocurre dicha acción (generalmente especificado por un cierto campo llamado *timestamp*), y un estado

asociado a la misma, que responde a cualquier otra pregunta de interés sobre esta (por ejemplo, el nivel en el que ocurrió la acción, el nombre del jugador, etc). Idealmente contienen la información necesaria para entender un "suceso atómico".

2. **Muestreo**: En lugar de enviar a un servidor un evento o colección de eventos que representen los sucesos que ocurren, en un muestreo se manda periódicamente un estado del juego, dejando a la fase de análisis interpretar qué ha sucedido entre medias, en los casos en los que sea relevante. Resulta importante comentar aquí que este método sólo tiene sentido en principio cuando el sistema dispone de un conjunto de variables de estado más o menos fijas que van cambiando de valor con el paso del tiempo, como puede ser la posición de un personaje en un nivel para juegos 3D.

[25] hace énfasis adicional en un aspecto interesante de configuración del sistema de telemetría, al que llama "iniciación". Con esto se refiere a la capacidad por parte de un gestor de la aplicación de activar o desactivar el proceso de registro de datos en función de la situación real de la misma. Por ejemplo, monitorizar la posición de un jugador puede dejar de resultar relevante una vez se da por cerrado el diseño final de un entorno, y en estos casos tener la capacidad de desactivar la recopilación de datos para estas trayectorias puede resultar conveniente.

Volviendo a la visión más general de la recogida de datos, tenemos en verdad dos posibles opciones. La primera es transferir la información a través del tracker de alguna de las formas ya introducidas anteriormente. Esto no es otra cosa que un registro fino de un comportamiento "momento a momento", en el caso del muestreo, o de sucesos registrados en momentos de interés, en el caso de los eventos. En esta opción, el código añadido al juego se centra en la toma de datos, y no en un procesamiento de los mismos de ninguna manera, por lo que estamos hablando de telemetría pura. La segunda opción es adoptar una estrategia agregada desde el propio juego, queriendo decir con esto que en lugar de enviar sucesiones de eventos o de estados, se opta por transmitir la información ya procesada al servidor. Esto es, ya transformada en métricas interpretadas. En cualquier caso, los datos correspondientes pueden anotarse en un fichero tipo log local, o ir mandándose por red de manera simultánea. Existen numerosas estrategias de envío, en el sentido técnico de la palabra, pero no ahondaremos en ellas en este capítulo.

Pongamos un ejemplo ilustrativo. Supongamos que estamos interesados en monitorizar datos relativos al movimiento de un jugador en un juego con niveles 2D en los que puede desplazarse usando botones direccionales. Una primera estrategia puramente por eventos podría ser considerar el envío de sucesos del tipo "Iniciado desplazamiento en dirección X" y "Finalizado desplazamiento en dirección X", cada uno con su timestamp correspondiente. De este modo, somos capaces de reconstruir completamente el movimiento de un jugador a lo largo de su paso por el nivel, y podemos inferir cualquier tipo de métrica al respecto tras cierto procesamiento posterior. Algo similar puede conseguirse por medio de un proceso de muestreo, registrando el estado del juego en intervalos fijos lo suficientemente pequeños. Esto último nos permite reconstruir "la mayoría" del comportamiento del jugador, en el sentido de que las acciones y cambios de estado dejan de ser continuos y pasan a verse restringidos a un dominio discreto. Por ejemplo, un desplazamiento a la derecha iniciado con timestamp 1.5 y finalizado con timestamp 2.3, en un sistema de muestreo que almacena el estado del juego en intervalos de un segundo, podría determinar un cambio de posición entre los segundos 1 y 2, y entre el 2 y el 3, pero no sería capaz de decirnos los detalles exactos del mismo (no se sabe que el movimiento duró 0.8 segundos, por ejemplo). Por otro lado, desde una estrategia agregada necesitamos tener claro qué queremos medir de antemano, para poder programar la lógica de registro correspondiente en el lado del juego.

Los eventos representan un mecanismo conveniente para gestionar las trazas de comportamiento e interacción del usuario con nuestro sistema, así como una base potente para el tratamiento sistemático en un análisis posterior. Proporcionan una gran versatilidad a la hora de realizar inferencias sobre lo

ocurrido en un juego, no producen pérdidas de información (si lanzamos un evento por cada suceso representativo durante la partida) y permiten decidir a posteriori qué queremos analizar o medir sin restricciones, pudiendo considerar tantas métricas nuevas como sea necesario, siempre y cuando estas puedan ser inferidas desde una traza de eventos. Lo mismo puede decirse, hasta cierto punto, del registro por muestreo. Sin embargo, debemos tener en cuenta en todo momento que esta conveniencia ocurre en el lado de la generación, pero no tanto en el de la extracción de conclusiones o del análisis. Estas estrategias pueden llegar a producir una gran cantidad de datos (por lo general muy superior a la que puede encontrarse en el caso agregado), e insistimos en que son trazas de sucesos o estados sin procesar, por lo que en principio es necesario realizar un esfuerzo de análisis posterior para extraer conclusiones adecuadas de los mismos. El caso agregado, por el contrario, requiere tomar primero una decisión sobre lo que se pretende medir, pero una vez implementado tiene la ventaja de que los datos así enviados están ya transformados en métricas de interés, con dimensiones normalmente bastante inferiores a las de los casos crudos.

Típicamente, si lo que se pretende es que el sistema de telemetría de un juego sea genérico, de cara a poderlo reutilizar para otras situaciones en el futuro, los datos agregados pueden llegar a ser mucho más variados y específicos en cuanto a forma que el envío de "sucesos atómicos" que ocurran en un juego, siendo estos últimos más sencillos de generalizar. Debido a esto la aproximación habitual es simplificar la toma de datos en la telemetría, a costa de mandar más información en forma de estos sucesos atómicos, y que la carga del análisis y la inferencia de métricas quede en el lado de un sistema de analíticas en el servidor, y no en el propio juego.

Habiendo introducido diversos consideraciones de interés general en el ámbito de la telemetría y las métricas, pasamos a comentar una posible metodología de referencia en la que contextualizar los conceptos vistos hasta ahora dentro del marco más general de las analíticas en juegos. El procedimiento a seguir a la hora de realizar estas analíticas suele basarse en un sistema comúnmente conocido como *descubrimiento de conocimiento (knowledge discovery)* en datos, de tipo cíclico e iterativo. Este se define como una secuencia de fases a seguir en un proceso en el que interesa deducir información relevante a partir de un conjunto de datos, refiriéndonos aquí de nuevo a [25]). Estas fases se pueden sintetizar de la siguiente manera:

1. **Definición de atributos**. En primera instancia se definen los objetivos de la iteración, los requisitos, y el resultado que debe satisfacer el proceso de descubrimiento de conocimiento. En esta etapa se determina el tipo de transmisión de datos a implementar (eventos, muestreo), se seleccionan los atributos del usuario a monitorizar, y se definen los dominios para cada atributo.

2. **Adquisición de datos**. Las definiciones anteriores han de implementarse en el sistema de telemetría empleado por la empresa o equipo responsable de la investigación, que puede ser un sistema externo a la organización, o uno propio desarrollado por la misma. Al finalizar el proceso de desarrollo (o de actualización del sistema previo en función a nuevos requisitos o atributos), se procede a recolectar nuevos datos administrando la aplicación a un conjunto de usuarios.

3. **Preprocesamiento de datos**. Durante este paso, la información telemétrica nueva es transformada en datos manejables y cargada en una base de datos (SQL o NoSQL en función de las necesidades del proyecto), desde la que es accesible para el análisis. Se realiza un proceso de "limpieza" de datos, posiblemente eliminando muestras no válidas.

4. **Desarrollo de métricas**. Tras el preprocesamiento, los datos de atributos se transforman en variables y métricas (variables agregadas, datos inferidos, mapas de calor, etc) como paso previo a un análisis más especializado.

5. **Análisis y evaluación**. Durante esta etapa se ejecuta el análisis escogido y se genera un modelo de los resultados, que es posteriormente evaluado para comprobar que cumple los objetivos requeridos y especificados durante las primeras fases.
6. **Visualización, informe**. Si fuera necesario, los resultados obtenidos se representan de manera visual con el fin de facilitar el proceso de comprensión de los mismos. Dichos resultados son presentados entonces a un diseñador, desarrollador, o cualquier persona que pueda tomar decisiones para aplicar el conocimiento adquirido en la modificación o mejora del sistema (por ejemplo, rediseñando un nivel, alterando los atributos a registrar, etc.).
7. **Implantación**. El conocimiento adquirido es implantado en la organización, lo cual suele iniciar una nueva iteración del proceso de descubrimiento de conocimiento.

Figura 2.1: Proceso de *knowledge discovery* en analíticas de juego

Veremos en la siguiente sección cómo este proceso de *knowledge discovery* se aplica, de forma implícita, en el contexto de un juego real.

2.2. Un caso de estudio

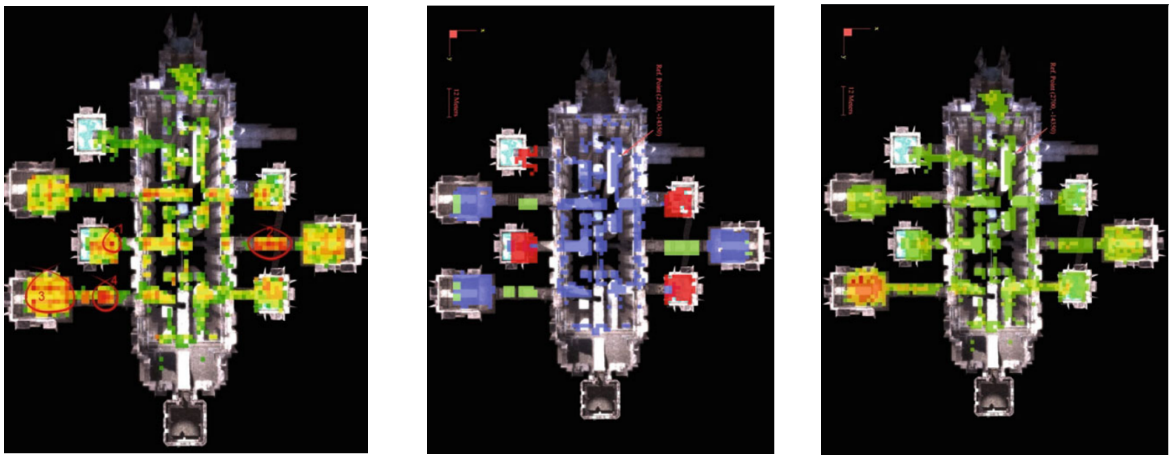
Resumamos pues los resultados clave de un caso de estudio para un juego comercial. En toda esta sección nos referiremos de forma exclusiva a [26] (capítulo 14), y nos limitaremos a comentar las peculiaridades del trabajo llevado a cabo en el mismo.

El estudio al que nos referimos fue conducido de forma posterior al lanzamiento del juego *Tomb Raider: Underworld*, y está centrado fundamentalmente en el análisis del desafío supuesto por las distintas secciones del título. De este modo, el objetivo de la investigación fue tratar de extraer conclusiones relevantes acerca del grado de dificultad a lo largo del juego, así como sobre las potenciales razones subyacentes a dicha dificultad. La forma escogida como primera aproximación a este problema fue considerar

las localizaciones y las causas de las muertes de los jugadores en distintos puntos del juego. El razonamiento tras esta idea es que investigar aquellas áreas donde los jugadores mueren de manera consistente puede poner de manifiesto una cierta falta de balance en cuanto a desafío planteado por las mismas. El entendimiento de las razones por las que esto puede llegar a ocurrir proporciona información valiosa sobre potenciales áreas problemáticas, y ayuda a diseñar nuevas áreas en las que en principio no se den estas dificultades.

Por lo general, este tipo de estudios se puede realizar tanto durante la producción como tras el lanzamiento de un título dado. En el segundo caso, las lecciones aprendidas pueden tenerse en consideración a la hora de lanzar un posible parche que solvente ciertos problemas, o de cara a producciones futuras.

Cada uno de los niveles en *Tomb Raider: Underworld* se compone de una serie de "unidades de mapa" para propósitos de logging de sucesos y métricas. La idea entonces es tratar de analizar los patrones de muertes sobre mapas dados, y para ello la estrategia empleada fue la elaboración de diversos *mapas de calor*. Un mapa de calor puede definirse como un sistema de visualización basado en cuadrículas, destinado a mostrar con qué frecuencia sucede un evento en cada una de las regiones de un entorno dado. En este caso, se dividió cada nivel en cuadrículas con un tamaño de celda suficientemente fino, y se elaboró un mapa de calor por cada tipo de evento asociado a la muerte de un jugador (un mapa por causa de muerte). De este modo, cada celda del mapa contenía una cuenta de cuántas muertes por la razón correspondiente habían tenido lugar en la misma.



(a) Muertes generales. (b) Tipos de muertes. (c) Número de causas.

Figura 2.2: Mapas de calor con información sobre las muertes de los jugadores

Presentamos aquí tres ejemplos de mapas agregados sobre un nivel concreto del juego; esto es, combinando información de ciertas capas dadas por mapas de calor de una única causa de muerte. La figura 2.2a muestra un mapa agregado de total de muertes por celda, considerando cualquier causa de muerte sin restricción. Las celdas verdes representan un menor número de muertes, mientras que las rojas indican una concentración importante. La figura 2.2b muestra las distribuciones de tres tipos de muertes aisladas (esto es, que no se dan simultáneamente en una misma celda). Estas son caídas (en verde), ahogamiento (en azul) y trampas (en rojo). Por último, la figura 2.2c muestra un mapa por número de distintos tipos de muerte sucedidas en cada casilla. De este modo, las celdas rojas representan áreas donde llegan a suceder hasta 6 tipos distintos de muertes, mientras que las verdes sólo simbolizan de una a dos.

Llegados a este punto, se pudo observar que por lo general las zonas con mayores concentraciones de muertes totales eran también aquellas con mayor número de causas de muerte distintas. En particular,

se detectaron cuatro zonas especialmente problemáticas, redondeadas en la figura 2.2a. El área 1 en esta imagen se corresponde con una región de tamaño relativamente pequeño, en la que se concentran dos tipos predominantes de muertes: muerte por ataque de un enemigo específico y muerte por caída en un salto. Una vez detectado una situación así, resulta importante estudiar si esta supone un problema (reduce el grado de satisfacción del jugador con el nivel) o si por el contrario se ve como un desafío aceptable. En principio, la única forma de comprobar esto es mediante encuestas de opinión a los jugadores sobre el tema. Si los jugadores resultan no estar satisfechos con el nivel de desafío de esta zona, el razonamiento previo nos da posibles opciones para solventar el problema, desde facilitar el salto de dicha zona a eliminar el enemigo. Un análisis semejante puede aplicarse al resto de zonas problemáticas. Del mismo modo, una elevada concentración de muertes en una región dada puede confirmarse como un problema importante si no se corresponde con la intención original del diseñador, sin ser tan esencial en este caso el feedback de los jugadores.

Habiendo desarrollado este caso, pasamos a comentar muy brevemente algunas de las peculiaridades de los sistemas de telemetría en los llamados juegos serios.

2.3. Telemetría en juegos serios

Con juegos serios (*serious games* en inglés), nos referimos a cualquier juego diseñado con un objetivo principal que no sea la diversión o el entretenimiento puros. En este ámbito, y en particular centrándonos aquí en aquellos juegos con cierto factor formativo o de evaluación, la motivación de esta colección sistemática de datos no es otra que el tratar de desarrollar medidas cognitivas de un jugador, que lleguen a ser lo suficientemente significativas en los siguientes sentidos:

1. Las medidas deben ayudar a los investigadores o desarrolladores a extraer conclusiones que expliquen el comportamiento o rendimiento de los jugadores analizados.

2. Deben exhibir alguna clase de relación con ciertos tipos de medidas complementarias. Estas pueden ser de naturaleza diversa, desde el nivel de conocimiento de una materia, si se trata por ejemplo de un juego destinado a la evaluación escolar, hasta el grado de destreza con distintos aspectos del videojuego, pasando por capacidades o cualidades intrínsecas del jugador. En nuestro caso particular, estaremos interesados en obtener relaciones con medidas de inteligencia.

Este tipo de captura sistemática de datos en videojuegos está tomando un papel cada vez más recurrente en diversos campos, como la educación o la formación médica, por poner algunos ejemplos. Refiriéndonos de nuevo a [23], en el caso de la educación, por ejemplo, la idea es que "los juegos pueden proporcionar una visión sobre los estados cognitivos de los estudiantes basándose en su comportamiento en los mismos." Ya hemos visto en la sección anterior que esta idea se puede abordar desde la perspectiva del registro de grano fino de un comportamiento "momento a momento" del jugador ("movimiento en la dirección X", "recogido objeto Y"), o desde una estrategia más agregada en la que interesa analizar el rendimiento general (resumen del nivel máximo alcanzado, número de muertes, etc). En cualquier caso, el supuesto es siempre que aquello que los jugadores hacen en un punto específico del juego es un reflejo parcial o directo de sus actuales procesos cognitivos [23]. Incluimos aquí capacidad de resolución de problemas, conocimiento sobre un campo, capacidad de toma de decisiones, etc.

Uno de los usos de la telemetría que más nos concierne en este documento es el de la medida de cualidades o características del usuario que no dependen del juego en sí. Un caso de estudio común puede encontrarse, de nuevo, en el ámbito educativo, donde la telemetría guiada por eventos permite

una medida *in situ* del rendimiento de un estudiante durante su paso por el juego, en contraste con medidas *ex-situ* que se limitan a tratar de comprender el efecto de dicha interacción como si el juego fuera una "caja negra". De este modo, ambas perspectivas resultan ser complementarias, queriendo decir con esto que mientras que una evalúa el impacto del sistema sobre el estudiante, otra busca entender cómo se ha llegado a producir dicho impacto desde las acciones registradas. Pensemos a modo ilustrativo en un experimento que consiste en administrar un juego educativo a un grupo de alumnos de primaria. Un estudio *ex-situ* en este caso podría centrarse en estudiar el efecto del juego sobre los usuarios, por ejemplo realizando pruebas de competencia en la materia antes y después del experimento, mientras que un estudio *in-situ* se podría centrar en identificar los comportamientos dentro del juego que justifican la progresión (o estancamiento) de los estudiantes durante el mismo, o incluso tratar de determinar distintos tipos de patrones de aprendizaje en el colectivo.

Presentamos ahora una serie de guías generales a seguir a la hora de diseñar sistemas de telemetría para juegos serios, en la línea de las pautas de [23], si bien muchas ideas aquí expuestas podrían aplicarse también en el contexto de otros tipos de juegos más centrados en el entretenimiento.

1. Hacer énfasis en comportamientos que reflejen el uso de demandas cognitivas. Con esto queremos decir que, en la medida de lo posible, los eventos registrados deben proporcionar información sobre demandas cognitivas sobre el jugador, incluyendo también los "fracasos" en la aplicación de las mismas. A modo de ejemplo, si un juego pretende evaluar la capacidad de un estudiante para resolver ciertas operaciones matemáticas, un evento interesante puede ser la acción de completar una operación con éxito, puesto que idealmente los diseños de niveles están pensados para que un jugador sólo pueda superar un obstáculo si tiene un cierto dominio sobre la situación. Resulta sin embargo igualmente legítimo anotar acciones fallidas, pues estas pueden aportar información esencial que permita explicar las razones por las que un jugador actúa como actúa, o proporcionar datos adicionales sobre su forma de abordar el problema.

2. Registrar los datos en el *tamaño de grano* más fino que resulte utilizable. Esto quiere decir que los eventos deben poder ser interpretados de manera relativamente aislada. Por ejemplo, la acción "Tecla pulsada" (sin parámetros) no aporta por lo general ningún tipo de información útil sobre el comportamiento del jugador, al no saber qué tecla fue pulsada o a qué instrucción se refería, mientras que al añadirle un parámetro que aporte información sobre qué tecla fue, o la acción que codificaba esta, sí que resulta usable.

3. Registrar *descripciones de comportamiento*, en lugar de inferencias sobre dicho comportamiento. Esto se propone por una cuestión de versatilidad y claridad: un evento "pulsar sobre el botón de menú" ofrece una información sin ambigüedades, que permite interpretarlo en el futuro como el analista de datos considere adecuado, mientras que un evento "saltó 6 veces" elimina parte de los datos que podrían haber resultado interesantes (como por ejemplo el *timestamp* en el que ocurrió cada salto para poder inferir cuánto tardó el jugador en realizar dichas acciones). Por lo general, si tratamos de limitarnos a acciones atómicas interpretables por sí mismas, permitiremos una mayor versatilidad de análisis en la fase de procesamiento.

Notamos que los puntos 2 y 3 de las indicaciones anteriores resumen buena parte de los comentarios sobre la recogida de eventos desarrollada en la sección anterior para juegos en general.

Con estas nociones, adoptamos una estrategia "híbrida". En última instancia, este trabajo pretende hacer uso de juegos comerciales destinados al entretenimiento, pero con objetivos más afines a los de los juegos serios. Por esta razón, trataremos de atenernos tanto a las pautas generales de sistemas de

telemetría en juegos generales, como a las introducidas en esta sección para juegos serios a la hora de decidir qué información optamos por enviar desde los juegos implementados.

2.4. Sistemas comerciales de analíticas

Las librerías comerciales de analíticas de juego proporcionan normalmente dos elementos clave a la hora de seguir el proceso de *knowledge discovery* en juegos. Por un lado, se tiene una API destinada a instrumentalizar el código del proyecto y poder enviar eventos a un servidor en la nube, gestionado por la propia compañía. Por otra parte, suelen disponer de un sistema para poder visualizar los datos agregados recolectados por medio de los llamados "paneles de control" (o *dashboards* en inglés) para poder extraer conclusiones sobre los datos de manera más o menos intuitiva. Comencemos presentando una de estas plataformas.

2.4.1. Unity Analytics

Unity Analytics se presenta como herramienta fuertemente concebida para un análisis con cierta componente "comercial" de los eventos registrados en el juego, con numerosos mecanismos de control de monetización, anuncios, jugadores activos, y beneficios obtenidos con la aplicación (estas son métricas de usuario o jugador, definidas en la sección 2.1). Por defecto, al activar el módulo de analytics en un proyecto, Unity comienza a registrar automáticamente una serie de *core events*, tales como `AppStart`, `AppRunning` o `DeviceInfo`, que se utilizan como base para inferir algunas métricas de interés general, como puedan ser el número de nuevas instalaciones, número de usuarios activos por día o mes, sesiones por usuario, o tiempo medio pasado en la app por parte de los jugadores.

Aparte de los anteriores, Unity ofrece un conjunto adicional de eventos "estándar", que en este caso pueden ser utilizados por parte del programador de la aplicación para enviar ciertas acciones y sucesos comunes al servidor. *Standard Events* incluyen la mayoría de los eventos considerados típicos en estos contextos, tales como `game_start`, `level_complete` (llamados eventos de progreso), `tutorial_start`, `tutorial_end` (eventos de control de primera experiencia de usuario), `chat_msg_sent`, `user_signup` (eventos que reflejan que el usuario se involucra con el juego), o `store_item_click` (eventos relacionados con la monetización de la app). Todo evento posee además un diccionario de parámetros adicionales, que pueden usarse para aportar toda la información que sea necesaria para contextualizar el suceso.

Para usar el sistema, una posible opción es desarrollar una clase estática que contiene llamadas estandarizadas para cada uno de los eventos contenidos en el listado "estándar" ofrecido por defecto. Lo ideal es encapsular todos los métodos relevantes en dicha clase, que ofrece una interfaz pública para la aplicación y es responsable del envío de eventos. A nivel de proyecto, según va creciendo la necesidad de enviar eventos desde un gran número de puntos del juego, aislar este componente en una clase estática separada aumenta la claridad y usabilidad del código a largo plazo. De este modo, para enviar un evento de tipo `level_start`, por ejemplo, podemos proceder de la siguiente manera:

```
// importar espacio de nombres
using UnityEngine.Analytics;

public class AnalyticsEvents{
    // ofrecer método público para lanzar eventos desde el resto de la aplicación
    public static void ReportLevelStart(string levelName, string levelType)
    {
```

```
        // envía un evento al servidor con nombre levelName y parámetros adicionales
            sobre el tipo de nivel
        Analytics.LevelStart(levelName, new Dictionary<string, object>
        {
            { "level_type", levelType }
        });
    }
}
```

La clase anterior sirve por tanto para instrumentalizar el código destinado a enviar eventos al servidor. Los siguientes pasos en el proceso son el análisis y la visualización de los datos obtenidos. Unity proporciona, al igual que tantos otros sistemas similares, un "panel de métricas" o *dashboard* para representar de forma agregada toda la información recopilada. Este panel es posible construirlo gracias al uso de los eventos estandarizados mencionados anteriormente, de modo que, por ejemplo, se sabe lo que es un evento `AppStart` y en consecuencia se puede saber cuántos jugadores activos se tienen en un juego diariamente, como puede verse en la figura 2.3.

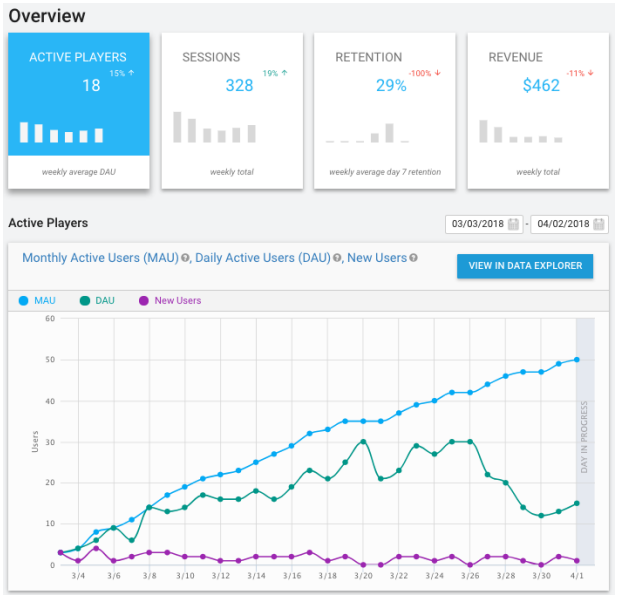


Figura 2.3: Panel de métricas en Unity Analytics

Si un juego particular tuviera métricas específicas de juego (refiriéndonos a las clasificaciones de la sección 2.1), como por ejemplo el "número de frutas comidas" en PacMan, el análisis y la visualización desde el *dashboard* se complican, al no ser estos eventos estándar soportados de forma nativa por la plataforma.

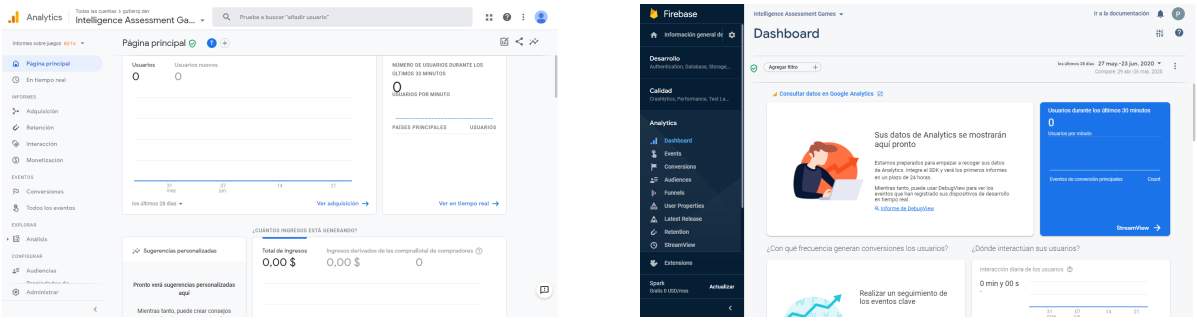
Por supuesto, Unity también permite la creación de eventos personalizados (*custom events*), con un máximo de diez parámetros, introducidos por medio del diccionario correspondiente. Volviendo al caso anterior, ahí el evento sería "Fruta comida", y si quisiéramos ver, por ejemplo, la métrica "Media de frutas comidas por nivel", nos veríamos obligados a llevar a cabo una agregación específica en el *dashboard*, lo cual como ya se ha dicho no está soportado para eventos no estándar.

Por otro lado, Unity Analytics permite la descarga de los datos crudos para los eventos recopilados (en formatos comunes como `CSV` o `JSON`), lo cual abre la posibilidad de realizar un análisis más específico si fuera necesario, pero esto debe hacerse manualmente por parte del usuario, como cabe esperar. La

descarga de estos datos, no obstante, no está dentro de las características de la versión gratuita de la plataforma, y es necesario hacer uso de alguno de sus planes de pago para poder realizar dicha descarga.

2.4.2. Google Analytics y Firebase Analytics

En la misma linea que el sistema nativo de Unity, existen opciones comerciales con licencias gratuitas, tales como Google Analytics [20] o Firebase Analytics [21]. Google Analytics es un servicio de analíticas destinado al *tracking* de tráfico y uso en diversos sitios y aplicaciones web, considerando por lo general datos agregados referentes al comportamiento de los usuarios y su interacción con el sistema. Asímismo ofrece una versión para aplicaciones Android e iOS, conocida como *Google Analytics for Mobile Apps*. Por otro lado, Firebase se presenta como una plataforma de desarrollo web y móvil perteneciente a Google. Esta puede verse como una "plataforma global" con un conjunto de herramientas en la nube que se necesitan típicamente durante el desarrollo de aplicaciones web y móvil, entre las que se encuentran las analíticas, entre muchas otras.



(a) Dashboard de Google. (b) Dashboard de Firebase.

Figura 2.4: Dashboards en plataformas comerciales

Ambas plataformas ofrecen plugins de compatibilidad con el motor de Unity, de uso muy semejante al ya ilustrado anteriormente para Unity Analytics. La única excepción es que estos obligan a importar GameObjects especiales en la escena en la que se pretenda lanzar eventos, y que contienen los scripts necesarios para llevar a cabo el envío al servidor. La experiencia con estos sistemas puede resumirse en los siguientes puntos:

- Tanto Google como Firebase Analytics ofrecen, al igual que Unity, un conjunto de eventos estandarizados de uso común, reservando un subconjunto de estos a "eventos de uso en juegos", en cierto modo semejantes a los ya mencionados anteriormente. Sin embargo, cabe destacar que ninguna de estas plataformas está concebida originalmente para juegos, por lo que es necesario adaptarlas mínimamente antes de poder utilizarse de forma cómoda en el contexto de un videojuego.
- La instalación y configuración de estos sistemas resultan notablemente más tediosas que en el caso de Unity Analytics, donde no hacía falta ningún tipo de paso previo más allá de vincular el proyecto con el servidor. Además, en diversas ocasiones, las llamadas a métodos de Firebase pueden resultar problemáticas, o incluso no llegar a registrarse todos los eventos enviados.
- Por lo general, los eventos de ambas plataformas sólo se actualizaban en el panel de gestión tras un número determinado de minutos/horas, por lo que los resultados no quedaban reflejados de manera inmediata. Además, al igual que ocurría con Unity Analytics, no existía una forma sencilla de obtener un archivo "crudo" fácilmente manejable de manera externa.

Resumen

- La telemetría engloba los mecanismos de especificación, captura o recopilación de cualquier tipo de suceso que pueda ocurrir en un juego. Los datos y eventos recogidos por medio de un sistema de telemetría pueden ser posteriormente transformados en métricas para emplearse en la fase de análisis de un juego.
- En este trabajo emplearemos el proceso de knowledge discovery como metodología de referencia, de tipo cíclico e iterativo, de cara a abordar las analíticas de los juegos implementados. Este proceso suele utilizarse de forma explícita o implícita en numerosos estudios de análisis de juegos, pudiendo llevar a distintas conclusiones de interés. Analizamos un caso particular en el análisis de las causas de muerte sobre niveles de *Tomb Raider: Underworld*.
- La naturaleza híbrida de los juegos desarrollados en este trabajo, inicialmente concebidos para el entretenimiento, pero usados aquí en un contexto de medición de capacidades cognitivas, hace que resulte interesante analizar las prácticas comunes tanto en juegos convencionales como en serious games, a la hora de abordar el tema de la telemetría.
- Unity Analytics y Google Analytics son algunos de los ejemplos más empleados de sistemas de analíticas en contextos afines, ambos basados en eventos.

Con este capítulo damos por concluida la primera parte de esta memoria destinada a estado del arte e investigación previa, para dar paso a un conjunto de capítulos dirigidos a describir en detalle el desarrollo técnico del trabajo.

Capítulo 3

Desarrollo de los juegos

Los juegos posibles a desarrollar para este proyecto fueron elegidos de entre los 10 juegos que se emplearon para el experimento base explicado en la sección 1.3. Esto nos permite, en la medida de lo posible, replicar el estudio original y hacer comparaciones entre los resultados obtenidos. Dado que se han estudiado tres factores cognitivos distintos (Gf, Gv y Gs), se pasó a escoger un juego destinado a evaluar cada uno de estos factores cognitivos.

A continuación introducimos brevemente los siete juegos posibles a desarrollar, junto con el factor que se presenta en cada uno de ellos. Tres juegos de la batería original fueron descartados por ser títulos comerciales muy conocidos y/o su complejidad para ser implementado en el presente trabajo.

- **Blek (Gf)** [27]: Juego 2D para móviles donde el jugador ha de realizar trazas con el dedo para poder resolver puzles.
- **Rail maze (Gf)** [28]: Juego 2D para móviles donde el jugador tiene que completar un escenario de vías de tren y otros elementos para que el tren pueda llegar al destino.
- **Ski jumping pro (Gs)** [29]: Juego 3D para móviles que es un simulador de saltos de esquí.
- **Unpossible (Gs)** [30]: Juego 3D para móviles donde hay que desplazarse por un circuito cilíndrico tratando de esquivar obstáculos girando la cámara mientras aumenta la velocidad.
- **Art of Balance (Gf/Gv)** [31]: Juego 3D para WiiU que consiste en tratar de encontrar el equilibrio apilando varios objetos sobre una plataforma.
- **Crazy pool (GV)** [32]: Juego 2D de física simulador de billar.
- **Edge (Gv)** [33]: Juego 3D para WiiU y móviles donde el jugador ha de atravesar un escenario manejando un cubo.

Las elecciones de desarrollo fueron las siguientes:

- Blek [27], versión para móvil para el estudio del razonamiento fluido (Gf).
- Edge [33], versión para ordenador para el análisis de habilidad visoespacial (Gv).
- Unpossible [30], versión para móvil para el estudio de la velocidad de procesamiento (Gs).

De esta forma se desarrollan 3 juegos para tratar por separado cada uno de los 3 factores posibles de la medida de la inteligencia. Se implementan los mismos niveles que los empleados en el estudio de referencia, tanto de la parte tutorial como del experimento real. La explicación de cada juego se realiza por separado y con detalle más adelante en este mismo capítulo. Cada uno de estos juegos fueron desarrollados en su totalidad en *Unity*, un motor de videojuegos multiplataforma desarrollado por Unity Technologies [34].

Este motor ofrece soporte para más de 25 plataformas distintas, incluyendo Windows PC, Android y WebGL, convirtiéndolo en un candidato ideal para proyectos concebidos sobre un amplio rango de sistemas. Esta versatilidad de elección en lo referente a plataformas es una de las razones principales por las que se optó por utilizar Unity para el desarrollo de los títulos escogidos. Los experimentos del estudio de referencia empleaban un conjunto extenso de consolas y dispositivos para la administración de los juegos en sus plataformas originales (Tablets, PCs, o WiiUs, entre otros). Si bien lo ideal para emular el experimento de partida hubiera sido desarrollar adaptaciones en los entornos originales donde se realizaron las sesiones, distintas limitaciones llevaron a una necesidad de realizar ciertas concesiones o adaptaciones al respecto. A modo de ejemplo, Edge fue administrado en su versión de WiiU en el experimento original, pero tratar de replicar esta decisión en nuestro caso suponía varias complicaciones técnicas y prácticas. Esto incluía la necesidad de un kit de desarrollo proporcionado por Nintendo para trasladar el juego a la plataforma, y la necesidad de reunir suficientes consolas para administrar los juegos a un ritmo asumible por el equipo. Siendo así poco realista restringirse a las plataformas originales en el desarrollo, el motor de Unity permite producir versiones para sistemas más accesibles en el presente trabajo, pudiendo realizar una mayor parte de la implementación sin preocuparse de la plataforma final.

La segunda razón de peso para la elección de Unity fue el hecho de que todos los integrantes del equipo estaban ya familiarizados con la aplicación en mayor o menor medida, permitiendo comenzar el desarrollo de forma ágil dentro de un entorno común. A largo plazo, utilizar un mismo motor permite compartir librerías de código de uso común entre juegos. Este es el caso del sistema de envío de eventos, implementado de forma genérica para ser soportado por los tres títulos. A continuación se procederá a explicar con detalle el desarrollo de cada uno de los tres juegos, sus principales características y la implementación llevada a cabo para cada uno.

3.1. Blek

Comencemos comentando el caso de Blek [27] (para el código asociado, ver [35]). En cierto modo, el juego se presenta como un "lienzo", en el que el jugador dibuja (con el ratón en PC, o con el dedo, en dispositivos táctiles) un trazo que pasa a "cobrar vida", repitiéndose en bucle hasta salirse de la pantalla o chocar contra un obstáculo. En cada nivel hay colocados un conjunto de círculos o bolas de colores que deben ser recogidos por el trazo en su recorrido por el mismo. Para superar un nivel, el trazo deberá recoger todos estos elementos antes de terminar su camino saliéndose de la pantalla por la parte superior o inferior de la misma. Para añadir más variedad, se introducen algunos objetos especiales que actúan como obstáculos, representados siempre como elementos de color negro. Cuando el trazo alcanza alguno de ellos, su avance se detiene inmediatamente. El trazo también se detiene si el usuario empieza a dibujar un nuevo trazo mientras que el anterior sigue en movimiento por la pantalla. En cualquiera de los casos, el nivel se reinicia, restableciendo cualquier círculo de color que hubiera sido recogido por la traza antes de detenerse. Una última peculiaridad es que el trazo "rebota" al entrar en contacto con los laterales izquierdo y derecho del nivel. Con el avance de los niveles, van apareciendo nuevos tipos de elementos, como por ejemplo las bolas de color con proyectiles, que lanzan un conjunto de pequeñas bolas de color al

ser tocadas por el trazo. Estas bolas de tamaño reducido tienen la capacidad de recolectar exactamente otra bola de color del nivel al tocarla, desapareciendo ambas en el proceso (notando aquí que estos proyectiles pueden atravesar libremente todo obstáculo de color negro). Por último, resulta importante indicar que, a la hora de dibujar el trazo correspondiente, el jugador no tiene permitido tocar ningún obstáculo del nivel (si así ocurriera, el trazo se detendría inmediatamente antes de cobrar vida), y en el caso de tocar una bola de color, se interrumpiría el dibujado, forzándolo a cobrar vida en el momento.

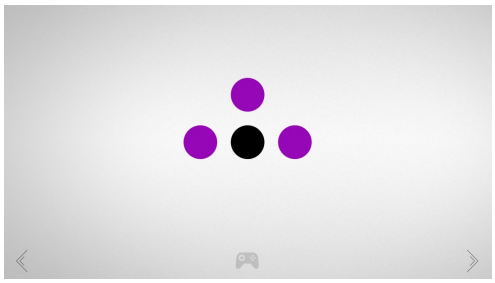


Figura 3.1: Ejemplo de nivel con un obstáculo y tres coleccionables.

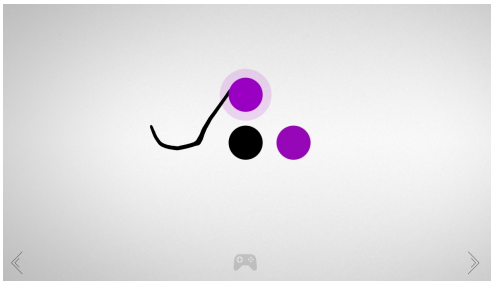


Figura 3.2: Traza en movimiento recogiendo el segundo coleccionable.

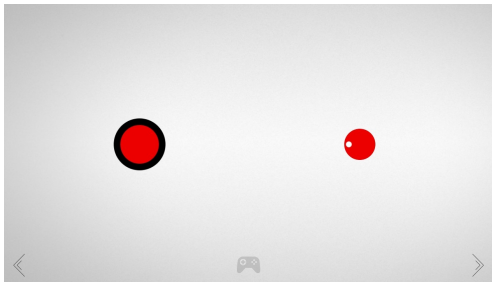


Figura 3.3: Ejemplo de nivel con una bola de color con proyectil.

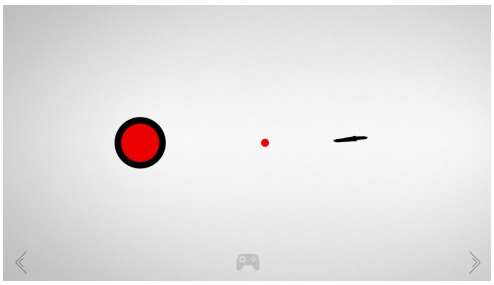


Figura 3.4: Proyectil en movimiento tras ser lanzado al contacto con trazo.

En este sentido, el juego se define claramente dentro de una categoría de puzzles, y resulta un candidato especialmente interesante para la evaluación del razonamiento fluido (Gf). La clave de esto es la forma en la que Blek presenta sus desafíos al jugador. Las distintas mecánicas van introduciéndose de manera progresiva, y nunca explicando en detalle su funcionamiento. Esto obliga al jugador a buscar la lógica subyacente a situaciones novedosas, con las que no está en principio familiarizado. Más aún, el juego satisface la condición de que el éxito en sus puzles no depende a priori del grado de conocimiento previo del jugador en un ámbito dado. En efecto, las reglas del juego son lo suficientemente únicas como para resultar novedosas incluso a aquellos con experiencia previa en títulos del mismo género. Y no solo la experiencia, tampoco el conocimiento específico, por ejemplo en matemáticas, resulta por lo general útil a la hora de enfrentarse a un problema en este juego. Finalmente, una gran parte de los niveles de Blek exige que el jugador sea capaz de detectar un cierto patrón u orden lógico para poder progresar. El éxito en dicha tarea pone de manifiesto la habilidad de inducción, generalización o abstracción de los jugadores, siendo todas ellas puntos clave en el concepto de razonamiento fluido.

3.1.1. Lógica del Jugador

Uno de los primeros desafíos a la hora de implementar Blek fue el diseño del "jugador", refiriéndonos con esto a la generación de las curvas con las que el participante de la sesión puede interactuar con los

niveles del juego. En esencia, el script de control del jugador se destina fundamentalmente a gestionar una máquina de estados finita (FSM por sus siglas en inglés) en la que se resumen los comportamientos a realizar en cada momento por parte de este generador de trazas. Seguimos aquí las indicaciones descritas en [36] sobre el patrón State.

Se contemplan cuatro estados básicos, que pasaremos a describir a continuación:

- `AWAIT_INPUT`: Es el estado de partida de la FSM y representa la idea de estar esperando al primer input proporcionado por el usuario. El sistema esperará estático hasta que el jugador toque la pantalla por primera vez, y entonces guardará una referencia al punto de contacto para su uso posterior. Tras esto, el estado de la FSM pasa a ser `FIRST_TOUCH`.
- `FIRST_TOUCH`: Representa un estado en la FSM destinado a determinar la intención del usuario al pulsar la pantalla por primera vez tras un periodo estático. Una primera pulsación de pantalla puede emplearse bien como medio para reiniciar un nivel en el que ya existe un cursor en movimiento (y luego levantar el dedo sin dibujar nada), o para comenzar a dibujar un trazo en cualquier momento del juego. La decisión sobre la transición a escoger se toma en función de la naturaleza del input del jugador. En cada frame en el que se mantenga pulsada la pantalla, se toma el punto de contacto y se compara con la referencia inicial registrada justo antes de pasar a este estado; si las posiciones resultan ser diferentes, se interpretará que el jugador ha decidido pasar a dibujar la traza (transición al estado `DRAW_LINE` y creación de un cursor de trazo) mientras que levantar el dedo sin haber cambiado de posición devolverá el sistema al estado por defecto (transición a `AWAIT_INPUT`).
- `DRAW_LINE`: Representa el estado de dibujo de traza. El estado se mantiene hasta que el usuario decide levantar el dedo de la pantalla o se detecta una colisión de la curva con algún objeto del nivel. En el caso de detectar la colisión con un objeto "negro" u obstáculo, se procede a destruir el cursor actual, reiniciar el nivel y lanzar una transición al estado `AWAIT_INPUT`. Por el contrario, si se detecta una colisión con una bola, se pasa automáticamente al estado `LOOP_LINE`, dando comienzo al bucle del trazo. En cada frame que la FSM se encuentra en este estado, el sistema agrega un punto adicional a la lista que define la trayectoria de la curva (almacenadas en principio en posiciones absolutas, por razones que se explicarán en el siguiente apartado) y desplaza el cursor para reflejar visualmente dicho cambio.
- `LOOP_LINE`: Representa el estado de "bucle" en el que la traza pasa a "cobrar vida" y desplazarse por la pantalla siguiendo el patrón almacenado en el cursor. Este estado puede ser interrumpido de varias maneras provocando diversas transiciones. En primer lugar, si se detecta que el nivel ha sido completado con éxito, se procede a destruir el cursor actual (en este caso esto sólo cumple un propósito estético y no funcional). Si se produce una colisión con un objeto "negro" o se detecta que el cursor está fuera de los límites permitidos de la pantalla, se reinicia el nivel, destruyendo el cursor actual y transicionando a `AWAIT_INPUT`. Por último, si se presiona la pantalla durante el proceso de "looping", se procede a gestionar la ambigüedad discutida anteriormente transicionando a `AWAIT_INPUT`, tras haber previamente destruido cursores y reiniciado el nivel.

Los métodos de gestión de la FSM tienen lugar en el cuerpo de las llamadas al update de cada frame en el script `PlayerController.cs`.

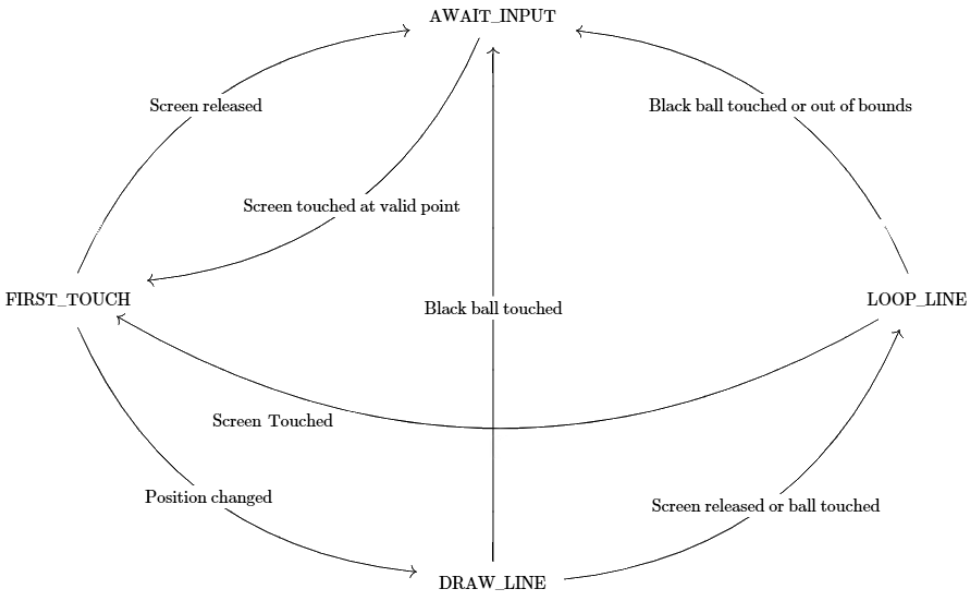


Figura 3.5: Diagrama de estados del jugador.

3.1.2. Control de trazo y rebotes

El segundo problema a tratar es la implementación del proceso de *looping* que tiene lugar en el estado `LOOP_LINE`. La complicación principal en este apartado es la gestión de los "rebotes" que sufre el cursor del trazo cada vez que alcanza las paredes izquierda y derecha de la pantalla, teniendo que pasar a "reflejar" la curva tras cada contacto con la pared.

Una primera aproximación a este problema sería hacer uso del motor de física de Unity para tratar de emular el flujo del trazo. Esto sin embargo, sería un caso de "sobreingeniería", pues la ventaja principal de dicho motor es el cálculo automático de la dinámica de cuerpos a partir de fuerzas, cosa que en está situación no existe de manera natural. Debido a esto se decidió realizar un control manual del trazo en el sentido que explicaremos a continuación. Comencemos definiendo los términos utilizados en el paso de actualización del cursor durante el bucle. Estos coinciden con los utilizados en el código correspondiente de `CursorController.cs`:

- `cursorTrajectories`: listado de posiciones de la curva actual (o lo que es lo mismo, puntos registrados durante la etapa de dibujo). Este listado se mantiene fijo durante todas las iteraciones del bucle.
- `loopPosition`: posición actual del cursor en el bucle, al recorrer los puntos de la lista de posiciones. Intuitivamente, representa "cuánto en la curva" llevamos recorrido en una iteración dada.
- `offset`: diferencia entre el primer punto en la iteración actual y el primer punto de la curva original, `cursorTrajectories[0]`. Al primer punto de la iteración actual lo llamaremos `reference`.
- `reflected`: flag que determina si la iteración actual comienza con el sentido original del trazo (flag desactivado) o con el sentido invertido o reflejado (flag activado). A modo de aclaración, la primera iteración del bucle siempre comienza con este flag inactivo, y estará activo, por ejemplo, cuando

una curva dibujada de izquierda a derecha "rebota" y pasa a reproducirse de derecha a izquierda en una iteración posterior.

- `locallyReflected`: flag que determina si una subsecuencia de la iteración actual se opone al sentido de esta. Esto es, si la curva "rebota" a mitad de una iteración, el sentido de la misma cambia en consecuencia.

Empleando estos términos, el algoritmo de rebote puede describirse de la siguiente manera:

1. Si `loopPosition` es `0`, esto significa que estamos al comienzo de una iteración del bucle, y por tanto debemos actualizar `reference` con el punto actual (la posición del transform del cursor), así como `offset` (`reference - cursorTrajectories[0]`). Además, si al acabar la iteración anterior la curva quedó marcada como localmente reflejada, alteramos el sentido "global" de esta, y desactivamos el flag `locallyReflected`. Por ejemplo, si una iteración comienza en el sentido original del trazo (`reflected` desactivado), y toca la pared en un instante intermedio, `locallyReflected` pasa a activarse hasta volver a tocar una pared o finalizar la iteración actual. Si la iteración ha acabado con `reflected` desactivado y `locallyReflected` activo, de acuerdo con la definición de `reflected`, este ha de pasar a estar activo, pues la nueva iteración comienza en sentido reflejado.
2. En cualquier caso, calculamos el siguiente punto "esperado" en la trayectoria. La siguiente posición en condiciones normales puede hallarse como el punto correspondiente de la traza original (`cursorTrajectories[loopPosition]`) más el offset de la iteración. Tenemos que tener en cuenta sin embargo el posible caso en el que `reflected` esté activado. La reflexión ocurre en este caso en el eje horizontal, por lo que reflejamos sobre `reference.x` haciendo `expectedPoint.x -= 2 * (expectedPoint.x - reference.x)`. Esta es una expresión común para reflexiones sobre una recta en el plano (en este caso, la recta vertical que pasa por `reference`).
3. Ahora bien, si el punto esperado en la trayectoria tras el cómputo anterior se "sale" de la pantalla del nivel, hemos encontrado una pared, y por tanto debemos volver a ejecutar una reflexión sobre la recta dada por la pared correspondiente, y activar `locallyReflected`.

Figura 3.6: Fase de dibujo

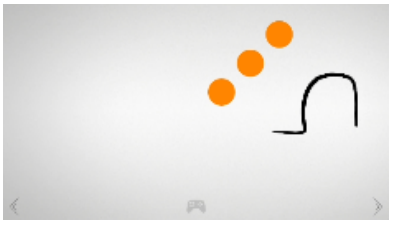


Figura 3.7: Iteración sin reflejo

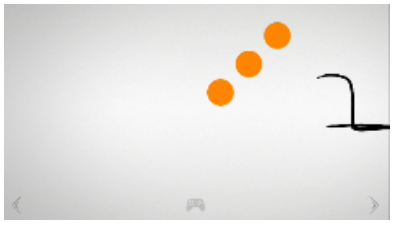


Figura 3.8: Reflejo local

Figura 3.9: Reflejo global

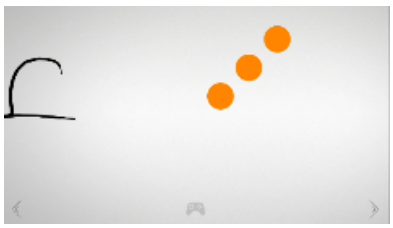


Figura 3.10: Reflejo global + local

Las imágenes anteriores muestran una secuencia completa desde la fase de dibujo, mostrando distintas configuraciones posibles por las que pasa una curva en su paso por el bucle.

3.2. Creación y estructura de niveles

Finalmente, resulta interesante comentar qué supone exactamente incorporar un nuevo nivel al juego. En principio, cada uno de los niveles de Blek se encuentra encapsulado en una escena única, en la que el diseñador podrá ir colocando un conjunto prefijado de prefabs para ir construyendo el puzle correspondiente. De este modo, proporcionamos objetos prediseñados para representar bolas de colores estándar, bolas con proyectiles, y obstáculos por medio de círculos negros. El rango de posibles tamaños y colores para estos elementos en el juego original hace que resulte poco práctico ofrecer un número limitado de assets con estas propiedades restringidas, por lo que optamos por establecer un tamaño y color base para cada prefab, dejando a discreción del diseñador el determinar el factor de escala o el material a emplear. Con esto queremos decir que, tras inspeccionar los tamaños de los elementos en el juego original, llegamos a la conclusión de que estos no seguían una clasificación fija, y que en su mayoría adoptaban medidas más o menos arbitrarias. Sí que existe, sin embargo, una cierta regularidad en el uso de materiales en las bolas de color, por lo que ofrecemos un conjunto limitado a aquellos encontrados en los primeros 50 niveles del juego. Cualquier otro material ha de ser incluido manualmente.

Por otro lado, si bien se ofrece un prefab base para el elemento "bola con proyectiles", las instancias concretas de estos objetos requieren de modificaciones mínimas para crear las distintas configuraciones que pueden ser interesantes desde el punto de vista del diseño. Este prefab se corresponde con el caso de un único proyectil colocado en un radio dado, pero en función del nivel a producir podría ser necesario incluir una bola con 5 proyectiles colocados en radios dispares, por ejemplo. Para ello, el script asociado a este componente permite añadir a mano el número de proyectiles a instanciar sobre la bola base, así como la posición en ángulos en la que se desea colocar cada uno de ellos. Otra posible variación, no mencionada en la descripción inicial del juego, es la posibilidad de añadir rotación a los elementos con proyectiles. Puesto que esto no caracteriza propiamente a estos objetos, se opta por ofrecer un componente o script de rotación, el cual, al ser añadido a una bola con proyectiles, permite especificar desde el editor la secuencia de rotaciones a ser ejecutada sobre la misma. Estas rotaciones vienen dadas por listas de parejas `(ángulo, tiempo)` que se ejecutan en bucle en cada nivel. De este modo, una secuencia `[(180, 2), (-180, 2)]` produce un giro de media circunferencia en 2 segundos, para luego deshacer dicho giro utilizando otros dos segundos. Este patrón se repite mientras el componente se mantenga activo.

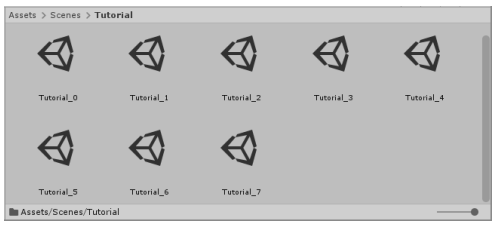


Figura 3.11: Secuencia de niveles tutorial.

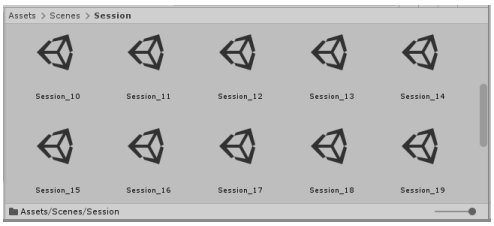


Figura 3.12: Secuencia de niveles sesión.

El diseño de un nivel se reduce así a crear una nueva escena, por lo general duplicando una plantilla base en la que ya se ha incluido el fondo por defecto y la cámara ya configurada, e ir añadiendo instancias de los prefabs descritos anteriormente a conveniencia. Lo último que nos falta por comentar es la determinación del orden de los niveles. Para ello optamos por realizar una clasificación de escenas

en *sets* de niveles (en este caso, de tutorial y de sesión, formados por 8 y 27 escenas, respectivamente), siguiendo una nomenclatura fija. Así, los niveles del set `Tutorial` se nombran `Tutorial_0`, `Tutorial_1`, etc. Cuando se inicia un set dado, se carga la escena de índice `0` en la secuencia, y a partir de ahí se continúa con niveles de índices ascendientes hasta no haber ya más de dicho set, momento en el que se considera completado.

3.3. Edge

Edge es un juego de plataformas 3D para WiiU que también tiene versión para móviles y Tablets [33], en este caso se adaptó para ordenador y Web (el código asociado a este juego puede encontrarse en el repositorio [37]).

3.3.1. Lógica del juego

El juego consta de varios niveles o desafíos en los que el jugador controla un cubo que puede moverse en 4 direcciones posibles. El mundo es un escenario en perspectiva isométrica compuesto por casillas discretas por las que irá desplazándose el cubo. El objetivo del jugador es tratar de llegar a la casilla final del nivel en el menor tiempo posible consiguiendo unos ítems o prismas que estarán distribuidos por todo mapa. Tal y como se comentó en la sección 1.3, en este juego trataremos de evaluar la habilidad visoespacial de los jugadores, es decir, la capacidad que poseen para representar mentalmente el escenario de juego presente.

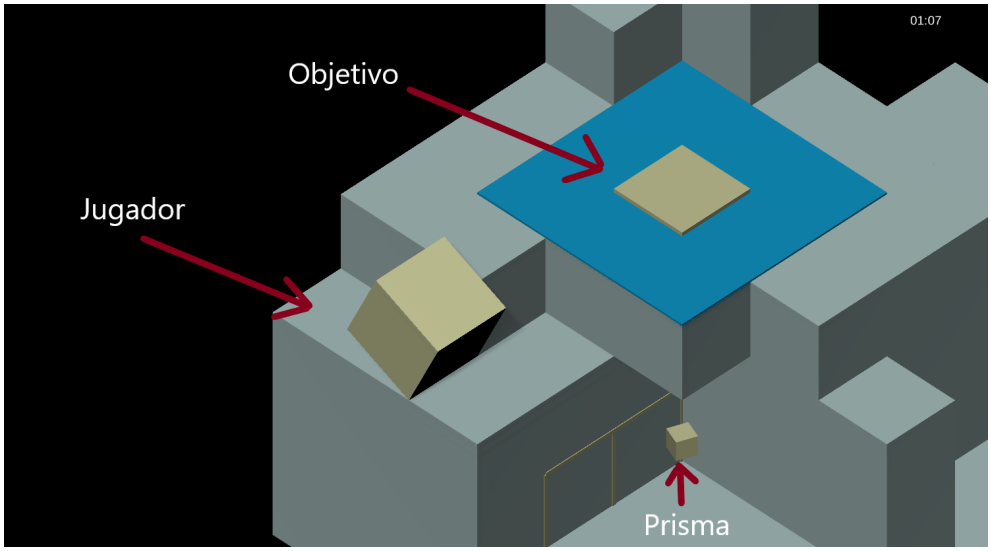


Figura 3.13: Ejemplo de escena de juego de Edge.

En cuanto a controles y mecánicas del juego, en esta versión para ordenador, el cubo se desplaza mediante las flechas del teclado, o mediante las teclas WASD. El cubo puede subir escalones, pero sólo de uno en uno y si no hay obstáculos encima. No podrá moverse si tiene un bloque justo encima de él o un bloque a cada lado que le impida el movimiento. También puede ser desplazado y empujado por obstáculos movibles o plataformas, algunas de ellas pueden ser accionadas con activadores. Otros elementos destacables de juego son activadores que empujan al cubo unas cuantas casillas en una dirección determinada y bloques de suelo quebradizo que se caen al poco tiempo de pasar por encima de ellos.

En ciertos niveles el jugador puede encogerse y convertirse en un mini-cubo. Este mini-cubo se maneja de igual forma y se le añade la funcionalidad de que puede trepar paredes y acceder a sitios donde el cubo de tamaño normal no podría. El jugador ha de volver al tamaño inicial en algún momento para poder concluir el nivel.

El jugador muere cuando cae por un borde del escenario, y vuelve a aparecer en un punto del mapa marcado por un *checkpoint*. Estos *checkpoints* o puntos de retorno son invisibles y están distribuidos por el mapa para facilitar la jugabilidad del usuario, al no tener que comenzar desde el principio cada vez que se produzca una muerte. El último *checkpoint* cogido será el punto donde aparezca el cubo si éste muere o cae al vacío.

Se puede ver en todo momento el cronómetro con el tiempo desde que comenzó el nivel y, en el caso de los niveles del experimento, el tiempo que falta hasta que éste concluya. En esta versión del juego adaptada, sólo se podrá superar cada nivel una vez. Se irán desbloqueando nuevos niveles a medida que se vayan completando hasta un total de 12. Una vez acabe el tiempo o se concluyan los 12 niveles, el juego acaba y no se puede avanzar más. Al final de cada nivel aparece un menú de información con los resultados conseguidos dependiendo de la cantidad de prismas cogidos con respecto al total que hubiera, y el tiempo. También se tiene en cuenta el número de muertes que haya habido desde que comenzó el nivel.

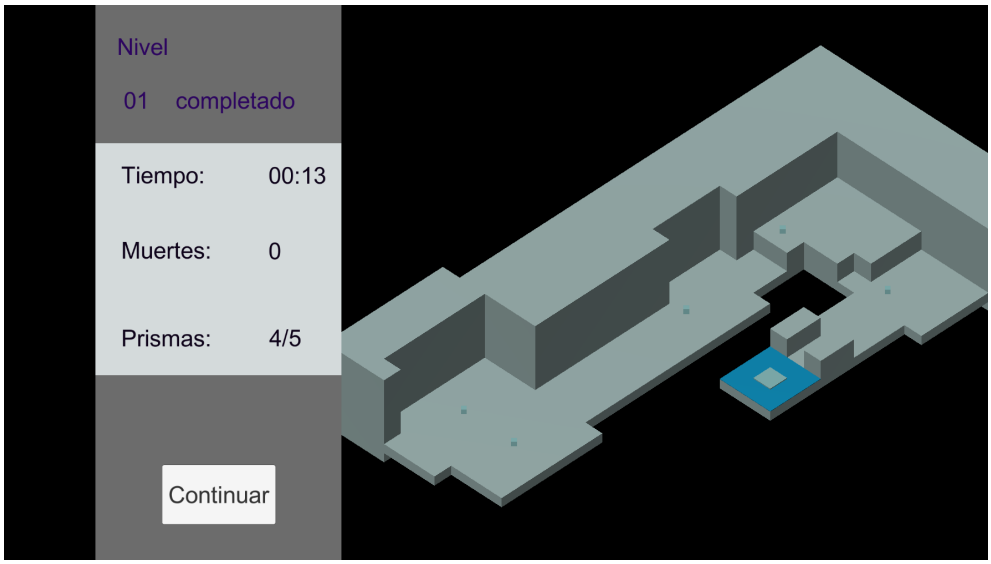


Figura 3.14: Menú de resultados mostrado al finalizar un nivel.

3.3.2. Implementación

A continuación se explicará con detalle cómo está programado y estructurado el juego. Las escenas de Unity creadas las podemos dividir en escenas de menús y escenas de juego. Éstas últimas son idénticas entre sí salvo por los elementos que aparecen en el nivel en cuestión (cada escena corresponde con un nivel). En total, el juego consta de 14 escenas:

- La primera escena, que será la que nos encontremos nada más ejecutar el juego, es el menú de instrucciones, donde se explica brevemente el objetivo del juego para que el usuario lo conozca y con un botón para que comience el juego en cuanto el usuario haya acabado de leerlas.

- Otra de las escenas corresponde al menú de información representado en la figura 3.14.
- Las otras 12 escenas corresponden a los 12 niveles de juego. Las escenas de juego se componen de los elementos de juego, explicados más abajo. La forma más sencilla de hacer un nuevo nivel es duplicar una escena de juego, eliminar el terreno y crear uno nuevo.

Elementos de juego:

- Player: cubo en 3D que representa a la entidad manejable por el jugador. Cambia de color constantemente siguiendo un patrón para que llame directamente la atención del usuario. El jugador puede avanzar en 4 direcciones con un movimiento de rotación del cubo. Los movimientos del player son todos discretos, es decir, el jugador avanza una posición exactamente igual a su anchura hacia la dirección indicada, salvo cuando es arrastrado o empujado por plataformas.
- Cámara y luces: la cámara seguirá al player en todo momento.
- Terreno: todos los demás objetos que componen el nivel, la mayoría como instancias o prefabs. Dentro del terreno encontramos objetos de suelo, objetos de pared, ítems (los objetos que va recogiendo el player), *checkpoints* o puntos de guardado y *deathzones*. Los *deathzones* son zonas donde se detecta que el jugador ha muerto para retornar al último *checkpoint*.
- Plataforma fin de nivel: es una plataforma azul de 3x3 que representa el final del nivel, el jugador ha de acabar justo en su centro para darlo por concluido.
- Menú de pausa: en realidad no será un menú de pausa como tal, ya que no para el tiempo del nivel. Esto se ha hecho para evitar que el jugador pueda pensar estrategias parando el tiempo a la hora de realizar el experimento. Este menú se activa con la tecla Escape y tiene opción de continuar el juego, salir del juego o retornar al último *checkpoint* cogido.
- Cronómetros: en blanco en la esquina superior derecha, aparecerá el tiempo del nivel transcurrido desde que éste comenzó y en la esquina superior izquierda en rojo la cuenta atrás de los 12 minutos de experimento.

Todo lo anterior es referido a los objetos visibles o interactivos dentro de un nivel. Ahora pasaremos a ver la implementación de Gestores o Managers, que son las instancias encargadas de comunicar información entre escenas y entre objetos:

- `PlayerController:` gestiona el input del usuario para el control del player mediante una máquina de estados y realiza la acción que corresponda si es posible. También realiza la gestión de físicas para saber si el jugador choca con un obstáculo o no mediante Raycasts. Los Raycast son "rayos" invisibles que se lanzan y notifican al jugador si llegan a chocar con algo en un alcance que se determine, en este caso de 1 unidad (el tamaño del cubo). No sólo tiene que lanzar Raycasts sobre la dirección en la que se quiere avanzar, también hay que tener en cuenta que:

  - Puede haber un obstáculo que impida hacer movimiento normal pero que pueda escalarse si es justo de 1 unidad de altura.

  - No puede avanzar si el jugador tiene un obstáculo justo encima suya, ya que no tiene espacio para realizar la rotación.

  - Por consiguiente, no puede avanzar a una casilla vacía que justo encima tenga un obstáculo.

- Al igual que el punto anterior, no puede subir a un escalón libre si justo encima de donde quedaría el player hay un obstáculo.

- También se lanzan constantemente Raycast en dirección vertical hacia abajo para ver si el player está sobre suelo o sobre vacío en el que tenga que caer.

El jugador dispone de varios estados en los que puede estar. Pero únicamente puede estar en uno en cada instante. Los estados son los siguientes:

- Parado: para realizar cualquier acción, el jugador tiene que estar en este estado. También es el único estado donde se lanzan Raycast para ver si pisa o no suelo para evitar que caiga cuando esté realizando movimientos ya que partiría la discretización de casillas y atravesaría paredes y obstáculos en ocasiones.

- MovW, movA, movS, movD: representan que el jugador está realizando un movimento normal hacia una de las 4 direcciones posibles. Una vez haya terminado de rotar los 90 grados pasará al estado parado.

- SubeW, subeA, subeS, suveD: al igual que los 4 anteriores, pero para cuando está subiendo escalones. El movimiento es distinto ya que se rota el cubo con respecto a la arista superior, no inferior, de la dirección indicada. Además, este movimiento es de 180 grados.

- Cayendo: en este estado se simula que el jugador cae por efecto de la gravedad manualmente, al no haber físicas como tal. El jugador cae a una velocidad constante en Y hasta chocar contra suelo o plataforma, o al llegar a un deathzone porque se cae fuera del nivel. Mientras está cayendo, el jugador no puede realizar ningún movimiento.

- Fin: estado especial que sólo tiene efecto cuando el cubo llega al objeto de 3x3 de fin de nivel, y se simula un efecto de que "flota" en el aire mientras gira.

El PlayerController también es el encargado de ajustar la posición en los 3 ejes del cubo cuando sea necesario, normalmente al acabar un movimiento o terminar de ser empujado por una plataforma, para asegurar que siempre se acaba en una casilla con un valor discreto y no acumular errores de posición.

- `PlayerMiniController:` controla el movimiento del player cuando se convierte en un cubo pequeño. Tiene prácticamente las mismas funciones y métodos que el PlayerController salvo algunas excepciones:

  - El jugador no sólo sube peldaños sino que los puede escalar verticalmente en el eje Y, es la principal ventaja de este modo para poder llegar a zonas que no podría el cubo normal. Por lo que incluye 4 estados más para escalar (escalaW, escalaA, escalaS y escalaD).

  - El jugador ahora no puede pasar al estado cayendo nada más pasar al estado parado, ya que eso originaría que cayera constantemente al intentar trepar paredes. Hay que dejar un tiempo de unos 200 milisegundos en los que el jugador esté sin pulsar ninguna tecla para que el jugador comience a caer.

  - El tamaño del nuevo cubo es un tercio del cubo original, por lo que los Raycasts que se lanzan cambian totalmente. Se reduce el alcance un 66 %, y hay raycasts que ya no son necesarios (no va a ocurrir el caso de que haya un obstáculo justo encima del jugador e impida que se mueva).

  - También se incluyen las acciones de Reducir Cubo y Agrandar Cubo, que hacen el efecto de que se hace pequeño el player o vuelve a su estado original después de ser pequeño. Tiene también sus propios estados ya que no interesa que el player pueda realizar movimientos mientras ocurre este efecto.

- `GameManager`: gestor de todas las variables que se necesitan entre escenas o que necesitan ser registradas, como son el número de niveles, los items recogidos en cada uno de los niveles, las muertes y los tiempos. Todo ello cuando ocurre se lo comunica al SessionManager mediante eventos. También se encarga del cambio de escenas y de controlar si se está jugando un tutorial o no, principalmente para controlar los tiempos que se puede estar jugando tutorial o experimento. El GameManager es un Singleton, se crea una sola vez en la primera escena de todas y ya no se destruye hasta que se salga de la aplicación.
- `LevelManager:` controla los sucesos que van ocurriendo en el nivel para comunicárselos al GameManager. También controla el tiempo actual del nivel mientras se está jugando, y cuándo se ha acabado un nivel.
- `SessionManager:` singleton que envía mensajes de eventos importantes al servidor. Estos eventos se los comunica sólo y exclusivamente el GameManager, por seguridad.

3.3.3. Problemas y complicaciones en el desarrollo

A continuación se explicará los distintos problemas destacables que fueron ocurriendo en el desarrollo de Edge y cómo pudieron resolverse.

- A nivel de programación, lo único realmente complejo fue el movimiento rotatorio del cubo, ya que era muy particular y dependiente de obstáculos el que se pudiera o no mover. El movimiento del cubo es una rotación sobre uno de sus bordes que hace que se desplace exactamente su longitud en una de las direcciones, de este modo deberíamos tener movimientos discretos. Pero una vez conseguido el movimiento esperado, resultó que debido a las unidades de posición de Unity no era tan sencillo, era necesario hacer correcciones de posición constantemente porque al llevar un tiempo jugando, el jugador al final llegaba a "salirse" de las casillas discretas. El jugador no acababa el movimiento exactamente en la posición que debía por un error minúsculo, prácticamente imperceptible hasta que se iba acumulando el error cuando se habían realizado ya varios movimientos. La solución fue ajustar la posición justo al haber acabado el movimiento que fuera. Esto fue aún un poco más complicado al tratar de ajustar después de haber sido empujado por una plataforma, había que hacer comprobaciones extras.
- Otro problema persistente fue el problema con las físicas. Debido al movimiento del cubo en casillas discretas, el juego no debía implementarse usando el motor de físicas de Unity, éste debía de ser nulo para evitar que el jugador chocara con bordes, opusiera resistencia a la hora de girar, etc. Por ello el movimiento del cubo debía de ser totalmente manual, sin aplicación de aceleraciones o torques como se intentó al principio, y la detección de obstáculos que impedirían el movimiento del jugador no debía de realizarse mediante colisiones sino mediante Raycast (explicados en el apartado 3.3.2). Mientras todo comportamiento fuera discreto no habría problema pero entonces surgieron las plataformas movibles, que no sólo podían impedir movimiento al jugador sino que podían arrastrarle de forma no discreta. Se trató de añadir físicas de Unity parcialmente para este aspecto y para la gravedad pero resultó inviable, salían nuevos conflictos constantemente. Al final la gravedad se simuló de forma manual al igual que el movimiento del jugador, y el arrastre de plataformas se pudo lograr mediante herencia de objetos en Unity, fue la opción que menos problemas conllevaba. Consistió en que, cada vez que una plataforma chocara con un jugador, éste se haría "hijo" de la plataforma de forma que se compartieran las transformaciones de movimiento, es decir, si la plataforma se movía 2 unidades a la derecha, todos sus hijos incluidos el jugador, se

moverían también esa cantidad en esa dirección. Luego cuando el jugador se moviera fuera de la plataforma, se desheredaría para que no ocurrieran las transformaciones cuando no debían. Esto solucionó el problema grande pero trajo consigo otros pequeños problemas que tratar a parte, como por ejemplo el que no se modificara el tamaño del jugador (ya que al hacerse hijo, se modifican tanto posiciones como tamaño y rotación), el desheredar automáticamente al llegar la plataforma a su tope para evitar el efecto imán, corregir posiciones tras empujes y el evitar atravesar plataformas si coincidía movimiento inacabado del jugador con el punto de impacto para el arrastre.

- Aún habiendo resuelto una gran cantidad de fallos que iban surgiendo en los niveles, había que tener en cuenta que probablemente hubiera fallos no encontrados aún, porque hay una infinidad de combinaciones de teclas de movimientos para cada uno de los niveles (la mayoría complicadas o imposibles de tratar de repetirlas). Esto podía dar como resultado algún bug problemático en el juego. Por ejemplo, que el jugador atravesara una pared por efecto de alguna plataforma, o que no detecte plataformas al caer si están en movimiento, etc. Por ello se decidió añadir al menú de pausa la opción de retornar el jugador al último *checkpoint* (una muerte provocada) por si se diera el caso de que en el experimento algún jugador no pueda moverse debido a un bug o algo semejante.

3.4. Unpossible

A diferencia de los otros dos juegos, la mecánica de Unpossible es bastante más sencilla: el jugador viaja por el exterior de un tubo curvado en el espacio y busca aguantar el mayor tiempo posible esquivando todos los obstáculos con los que se encuentra. El movimiento hacia delante no es controlable por el jugador, lo único que puede hacer es girar hacia la izquierda o hacia la derecha sobre el tubo. Cada vez que el jugador se choca con un obstáculo, este muere y "resucita" al comienzo del nivel correspondiente.

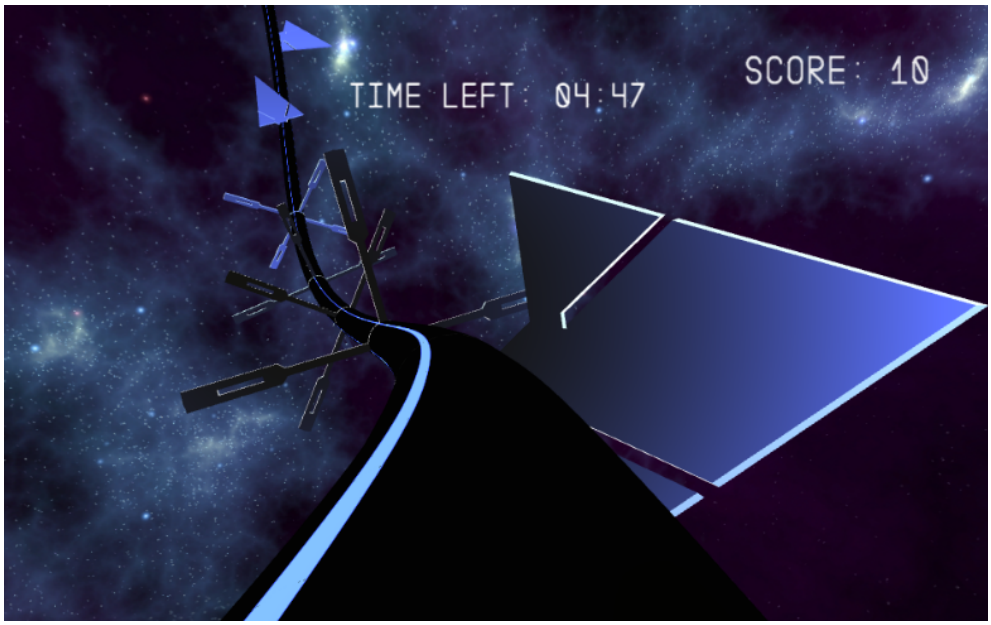


Figura 3.15: Una escena de Unpossible

A medida que avanza en el juego, la secuencia de obstáculos adquiere más complejidad y dificultad, requiriendo al jugador un tiempo de reacción cada vez más rápido y una concentración constante para

controlar sus movimientos. Por lo tanto, es un candidato muy prometedor para medir la velocidad de procesamiento (Gs), y probablemente también cierta cantidad de capacidad visoespacial (Gv). El código que corresponde con la implementación del juego se puede encontrar en [38].

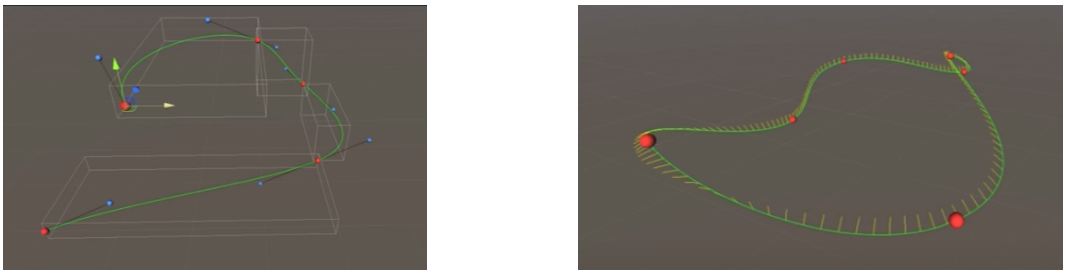
3.4.1. Trayectoria de un nivel

El componente principal del juego es el tubo curvado que define la trayectoria del jugador. La curvatura constituye una parte esencial de la jugabilidad, ya que dificulta visualizar los siguientes obstáculos de la trayectoria, además de requerir al jugador una velocidad de reacción rápida y variable, dependiendo de dónde esté situado en la curva. Adicionalmente, esta añade ruido a la perspectiva del jugador, causando una sensación de "mareo" y desenfoque o fatiga visual. También hace que el juego sea más interesante, eliminando la monotonía de seguir una trayectoria recta en todo momento. Todas estas razones exigen un cuidado especial a la hora de desarrollar esta componente, pues resulta crítica en el acabado final de la aplicación. En el proceso, nos encontramos ante dos problemas principales a resolver: (1) conseguir representar una curva de manera efectiva en Unity (partiendo del hecho de que el motor no soporta esta función de forma nativa), y (2) lograr encontrar un sistema adecuado para simular una trayectoria en principio infinita.

Librería Path Creator

El motor de Unity no proporciona ninguna herramienta integrada para crear objetos curvados. Después de investigar varias librerías desarrolladas independientemente por miembros de la comunidad, optamos por el paquete **Bezier Path Creator** [39] como base. Es una herramienta bastante potente para crear curvas de Bézier en el editor de Unity, tanto en espacio 2D como en 3D. Aunque la librería permite crear curvas exclusivamente desde código, elegimos utilizar el editor, al ser este más conveniente y permitirnos controlar de manera visual que la curvatura sea la adecuada para el nivel.

La creación de una curva se consigue añadiendo y ajustando puntos de anclaje y de control. La herramienta dispone de una opción para mostrar los vectores normales de la curva, cuyos ángulos son modificables por el diseñador. La figura 3.16 ilustra el modo de edición de las respectivas funcionalidades.



(a) Definir una curva en el editor (b) Modificar vectores normales de una curva

Figura 3.16: Librería Bezier Path Creator

Explicamos aquí brevemente las clases principales de la librería y los métodos que hemos modificado o usado directamente en el juego.

- **BezierPath**: Una curva de Beziér es un sistema para representar una curva parametrizada empleado muy frecuentemente en gráficos por computador. Puede tener un número arbitrario de puntos

de control pero la más usada es la del orden cúbico (con cuatro puntos de control). La clase *BezierPath* representa la curva como una sucesión de curvas cúbicas de Beziér. Almacena los puntos de control y permite realizar algunas operaciones básicas, como la suma de dos curvas, insertar o eliminar un punto de control en una posición dada, modificar el ángulo de los vectores normales, etc.

- **VertexPath**: La clase *VertexPath* es la que guarda los puntos en el espacio a lo largo de una curva de Beziér, proporcionando una representación computacional de la curva construida matemáticamente. Además de almacenar también los vectores tangentes y normales en cada punto, ofrece métodos para consultar las coordenadas en el espacio de un punto, el vector tangente o normal dada la posición relativa en la curva, u obtener el punto de la curva más cercano a un punto dado. Cambiamos algunos tipos devueltos para incorporar su uso en el juego. Veremos que estos métodos resultan especialmente útiles en la implementación.
- **PathCreator**: Es la clase propiamente accesible de la librería, la cual debemos adjuntar a un GameObject en Unity para crear una curva. Almacena los datos del editor en una clase auxiliar *PathCreatorData* y también la curva de Beziér y la curva discretizada correspondiente. Por lo tanto, sirve como el punto de acceso a las clases de las curvas y a sus métodos en el juego.

Curva cerrada

El segundo reto es modelar el tubo, que a priori es infinito en el juego original, ya que el jugador puede seguir jugando tanto tiempo como le sea posible siempre y cuando no se tope con ningún obstáculo. Aunque intentamos mantenernos fieles a los niveles usados en el experimento original, resulta imposible reproducir la curvatura del tubo con exactitud. En efecto, se encontraron fuentes de video de varias versiones de un mismo nivel, lo que lleva a pensar que tanto los obstáculos como la trayectoria de los niveles son aleatoriamente generados. Por esta razón, solo calcamos las formas y las distancias de separación de los obstáculos de cada nivel (de una fuente concreta), liberando esta restricción sobre el desarrollo de la curva. La decisión final es definir la trayectoria de un nivel como una curva cerrada (cíclica), debido a varias consideraciones:

- En primer lugar, este tipo de diseño permite que el jugador recorra la curva "en bucle", produciendo una sensación de infinitud. Esto se traduce en que un nivel puede jugarse, en principio, sin límites de tiempo, pues técnicamente dejan de existir los puntos finales de curva.
- Al mismo tiempo, evitamos generar y destruir constantemente objetos a lo largo del trascurso del juego. Esto puede causar una sobrecarga y retardo que empeora la experiencia de juego, ahorrando sin duda también recursos y tiempo de procesamiento y de renderizado.
- En realidad el jugador no es consciente de esta decisión y no puede predecir la curvatura del tubo durante el juego, lo cual tiene el mismo efecto que un tubo infinito aleatoriamente generado.

3.4.2. Modelado 3D

Una parte de la implementación del juego es la creación de modelos 3D, tanto de los obstáculos como del tubo sobre el que gira el jugador.

Creación de obstáculos

En el juego original existen muchos obstáculos de diferentes formas. Unity no ofrece objetos 3D más allá de un conjunto limitado de elementos primitivos (esferas, cubos, etc.), así que hemos elegido Blender como programa de modelado para crear los objetos base. La figura 3.17 muestra todos los obstáculos creados para el juego.

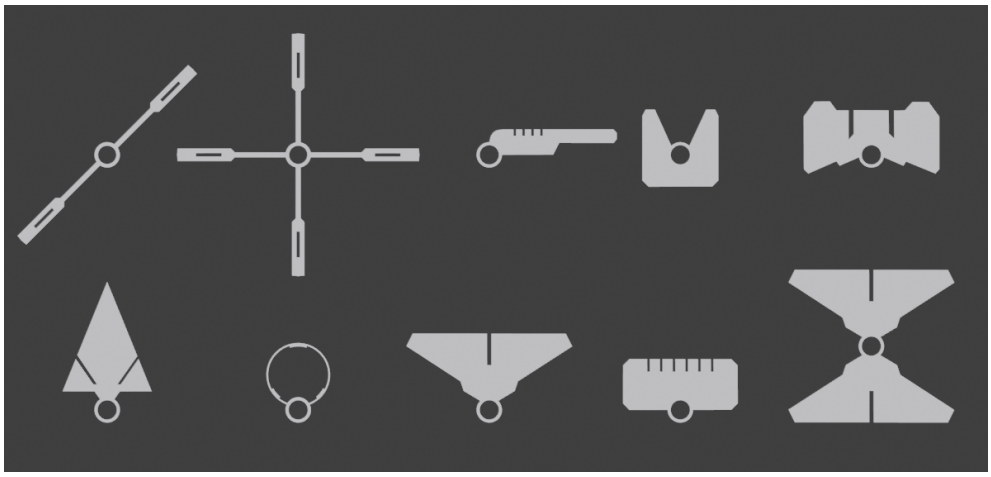


Figura 3.17: Colección de modelos de obstáculos en Blender

Como se observa en la figura, todos los obstáculos comparten un hueco con forma de "anillo", reservado para encajar en el tubo, de modo que el resto de su cuerpo sobresale hacia fuera del tubo, obstruyendo el "camino" del jugador sobre el tubo. La figura 3.18 ilustra el resultado de incorporar un obstáculo en la escena.

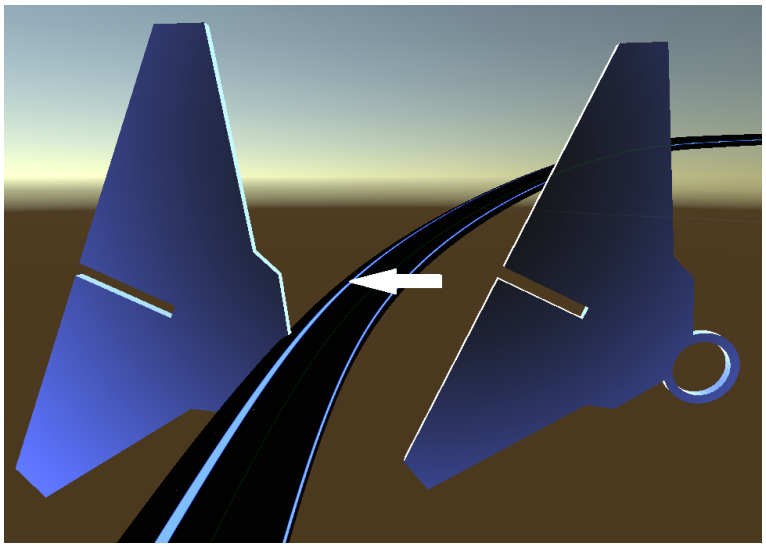


Figura 3.18: Un modelo en escena de juego

Diseño de obstáculos

Una vez importados los modelos como assets en Unity, pasan a ser objetos inmodificables, a los que añadimos un material para darle el aspecto visual deseado. Diseñamos un efecto neón con el fin de emular la estética del juego original.

En Unity, los objetos necesitan tener un material que define cómo tienen que ser renderizados, y los *shaders* son programas cortos utilizados por los materiales, que definen su modo de renderizado. Normalmente, todos los materiales utilizan "Standard Shader", que es suficiente para definir numerosas propiedades básicas. En nuestro caso, se optó por crear un shader propio para el efecto neón que se pretendía conseguir.

Primero necesitamos incorporar los paquetes "Shader Graph" [40] y "Lightweight RP (Render Pipeline)" [41]. El primero nos permite crear shaders fácilmente de manera visual mediante un grafo en vez de usando código, y el segundo es un *pipeline* que optimiza el comportamiento en tiempo real del *built-in renderer*. Creamos un shader "PBR Graph" (Physically Based Rendering Graph) que contiene los siguientes 5 nodos: la intensidad del efecto neón, el color, un nodo de efecto Fresnel que toma la intensidad como entrada, un nodo que combina el efecto Fresnel con el color, y por último el nodo maestro del shader. Las propiedades color e intensidad las definimos con unos valores iniciales, pero son configurables en el material que utiliza este shader. El grafo completo se observa en la figura 3.19.

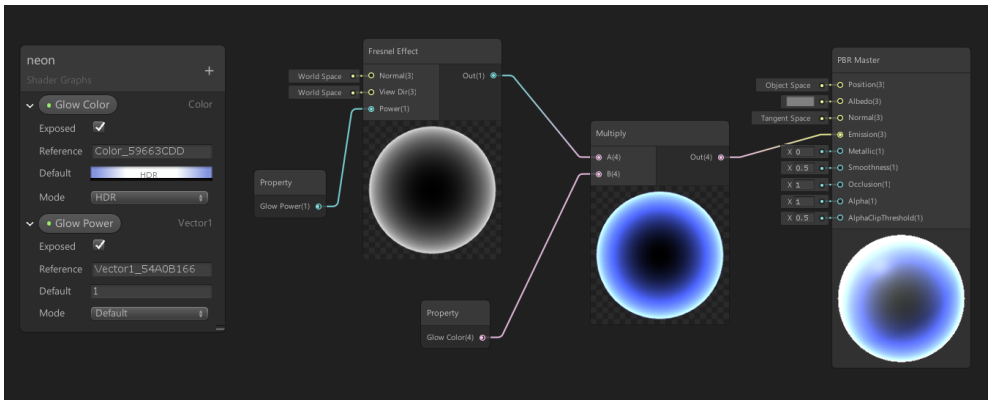


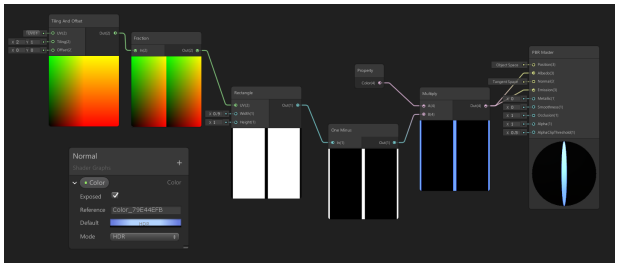
Figura 3.19: Shader Graph efecto neón

Creación del tubo

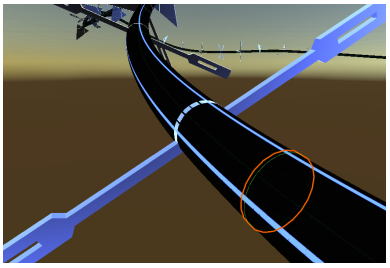
Una vez obtenida la curva de la trayectoria, necesitamos renderizar el tubo. En lugar de emplear técnicas avanzadas (como programar un shader de teselación para renderizar el tubo entero), decidimos utilizar segmentos cilíndricos de altura muy reducida apilados a lo largo de la curva diseñada. Programamos un shader para determinar el aspecto visual de cada segmento. La generación de los segmentos se controla mediante una extensión personalizada del editor de Unity (que explicaremos en el siguiente apartado).

3.4.3. Generación de obstáculos a lo largo de la curva

Necesitamos una mecánica efectiva para la creación de los obstáculos a lo largo de la curva en lugar de trasladar cada objeto manualmente y encajarlo en el tubo. Creamos el script "*ObstacleGenerator.cs*"



(a) Shader del tubo



(b) Tubo con obstáculos en el editor

Figura 3.20: Creación del tubo

para controlar este proceso.

Con la ayuda de la librería mencionada en 3.4.1, podemos recuperar la posición que corresponde a una distancia dada desde el punto inicial u origen de la curva, así como los vectores tangente y normal en el punto asociado. Con esas direcciones podemos obtener la rotación que debe tener el obstáculo y colocarlo sobre el tubo con su posición y rotación correspondientes. Para facilitar el proceso, extendemos el editor de Unity con el script "*ObstacleGeneratorEditor*" que tendrá la interfaz mostrada en la figura 3.21. De esta manera automatizamos la creación de los obstáculos con un botón (*GenerateNext*), proporcionando simplemente la información de su tipo, la distancia de referencia (*NextDistance*) y la diferencia entre la distancia del nuevo obstáculo que queremos crear y la de referencia (*DistanceGap*). La distancia de referencia para el siguiente obstáculo se actualiza automáticamente sumándose la cantidad de *DistanceGap*. El proceso de creación de un nuevo obstáculo se puede esquematizar de la siguiente manera:

1. Se introducen en el editor los datos correspondientes al tipo de obstáculo a añadir, la distancia sobre la que deseamos instanciarlo, y la separación entre este y el próximo obstáculo a añadir.

2. Al pulsar el botón de la herramienta, el sistema coloca el obstáculo descrito sobre la curva, e incrementa la distancia de referencia de acuerdo con los valores especificados. Por ejemplo, si decidimos instanciar un obstáculo en distancia = 0, con distancia de separación = 10, la nueva distancia de referencia para el próximo objeto pasa a ser = 10. Notamos que esto ocurre por un tema de conveniencia para el diseñador, evitándole así tener que introducir las distancias manualmente en secuencias de elementos equiespaciados.

3.4.4. Control

Como se ha explicado anteriormente, en este juego, el jugador sólo dispone de dos acciones simples: girar hacia la izquierda sobre el tubo y girar hacia la derecha. Implementamos el juego tanto para PC como para dispositivos móviles, antes de conocer los entornos específicos del experimento.

En la versión para PC del juego, dichas acciones se realizan en respuesta a entradas por teclado (flechas de dirección ← y →). En la versión móvil, en cambio, se implementan los dos tipos de control del juego original: el control táctil (refiriéndonos al tipo de control tocando la pantalla del dispositivo) y el control giroscópico (refiriéndonos al tipo de control girando el dispositivo como si fuera un volante). El manejo del control táctil es directo, pero para implementar el control giroscópico nos vimos obligados a tomar ciertas decisiones de diseño.

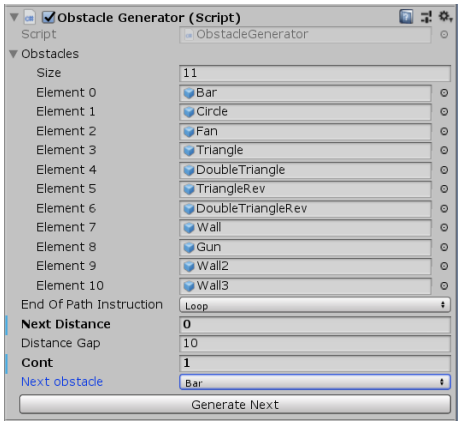
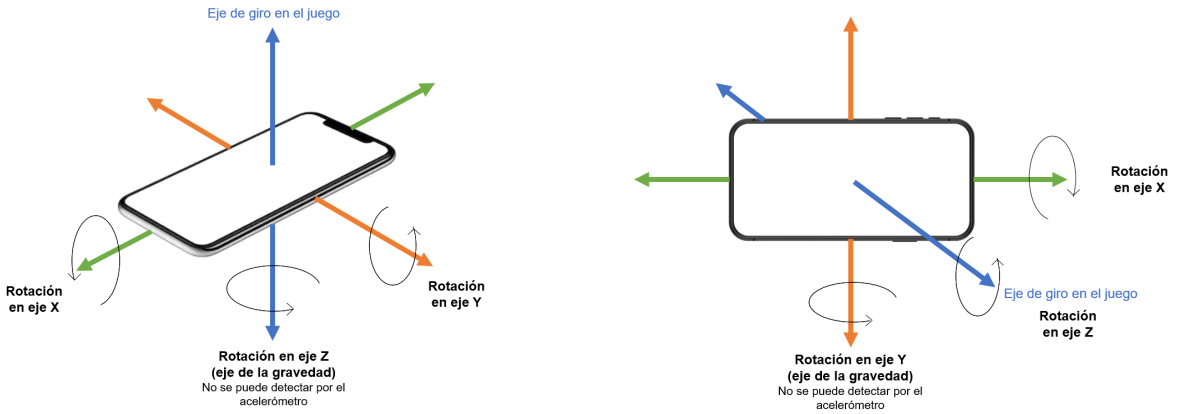


Figura 3.21: Generador de obstáculos en editor

Hay dos sensores en el dispositivo móvil que ofrecen esta funcionalidad: el acelerómetro y el giroscopio. Actúan como puntos de acceso a distintos tipos de información, como la aceleración y orientación del dispositivo.

El acelerómetro se encuentra en una mayoría de los dispositivos actuales y permite recuperar datos muy precisos sobre la aceleración de los mismos. Cuando el dispositivo se encuentra en reposo, recupera como aceleración única aquella asociada a la gravedad, y esto permite detectar la orientación del terminal. Sin embargo, este sensor no es capaz de detectar giros cualesquiera sobre el eje de la gravedad. Por lo tanto, solo es adecuado usarlo en el juego siempre y cuando se asuma que el dispositivo está colocado de manera vertical, como se indica en la figura 3.22b. Esto supone una limitación en tanto a que el jugador no puede colocar el móvil sobre una superficie plana (figura 3.22a), o jugar tumbado con la pantalla orientada hacia el suelo.



(a) Dispositivo sobre la mesa (giro no detectable) (b) Dispositivo en vertical (giro detectable)

Figura 3.22: Detectar giros con el acelerómetro

Una alternativa para eliminar esta restricción es utilizar el giroscopio. Frente al acelerómetro, este último añade otra dimensión de información, que es la rotación del dispositivo. Permite recuperar dicha rotación en cualquier eje, por lo que en principio es la opción ideal. No obstante, aunque este sensor no es una tecnología muy reciente, los dispositivos más económicos tienden a optar por no incorporarlo, debido a su coste relativamente elevado. Esto puede causar problemas críticos en el experimento para

aquellos participantes que no dispongan de un móvil con estas prestaciones. Con todas estas consideraciones, decidimos finalmente utilizar el acelerómetro. Su manejo en Unity es bastante conveniente, pues proporciona directamente los valores de la aceleración en cada eje respecto del móvil.

3.4.5. Implementación del movimiento

Una vez introducidos los diferentes esquemas de control del jugador, pasemos a estudiar su implementación en esta subsección.

Actualización de la posición y rotación del jugador

Cuando se recibe un input de entrada por parte del jugador, recurrimos a la función *RotateAround* [42] para llevar a cabo un giro sobre un punto del tubo en ese instante. Esta recibe el punto central sobre el que se pasará a realizar el giro, y el vector dirección su eje. Pero la aplicación no es trivial: la posición y rotación después de hacer el giro no se mantienen fijas para el siguiente fotograma del juego, ya que el jugador habrá cambiado de posición si la trayectoria presenta curvatura no nula (en caso de una recta, se mantendrían fijas). Si se aplica otro giro en el siguiente fotograma, el jugador eventualmente se alejará del tubo.

Para entender esto mejor, fijémonos en la figura 3.23. Consideramos que la curva trazada aquí representa la trayectoria del tubo. Denotemos por o el punto de la curva que actúa como el centro de rotación del jugador en el instante o fotograma actual, t el vector tangente a la curva en o, y n el vector normal a la curva en o. Supongamos que después de aplicar un giro, el jugador se encuentra en el punto indicado por el vector v (lo llamamos a partir de ahora vector *offset*; es la posición absoluta del jugador menos el origen o). En el siguiente instante del juego, el jugador avanzará y supongamos que el centro del giro se encontrará en el punto o'. El vector v ya no representa por tanto el offset de la posición del jugador desde el nuevo centro, sino el vector v'. Necesitamos realizar este cambio antes de aplicar el siguiente giro.

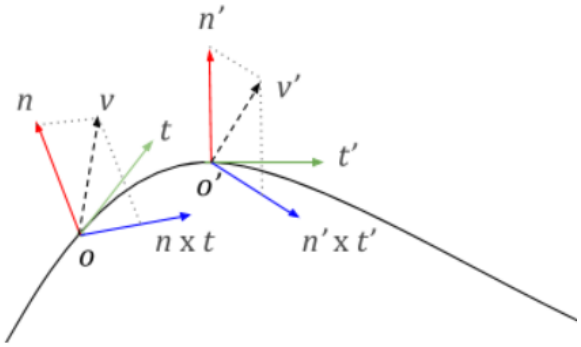


Figura 3.23: Cálculo de posición. Base local

Si nos fijamos en la base local en cada punto de la curva formada por el vector normal, el vector tangente y el producto vectorial entre ellos, entonces el vector que representa la posición del jugador tiene las mismas coordenadas en todas ellas (aunque este es distinto si consideramos coordenadas mundiales). Es decir, v y v' tienen las mismas coordenadas en la base local $\{n, n \times t, t\}$ en el punto o que en la

base local $\{n', n' \times t', t'\}$ en el punto o'. Por lo tanto, después de aplicar el giro, podemos obtener las coordenadas locales de v en la base $\{n, n \times t, t\}$, y volver a cambiarlas a coordenadas mundiales desde la base $\{n', n' \times t', t'\}$ en el siguiente instante, antes de procesar el siguiente giro.

Respecto de la rotación del jugador, se calcula fácilmente a partir de su posición. En todo punto, la rotación se computa como un Quaternion que tiene la dirección "forward" la del vector tangente y "upward" la del vector offset v.

Resumen

- Los juegos desarrollados en este trabajo fueron escogidos entre una lista de diez títulos comerciales considerados en el estudio de referencia. Más concretamente, se optó por implementar Blek, Edge y Unpossible, por sus respectivas afinidades con las capacidades cognitivas evaluadas en el estudio original.
- Se decidió desarrollar los juegos en el motor de Unity por la facilidad que este ofrece a la hora de cambiar entre distintas plataformas y la familiarización que ya tenía el equipo con el motor.

Habiendo comentado los detalles más importantes en la implementación de los juegos escogidos, en el siguiente capítulo pasaremos a detallar la primera versión del sistema de telemetría implementado para instrumentalizar estos juegos.

Capítulo 4

Desarrollo del sistema de telemetría

El objetivo del actual capítulo es realizar una muy breve descripción de los conceptos más importantes puestos en práctica durante la fase 2 del proceso de descubrimiento de conocimiento: la adquisición de datos. Nos incumbe aquí por tanto la implementación del sistema de telemetría o *tracker* a emplear por el equipo, incluyendo los métodos de transmisión (exclusivamente por eventos en este caso) y la infraestructura de datos (backend, servidor, BBDD) sobre la que se apoyará la fase de preprocesamiento de la etapa 3 del proceso de knowledge discovery.

4.1. Sistema de Telemetría

Tomando los puntos más importantes descritos en el capítulo 2, se optó por las siguientes decisiones de diseño y requisitos fundamentales:

1. El sistema de telemetría, siguiendo el razonamiento del capítulo 2, debe estar basado en eventos de cara a: (1) ser lo más general posible en cuanto a los tipos de juegos que es capaz de soportar y (2) ser lo suficientemente versátil como para permitir la inferencia de distintos tipos de métricas en un procesamiento posterior, incluso aquellas que no fueran inicialmente consideradas. Notamos que la condición (1) hace referencia a una cuestión de diseño de la infraestructura del sistema en sí, mientras que (2) pretende actuar como pauta de seguimiento general a la hora de seleccionar qué sucesos enviar desde los juegos por medio de dicho sistema.

2. Los eventos, dada la naturaleza de los experimentos a llevar a cabo, deberán tener como campos obligatorios, al menos, *identificadores* para el juego en el que ocurrieron y el *usuario* que los produjo, un *timestamp* marcando el instante (siguiendo el formato UNIX) en el que sucedieron, un *nombre* representativo que lo diferencie de otros eventos del juego, y un diccionario o lista de posibles *parámetros* adicionales que pudieran ser necesarios para el manejo y contextualización de los mismos.

3. El envío de eventos debe ser una tarea lo más sencilla posible desde la perspectiva del programador de cada juego, asemejándose su modo de uso idealmente al del mostrado en el caso de Unity Analytics en la sección 2.4.1. Esto quiere decir que la instrumentalización en el código debe ser clara e intuitiva de utilizar. Además, el código correspondiente debe ser reutilizable desde cualquier juego adicional implementado en el mismo motor sin adaptaciones adicionales.

Siguiendo las pautas anteriores, se desarrollaron dos sistemas de envío de eventos al servidor, uno para uso desde dispositivos móviles/ aplicaciones de escritorio, y otro adaptado a WebGL.

4.1.1. Implementación Móvil/ Escritorio

La implementación inicial del sistema de envío de eventos o *tracker* desde los juegos se desarrolló de manera independiente a cada uno de los juegos, y sus módulos correspondientes fueron encapsulados en forma de un archivo `.dll` que se distribuyó entre los integrantes del equipo para su incorporación en el proyecto. Tras ser importado en Unity, el módulo `.dll` expone un conjunto de clases y métodos, mayormente inspirados en las tecnologías descritas en la sección 2.4.1 para el proceso de envío o *log* de eventos. El código de la versión aquí comentada puede encontrarse en el repositorio [43].

El objeto principal aquí es el `ServerEventManager`, que puede instanciarse proporcionando la url con el endpoint del servidor al que se pretenden enviar los eventos del juego:

```
using namespace ServerEvents;
eventManager = new ServerEventManager(url);
```

Este gestor proporciona un método `logEvent` que recibe como único parámetro un objeto de tipo `ServerEvent`, con los campos descritos en los requisitos anteriores. Los parámetros opcionales se pueden incluir en forma de lista de objetos de tipo `ServerEventParameter`, que esencialmente funcionan como entradas de un diccionario de strings o un archivo `JSON`.

```
ServerEvent gameEvent = new ServerEvent(userId, timestamp, gameName, eventName,
    parameters);

ServerEventParameter[] parameters = {
    new ServerEventParameter(ServerEventParameter.LEVEL_NUMBER, "3"),
    new ServerEventParameter("CUSTOM_EVENT", "VALUE"),
    ...
};

eventManager.LogEvent(gameEvent);
```

Las clases de eventos y parámetros sirven meramente como contenedores de datos, destinadas a ser serializadas y transmitidas vía web. Adicionalmente, ofrecen un listado público de eventos "estándar" en la línea de los incluidos en Unity Analytics, aunque bastante más reducido, de uso común en los tres juegos a implementar:

```
#region event_constants
    public const string EXPERIMENT_START = "EXPERIMENT_START";
    public const string EXPERIMENT_END = "EXPERIMENT_END";
    public const string TUTORIAL_START = "TUTORIAL_START";
    public const string TUTORIAL_END = "TUTORIAL_END";
    public const string LEVEL_START = "LEVEL_START";
    public const string LEVEL_END = "LEVEL_END";
    public const string PLAYER_DEATH = "PLAYER_DEATH";
    public const string GAME_PAUSE = "GAME_PAUSE";
    public const string GAME_RESUME = "GAME_RESUME";
#endregion
```

Resulta pertinente comentar que los eventos de inicio y final de nivel suelen ir acompañados de un parámetro numérico, para determinar el número del nivel al que hacen referencia. Aparte de estos eventos

estandarizados, cada uno de los desarrolladores empleó un conjunto de eventos/parámetros adicionales exclusivos para sus juegos, que pasaremos a listar a continuación:

- **Blek**: se consideran los eventos adicionales `first_touch` (el jugador toca la pantalla por primera vez tras un periodo estático), `begin_drawing` (inicio de dibujado de un trazo), `begin_looping` (inicio de la fase de repetición de trazo) y `black_touched` (colisión con un obstáculo del juego). Ninguno de ellos se envía en ningún caso con parámetros adicionales.
- **Edge**: se consideran los eventos `got_item` (cubo coleccionable conseguido), `got_checkpoint` (alcanzada nueva marca de progreso en el nivel), y un parámetro adicional `num_moves` en los eventos de fin de nivel destinado a ofrecer un total de movimientos realizados en el mismo.
- **Unpossible**: no se consideran eventos adicionales, pero se añaden nuevos parámetros a los evento de tipo `player_death` con información sobre los giros y pulsaciones realizados por el usuario en cada dirección. Del mismo modo se incorporan parámetros para determinar el punto de la curva en el que el jugador murió en el intento correspondiente.

En cuanto a la implementación del `ServerEventManager`, una de las mayores dificultades fue mantener el carácter estático de la clase, sin vincularla como componente a un GameObject del juego. Esto último es la solución escogida por Google Analytics y Firebase para lanzar eventos desde Unity (véase la sección 2.4.2). No obstante, a nivel de uso, consideramos mucho más cómoda la versión nativa de Unity Analytics, donde se puede utilizar el gestor de eventos directamente, sin necesidad de añadir un objeto adicional a la escena. Sin embargo, las peticiones `HTTP` en el motor han de ser realizadas por medio de corrutinas, las cuales exigen a su vez ser llamadas desde un script derivado de `MonoBehaviour`.

La manera elegida para "burlar" esta restricción fue, quizá paradójicamente, utilizar lo que llamamos un `CoroutineLauncher`, una clase encargada de instanciar un objeto provisional sobre la escena al que se vincula una corrutina de envío de eventos al servidor. Si bien esto puede parecer contradictorio con lo dicho anteriormente, recordemos que lo que se busca aquí es evitar al desarrollador la necesidad de añadir un objeto encargado del proceso de registro de eventos en cada escena en la que necesite esta funcionalidad, pudiendo simplemente hacer uso de una clase estática accesible desde cualquier script de la aplicación. Así pues, el método `logEvent` se reduce a

```
public void LogEvent(ServerEvent gameEvent){
    CoroutineLauncher.StartCoroutine(eventClient.SendRequest(gameEvent));
}
```

donde `eventClient` no es más que el objeto encargado de realizar la petición `HTTP`, y `SendRequest` devuelve la corrutina que pasará a "inyectarse" en dicho GameObject provisional. Internamente:

```
public static Coroutine StartCoroutine(IEnumerator coroutine)
{
    // Create a new gameObject to hold our temporary MonoBehaviour.
    GameObject coroutineLauncher = new GameObject();
    // Add an instance of a MonoBehavior-extending component to the coroutineLauncher.
    CoroutineLauncherInstance routineHandler =
        coroutineLauncher.AddComponent(typeof(CoroutineLauncherInstance)) as
            CoroutineLauncherInstance;
    // Use our handler monobehavior to start the coroutine from an actual gameObject.
    return routineHandler.LaunchCoroutine(coroutine);
}
```

Una instancia de lanzador de corrutinas no es más que el propio script añadido al objeto generado, en el que se ejecuta la corrutina. Se garantiza además que este no desaparezca en un posible cambio de escenas, y que se destruya al finalizar la tarea para la que fue creado (y no antes).

En cuanto al ya mencionado `eventClient`, internamente hacemos uso de una función auxiliar que transforma un objeto de evento (`ServerEvent`) en un formato válido para el cuerpo de la petición, en este caso `JSON`. A modo aclaratorio, por un fallo en el sistema de Web Requests de Unity en el momento de desarrollo, es necesario realizar un "workaround" sobre las peticiones de tipo POST con un *payload* en `JSON`, que consiste en generar la petición como `PUT` para posteriormente modificar de forma manual su tipo.

```
private UnityWebRequest QueryRequest(object obj)
{
    // convert to a valid json-formatted string that can be accepted by server.
    string json = JsonUtility.ToJson(obj);

    // encode payload as an array of bytes so that it can be easily added to the request
        body.
    byte[] payload = Encoding.UTF8.GetBytes(json);

    // Generate POST request.
    UnityWebRequest request = UnityWebRequest.Put(url, payload);

    // attach upload/ download handlers and set content-type header to json.
    request.uploadHandler = new UploadHandlerRaw(payload);
    request.downloadHandler = new DownloadHandlerBuffer();
    request.SetRequestHeader("Content-Type", "application/json");
    request.method = UnityWebRequest.kHttpVerbPOST;

    return request;
}
```

Una vez comentados los aspectos más importantes de la librería de envío de eventos, pasemos a introducir los detalles fundamentales del backend desarrollado para recibirlos.

4.2. Desarrollo de infraestructura del servidor

La infraestructura utilizada se compone de dos elementos principales: una aplicación en Express para exponer una API con los distintos endpoints necesarios para interactuar con nuestro sistema, y una base de datos para gestionar los distintos objetos considerados.

Comenzamos describiendo las tecnologías empleadas para la base de datos. Como ya se detalló en las secciones anteriores, el tipo de datos principal a registrar en los juegos es el `Evento`, concebido como suceso atómico de cara a almacenar las trazas de interacción de los participantes con los juegos. La naturaleza variable de los eventos, que en principio pueden contener un número arbitrario de parámetros, disuade de utilizar una base de datos `SQL` convencional, a favor de bases de datos con estructuras menos rígidas como las `NoSQL`, que permiten representaciones más sencillas de manejar en nuestro caso. En consecuencia, se optó por emplear una base de datos *MongoDB* [44], que además ofrece planes gratuitos para gestión de conjuntos de datos en linea de alrededor de medio giga, más que suficiente para la tarea que nos ocupa. La base de datos se hospeda de este modo en la plataforma *Atlas*, ofrecida de forma nativa en la web de MongoDB.

En el caso del backend, se optó por montar un servidor escrito en *NodeJS*, basado en *Express* [45].

La decisión en esta parte fue cuestión de experiencia previa con las tecnologías y sencillez de uso en el desarrollo. En lugar de utilizar directamente el módulo nativo de MongoDB en el servidor, se decidió emplear *Mongoose* [46], un módulo que actúa como *wrapper* de Mongo y que permite aportar una cierta estructura a los elementos de la base de datos por medio de esquemas y modelos definidos por el programador. Esto facilita enormemente la tarea de trabajar con tipos de datos para los que se tiene una definición previa clara, como es el caso para nuestros eventos, y permite realizar operaciones sobre entidades asociadas a un cierto modelo, como búsquedas o modificaciones. A continuación mostramos la definición del esquema del modelo `Event` en Mongoose a modo ilustrativo:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const parameterSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    value: {
        type: String,
        required: true
    }
}, { noId: true} );
const eventSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    gameName: {
        type: String,
        required: true
    },
    timestamp: {
        type: Number,
        required: true
    },
    userId: {
        type: String,
        required: true
    },
    orderInSequence: {
        type: Number,
        required: true
    },
    parameters: [parameterSchema]
});
module.exports = mongoose.model('Event', eventSchema);
```

Como se puede comprobar, definir un esquema de estas características se reduce a una tarea de especificación. Notamos que lo que verdaderamente interesa desde fuera del módulo es el modelo compilado basado en dicho esquema, que es lo que nos permitirá realizar las operaciones pertinentes. Resulta importante hacer énfasis en un campo adicional que no se mencionó en secciones anteriores, `orderInSequence`. Este atributo se define como un parámetro numérico de todo evento que especifica la posición del mismo en la secuencia definida por una partida, y se añade como forma de ordenar los eventos de manera no ambigua en el postprocesado. Aquí hay que tener en cuenta que los métodos que ejecutan peticiones `HTTP` son por lo general asíncronos, y no se garantiza por tanto que los eventos alcancen el servidor de forma ordenada. Esto en principio no debería resultar problemático si decidimos ordenar por el campo

`timestamp`, pero dada la naturaleza de los eventos registrados, no resulta raro encontrar más de un evento que se produzca en un mismo segundo, generando así una situación de ambigüedad si no contamos con un campo explícito que dicte la numeración en la secuencia.

Una vez dicho esto, pasamos a resumir los dos *routers* básicos de la aplicación:

- **eventRouter**: encargado de la gestión de eventos en el sistema. Expone una única ruta `/events` (`POST`: enviar un evento al servidor). Los endpoints de tipo `GET` fueron eliminados para el paso de pos-procesamiento final, pues se pasó a descargarlos en archivos `JSON` generados por la herramienta de gestión *MongoDB Compass*, directamente desde la base de datos.
- **fileRouter**: encargada de procesar los datos de cada juego para generar un archivo `CSV` con las variables inferidas en cada caso. Actualmente en desuso tras decidir realizar la fase de pos-procesamiento íntegramente en Python a partir de los archivos `JSON` producidos en Compass. Exponía endpoints para la generación y descarga de dichos archivos para cada uno de los juegos.

La funcionalidad del router de eventos resulta especialmente sencilla, pudiendo resumirse en la de su endpoint principal:

```
router.post('/events', async (req, res) => {
    const event = new Event({...req.body})
    try {
        await event.save()
        res.status(201).send(event)
    } catch (e) {
        res.status(400).send(e)
    }
})
```

En esencia, nos limitamos a ejecutar una operación básica de guardado en la base de datos y, en caso de éxito, devolvemos al cliente el mismo evento como confirmación. Este backend, inicialmente muy limitado, será extendido en el siguiente capítulo para incorporar nuevas funcionalidades, como la gestión de usuarios de la plataforma web actualmente en uso, como se pasará a explicar allí.

Resumen

- Siguiendo las indicaciones desarrolladas en el capítulo 2, se diseñó un sistema de telemetría genérico para llevar a cabo una instrumentalización de los juegos escogidos. Este sistema estaba en principio concebido para su uso en versiones Android y PC de los títulos implementados, y se distribuyó a los integrantes del equipo en forma de archivo `.dll` para su uso desde Unity.
- La librería correspondiente exponía una serie de objetos y métodos para el envío de sucesos a los endpoints del servidor de manera similar a la estrategia empleada por Unity Analytics. Esta ofrecía un conjunto de eventos estandarizados para uso común desde los tres juegos, y permitía asimismo el uso de eventos personalizados, exclusivos para cada juego.
- De forma paralela, se desarrolló un backend muy sencillo en Express con base de datos MongoDB para el almacenamiento y gestión de eventos. La infraestructura resultante se corresponde con la parte técnica de la fase de adquisición de datos del proceso de descubrimiento de conocimiento.

En el siguiente capítulo estudiaremos el protocolo original del experimento de referencia, e introduciremos numerosos cambios y mejoras sobre el backend desarrollado en esta parte de la memoria debidos a una necesidad de revisión de las primeras fases del proceso de knowledge discovery.

Capítulo 5

Desarrollo del experimento

Una vez satisfechos los requisitos técnicos de la fase de adquisición de datos, esto es, la implementación e instrumentalización de los juegos, y el lanzamiento del servidor para la recogida de eventos, pasamos a especificar los requisitos logísticos de la misma. El comienzo del capítulo está dedicado a presentar el protocolo que define el experimento inspirado por el estudio de referencia. A continuación introducimos los nuevos requisitos técnicos de cara a realizar ciertas modificaciones necesarias como consecuencia del cambio situacional en el trabajo. Una vez detallados los cambios realizados, introducimos una serie de consideraciones sobre las nuevas condiciones del experimento y describimos el desarrollo del mismo. Por último, se define el conjunto de métricas a inferir a partir de los eventos registrados para incorporar su uso en la fase de análisis.

5.1. Protocolo del experimento original

De cara al objetivo principal de este trabajo, resulta interesante tratar de reproducir el experimento del estudio de referencia [8] de la manera más fiel posible, si bien instrumentalizando el registro de datos e incorporando métricas y análisis adicionales, que discutiremos más adelante. Los experimentos ahí considerados consistían en un conjunto de pasos que listamos a continuación:

- En primera instancia, se introduce una primera sesión obligatoria a modo de *tutorial*, para que el participante pueda familiarizarse con las mecánicas de un juego dado.
- Una vez completado el tutorial correspondiente, se lleva a cabo una fase de *sesión* de experimento, donde pasa a registrarse el comportamiento del jugador para inferir las variables de interés incluidas en un análisis posterior.
- Tras finalizar las fases de sesión de todos los juegos, se administra un conjunto de *tests de inteligencia* estandarizados al participante, uno por cada capacidad cognitiva considerada en el estudio.
- En última instancia, se insta a los participantes a cumplimentar un *cuestionario de hábitos de juego*, con vistas a una posible normalización de los resultados en el análisis.

En el capítulo 3, presentamos los detalles técnicos de cada juego, sin especificar los niveles incluidos ni la duración de los mismos. Listamos aquí brevemente las condiciones de los tutoriales y de las sesiones consideradas para cada uno de los juegos escogidos:

- **Blek**: Se presentan 8 niveles a modo de tutorial, a completar sin límite de tiempo por parte del participante, y una segunda parte de 10 minutos de sesión en la que el usuario deberá completar tantos niveles como pueda de un total de 27.
- **Edge**: Al igual que en Blek, se presentan 3 niveles de tutorial sin límite de tiempo, seguidos de 12 minutos de sesión en los que el usuario deberá completar todos los niveles que pueda sobre un total de 9.
- **Unpossible**: Se introduce un nivel de tutorial de 2 minutos para que el jugador se habitúe a la mecánica del juego sobre un nivel de menor dificultad. La sesión del experimento real se alarga hasta 5 minutos en la que el jugador deberá tratar de aguantar el mayor tiempo posible sobre un segundo nivel.

En este punto del trabajo, se pasó a planificar el experimento mediante la redacción de un protocolo de actuación, adjunto como anexo A. Este se elaboró tratando de seguir en la medida de lo posible las directrices del experimento de partida, con ciertas adaptaciones a nuestro contexto particular. El protocolo contenía indicaciones referentes a los pasos a seguir antes y durante las sesiones por parte de los miembros del equipo, los materiales necesarios para poder llevarlas a cabo (tablets y ordenadores), y los requisitos de espacio correspondientes (salas, laboratorios). Las limitaciones materiales y personales llevaron a concebir la organización de los experimentos en varias "tandas" de 15 a 20 participantes por día, en lugar de pretender recoger en una única sesión muestras de los 100 participantes que inicialmente se establecieron como objetivo. Se planeó una primera sesión con un conjunto limitado de sujetos, destinada fundamentalmente a servir como experimento piloto y test, para comprobar que tanto el protocolo como la infraestructura web eran adecuados de cara a proseguir con futuras "tandas", y más participantes.

Sin embargo, debido a las nuevas condiciones por la pandemia del COVID-19, surgió la necesidad de realizar una serie de modificaciones sobre los requisitos del protocolo original. Esto se tradujo esencialmente en una revisión de las fases iniciales del proceso de *knowledge discovery* seguido a lo largo del trabajo, así como en la búsqueda de alternativas a la hora de desarrollar una nueva iteración del proceso. Recordemos que el tracker a emplear para el registro de datos especificado en el capítulo anterior estaba concebido inicialmente para su uso en versiones móvil y de escritorio, lo cual deja de ser una opción en esta iteración. La revisión optó por una estrategia web para la administración de los juegos, que pasaremos a detallar en la siguiente sección.

5.2. Modificaciones en la fase de adquisición de datos

Comenzando con la especificación, los nuevos requisitos para el proyecto podrían resumirse en los puntos siguientes:

1. Los tres juegos escogidos deben estar centralizados en una única plataforma web accesible por cualquier usuario por medio de un navegador web. Las limitaciones del motor de Unity para WebGL nos llevan a ser aún más específicos: al no soportar este dispositivos móviles, sólo se tendrá como objetivo crear una plataforma dirigida a navegadores de escritorio.
2. Pasa a ser necesario garantizar un cierto seguimiento y control de usuarios con el fin de identificar de forma adecuada a los jugadores de cada partida. En consecuencia, el sistema debe implementar una gestión básica de perfiles de usuario, ofreciendo asimismo la correspondiente pantalla inicial de registro/ acceso a cuenta.

3. No se exigirá a los nuevos usuarios un registro con dirección de correo o similares, por un tema de protección de datos, unido a una necesidad de facilitar el proceso de entrada al sistema en la medida de lo posible. No obstante, sí que será necesario almacenar un identificador único por cada jugador, así como su género y su edad, pero se ha de garantizar el anonimato de todo usuario, siempre y cuando estos escojan un identificador de usuario adecuado.

4. Con el fin de ajustar los datos de cada partida, el usuario deberá responder a un breve cuestionario inicial en el que indique su grado de conocimiento y experiencia previos con cada uno de los juegos del sistema.

Habiendo listado los requisitos mínimos del sistema a implementar, pasemos a detallar las decisiones en cuanto a tecnologías y diseño alcanzadas en esta etapa del proceso.

5.2.1. Backend y Base de datos

Comenzamos describiendo los cambios fundamentales realizados sobre el backend de la plataforma. Como ya se detalló en el capítulo 4, el tipo de datos principal a registrar en los juegos es el *Evento*, y los nuevos requisitos del sistema nos llevan a incluir un tipo *Usuario* para almacenar los datos relativos a las sesiones. Consecuentemente, se introduce un nuevo *router*, `userRouter`, encargado de realizar la gestión de usuarios en el sistema. Expone las rutas `/users` (`POST`: dar de alta a un nuevo usuario), `users/login` (`POST`: iniciar sesión), `/users/logout` (`POST`: cerrar sesión), y `/users/me` (`GET`: obtener perfil del usuario actual, `PATCH`: actualizar perfil de usuario, `DELETE`: eliminar perfil de usuario).

Como ya se hizo anteriormente para eventos, definimos un nuevo esquema en Mongoose para representar usuarios, con numerosos campos para almacenar la información ya especificada en los requisitos, y obtenida por medio del proceso de registro. Esta incluye datos como el sexo, la edad, o un identificador personal de usuario. Se añade, asimismo, un campo de contraseña debidamente encriptada en el paso de generación del usuario. Esto se hace de la siguiente manera, utilizando el módulo `bcrypt` de `npm`:

```
userSchema.pre('save', async function(next){
    const user = this
    if(user.isModified('password')){
        user.password = await bcrypt.hash(user.password, 8)
    }
    next()
})
```

Esencialmente, el código anterior garantiza que las contraseñas de usuario pasen por una etapa de encriptado antes de ser almacenadas en la base de datos. Normalmente, ocho rondas de encriptado suelen considerarse suficientes para la mayoría de usos típicos. Las contraseñas, por tanto, no pueden ser consultadas directamente por el gestor del sistema. Lo que se hace en el momento de realizar un login por parte de un usuario es utilizar el mismo módulo de encriptado, para generar un *hash* del string introducido como contraseña:

```
userSchema.statics.findByCredentials = async (userId, password) => {
    const user = await User.findOne({userId})
    if(!user){
        throw new Error('Unable to login')
    }
    const isMatch = await bcrypt.compare(password, user.password)
    if(!isMatch){
        throw new Error('Unable to login')
```

```
    }
    return user
}
```

De este modo, cada vez que se produce un intento de login, se busca en la base de datos el usuario con ID igual al proporcionado, y se compara una versión "hasheada" de su contraseña real con la dada para comprobar la equivalencia.

El esquema de usuario contiene también campos que podemos considerar "de progreso" (`gameX_completed`) y que determinan si el usuario en cuestión ha completado cada uno de los juegos, además de campos de respuestas a los cuestionarios sobre experiencia previa con cada uno de los juegos de la plataforma. El último campo, de carácter esencial en el proceso de autenticación, es el de `tokens`, que mantiene un listado de sesiones abiertas/ activas para un usuario dado. Los tokens de autenticación pueden generarse mediante el proceso siguiente con ayuda de la librería `JSON Web Tokens` (`JWT`):

```
userSchema.methods.generateAuthToken = async function() {
    const user = this
    const expiration = process.env.DB_ENV === 'testing' ? 100 : 604800000
    const token = jwt.sign({'userId' : user.userId}, process.env.JWT_SECRET, {
        expiresIn: process.env.DB_ENV === 'testing' ? '1d' : '7d',
    })
    user.tokens = user.tokens.concat({ token })
    await user.save()
    return { token, expiration }
}
```

El mecanismo de autenticación por medio de tokens se completa con una pieza de *middleware* colocada en todos los endpoints, a excepción de las rutas de login y logout, por razones obvias. Esto no es otra cosa que una función preliminar por la que pasa una petición antes de alcanzar el endpoint correspondiente, que en nuestro caso actúa como un "filtro" para impedir el avance de peticiones no autorizadas:

```
const auth = async (req, res, next) => {
    try {
        const token = req.cookies.token || ''
        const decoded = jwt.verify(token, process.env.JWT_SECRET)
        const user = await User.findOne({ userId : decoded.userId})

        if(!user){
            throw new Error()
        }
        req.user = user
        req.token = token
        next()
    } catch (e) {
        res.status(401).send({error : 'Please authenticate'})
    }
}
```

Los *JSON Web Tokens* (`JWT` para abreviar) pueden definirse, de acuerdo con [47] como un estándar abierto (RFC 7519) que define un modo compacto y autocontenido de transmitir información de manera segura entre aplicaciones como objeto `JSON`. Los `JWTs` pueden firmarse utilizando lo que se conoce como un "secreto" (en el código anterior, `JWT_SECRET`) de manera que sólo la parte que firma el token es capaz de verificar su validez. Los datos encapsulados en el token pueden, sin embargo, ser accedidos de forma sencilla si no se someten a un proceso de encriptado previo, pero en nuestro caso particular, al no incluir en ellos ningún tipo de información sensible, sólo nos interesa comprobar que un token en cuestión ha sido

firmado por nuestra aplicación y no por un tercero, sin importar especialmente los datos almacenados en él.

Volviendo al código de arriba, cuando generamos un token de autenticación, guardamos la fecha de expiración del mismo, a partir de la cual dejará de ser considerado válido por nuestro verificador, así como el nombre de usuario al que pertenece, y lo firmamos con un secreto arbitrario, fijo en el backend. El token pasa a añadirse a la lista `tokens` del usuario, y permanecerá ahí mientras que este no cierre sesión o se sobrepase el momento de expiración. La comprobación, realizada por el middleware `auth`, de que un usuario está correctamente autenticado, se reduce por tanto a buscar en la base de datos un usuario con el nombre del proporcionado en el cuerpo del token, y comprobar si alguno de los elementos de su lista es verificable con la firma del sistema (y sigue siendo válido).

Una ventaja adicional de este método es que antes de continuar la cadena de llamadas hasta la ruta solicitada en el servidor, podemos añadir datos adicionales al cuerpo de la petición, como el usuario buscado en la base de datos, para evitar tener que volver a determinarlo en la propia llamada de la ruta. Llegados al endpoint correspondiente, podemos asumir asó directamente que `req` contiene ya un objeto con el usuario de la petición.

Una última idea clave para entender el funcionamiento del proceso de autenticación es la de las Cookies de tipo `HTTP-only`. Una primera aproximación ingenua a la gestión de los tokens descritos anteriormente sería devolver un token al cliente, en el cuerpo de la respuesta a una llamada con éxito a los endpoints de login o registro. El cliente entonces pasaría a almacenar el token de sesión de manera local en el propio navegador, pero el utilizar esta estrategia hace que los tokens sean vulnerables a ataques `XSS` por parte de terceros. Una de las soluciones más comunes a este problema es el envío de Cookies `HTTP-only` por parte del servidor al cliente:

```
router.post('/users/login', async (req, res) => {
    try {
        const user = await User.findByCredentials(req.body.userId, req.body.password)
        const { token, expiration } = await user.generateAuthToken()
        res.cookie('token', token, {
            expires: new Date(Date.now() + expiration),
            httpOnly: true
        })
        res.send(user)
    } catch (e) {
        res.status(400).send()
    }
})
```

que encapsule el token correspondiente. Este tipo de Cookies no es directamente accesible desde el código del cliente, lo cual se traduce en una mayor seguridad. No obstante, estas pueden ser enviadas en las peticiones `HTTP` correspondientes activando el campo de opciones asociado en los parámetros de la llamada. En en caso de fetch, por ejemplo, esto se consigue introduciendo `includeCredentials : true`. En el caso del logout, basta con eliminar la Cookie desde el código del servidor, así como el token de sesión pertinente en la lista de tokens de usuario en la base de datos.

El código actual del backend descrito aquí puede encontrarse en el repositorio [48]. Continuamos comentando las modificaciones en la infraestructura de adquisición de datos en el proceso de *knowledge discovery*, pasando ahora a los cambios en la instrumentalización por código desde los juegos.

5.2.2. Instrumentalización web

La decisión de migrar los juegos a una plataforma web permitió realizar diversos cambios en el sistema descrito en el capítulo 4 para simplificar el proceso de envío, además de eludir ciertas dificultades producidas en el navegador al tratar de incorporar las credenciales de usuario desde el motor de WebGL.

Los diversos intentos de mantener el modelo anterior en web nos llevaron a la conclusión de que, quizá, la manera más conveniente de realizar las peticiones `HTTP` de envío de eventos era delegando esta tarea a la propia aplicación web. Esto conlleva por tanto un abandono de la librería introducida en el capítulo 4 sobre las versiones web. Cuando hablamos aquí de delegar el envío de eventos a la aplicación web, nos referimos a establecer un puente entre el motor de Unity y Javascript, de manera que mandar un evento desde un juego se limita a elevarlo a la web donde está hospedado, que a su vez se hace cargo del envío al servidor. Estos envíos desde la web son sensiblemente más simples que los concebidos para el caso móvil/escritorio, y trivializan la introducción de credenciales de acceso sobre las peticiones `HTTP` a los endpoints del servidor.

Para este tipo de comunicación Unity-Javascript, se utiliza una estrategia semejante a la descrita en [49]. Para ello, se introducen los siguientes métodos externos en Unity:

```
[DllImport("__Internal")]
private static extern void LogGameEvent(string eventJSON);

[DllImport("__Internal")]
private static extern void GameOver();
```

que no son más que referencias a funciones de Javascript definidas en el fichero `WebEvents.jslib` en la carpeta de Plugins:

```
mergeInto(LibraryManager.library, {
    LogGameEvent: function (eventJSON) {
        ReactUnityWebGL.LogEvent(Pointer_stringify(eventJSON));
    },
    GameOver: function() {
        ReactUnityWebGL.GameOver();
    }
});
```

El objeto `ReactUnityWebGL` proviene de una librería `npm` externa [50] de la que hacemos uso para gestionar la comunicación entre Unity y el proyecto de React en la web. La introducción de un método `GameOver` desde el juego sirve la función de notificar al navegador del final de la partida, para propósitos de cambio de página. En cuanto al uso en el frontend de estos métodos, la librería anterior permite realizar llamadas del siguiente modo:

```
this.unityContent.on("LogEvent", eventJSON => {
    const event = JSON.parse(eventJSON);
    fetch(API_BASE_URL + 'events', {
        method: 'POST',
        body: JSON.stringify(event),
        credentials: 'include',
        headers: {
          'Content-Type': 'application/json'
        }
    })
    .then(response => console.log(response))
    .catch(err => console.log(err));
```

```
});
```

lo que proporciona una manera sencilla de realizar las peticiones `HTTP` correspondientes al servidor sin tener que aumentar la complejidad en Unity. De este modo, el procedimiento de envío de eventos se puede resumir de la siguiente manera:

1. Desde un script de Unity (ya no hay necesidad de que este sea `MonoBehaviour`, pues desaparece el uso de corrutinas), en el que se haya declarado el método externo `LogEvent`, se realiza una llamada a este último con un objeto `WebEvent` como parámetro, totalmente análogo a `ServerEvent` en el caso anterior.
2. En el método de la librería de JavaScript, el objeto se transforma en `string` por cuestiones de compatibilidad, y se lanza un evento parametrizado de `ReactUnityWebGL` con dicho string como argumento.
3. El componente de Unity en React recibe el evento y ejecuta el código de envío al servidor, tras generar el payload correspondiente en formato `JSON`. Para ello, dicho comportamiento se registra previamente en la inicialización del componente del juego en el frontend.

Pasemos a continuación a presentar la última pieza necesaria para completar la infraestructura de la adquisición de datos.

5.2.3. Frontend y plataforma web

Como ya se ha comentado por encima en la sección anterior, se optó por emplear *React* [51] como framework para la creación del frontend, que se puede encontrar hospedado por Heroku en [52], y el código asociado en el repositorio [53]. Esta decisión se reduce a una cuestión de comodidad y agilidad en el desarrollo de la plataforma. Esta es una librería de JavaScript destinada a facilitar la producción de interfaces de usuario interactivas para aplicaciones web de una sola página, y tiene la ventaja de estar fuertemente basada en componentes, lo que promueve la reutilización de contenido de manera sencilla. En esta sección realizaremos un breve repaso de la estructura en componentes de la aplicación, comentando algunos de los puntos más importantes.

La aplicación principal utiliza el `Router` de React para distinguir entre dos "pantallas" base sobre las que se construyen todas las demás: `Iag` (Intelligence Assessment Games) y `Login`. La primera es la encargada de albergar el propio contenido de la plataforma, mientras que la segunda se centra en las etapas de login y registro de usuarios.

Empecemos comentando la componente asociada a `Login`. Esta mantiene un estado interno con todas las distintas variables ligadas a los campos modificables por el usuario por medio de distintos formularios, mientras que los botones de inicio de sesión o registro lanzan los métodos asociados para realizar peticiones `HTTP` al servidor. Los errores en el proceso se traducen en mensajes por pantalla al usuario, especificando la naturaleza del problema, mientras que un inicio de sesión con éxito cambia la ruta actual a `/questions`, con el cuestionario sobre experiencia previa con los juegos.

Cabe mencionar que el sistema almacena de manera local una variable `isLoggedIn` que determina si el jugador debería ser redirigido a una u otra página en función de su valor. De este modo, si la variable está activa, cualquier intento de acceso a la pantalla de registro producirá un redireccionamiento al perfil del usuario, mientras que la situación opuesta redirige todo intento de acceso a secciones del perfil a la pantalla mostrada en la figura 5.1.

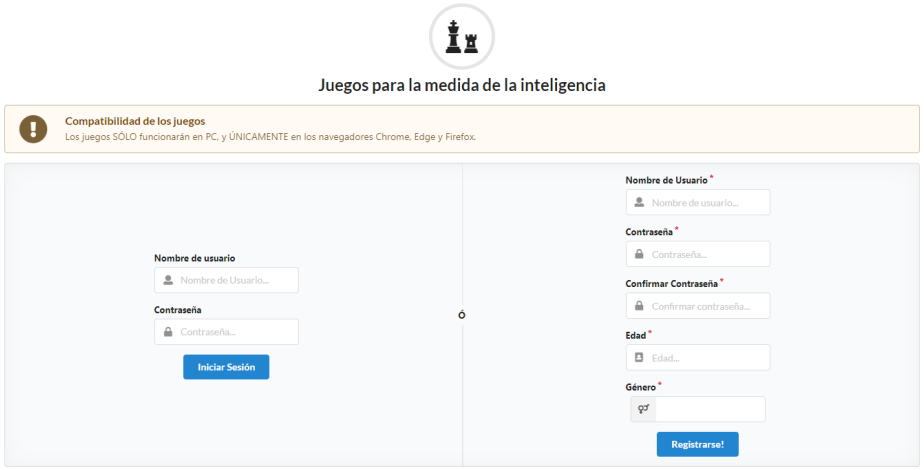


Figura 5.1: Pantalla de Login y Registro de la plataforma

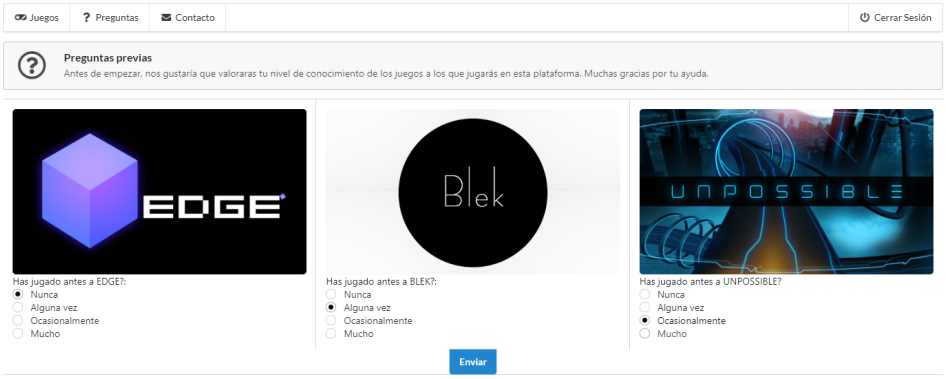


Figura 5.2: Pantalla cuestionario sobre experiencia previa

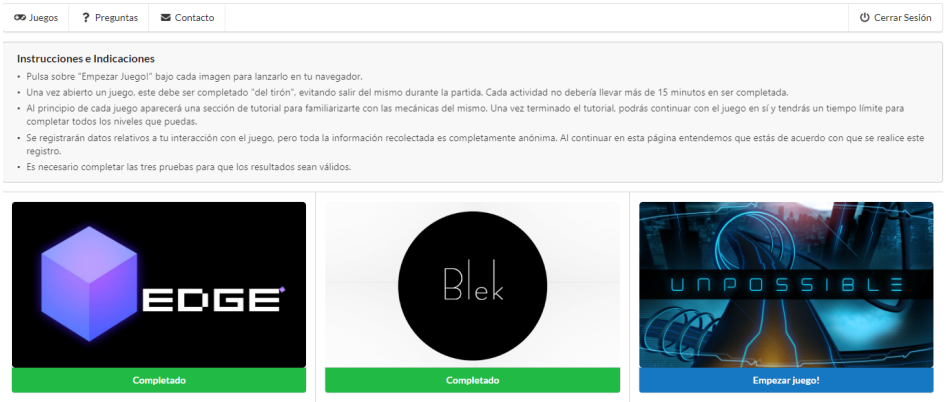


Figura 5.3: Pantalla de instrucciones y acceso a juegos

Tras un registro con éxito, o un inicio de sesión de un jugador que aún no haya completado el las preguntas previas, el sistema redirige al usuario a la página de cuestionario mostrada en la figura 5.2, donde este puede especificar su grado de conocimiento de cada uno de los juegos de la plataforma. Nos

encontramos aquí ya dentro del componente principal, `Iag`, que mantiene un subcomponente de menú de navegación en todas las pantallas en las que el usuario tiene una sesión activa. Cerrar sesión, por su parte, referencia a un método destinado a deshabilitar la variable `isLoggedIn` en el navegador, así como a realizar una petición `HTTP` al servidor en el endpoint `/users/logout`, ya explicado en la sección 5.2.1. Esto a su vez produce una redirección a la página de inicio de sesión.

Una vez enviado el formulario al servidor y habiendo actualizado la base de datos, se muestra la pantalla principal del perfil de usuario (figura 5.3), gestionada por el componente `Games`, que lista las instrucciones del experimento, una serie de consejos para los jugadores, y enlaces a cada uno de los items a completar a lo largo de la sesión. Estos pueden ser jugados en cualquier orden, quedando marcados en verde como "Completado" al finalizarlos. Sin embargo, no se restringen la posibilidad de iniciar partidas adicionales, en caso de que el usuario quisiera volver a probar alguno de ellos.

Figura 5.4: Sección de vídeo de introducción a Blek

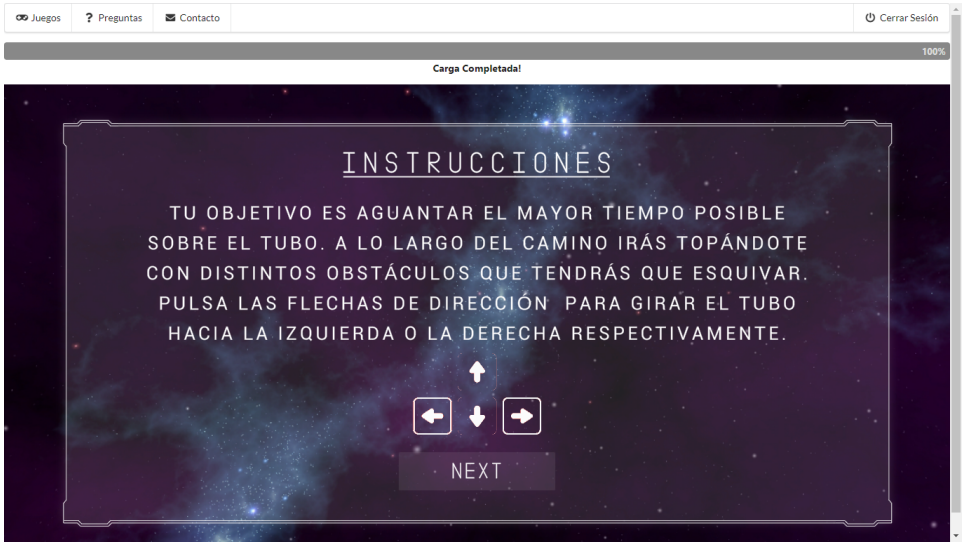


Figura 5.5: Pantalla de instrucciones de Unpossible

Al hacer click sobre alguno de los títulos jugables, el sistema pasa a cargar el componente correspondiente a cada juego, denotados `Blek`, `Edge` y `Unpossible`, respectivamente. Los tres son esencialmente idénticos, a excepción de Blek, que introduce un vídeo previo para explicar de manera más intuitiva el

manejo del juego. Esta decisión se tomó tras una primera prueba con un número limitado de usuarios, al comprobar que varios de ellos consideraban las mecánicas de Blek sensiblemente difíciles de entender. En cualquier caso, la comunicación de Unity WebGL con React ya se aclaró en el apartado de implementación correspondiente, y el método `GameOver` que quedó sin explicar entonces simplemente se limita a redirigir al usuario a su perfil principal.

Las versiones web de los juegos pasan a estar rediseñadas para tener un flujo puramente lineal: los menús con varias opciones quedan reemplazados por botones únicos para continuar con la sesión, y los tres juegos quedan estructurados en tutorial $\implies$ sesión, con una pausa intermedia entre ambas partes. La barra de carga en la parte superior de la pantalla se introdujo por un tema de feedback visual al usuario. Durante las primeras pruebas llegamos a la conclusión de que los juegos tardaban unos segundos en cargarse por completo en algunos navegadores, dando la sensación inicial de que la página no funcionaba o se había quedado congelada. La incorporación de esta información visual garantiza así que los jugadores esperen a que la carga termine de completarse.

5.3. Nuevo entorno del experimento

Con el fin de entender las implicaciones de la nueva plataforma escogida para el experimento final, realicemos una serie de observaciones sobre las posibles ventajas e inconvenientes que esta podría tener frente al experimento físico. En cuanto a puntos positivos, podemos encontrar los siguientes:

- **Comodidad de administración**. La distribución de los juegos por medio de una plataforma web facilita notablemente el proceso de administración de los mismos, tanto para los participantes como para los organizadores, al no existir ya la necesidad de realizar sesiones presenciales.
- **Transporte**. En la linea del punto anterior, el hecho de poder administrar los juegos a los participantes remotamente desde su casa fomenta la participación general, al reducir el "esfuerzo percibido" por parte de la población hacia los experimentos.
- **Ausencia de límite de participantes por sesión**. Desaparece la necesidad de realizar el experimento en "tandas", como se comentó anteriormente, permitiendo un número en principio ilimitado de sesiones diarias, siempre y cuando el servidor lo permita.
- **Ausencia de límite de recursos**. En la misma linea, desaparece el problema de la limitación de recursos, al no ser ya necesario proporcionar dispositivos específicos como ordenadores o tablets. Cada participante podría jugar simplemente desde sus propios dispositivos.

Pasando a puntos negativos, podemos resumirlos del siguiente modo:

- **Falta de control**. La flexibilidad ya comentada anteriormente viene asociada de forma directa a una clara falta de control sobre los sujetos durante los experimentos. Esto se traduce en que resulta virtualmente imposible llevar un seguimiento o monitorización del comportamiento de los usuarios a lo largo de las sesiones, pudiendo darse un gran número de sucesos que interrumpan o afecten negativamente al curso del experimento. Estos pueden ser interrupciones (llamadas telefónicas entrantes), problemas técnicos (fallos de red o del dispositivo), o ayudas no controladas por parte de familiares y amigos, que pueden distorsionar los resultados reales del sujeto, por nombrar algunos casos comunes. Del mismo modo, resulta poco factible proporcionar apoyo técnico a todos los participantes de forma remota.

- **Consentimiento**. En el experimento de referencia, cada participante debía firmar una autorización o consentimiento para realizar los experimentos de forma confidencial, lo cual resulta más complicado en la versión online de los mismos.
- **Unificación de plataformas**. Como ya se comentó en la sección 5.2, los tres juegos han de estar centralizados en una única plataforma web, que por limitaciones del motor de Unity en WebGL no puede ser compatible con dispositivos móviles. Esto supone la necesidad de realizar modificaciones en los esquemas de control de Blek y Unpossible, juegos originalmente planteados para dispositivos móviles. De este modo, los controles de Blek han de ser adaptados a ratón (en lugar de dedo) y en el caso de Unpossible se opta por un manejo exclusivamente por teclas direccionales.
- **Requisitos web**. Como ya tratamos en profundidad en la primera parte de este capítulo, resulta necesario desarrollar un cierto sistema de autenticación para la gestión de usuarios, y pasa a haber una necesidad de desarrollar un frontend completo para la distribución web.
- **Tests de inteligencia**. Los tests de inteligencia administrados en el experimento de referencia no pueden ser transmitidos de manera online, por estar estos protegidos por copyright. Los formularios asociados sólo pueden ser completados de manera física y ser devueltos a los organizadores para la evaluación, lo cual no sería posible en el nuevo sistema. Si bien inicialmente se planteó el empleo de tests online a modo de sustitución para los estandarizados, no fue posible encontrar ninguno lo suficientemente completo como para verdaderamente justificar su uso.

Se asume aquí que las desventajas listadas en este apartado derivarán de forma inevitable en ciertas imprecisiones en los datos, en partidas no válidas por cualquiera de los motivos ya expuestos. Con estas consideraciones en mente, procedemos a describir el proceso seguido en el experimento online para el registro de datos de los participantes voluntarios.

5.3.1. Desarrollo del experimento

En términos generales, podemos distinguir el experimento en dos fases:

1. En primera instancia, se realizó una distribución inicial de la plataforma web, limitada a un grupo reducido de participantes. En esta primera fase se pretendía llevar a cabo un primer estudio piloto para evaluar los potenciales problemas del nuevo entorno del experimento. Si bien se siguió un proceso similar durante el propio desarrollo de los juegos, a fin de detectar posibles errores y verificar el correcto funcionamiento del backend y la base de datos, se consideró oportuno añadir esta fase previa de prueba con vistas a enmendar fallos imprevistos, o relacionados con el frontend. Esta fase fue completada a lo largo de una semana a principios de mayo, sobre un conjunto de 17 usuarios. A partir del feedback de dichos participantes, se incorporaron distintos cambios en la plataforma con el fin de mejorar la experiencia de usuario, tales como modificar distintos aspectos de la aplicación web de cara a mejorar su usabilidad, o reducir la cantidad de texto dedicado a instrucciones dentro de los juegos. Asimismo, se abrió la posibilidad de repetir los juegos completados, con el fin de recoger muestras adicionales de posible interés para análisis futuros.
2. Una vez incorporados estos cambios, se inició una fase de distribución general de la web, con el fin de difundir la plataforma entre el mayor número posible de usuarios y participantes.

Una vez finalizado el periodo de tiempo originalmente concebido para la recolección de datos, se procedió a dar comienzo a la fase de análisis sobre las trazas de eventos registradas.

5.3.2. Resultados obtenidos

En esta sección, presentamos los resultados obtenidos en cuanto al número de personas que participaron en los experimentos online. Se registraron un total de 101 usuarios, de los cuales 74 completaron al menos uno de los juegos produciendo un resultado válido. La tabla 5.1 resume los datos de interés. La primera columna (nº Jugadores) se corresponde con el número de participantes que llegaron a jugar a cada juego; la segunda indica el número de usuarios que completaron correctamente el juego en cuestión; la última, el número partidas válidas que se prestan a ser incluidas en un análisis posterior.

	nº Jugadores	nº Completados	nº Válidos
Blek	73	62	61
Edge	75	61	59
Unpossible	62	59	58
Los 3 juegos	70	53	53

Tabla 5.1: Resumen de datos registrados

5.4. Desarrollo de métricas

Habiendo registrado los eventos de los participantes, pasamos a la tercera fase del proceso de descubrimiento de conocimiento: el desarrollo de métricas. Como ya se ha comentado en repetidas ocasiones, estas métricas adicionales tienen el objetivo de introducir una mejora sobre el estudio de referencia, la cual se detallará en el siguiente capítulo. En cuanto a las métricas tenidas en cuenta en el experimento de partida, podemos encontrar en esencia dos: el número de niveles completados en el caso de Blek y de Edge, y el número de muertes o intentos para Unpossible. Estas métricas fueron mantenidas en nuestro caso, y calculadas a partir de los eventos registrados: el número de niveles completados se obtiene como suma del número de eventos `LEVEL_END` previos al evento `EXPERIMENT_END`, mientras que el número de muertes en el caso de Unpossible se deriva de la suma de eventos `PLAYER_DEATH` durante el tiempo de sesión.

Por otra parte, proponemos el siguiente conjunto de métricas adicionales. Cabe mencionar que, al igual que las dos métricas originales, todas las métricas aquí listadas pueden ser inferidas exclusivamente a partir de los eventos enviados al servidor por parte de cada juego.

- **Blek**:
  - **Tiempo por nivel**. Se obtiene como diferencia entre los tiempos de los eventos `LEVEL_START` y `LEVEL_END` para cada uno de los niveles completados por un jugador, y representa el tiempo total empleado por este para finalizar un nivel dado.
  - **Tiempo de reflexión por nivel**. Se obtiene como diferencia de tiempos entre el evento `LEVEL_START` y el siguiente evento que tenga lugar después de este. Representa el tiempo que un jugador emplea en analizar un nivel de manera estática antes de comenzar a interactuar de forma directa con él.
  - **Total de curvas o trazos por nivel**. Se obtiene como suma de todos los eventos de tipo `BEGIN_DRAWING` lanzados por el sistema en el contexto de un nivel dado. Representa el número de curvas o trazas realizadas por un jugador antes de completar un nivel.

A partir de las anteriores, se pueden inferir métricas agregadas como el **tiempo medio por nivel**, el **tiempo medio de reflexión por nivel** o **media de curvas trazadas por nivel** como las medias aritméticas de las métricas correspondientes.

- **Edge**:
  - **Tiempo por nivel**. Análogo a la métrica correspondiente en Blek.
  - **Muertes por nivel**. Se calcula como el número de eventos `PLAYER_DEATH` producidos hasta el evento `LEVEL_END` del nivel correspondiente. Representa el número de veces que ha muerto el jugador en el nivel asociado.
  - **Prismas recogidos por nivel**. De forma análoga al caso anterior, se calcula como el número de eventos `GOT_ITEM` producidos hasta el evento `LEVEL_END`.
  - **Movimientos realizados por nivel**. Se toma el valor del parámetro correspondiente registrado en el evento `LEVEL_END` del nivel actual.
  - **Checkpoints alcanzados por nivel**: Se calcula como suma de los eventos `GOT_CHECKPOINT` antes del evento `LEVEL_END`. Este valor es posteriormente dividido por el número total de checkpoints del nivel actual, para representar el porcentaje completado del nivel.

  Las métricas agregadas que pueden resultar de interés en este juego son, por ejemplo, el **número medio de muertes por nivel** y el **último checkpoint alcanzado**.

- **Unpossible**:
  - **Tiempo aguantado por intento**: Se calcula como la diferencia entre los tiempos de eventos consecutivos `PLAYER_DEATH`, salvo la primera y la última muerte. En el primer caso, se trata de la diferencia entre los tiempos del primer evento `PLAYER_DEATH` y del evento `EXPERIMENT_START`. En el segundo, entre el evento `EXPERIMENT_END` y el último `PLAYER_DEATH`. Hay que tener en cuenta que el juego incorpora un contador de cuenta atrás de tres segundos después de una muerte y antes de iniciar el siguiente intento, para que el jugador tenga un tiempo de "descanso", simulando el intervalo transcurrido hasta que vuelve a "dar a empezar". Esta cantidad debería ser restada en el cálculo correspondiente.
  - Por otra parte, las siguientes cinco métricas: el **número de obstáculos superados por nivel**; la **cantidad de giro hacia la izquierda por intento**; la **cantidad de giro hacia la derecha por intento**; el **número de pulsaciones de la tecla flecha izquierda por intento**; y el **número de pulsaciones de la tecla flecha derecha por intento** se calculan a partir de los parámetros correspondientes del evento `PLAYER_DEATH` o `EXPERIMENT_END`. Cabe mencionar que la primera de ellas es directamente proporcional a la métrica anteriormente introducida (tiempo aguantado por nivel), por lo que se omitirá en el análisis posterior.
  - Las métricas anteriores se han desarrollado también por unidad de minutos, en vez de por cada muerte del jugador. En principio, estos valores son solo reportados en los eventos `PLAYER_DEATH` producidos durante la sesión, y no por medio de un muestreo minuto a minuto. Por lo tanto, para el cálculo suponemos un comportamiento más o menos lineal del jugador en referencia a estos valores, y se realiza una aproximación agregada en cada intervalo.

  Respecto a las métricas agregadas de interés, en este caso no se tratan de las medias, sino los máximos, como el **Tiempo máximo aguantado en un intento**.

Resumen

- Inicialmente se redactó un primer protocolo, en la línea del de los experimentos del estudio de referencia, para la administración de los videojuegos escogidos y un conjunto estandarizado de tests de inteligencia.
- Las condiciones particulares del año llevaron a realizar una revisión de las primeras fases del proceso de knowledge discovery, destinada a tratar de encontrar alternativas para la adquisición de datos. Tras considerar las implicaciones de distintas estrategias, se optó por migrar los juegos a una plataforma web para su distribución en experimentos remotos.
- Esto supuso una reestructuración de la infraestructura previa del sistema de telemetría, con cambios correspondientes tanto en la instrumentalización de los juegos como en la estructura del servidor. Adicionalmente, se desarrolló un frontend para albergar los títulos con un sistema básico de gestión de usuarios.
- El experimento online se realizó en dos fases: una prueba piloto con un número reducido de participantes y una etapa de difusión pública. En un periodo de tres semanas, se registraron un total de 101 usuarios, si bien no todos ellos llegaron a completar todos los juegos.
- Una vez conducido el experimento online, se elaboró un conjunto de métricas a inferir a partir de las trazas de eventos de cada jugador, de cara a su uso posterior en la fase de análisis.

Dando por completadas las fases de adquisición de datos y desarrollo de métricas, en el siguiente capítulo pasaremos a desarrollar una de las etapas más interesantes del proceso: el análisis y evaluación de resultados.

Capítulo 6

Análisis de datos

Recordemos que el objetivo principal de nuestro trabajo es estudiar la posibilidad de introducir una mejora en el estudio de referencia [8] por medio de la reimplementación de un subconjunto de juegos empleados. Esto es, poder llegar a la correlación entre el rendimiento en los videojuegos y la inteligencia general, considerando toda la información adicional registrada. En este capítulo nos centraremos en desarrollar la quinta fase del proceso de knowledge discovery, correspondiente al análisis y evaluación de los datos obtenidos, de cara a analizar esta viabilidad. Como primer paso, realizamos un procedimiento estadístico sobre las métricas propuestas en el capítulo anterior, en busca de su relación con el factor inteligencia. En segunda instancia, proponemos un problema de predicción de la puntuación final del jugador, utilizando datos de los primeros minutos de las sesiones completas, para posiblemente acortar la duración en un futuro experimento. Cabe destacar que durante el desarrollo de la memoria, omitiremos la explicación en detalle de muchos conceptos técnicos, ya que quedan fuera del ámbito de este trabajo.

6.1. Análisis estadístico. Medida de la inteligencia

Cuando asumimos que la inteligencia es lo que subyace en los resultados obtenidos por un individuo en algún tipo de prueba, ya sea este un test de inteligencia o la puntuación obtenida en un videojuego, necesitamos utilizar una técnica estadística llamada **Análisis Factorial**. Un análisis de este tipo supone la existencia de un factor latente oculto (en este caso, la inteligencia) que no se puede observar directamente, pero sin embargo sí que se ve reflejado en el conjunto de variables registradas, explicando la variación y covariación entre ellas. Esta idea fue propuesta por Spearman a principios del siglo XX [18], y constituye un hito importante tanto en el campo de la psicología como en el de estadística. Fue él quien, al mismo tiempo, propuso la existencia de un factor general de inteligencia (también conocido como factor g) [18], que sintetiza las correlaciones entre diferentes tareas cognitivas. Como se ha explicado en el capítulo 1, este factor latente es la mejor estimación de la capacidad intelectual general, siempre que se haya obtenido a partir de un número representativo de tareas mentales. En la siguiente subsección, se presenta una breve explicación sobre este tipo de análisis y sus dos variantes principales: CFA y EFA. A continuacion, se detalla el razonamiento seguido para seleccionar el tipo más adecuado, y cómo se aplicó el proceso de análisis para nuestro enfoque.

6.1.1. Teoría del análisis factorial

No entraremos en detalles sobre el modelo estadístico o las bases matemáticas del análisis factorial, pero la idea básica es que cada variable observada (o indicador) puede verse como una función lineal de uno o más factores comunes y un término de error. La varianza de este indicador se divide en dos partes: (1) una **varianza común**, o varianza explicada por el factor, que se estima en base a la varianza compartida con otros indicadores en el análisis, y (2) una **varianza única**, que es la combinación de una varianza específica al indicador y una varianza de error de medición. El análisis factorial busca analizar la varianza común a todas las variables y encontrar los coeficientes de cada ecuación lineal. Los dos tipos de análisis más populares son el **Análisis Factorial Exploratorio** ("Exploratory Factor Analysis" o EFA) [54, 55] y el **Análisis Factorial Confirmatorio** ("Confirmatory Factor Analysis" o CFA) [56]. Ambos buscan reproducir las relaciones observadas entre el conjunto de variables con los factores latentes, pero se tratan de análisis distintos tanto conceptualmente como estadísticamente.

EFA

Generalmente, se utiliza un EFA en la etapa temprana del desarrollo de un análisis factorial, mientras que el CFA se emplea habitualmente en fases posteriores después de que la estructura subyacente se haya establecido a través de un análisis exploratorio. En este primer análisis, se identifican las interrelaciones entre un número relativamente grande de variables, en busca de reducirlas en unos factores que expliquen la mayor varianza. Como indica su nombre, el objetivo principal es explorar, porque no se establece ningún esquema hipotético a priori en las relaciones entre las variables observadas y los factores latentes. Se establecen relaciones entre todos los indicadores y todos los factores, y los pesos (o cargas) obtenidos como resultado en el modelo se utilizan para interpretar los factores latentes encontrados.

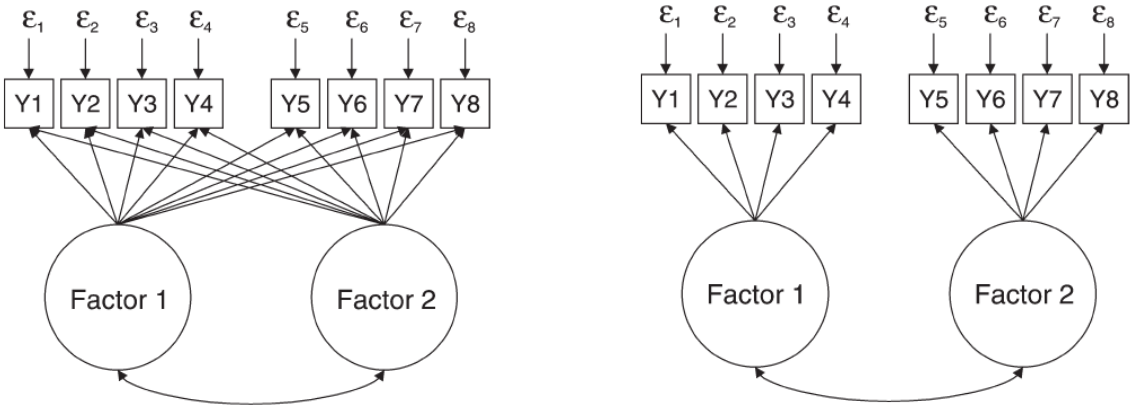
El proceso a seguir suele incluir las siguientes etapas: (1) Elegir el método para extraer los factores latentes. (2) Determinar el número de factores latentes. (3) Aplicar una técnica de rotación en el caso de desear interpretar los factores resultantes. (4) Elegir un método de computar puntuaciones factoriales para un análisis posterior posible.

CFA

En contraste, el CFA es una herramienta que se usa para confirmar o rechazar una hipótesis previamente establecida sobre un modelo factorial, donde todos los aspectos están previamente especificados: desde el número de factores, hasta el patrón de las cargas en los distintos factores. Se comprueba el modelo planteado sobre los datos de entrada y se pueden obtener índices de ajuste para evaluar la consistencia del modelo. Aunque no se identifican pasos claros a seguir como en el caso anterior, podemos tratar el procedimiento general de la siguiente manera: (1) Definición de las construcciones individuales. (2) Desarrollo de la teoría sobre el modelo. (3) Especificar el modelo de medición. (4) Evaluación de la validez del modelo.

Insistimos de nuevo que la fundamentación teórica de EFA y CFA queda fuera del alcance de este trabajo. Pero para entender mejor la diferencia entre estas dos técnicas, incluimos una breve comparativa.

En la figura 6.1, se presentan los diagramas que corresponden con dos modelos ejemplo, donde se consideran ocho indicadores observables, de los cuales se extraen dos factores latentes. Las variables se representan por cuadrados cuyas etiquetas empiezan por la letra Y, mientras que los factores se representan por círculos. Los diferentes valores ϵ indican los errores residuales de cada variable. Por otra parte, las flechas unidireccionales representan una relación directa entre indicadores y factores, mientras



(a) Representación gráfica de un modelo ilustrativo de EFA

(b) Representación gráfica de un modelo ilustrativo de CFA

Figura 6.1: Comparativa entre EFA y CFA

que las flechas bidireccionales representan varianzas o covarianzas.

Por lo tanto, siguiendo lo que hemos comentado anteriormente, todos los indicadores en EFA se cargan libremente en todos los factores, pues interesa buscar las relaciones entre ellas e interpretar los factores resultantes. En cambio, las relaciones entre indicadores y factores son previamente establecidos en un modelo CFA según una hipótesis. Si bien las cargas primarias y las correlaciones de factores se estiman libremente, no se especifican otras relaciones adicionales.

6.1.2. Aplicación del proceso de análisis

Pasamos a explicar las consideraciones necesarias a tener en cuenta antes de aplicar el proceso de análisis. En este enfoque, solo podemos utilizar los datos de aquellos participantes que han completado los tres juegos durante el experimento, por lo que disponemos de un $N = 53$.

El CFA es la técnica usada originalmente en el estudio de partida para confirmar la correlación entre la inteligencia general y los videojuegos. En nuestro caso, sin embargo, no se trata del modelo idóneo a aplicar por las siguientes razones. En primer lugar, disponemos ahora de un total de 340 variables registradas contabilizadas como se ha explicado en 5.4 (sin incluir las variables agregadas). La relación entre ellas y el factor de la inteligencia desconocemos a priori, por lo que necesitamos elegir las más apropiadas para evaluar. Adicionalmente, un análisis de tipo CFA requiere un conjunto de datos mayor del que disponemos, para conseguir resultados fiables. Como alternativa, elegimos el análisis exploratorio EFA, cuyo enfoque es más adecuado para ser aplicado en el conjunto de datos en cuestión.

Pero de forma previa a aplicar este método directamente, necesitamos hacer las dos observaciones importantes:

- Si bien el método es más estable que el CFA, requiere un mínimo de 50 casos de observación, preferiblemente mayor que 100 [54].
- Por cada variable que se considera en el análisis, se sugiere unos 10-15 casos de observación. Si las comunalidades (las proporciones de la varianza explicada por los factores comunes en una variable) conseguidas son bajas y los factores no bien determinados, se recomienda incluso una proporción mayor de al menos 20 casos [55].

Esto nos advierte que, aunque en principio es viable aplicar EFA en nuestro conjunto de datos, no debemos olvidarnos de la posibilidad de obtener resultados no deseados. Además, únicamente se nos permite aplicar EFA a como mucho 6 variables observadas, las cuales debemos elegir entre todos los indicadores de manera sensata. Para empezar, es razonable incluir las tres variables consideradas en el estudio original en nuestro subconjunto (número total de muertes en Unpossible y número de niveles alcanzados en Blek y en Edge), ya que se ha demostrado un valor de correlación bastante prometedor con el factor g.

Antes de pasar a seleccionar los tres indicadores restantes, debemos asegurarnos de que las variables adicionales registradas de cada juego son en efecto medidas fiables para los factores inobservables. Es decir, es necesario que sus valores en todos los niveles (o intentos en caso de Unpossible) sean estables y consistentes, con una alta correlación entre ellos mismos. Como el siguiente paso, entre los indicadores más fiables, proponemos seleccionar el que mejor prediga las variables resumen de cada juego a incluir en el subconjunto final. En este momento, podemos aplicar las etapas del análisis EFA sobre dicho subconjunto, omitiendo el último en nuestro caso. A continuación, ofrecemos una explicación más detallada para cada fase descrita.

Test de fiabilidad

Una medida muy popular para cuantificar el nivel de fiabilidad de una variable es la **Alfa de Cronbach**. Este indicador mide la consistencia interna de los valores que forman una variable, es decir la covariación entre sí. Es una medida que toma valores entre 0 y 1. Con un valor próximo a 1, afirmamos tener evidencia de la fiablidad de dicha variable. Diferentes fuentes han publicado diversos valores admisibles de alfa, desde 0.70 a 0.95 [57, 58]. En las tablas 6.1, 6.2 y 6.3, encontramos un resumen de prueba de fiabilidad para los tres juegos. Esta prueba es realizada para todas las métricas desarrolladas anteriormente en 5.4. Las variables sombreadas en las tablas son las que se deben considerar para aplicar el análisis EFA.

Métricas propuestas	α Cronbach
Tiempo de reflexión por nivel	0.911
Tiempo de superar el nivel	0.233
Número de curvas por nivel	-0.004

Tabla 6.1: Test de fiabilidad para Blek

Métricas propuestas	α Cronbach
Tiempo de superar el nivel	0.409
Número de muertes por nivel	-0.112
Cantidad de movimiento por nivel	0.582
Número de prismas recogidos el nivel	0.833
Checkpoints alcanzados por nivel	0.830

Tabla 6.2: Test de fiabilidad para Edge

Métricas propuestas	α Cronbach
Tiempo aguantado por intento	-20.288
Giros izda. por intento	-0.062
Giros dcha. por intento	-0.109
Teclas izda. pulsadas por intento	-0.857
Teclas dcha. pulsadas por intento	-1.264
Giros izda. por minuto	0.932
Giros dcha. por minuto	0.830
Teclas izda. pulsadas por intento	0.788
Teclas dcha. pulsadas por intento	0.823

Tabla 6.3: Test de fiabilidad para Unpossible

Regresión lineal por pasos

Para proceder con la elección de una variable concreta por juego cuya relación con el indicador resumen correspondiente sea más estrecha, empleamos una regresión lineal por pasos. En cada paso, el modelo considera una variable del conjunto de predictores elegidos para añadir o eliminar en el análisis. Usamos como criterio de evaluación R^2 ajustado, que en resumen indica la correlación entre los valores predichos por el modelo y los valores reales de la variable a predecir. Cuanto más alto es este valor, más ajustado es el modelo de regresión.

Lo último que se debe comprobar es la colinealidad entre las variables, ya que es un problema muy grave en problemas de regresión, especialmente si la muestra es pequeña (como es nuestro caso). Predictores colineados con la variable a predecir tienen un intervalo de confianza muy amplio y no son significativos para la predicción. Cuando dos o más variables supuestamente independientes están en efecto correlacionadas, el **índice de condición** será mayor que 1. Las variables con un valor de 1 son independientes, con un valor mayor que 15 presentan un problema y si supera 30 son muy cuestionables y deberían ser descartadas del análisis.

De esta manera, las variables seleccionadas a partir del modelo de regresión para cada juego son: el tiempo de reflexión antes de pintar el primer trazo en el nivel 12 de Blek, el porcentaje de checkpoints alcanzados en el nivel 7 de Edge y el máximo tiempo aguantado en un intento durante el experimento en Unpossible. La tabla 6.4 muestra los estadísticos más importantes que justifican la elección de dichas variables.

	Variable dependiente	Variable predictor	p-valor	R^2 ajustado	Índice de condición
Blek	nº niveles	Tiempo de pensar en el nivel 12	0	0.579	2.207
Edge	nº niveles	Checkpoints alcanzados en el nivel 7	0	0.767	3.395
Unpossible	nº muertes	Máximo tiempo aguantado durante el experimento	0	0.536	5.783

Tabla 6.4: Resumen regresión lineal por pasos

Extracción de factores

A la hora de extraer los factores en EFA, hay muchos métodos disponibles a elección, como por ejemplo el Análisis de Componentes Principales (PCA), el método de Máxima Verosimilitud (ML), la Factorización de Ejes Principales (PAF) o la Factorización Alfa. Los examinamos brevemente para selecionar el más adecuado.

El método PCA supone que no existe una varianza única en cada indicador, lo que conduce a que la varianza total es igual a la varianza común. Por este carácter, aunque provee ventajas frente a otros métodos como el tener menor probabilidad de extraer soluciones indebidas o ser capaz de calcular una puntuación para cada participante en un componente principal, no es la técnica más apropiada para nuestro análisis.

Entre los demás métodos, los más usados son ML y PAF. Una ventaja principal de ML es que permite evaluar estadísticamente la aptitud de una solución factorial con un amplio rango de índices de bondad de ajuste (*goodness-of-fit*). Por lo tanto, es también la técnica empleada más habitualmente en un análisis CFA, pues ayuda a confirmar el modelo propuesto. Para aplicar este método, se requiere que las variables tengan una distribución normal multivariada. Si no fuese así, los resultados estimados pueden resultar no fiables. Ocasionalmente tiende a producir soluciones incorrectas o resultados fuera de rango como un indicador con una comunalidad mayor que 1.

Por otro lado, PFA no requiere esta suposición sobre la distribución de los datos, y es más robusto a soluciones inválidas. La desventaja es que no proporciona índices de bondad de ajuste para determinar la aptitud del modelo factorial. Por esta razón, si los supuestos necesarios se cumplen para ML, este último se trata de una opción más favorable a considerar.

Ejecutamos pruebas de normalidad como la de Shapiro-Wilk (se ha concluido ser el mejor índice a este propósito [59]) y gráficos de Q-Q normal sobre las variables elegidas. Si bien no todas presentan una distribución normal, es muy cercana a ella. Confiamos en que con una muestra mayor se observaría la normalidad. Como conclusión, podemos empezar con el método de Máxima Verosimilitud.

Selección de factores

Este paso es probablemente el más importante en un EFA, ya que tanto una sobrefactorización o un número insuficiente de factores pueden invalidar el modelo y sus estimaciones. Aunque en principio se exploran las diferentes posibilidades, también podríamos seguir una aproximación de carácter confirmatorio [54], estableciendo un número de factores determinado a priori, para replicar los resultados de la investigación previa (suponer un único factor que indica la inteligencia general).

No hay una regla que prediga el mejor número de factores a extraer, y muchas veces esta elección se delega al analista de datos. Sin embargo, se han establecido numerosos criterios sirven de ayuda a la hora de determinarlos. Normalmente, el proceso se guía por los autovalores o valores propios de las matrices de correlación (tanto sin reducir como reducido). Hay tres reglas comúnes que se basan en ello: la regla de Kaiser-Guttman, el gráfico de sedimentación (*scree plot*) y el análisis paralelo. No obstante, nos centraremos en el método ML y basaremos nuestra elección en los índices de bondad de ajuste y la inferencia estadística que proporciona.

Según [56], para aplicar el método ML, el número de elementos en la matriz de correlación o de covarianza de los datos de entrada debe ser mayor o igual que el número de parámetros que se van a

estimar en el modelo factorial. Estos valores están dados por las siguientes fórmulas:

$$a = p * (p + 1)/2 \tag{6.1}$$
$$b = p * m + m * (m + 1)/2 + p - m^2 \tag{6.2}$$

donde a es el número de elementos en la matriz de correlación, b el número de parámetros en el modelo factorial, p denota el número de indicadores y m corresponde al número de factores latentes a considerar. Por la observación realizada anteriormente, solo podemos considerar 6 variables registradas, es decir ($p = 6$). En consecuencia, solo se cumple la condición $a \geq b$ cuando $m = 1$ o $m = 2$. Es decir, solo podremos establecer un modelo unifactorial o bifactorial. Si elegimos un factor, su interpretación podría ser el factor g o la inteligencia que subyace a la realización de los videojuegos. Si elegimos dos factores, tenemos que darle significación a las cargas factoriales e interpretar los factores resultantes. Estableceremos los dos modelos y compararemos sus resultados.

Rotación de factores

Cuando se extrae más de un factor, se rotan los factores resultantes para facilitar su interpretación. La razón es que para cualquier modelo multifactorial, hay un número inmenso de modelos equitativamente válidos y adecuados, en los que se diferencian las cargas factoriales. El objetivo de este paso es aplicar una transformación matemática que mantiene el χ^2 del modelo mientras que mejora la interpretabilidad mediante la llamada estructura simple. En dicha estructura, se intenta que cada variable solo tenga carga alta en un factor y para cada factor siempre haya variables que tengan una carga alta.

Se puede diferenciar dos tipos de rotaciones: las ortogonales y las oblicuas. En el primer tipo (al cual pertenecen rotaciones tipo varimax, quartimax y equimax), se presupone que los factores son independientes y no tienen correlación entre sí. En el segundo (rotaciones tipo oblimin direct, promax, orthoblique) se producen factores correlacionados. Para más detalle sobre las rotaciones consúltese [54, 55].

Según los estudios anteriores, las rotaciones oblicuas son capaces de proporcionar soluciones más precisas y reproducibles. Además, en un tipo de análisis psicológico sobre la inteligencia, hay suficiente evidencia sobre la correlación entre los factores subyacentes, por lo que elegimos la rotación oblicua "oblimin" en el modelo bifactorial.

Resultados

Antes de presentar finalmente los resultados obtenidos, necesitamos conocer algunos métodos de evaluación. Para comparar los modelos EFA, utilizamos la medida Kaiser-Meyer-Olkin (KMO) de adecuación de muestreo y la prueba de bondad de ajuste chi-cuadrado (χ^2). El índice KMO puede ser interpretado siguiendo distintos criterios. Los más utilizados son: valores menores de 0,5 se consideran inaceptables; entre 0,5 y 0,59 se consideran pobres; entre 0,6 y 0,79, regulares; y entre 0,8 y 1 muy satisfactorios. Por otra parte, la hipótesis nula en la prueba χ^2 es que no hay diferencia entre el modelo diseñado y los datos observados, por lo que en vez de rechazarla, buscamos que sea estadísticamente no significativo en este caso ($p > 0{,}01$).

Realicemos una comparación entre los modelos obtenidos antes de pasar a dotar de significado a los factores. Vemos en la tabla 6.5 que, en el modelo unifactorial, aunque el índice KMO es aceptable, obtenemos $\chi^2 = 51{,}555$ y $p = 0$. Esto significa que ha rechazado la hipótesis nula, lo que sugiere que no

	KMO	Varianza total explicada	χ^2	p-valor
Modelo unifactorial	0.765	55.410 %	51.555	0
Modelo bifactorial	0.765	72.427 %	11.379	0.023

Tabla 6.5: Comparación de modelos

	Factor 1	Factor 2
Unp_Exp_maxTime	0.745	0.335
Unp_Exp_Deaths	-0.703	-0.296
Edge_Exp_Checkpoints7	0.891	0.500
Edge_Exp_Levels	0.938	0.429
Blek_Exp_Think_12	0.308	0.819
Blek_Exp_Levels	0.633	0.941

Tabla 6.6: Matriz de estructura

	Factor 1	Factor 2
Factor 1	1.000	0.463
Factor 2	0.463	1.000

Tabla 6.7: Matriz de correlaciones factorial

es el modelo más adecuado y probablemente es necesario considerar más factores. Además, el porcentaje de la varianza total explicada es otro aspecto muy importante a tener en cuenta para determinar si es necesario continuar extrayendo factores. No existe un criterio fijo, pero como umbral se suele establecer un mínimo de 60 % [54].

En el modelo bifactorial, conseguimos $\chi^2 = 11{,}379$ y $p = 0{,}023$. Es estadísticamente no significativo ya que $p > 0{,}01$, permitiéndonos concluir que ofrece un ajuste razonable a los datos de muestreo. La varianza total explicada también ha aumentado a un valor satisfactorio, superando el umbral establecido.

Pasamos ahora a interpretar los dos factores encontrados fijándonos en la matriz de estructura y la matriz de correlación entre los factores. El primer factor, al tener una carga tan alta en todas las variables excepto una (figura **??**), lo podemos identificar como el factor inteligencia o capacidad cognitiva general que subyace en los videojuegos. El peso factorial del tiempo de reflexión en el nivel 12 de Blek en dicho factor es $< 0{,}60$ (umbral considerado para mantener una potencia 80 %, dado el N tan bajo en nuestro caso [54]), por lo que ha de ser ignorado en su interpretación. El segundo factor se relaciona altamente con el juego Blek, medianamente con Edge y levemente con Unpossible, por lo que podría representar el factor de razonamiento lógico-espacial, excluyendo el componente de velocidad que representa Unpossible. Los dos factores comparten una correlación de 0.463, como cabe esperar, ya que el primero explica el segundo.

Mostramos en la siguiente figura el resumen del modelo bifactorial.

6.2. Análisis predictivo. Problema de regresión

A la par del análisis anterior, percibimos el problema de que no todos los participantes han completado la sesión de los tres juegos. Una razón común reportada por los jugadores es que la duración del experimento es demasiado extensa y, al ser el plataforma on-line, no podemos garantizar que ellos terminen los juegos debidamente. A raíz de este fenómeno, surge la pregunta de si sería posible realizar una estimación de la inteligencia de los jugadores a través de experimentos más cortos, ahora que disponemos más información que en el experimento original. Lo que se plantea en esta sección es predecir, mediante algoritmos de aprendizaje automático, la evolución de las partidas y resultados de distintos usuarios, fijándonos únicamente en un intervalo limitado de tiempo. En efecto, si considerando un experimento

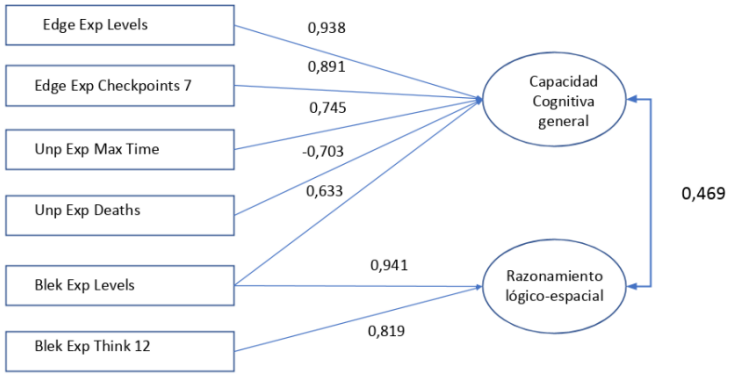


Figura 6.2: Modelo bifactorial

"truncado" somos capaces de dar una estimación robusta del progreso final de un jugador, resultaría lógico pensar que es posible reducir la duración de las sesiones en el experimento. Además, este nuevo enfoque del análisis nos abre la posibilidad de reutilizar datos de participantes con juegos parcialmente completados, por lo que el número de datos útiles se incrementa ligeramente con respecto al caso anterior.

A continuación presentamos las técnicas tanto del aprendizaje profundo como del aprendizaje automático tradicional utilizadas en el análisis, así como los resultados obtenidos.

6.2.1. Deep Learning

Empezamos haciendo una pequeña introducción al *deep learning* o aprendizaje profundo, que se trata de un subcampo del famoso *aprendizaje automático*. Para entender lo que le caracteriza o diferencia de otros algoritmos de aprendizaje automático, veamos primero en qué consiste un algoritmo de este campo general. En el paradigma de programación clásica, se predefine una regla para encontrar las respuestas que corresponden a los datos de entrada. En contraste, un sistema de aprendizaje automático tiene el objetivo de encontrar las reglas que producen las salidas esperadas según los datos de entrada, de acuerdo con [60]. De esta manera, necesitamos: (1) el conjunto de datos de entrada, que en una tarea de aprendizaje automático suele ser grande y complejo, (2) las salidas esperadas correspondientes, y (3) una manera de evaluar la regla o el algoritmo encontrado, que suele ser la distancia entre la salida producida y la salida esperada. Esta última medida es usada como retroalimentación para ajustar el algoritmo encontrado, el proceso que da nombre a "aprendizaje". Durante el entrenamiento de un modelo, lo que se busca es una transformación o *representación* útil de los datos, para facilitar el proceso de encontrar una relación con las salidas.

El aprendizaje profundo no es más que "aprender" esta representación de datos mediante una cascada de *capas*, de ahí el nombre de "profundo", que hace referencia a la idea de tener numerosas capas apiladas. Estas representaciones en capas se aprenden casi siempre a través de modelos conocidos como *redes neuronales*.

En resumidas cuentas, el objetivo de dichas capas es transformar los datos de entrada en una representación más abstracta, hasta aproximarse eventualmente a las salidas. Esta transformación está parametrizada por unos *pesos* caracterizados de las capas y el proceso de aprendizaje consiste en encontrar los valores adecuados para cada peso. Conforme al esquema general de aprendizaje automático visto anteriormente, el proceso de aprendizaje es evaluado por una *función de pérdida o función objetivo*, y

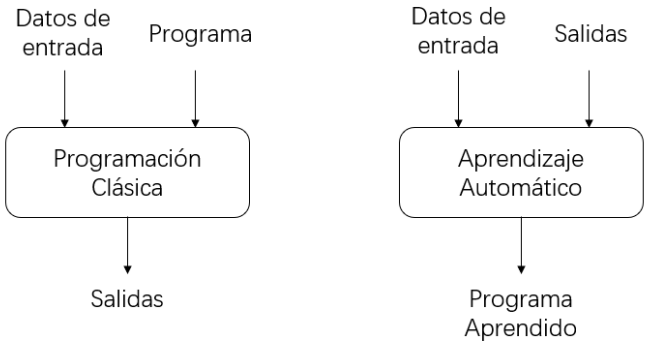


Figura 6.3: Aprendizaje automático

la puntuación obtenida se retroalimenta en el sistema siguiendo un algoritmo llamado *retroprogamación*. La siguiente figura de [60] ilustra con claridad el modelo de una red neuronal.

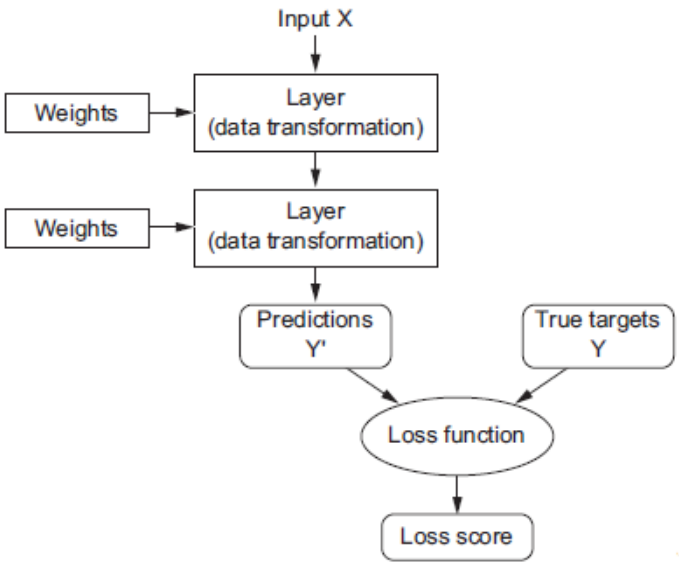


Figura 6.4: Modelo red neuronal

Para dar una idea de lo potente que es esta herramienta, podemos referirnos al teorema de aproximación universal [61, 62], que indica que una red neuronal simple con una capa oculta ya es capaz de aproximar cualquier función real continua.

En el resto de esta sección, hablaremos sobre la arquitectura y cada componente principal de las redes neuronales (capas, funciones objetivas, optimizadores, hiperparámetros, etc.), presentaremos cómo se ajustan diferentes modelos para optimizar su rendimiento en el contexto de cada juego, y expondremos los mejores resultados obtenidos. Utilizamos el lenguaje Python, de cara a emplear la librería *Keras* [63], uno de los marcos de aprendizaje profundo más populares para comenzar con el experimento.

Preprocesamiento de datos

Antes de entrar en detalle sobre cómo establecer modelos concretos para nuestro problema, necesitamos transformar los datos de lo que disponemos ahora. Según [60], siempre y cuando el tiempo tiene importancia en los datos, deberíamos almacenarlos en un tensor o array tridimensional con un eje temporal. Es decir, hay una dimensión que se encargaría de guardar los *timestamps* y los datos de observación serán una secuencia de vectores bidimensionales. Por convención, la dimensión del tiempo es siempre la segunda.

Siguiendo esta pauta, en cada juego, preparamos nuevos datos desde los eventos "crudos" de la base de datos, registrando variables cada cierto intervalo de tiempo. Como estos parámetros son dependientes del juego y experimentamos con diferentes versiones, explicaremos más en detalle cada caso concreto:

- **Blek**: El experimento de Blek dura 10 minutos. Probamos a recoger los datos de los primeros 5 y 6 minutos, con intervalos de 3 segundos. En cada timestamp, se registran: el nivel actual, una variable booleana que indica si el jugador está pensando o no (guardada como un valor de punto flotante), y el número de curvas intentadas hasta el momento en el nivel actual.
- **Edge**: En caso de Edge, la prueba se alarga hasta 12 minutos. Probamos a registrar los primeros 6 y 8 minutos de datos, con intervalos de 5 segundos, por el ritmo más lento que suceden los eventos (los checkpoints y prismas recogibles están más separados que una distancia recorrible en 5 segundos). Las variables registradas en cada timestamp son: el nivel actual, el porcentaje de checkpoints superados hasta el momento del nivel actual, el porcentaje de prismas recogidas hasta el momento en el nivel actual, y el número de muertes acumuladas hasta el momento en el nivel actual.
- **Unpossible**: De los 5 minutos que dura una sesión del experimento, probamos a recoger datos de los primeros 1.5, 2 y 2.5 minutos, con intervalos desde 1 hasta 5 segundos. En cada marca temporal, registramos las siguientes 5 variables: el número de muertes acumuladas hasta el momento, el número de veces que ha pulsado el jugador la flecha izquierda/derecha desde la última muerte hasta el momento, y la cantidad de giro en cada dirección desde la última muerte hasta el momento. Un aspecto especial de este juego que cabe mencionar es que estas últimas variables solo son reportadas al servidor cuando el jugador muere o se ha terminado el experimento, por lo que no disponemos de sus valores reales en cada timestamp. Para estimarlos, suponemos que tienen un comportamiento lineal, y calculamos la proporción correspondiente en cada momento.

De esta manera, las dimensiones de los datos se resumen en la tabla 6.8.

	(1ª dimensión) nº de observaciones	(2ª dimensión) nº de timestamps	(3ª dimensión) nº de variables
Blek	68	100	3
Edge	63	72	4
Unpossible	60	75	5

Tabla 6.8: Blek 5 minutos, timestamp de 3 segundos. Edge 6 minutos, timestamp de 5 segundos. Unpossible 2.5 minutos, timestamp de 2 segundos

En general, datos que pueden tomar valores muy grandes o datos muy heterogéneos pueden causar problemas de convergencia en el método del descenso del gradiente (el algoritmo utilizado en las redes

neuronales para minimizar la función de pérdida, para más detalle consultar [60]). Por lo tanto, después de convertir los datos en el formato vectorial más adecuado para el problema, necesitamos aplicar el último paso antes de procesarlos: la **normalización**. Este proceso consiste en ajustar los valores para que tengan una escala común, en ocasiones incluso intentar alinear distribuciones de ellos. En una tarea de aprendizaje automático, se suele exigir que todas las variables tengan una media de 0 y desviación estándar de 1. Esto se consigue restando a cada valor de las variables su media y dividiéndolo posteriormente por su desviación estándar. Tras esta transformación, podemos pasar a determinar la arquitectura concreta de la red neuronal.

Selección del tipo de capas

Para diferentes tipos de formatos de datos de entrada, o mejor dicho diferentes tipos de tareas de procesamiento, se ha comprobado que ciertas estructuras de capas funcionan mejor que otras. Por ejemplo, para vectores 2D de datos simples, de formato *(observaciones, variables registradas)*, se suelen utilizar capas densas. Para datos de secuencia, especialmente cuando el tiempo es involucrado, se han diseñado las redes neuronales recurrentes para abordar su procesamiento [64]. En la práctica, experimentaremos con las dos opciones y compararemos los resultados obtenidos. En todas las capas utilizamos la función de activación *ReLu*, la más popular de hoy en día [65].

Compilación del modelo

Una vez determinadas las capas a utilizar, podemos proceder a compilar el modelo. Sin embargo, todavía es necesario especificar varios parámetros, como es la función de pérdida, el optimizador y la métrica a usar. Tanto la función objetivo como la métrica son elecciones intrínsecas al problema. El primero da el valor que representa lo cerca que está nuestra predicción al valor real, y es lo que se busca minimizar en el proceso de entrenamiento. El segundo se utiliza para evaluar el rendimiento del modelo, y puede ser distinto a la función de pérdida. Para los problemas más populares como es la clasificación o regresión, se han establecido unos criterios para elegir la función de pérdida más adecuada. Según [64], en un problema de regresión como el nuestro, se suele emplear el error cuadrático medio ("mean squared error" o mse) como función objetivo, sin la necesidad de incluir una función de activación en la última capa. En cambio, se suele utilizar el error absoluto medio ("mean absolute error" o mae) como métrica.

Respecto al optimizador, se trata también de un hiperparámetro ajustable junto con su *tasa de aprendizaje*, de lo que hablaremos en las siguientes subsecciones. En un principio, utilizaremos *rmsprop* con la tasa de aprendizaje por defecto. Si en los modelos entrenados observamos patrones claros de una tasa de aprendizaje inapropiada, como por ejemplo cuando la curva de entrenamiento se estanca en una solución subóptima y no mejora, o cuando presenta un salto y vuelve a "olvidar" todo lo aprendido [64], ajustaremos su valor a uno más adecuado.

Validación del modelo

A la hora de evaluar un modelo establecido de aprendizaje automático, siempre tenemos que dividir el conjunto de datos disponibles en tres subconjuntos: los datos de entrenamiento, los datos de validación y los datos de prueba. Como sus nombres indican, se debería entrenar el modelo con los datos de entrenamiento, evaluarlo en los datos de validación y ponerlo a prueba con el conjunto de test. Disponer de ciertos datos de validación es crucial en una tarea de aprendizaje automático porque permite posibles ajustes del modelo para mejorar su rendimiento, pero al mismo tiempo evita el filtro de la información

de los datos de prueba. Cada vez que volvemos a evaluar el modelo después de ajustar su configuración, aprende cierta información sobre el conjunto de validación, lo que podría resultar rápidamente en un sobreaprendizaje en los datos de entrenamiento, un problema que discutiremos en la siguiente subsección. Por lo tanto, el conjunto de prueba debe estar compuesto de datos completamente nuevos y nunca vistos por el modelo.

La mayor dificultad es que disponemos de un muestreo de datos muy reducido. El conjunto de validación podría verse muy condicionado a los datos incluidos en él, produciendo una variación muy alta en la "puntuación" de validación obtenida. Esto puede generar un modelo con poca fiabilidad, muy dependiente de la división de datos. En situaciones como estas, es recomendable usar una **validación cruzada**. Es una técnica para garantizar que la partición de datos no influye en la evaluación del modelo.

Existen muchas técnicas de validación cruzada, las más adecuadas para usar en este caso es la de K **repeticiones** (K-fold cross-validation) o **iterada de K repeticiones con reordenamiento** (iterated K-fold with shuffling) [60]. La primera consiste en dividir los datos disponibles (sin considerar los de prueba) en K subconjuntos. Cada uno de ellos se utiliza como datos de validación una vez en un total de K iteraciones, mientras que los restantes $(K-1)$ subconjuntos se utilizan como datos de entrenamiento. De esta manera, obtenemos K modelos distintos, y la puntuación final de validación se calcula como la media aritmética de los resultados de los K modelos. Típicamente se elige $K = 4$ o $K = 5$. Por su parte, la técnica iterada de K repeticiones con reordenamiento no es más que repetir una validación cruzada de K repeticiones varias veces, con un reordenamiento aleatorio de los datos antes de proceder a dividirlos. En vez de K modelos distintos, se obtendrán un total de $P \times K$, siendo P el número de repeticiones.

En nuestro análisis, elegimos la validación cruzada de 4 repeticiones, sin iteraciones añadidas.

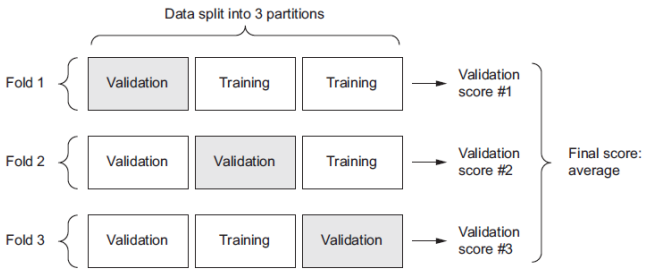


Figura 6.5: Validación cruzada de K repeticiones (imagen de [64])

Ajuste del modelo

Habiendo establecido el método de evaluación del modelo, podemos introducir técnicas de ajuste del modelo, como afinar los hiperparámetros o regularizar el modelo, para intentar conseguir el mejor resultado posible. Se trata de un proceso iterativo de modificar, entrenar y evaluar en los datos de validación, hasta alcanzar un modelo lo suficientemente satisfactorio.

Un aspecto muy importante a tener en cuenta a la hora de entrenar y evaluar un modelo es el *overfitting* o sobreajuste. Es un problema común que ocurre en todas las tareas de aprendizaje automático. Consiste en que el modelo "sobreentrena" y aprende características demasiado específicas a los datos de entrenamiento. Recordemos que el objetivo final de entrenar un modelo es poder predecir resultados sobre nuevos datos a partir de lo aprendido con los datos de observación, por lo que el poder de *generalización* es crucial. No debemos dejar que los ruidos o patrones específicos de los datos de entrenamiento repercutan negativamente en la predicción de nuevos datos. Lo ideal es poder encontrar el equilibrio

entre optimización y generalización, pero este último no es algo que podamos controlar activamente. Lo que sí está en nuestras manos y normalmente suele recomendarse es reducir el sobreajuste. Además de soluciones naturales como aumentar el conjunto de datos de entrenamiento, existe una serie de técnicas de *regularización* aplicables que presentaremos en un instante.

Siguiendo los consejos de [60, 64], consideramos los siguientes hiperparámetros ajustables:

- **Número de capas y neuronas por capa**: Estos dos parámetros constituyen los *parámetros aprendibles (learning parameters)* del modelo y determinan su capacidad. Los modelos con menor capacidad son más robustos al sobreajuste porque cuanto más capas o unidades por capa posea, más facilidad tiene para memorizar las características de los datos de entrenamiento. Sin embargo, tampoco debe por esta razón restringirse a valores muy reducidos. Una capacidad insuficiente puede causar *underfitting* o subajuste. En los tres juegos, probamos tanto redes recurrentes como redes prealimentadas de una capa oculta y dos capas ocultas, incrementando sus capacidades de 8 unidades a 64 unidades en potencias de dos.
- **Regularización**: La regularización es, como se ha comentado, una técnica para mejorar la capacidad de generalización del modelo y reducir el sobreajuste. La idea principal es añadir un cierto "coste" a la función objetivo del modelo proporcional a los pesos de las capas, para que estos tomen únicamente valores pequeños. Existen los regularizadores $L1$, $L2$ o híbridos. Se asocian a las normas $L1$ o $L2$ de los pesos. En modelos concretos, probamos diferentes regularizadores tipo $L2$, con valores entre 10^{-3} y 10^{2} para el coeficiente multiplicativo de los pesos, hasta no observar un sobreajuste en el modelo.
- **Dropout**: Se trata de otra técnica de regularización que consiste en poner al valor 0 aleatoriamente algunas salidas de las capas durante el entrenamiento. De esta manera, se introduce ruido en las capas para evitar posibilidades de que la red neuronal intente "memorizar" patrones de los conjuntos de entrenamiento y por tanto producir un sobreajuste. Se suele utilizar un ratio entre 0.2 y 0.5. Probamos modelos sin dropout y con diferentes ratios de dropout en dicho rango.
- **Capas bidireccionales**: Utilizar capas bidireccionales podría aumentar la precisión en algunos casos. Consiste en utilizar dos redes recurrentes normales, cada una procesando la secuencia de datos en una dirección, para al final combinar sus resultados. En principio, esta técnica es más adecuada a tareas de procesamiento de lenguaje natural, donde es interesante explotar el orden de la secuencia de datos (palabras). Teniendo esto en cuenta, nosotros lo probamos de todas maneras. En caso de no conseguir una mejora, reconfirmamos la idea de que no es apropiada para secuencias temporales.
- **Tasa de aprendizaje**: Este valor controla los "pasos" que se dan en el algoritmo del descenso del gradiente comentado anteriormente en el optimizador, y por tanto la velocidad a la que aprende el modelo. Una tasa demasiado grande hace que se salten los óptimos y por consiguiente se aumente el error de entrenamiento. En contraste, una tasa muy pequeña no solo ralentiza el proceso de aprendizaje, sino también puede hacer que el modelo se atasque en un óptimo local y no consiga mejorar más su rendimiento. Típicamente se prueba valores dentro del conjunto $\{0{,}1, 0{,}01, 10^{-3}, 10^{-4}, 10^{-5}\}$, de mayor a menor, hasta encontrar la tasa más adecuada al problema.

Resultados

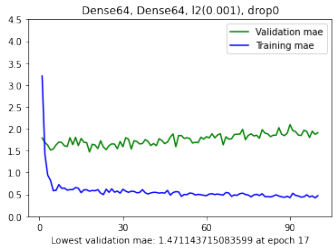
En primer lugar, incluimos en la tabla 6.9 la variable a predecir en cada juego y el rango de valores que pueden tomar en el conjunto de datos, con la finalidad de poder interpretar mejor la métrica empleada

(mae). En cada caso, tenemos en cuenta dos modelos base: uno de regresión lineal y otro modelo de simplemente predecir el resultado como la media de las salidas de entrenamiento. El objetivo es que los modelos entrenados de las redes neuronales obtengan un mejor rendimiento que estos resultados.

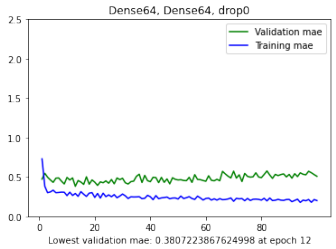
Con todos los modelos entrenados y ajustados, incluimos en las tablas 6.10 y 6.11 los mejores encontrados para cada juego, tanto de redes recurrentes como de redes prealimentadas. Las figuras 6.6 y 6.7 muestran las curvas de aprendizaje correspondientes a cada modelo.

	Variable a predecir	Rango	MAE Regresión lineal	MAE Predicción media
Blek	nº niveles	[1,26]	4.588	5.433
Edge	nº niveles	[1,8]	0.605	1.084
Unpossible	nº muertes	[3,25]	5.043	3.627

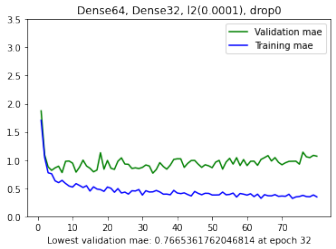
Tabla 6.9: Modelos base



(a) Blek: 2 capas densas de 64 unidades cada una, con regularizador $l2(0{,}001)$, sin dropout



(b) Edge: 2 capas densas de 64 unidades cada una, sin regularizador ni dropout



(c) Unpossible: 2 capas densas de 64 y 32 unidades cada una, con regularizador $l2(0{,}0001)$, sin dropout

Figura 6.6: Curvas de aprendizaje de los mejores modelos prealimentados

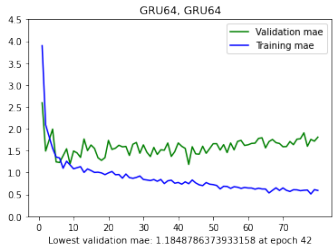
	Configuración	Epochs	MAE validación	MAE prueba
Blek	2 capas, 64 unidades/capa, reg l2(0.001)	17	1.47	2.99 - 4.86
Edge	2 capas, 64 unidades/capa, sin reg, sin dropout	12	0.38	0.66 - 1.14
Unpossible	2 capas, 64,32 unidades, reg l2(0.0001)	32	0.76	1.53 - 2.17

Tabla 6.10: Mejores modelos de red neuronal prealimentada

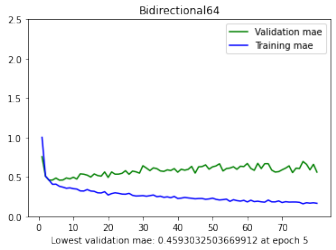
	Configuración	Epochs	MAE validación	MAE prueba
Blek	2 GRU, 64 unidades/capa, sin reg, sin dropout	42	1.18	2.61 - 4.60
Edge	2 GRU, 64,32 unidades, sin reg, sin dropout	5	0.44	0.90 - 1.06
Unpossible	2 GRU, 64 unidades/capa, learning rate(0.0001)	48	0.88	1.59 - 1.92

Tabla 6.11: Mejores modelos de red neuronal recurrente

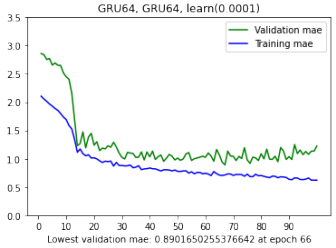
En una vista general, observamos que las redes neuronales han mejorado bastante los modelos de referencia de los que partimos, sobre todo en Blek y en Unpossible. En Edge no se aprecia en gran medida la reducción del error, por la simple razón de ser ya una escala pequeña con un rango más reducido. De hecho, en el conjunto de prueba llega a una predicción peor que el modelo de regresión lineal. Analizaremos este comportamiento más en profundidad.



(a) Blek: 2 capas GRU de 64 unidades cada una, sin regularizador ni dropout



(b) Edge: 2 capas GRU de 64,32 unidades cada una, sin regularizador ni dropout



(c) Unpossible: 2 capas GRU de 64 unidades cada una, sin regularizador ni dropout, con learning rate 0.0001

Figura 6.7: Curvas de aprendizaje de los mejores modelos recurrentes

En cuanto a las gráficas, todos los modelos encontrados tienen un comportamiento similar: la curva de aprendizaje del conjunto de entrenamiento desciende rápidamente, mientras que la de validación se mantiene estable después de pocos epochs, presentando un ligero sobreajuste. Añadir técnicas de regularización y Dropout han ayudado a reducir la brecha entre los resultados de entrenamiento y validación, pero a cambio de un rendimiento peor. Generalmente, cuando el sobreajuste es pequeño y el error en validación se estanca, indica que el modelo no tiene suficiente capacidad [60]. Sin embargo, probando redes con más capas y/o neuronas por capa no ha resultado en una mejora.

Es interesante hacer una comparativa entre modelos de redes prealimentadas y de redes recurrentes. En general, no se aprecia una ventaja clara del empleo del segundo, que teóricamente es más apto para datos en serie. Sí que reduce ligeramente el sobreajuste en el comienzo del entrenamiento, pero se debe más bien al avance más lento de aprendizaje en los datos de entrenamiento. Sólo se presenta una ligera mejora en el caso de Blek. Sin embargo, por la alta varianza obtenida en el conjunto de prueba, no podemos afirmar la dominación de redes recurrentes con certeza. Respecto a la introducción de capas bidireccionales, no ha ayudado en ningún caso a mejorar el rendimiento en una medida significativa. Esto está en línea a nuestra hipótesis previa, por tratarse de datos donde el peso del eje temporal es importante.

Cabe detenernos en el patrón que presenta Blek en los resultados de prueba, por tener una varianza mucho mayor que otros juegos. Aunque el rango de los valores a predecir es más grande, este problema no se aprecia en Unpossible, cuyo rango es similar. Esta varianza tan grande puede indicar que es un conjunto de datos bastante más complicado y heterogéneo que otros casos, o que las variables incluidas en el análisis no ayudan a la predicción. Se justifica con el primer análisis estadístico que realizamos, donde vimos que la variable de número de curvas no tiene casi correlación con el nivel alcanzado, sino el tiempo de reflexión. Y este último no es proporcionado directamente como dato de entrada, sino tiene que ser inferido de la variable booleana que introducimos como experimento.

Otro fenómeno general es que todos los modelos se comportan bastante peor en los datos de prueba, habiendo conseguido resultados bastante prometedores en validación. Esta diferencia es notablemente mayor en Blek, cuyo error alrededor de 1 en los datos de validación en los mejores modelos puede llegar a un valor de casi 5 en el conjunto de prueba. Este problema puede ser debido al número limitado de observaciones que tenemos. En este caso, la división aleatoria de los datos en tres subconjuntos puede condicionar altamente en los resultados obtenidos. Por ejemplo, en un conjunto de datos suficientemente grande, las divisiones siguen aproximadamente la misma distribución, presentando las mismas características. Al contrario, hay menor probabilidad de que cada subconjunto tenga la misma distribución, y

por tanto pueden contener patrones muy distintos que los modelos no son capaces de aprender solamente de los datos de entrenamiento. Los casos límites también pasan a tener mayor poder en un conjunto de datos pequeño e influye fuertemente en la puntuación final.

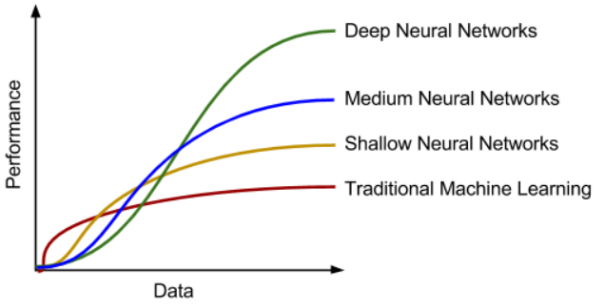


Figura 6.8: Relación entre el rendimiento del modelo y el tamaño del conjunto de datos, según [66]

Llegado a este punto, quedan pocas conclusiones que podemos extraer de este análisis. Tanto estudios empíricos como teóricos han demostrado el impacto de la insuficiencia de datos en redes neuronales [67, 68]. Aunque hay continua investigación sobre cómo aplicar eficientemente técnicas de aprendizaje profundo en un conjunto de datos reducido [69, 70, 71], afirman que algoritmos tradicionales de aprendizaje automático suelen tener un comportamiento considerablemente más prometedor, como indica la figura 6.8. Por lo tanto, en vez de indagar más en diferentes configuraciones de redes neuronales, pasamos a poner a prueba una técnica más simple, los *random forests*, con la esperanza de conseguir un mejor rendimiento en nuestros datos.

6.2.2. Random Forest

Random Forest (RF), también conocido como bosques aleatorios [72], es un algoritmo ampliamente usado tanto en problemas de clasificación como de regresión. A diferencia de las redes neuronales, pertenece al ámbito de *shallow learning*, es decir el conjunto de técnicas tradicionales de aprendizaje automático. Debido a su simpleza de entrenar y ajustar junto con su capacidad de obtener una precisión muy alta, se trata de un algoritmo muy popular de este campo. Brevemente explicando, consiste en un método de *ensamble* donde se entrenan varios árboles de decisión y se agregan sus resultados para obtener un rendimiento más alto. Desarrollaremos los conceptos de método de ensamble y árbol de decisión para darle una mejor comprensión a este modelo.

Árboles de decisión

Empezamos haciendo una breve introducción a los árboles de decisión [73]. Son modelos que tienen un diagrama en forma de árbol, utilizado para clasificar o predecir resultados de los datos de entrada. Debido a su simplicidad y su facilidad de visualización e interpretación, es un modelo bastante popular de aprendizaje automático.

Un árbol llega a una estimación realizando una serie de preguntas de verdadero y falso a los datos de entrada, donde cada decisión ayuda a reducir el número de resultados posibles correspondientes a dichos datos, hasta que el modelo obtenga suficiente seguridad de realizar una predicción. En cada nodo del árbol, los datos se dividen en dos grupos más pequeños siguiendo un criterio para elegir las propiedades más significativas, de tal manera que los grupos resultantes sean al mismo tiempo lo más homogéneos y

lo más alejado del otro grupo posible. Las hojas del árbol ya no representan una decisión, sino almacenan la categoría o el valor de predicción de los datos que llegan hasta ellas. Por lo tanto, se trata de un modelo muy explicable por sí, aunque esta explicabilidad se pierde en el bosque aleatorio al combinar numerosos árboles.

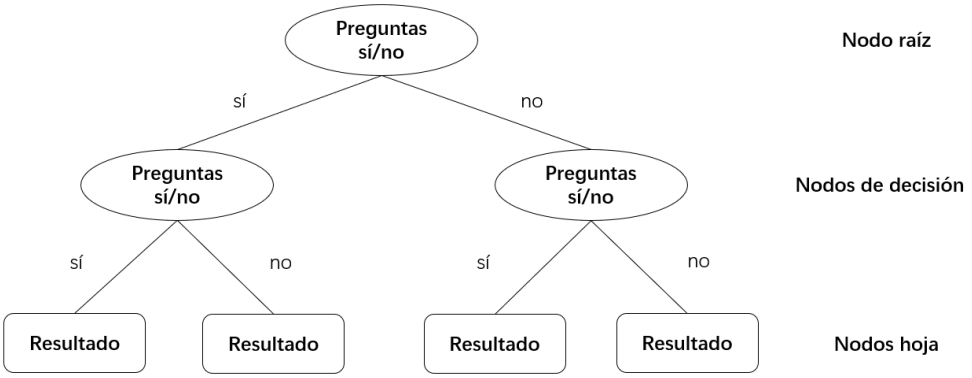


Figura 6.9: Árbol de decisión

Métodos combinados

Muchas veces, en vez de usar algoritmos individuales para enfrentar un problema de aprendizaje automático, se emplean métodos combinados o métodos de ensamble para obtener un rendimiento mayor combinando un conjunto de diversos modelos. Un método muy común es la *agregación Bootstrap* o *Bagging* [74], y es el que usa RF para combinar árboles de decisión aleatorios.

Hay que señalar que los árboles de decisión son modelos con una variación muy alta, ya que son muy sensibles a los datos de entrenamiento. Si se presenta un cambio en estos datos, el árbol resultante puede ser muy diferente y por tanto se obtienen predicciones muy diferentes. El método de bagging reduce la variación de los modelos independientes entrenando a cada uno de ellos con un subconjunto aleatorio del conjunto de entrenamiento con reemplazo. Es decir, los elementos del conjunto pueden aparecer repetidos en subconjuntos como dato de entrada. Así, se obtienen numerosos árboles de rendimiento variable y los resultados de cada uno de ellos se promedian para obtener el resultado final del bosque. Si se trata de un problema de clasificación, se obtiene por voto como la mayoría de los resultados. En caso de regresión, como la media aritmética de los resultados.

En general, la combinación de resultados de varios modelos en ensamble funciona mejor si los modelos no están correlacionados o muy poco correlacionados. En base a ello, el propio creador de RF y del método bagging, Breiman [72], introduce todavía una mejora: descorrelaciona los árboles de decisión mediante la elección de un subconjunto aleatorio de las variables a considerar en cada decisión de los árboles. Es decir, cada vez que se elige una pregunta en los nodos de decisión, se fija en solo una parte de las posibles variables de los datos. De este modo, los árboles entrenados en el bosque son más independientes entre sí, dejando fuera la posibilidad de que todos empiecen por unas pocas características con un poder predictivo muy alto. Resumiendo la explicación en una frase, los modelos RF son un conjunto de árboles de decisión "baggeados" cuyas selecciones se basan en un subconjunto limitado de variables.

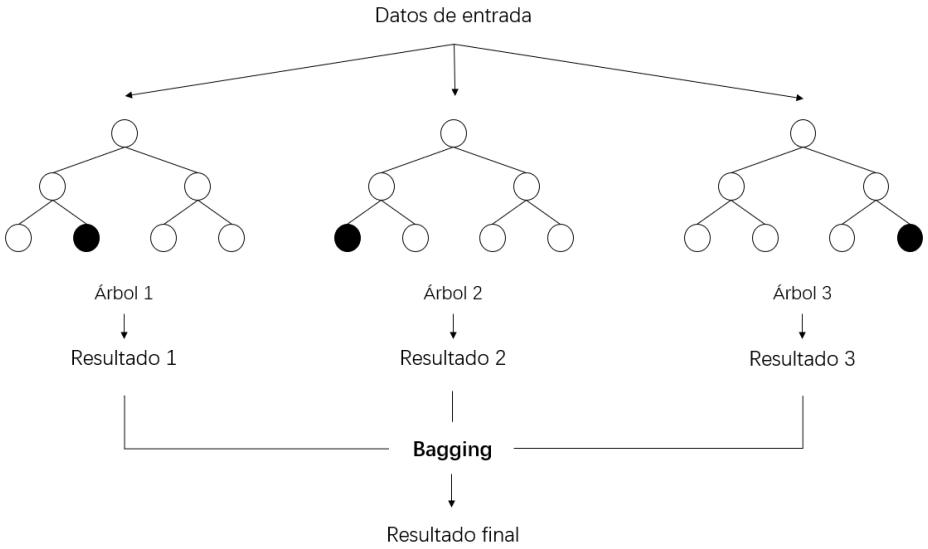


Figura 6.10: Ilustración de un modelo Random Forest

¿Es buena la elección de Random Forest?

Para justificar nuestra elección de este modelo, presentamos a continuación una serie de ventajas que ofrece RF frente a otros modelos de aprendizaje automático según las fuentes [72, 75, 76]:

- Es un modelo que no suele presentar sobreajuste, cuya importancia ya hemos discutido en 6.2.1. Mientras que un árbol de decisión tiende a sobreaprender, en un RF este riesgo se reduce por medio de la promedia de los resultados de múltiples árboles.
- Es capaz de tratar tanto conjuntos de datos grandes o pequeños, siempre manteniendo un rendimiento alto. Por lo tanto, cabe la posibilidad de que consiga una mejora frente a las redes neuronales en nuestro conjunto de datos.
- Además, el hecho de que se elige un subconjunto de las variables a la hora de tomar decisiones convierte RF en un modelo muy adecuado para los problemas de tipo *small n large p*. Es decir, donde hay muchas más variables que observaciones, como es el problema en cuestión.
- El RF estima bien incluso con datos faltantes o perdidos y puede garantizar una precisión bastante alta. En nuestro problema, pese a las pocas observaciones obtenidas, se observan datos inválidos en varios sujetos por un problema de la comunicación con el servidor o porque no han realizado el experimento completamente debido a la falta de control. Un RF no se ve demasiado condicionado por estos datos ausentes.
- No se requiere una hipótesis previa sobre la distribución de factores explicativos. En este aspecto es similar a las redes neuronales. Podemos alimentar el modelo con todos los datos sin tener que cumplir ciertos requerimientos.
- Permite la mezcla de factores categóricos y numéricos sin la necesidad de un preprocesamiento previo. Los datos tampoco necesitan ser reescalados o transformados de ninguna forma. Esto puede resultar favorable ya que nuestros datos presentan rangos de valores muy distintos.

- Es paralelizable, lo cual permite reducir el tiempo de entrenamiento en el futuro si disponemos de un número significativo de datos.
- Su velocidad de entrenamiento y de predicción es más rápida que muchos otros modelos. Además, como en cada decisión solo se considera subconjuntos de las variables, posibilita trabajar con numerosas variables, como ocurre en nuestro caso al aplanar las dimensiones de los datos.

Si solo tenemos las ventajas en cuenta, es un modelo bastante adecuado para nuestro problema. No obstante, no debemos olvinarnos de los posibles problemas que puede tener. En primer lugar, el tamaño del conjunto de datos sigue suponiendo un riesgo. Siempre hay más probabilidad de que el modelo sobreaprenda sobre los pocos datos que tenemos y por tanto no tenga suficiente poder de generalización ante nuevos sujetos en la predicción real. Además, en general el método de bagging no es ideal para una muestra reducida, debido al reemplazo que realiza para formar nuevos subconjuntos de entrenamiento. Hay una probabilidad más alta de que una observación se repita y puede originar subconjuntos de entrenamiento muy sesgados o repetitivos. Esto puede resultar un impacto en el promedio de los resultados, aunque los árboles sean no correlacionados e independientes.

Otra problemática es que el RF no está diseñado para datos en serie. Existen otros modelos más adecuados para este tipo de estructuras, como las redes neuronales recurrentes probadas anteriormente. Sin embargo, como disponemos de pocos registros temporales, cabe la posibilidad de que RF sea capaz de identificar variables más discriminatorias en ciertos momentos del juego.

En resumen, es un modelo que merece una oportunidad. Veremos los resultados que podemos conseguir con él.

Resultados

Realizamos la implementación con la librería *scikit-learn* [77] en Python. La función `RandomForestRegressor` es la que necesitamos usar y por defecto tiene una configuración estándar si no especificamos los hiperparámetros. Sin embargo, podemos ajustar los siguientes en busca de un rendimiento mayor:

- **Número de variables a considerar en cada separación**: Es probablemente el parámetro más importante en un RF. Como comentamos anteriormente, ayuda a descorrelacionar los árboles entrenados. En un problema de regresión, [72] recomienda fijar este valor al tercio del número de variables totales.
- **Proporción del conjunto de entrenamiento que se usa en bagging**: Por defecto se elige un subconjunto del mismo tamaño que el conjunto de entrenamiento para entrenar a cada árbol en el bosque. Una proporción menor puede llevar a árboles más diferentes. Probamos diferentes porcentajes desde un 10 % hasta un 100 % con un salto de 10 %.
- **Profundidad máxima de los árboles**: Árboles con más profundidad sobreaprenden más de sus datos de entrenamiento, pero así están menos correlacionados entre sí. Diferentes valores desde 10 hasta 50 son puestos a prueba.
- **Número de árboles en el bosque**: En general, cuanto más árboles incluimos en el bosque mejor suele ser el rendimiento obtenido. Se suele incrementar este valor hasta que no presenta ninguna mejora. Probamos un rango de 10 a 1000.

Para evaluar un modelo concreto, utilizamos aquí otra vez validación cruzada, explicada en la sección 6.2.1. Hay que tener en cuenta que con los mismos parámetros configurados, podemos obtener resultados

distintos de una ejecución a otra por la propiedad estocástica del modelo. Nos aseguramos de que con varias ejecuciones del código obtenemos resultados parecidos, así que los modelos son estables.

En los tres casos, los hiperparámetros por defecto alcanzan un resultado satisfactorio. Los errores mae son presentados en la tabla 6.12.

	MAE validación	MAE prueba
Blek	3.24 ± 0.37	2.86 - 3.15
Edge	0.91 ± 0.20	0.88 - 0.95
Unpossible	1.67 ± 0.39	1.42 - 1.74

Tabla 6.12: Resultados de Random Forest

Observamos que, como intuimos, los modelos de RF se comportan de manera similar o incluso mejor que los modelos de redes neuronales. Podemos concluir que, con un conjunto de datos tan reducido como el en cuestión, no es necesario recurrir a técnicas tan complejas de aprendizaje profundo. Un algoritmo tradicional de aprendizaje automático como el de random forest tiene la misma capacidad de predicción.

Resumen

- De cara a confirmar las métricas propuestas desarrolladas en la fase anterior, planteamos un primer análisis desde el punto de vista estadístico, en busca de las relaciones entre las nuevas variables y el factor de inteligencia general.
- Adoptamos un análisis factorial exploratorio ante el confirmatorio, debido al carácter del problema y el número extenso de variables.
- Antes de aplicar el proceso, incluimos un estudio sobre la fiabilidad de las variables y una regresión lineal por pasos para seleccionar las 6 variables más representativas entre todas. Entre los dos modelos factibles (unifactorial y bifactorial), el bifactorial se ha considerado más adecuado en muchos aspectos. Consiguió explicar un total de 72,427 % de la varianza total de todos los indicadores considerados. Una interpretación posible a los factores es que el primero corresponde con la inteligencia general y el segundo con la habilidad lógico-espacial.
- Un segundo enfoque consiste en intentar reducir la duración establecida de las sesiones del experimento, mediante la predicción de resultados de los jugadores en partidas truncadas.
- Utilizamos técnicas de aprendizaje profundo para tratar los datos secuenciales en el eje temporal, consiguiendo mejorar los modelos base. Sin embargo, los resultados en el conjunto de prueba fueron peores, a causa del tamaño reducido del conjunto de datos.
- Con el fin de encontrar una técnica más adecuada para este tipo de datos, empleamos modelos de random forests. Se han conseguido resultados similares al análisis anterior.

Conclusiones y trabajo futuro

El estudio conducido en [8] obtuvo resultados prometedores al tratar de establecer un factor de correlación entre el rendimiento en juegos comerciales de entretenimiento, y la inteligencia general. Pese a los hallazgos favorables de dicho trabajo, este planteaba algunos puntos de mejora. En particular, las medidas tomadas eran mayormente unidimensionales, y los datos fueron recolectados de forma manual, limitando las posibilidades de análisis y dificultando el proceso de registro de datos. El presente trabajo se desarrolló con el fin de reproducir los resultados del experimento original, si bien a menor escala, intentando al mismo tiempo mejorar estos puntos débiles y ampliar el rango de posibles análisis a realizar sobre los datos obtenidos. Para ello, se realizó una revisión bibliográfica inicial, destinada a comprender los mecanismos clásicos empleados para medir la inteligencia general. Al mismo tiempo, se incluyó un estudio previo de los distintos tipos de sistemas de telemetría y analíticas en juegos utilizados en contextos similares.

El estudio de partida consideraba tres habilidades cognitivas fundamentales definidas en el segundo estrato del modelo CHC, concretamente el razonamiento fluido (Gf), la capacidad visuoespacial (Gv), y la velocidad de procesamiento (Gs). Optamos por implementar tres de los diez títulos considerados originalmente en la batería de tests del experimento de referencia, uno para cada uno de estos tres factores cognitivos. Blek se presenta como un candidato natural para la medida del razonamiento fluido, al incorporar una mecánica innovadora y un exigente proceso de aprendizaje a lo largo de sus niveles; Edge, por su parte, se corresponde de manera clara con la capacidad visuoespacial al requerir navegar de forma más o menos compleja por un espacio tridimensional en perspectiva isométrica; finalmente Unpossible, pese a sus mecánicas relativamente simples, posee una relación más directa con la velocidad de procesamiento por requerir concentración constante y gran velocidad de reacción en intervalos de tiempo irregulares. De este modo, estos tres juegos se consolidaron como los candidatos más prometedores para evaluar cada una de las habilidades cognitivas listadas.

El objetivo principal de esta implementación era permitir instrumentalizar por código el envío de datos relativos al comportamiento de los jugadores, y a sus interacciones con los juegos. Para ello, se desarrolló un sistema de telemetría general para la transmisión de eventos a un servidor propio, tomando como referencia distintos productos comerciales como Unity Analytics o Google Analytics. La infraestructura correspondiente a este sistema se fue produciendo de manera progresiva, y está constituida por tres componentes esenciales. Estos son: una librería de envío de eventos genéricos desde juegos implementados en Unity, un backend destinado a la recepción de dichos eventos y a su almacenamiento en una base de datos no relacional, y un frontend para hospedar y distribuir el conjunto de títulos al público general de cara a conducir un experimento web.

Una vez satisfechos los requisitos técnicos del proyecto, se procedió a lanzar dicho experimento online. La población a la que iba dirigida la plataforma no presentaba en principio ninguna limitación de edad o contexto, por lo que se acabó consiguiendo datos de un colectivo notablemente heterogéneo. Una vez

recogidos los datos del experimento, se procedió al desarrollo de métricas de cara a la fase de análisis. Para cada uno de los juegos, se computaron no sólo las métricas originales consideradas en el estudio de referencia, sino también un conjunto de métricas adicionales propias. 62, 59 y 58 participantes produjeron trazas de eventos válidas para Blek, Edge y Unpossible, respectivamente, si bien sólo 51 llegaron a completar las tres sesiones correctamente. A partir de estos datos, condujimos dos tipos fundamentales de análisis. El primero de ellos se centraba en buscar factores latentes subyacentes a los indicadores registrados por medio de un análisis factorial exploratorio o EFA. Por otro lado, el segundo tenía como objetivo predecir el resultado final de un jugador dado sobre partidas truncadas en las que sólo se consideraban los datos asociados a los primeros minutos del experimento.

En el análisis EFA, de las 6 variables consideradas para representar el rendimiento en los juegos, obtuvimos un modelo unifactorial y un modelo bifactorial. El segundo de ellos lograba explicar un 72,427 % de la varianza total de todos los indicadores considerados, en contraste con el 55,410 % obtenido por el primer modelo. Interpretando las cargas de los factores tras aplicar una rotación oblicua, identificamos los dos factores del modelo bifactorial como el factor de inteligencia general y un factor lógico-espacial, con una alta proyección en Blek. Estos dos factores muestran una clara correlación mutua (0.463), en el sentido de que el segundo refleja en cierto modo el primero. Este resultado resulta muy positivo, al confirmar la hipótesis de que el grado de inteligencia subyace en buena medida a las variables registradas.

El otro gran objetivo de nuestro trabajo era estudiar si resulta factible reducir la duración de los experimentos en cada juego, tratando de predecir los resultados finales a partir de partidas truncadas de los participantes. En este contexto, convertimos en primera instancia los datos registrados en una representación secuencial. Pese al tamaño reducido del conjunto de datos, logramos utilizar con éxito técnicas de aprendizaje profundo, probando tanto modelos de redes neuronales prealimentadas como de redes recurrentes con diferentes configuraciones. Los resultados no llegaron a mostrar una preferencia clara entre ambos tipos. Por lo general, los modelos entrenados superan significativamente a los modelos base, a excepción de en el caso de Edge, donde el margen de error ya era bastante bajo. No obstante, estos modelos obtienen resultados mucho peores sobre los conjuntos de prueba, mostrando en ocasiones variaciones muy elevadas.

Identificamos este fenómeno como consecuencia del tamaño reducido de nuestro conjunto de datos. Los modelos de aprendizaje automático apenas suelen considerarse en el contexto de conjuntos de datos tan pequeños, donde un número insuficiente de muestras de entrenamiento pueden comprometer el éxito del aprendizaje. Distintos estudios han demostrado que las técnicas tradicionales de aprendizaje automático suelen ser mejores para este tipo de datos. Como alternativa, empleamos *random forests* considerando sus ventajas sobre otros algoritmos usuales. Por este método, se obtuvo un resultado similar en los datos de prueba. Como conclusión, en nuestro caso específico, no resulta necesario recurrir a técnicas de aprendizaje profundo, pues los algoritmos tradicionales de aprendizaje automático como los *random forests* resultan ser suficientes para alcanzar los mismos resultados. Sin embargo, si se dispone de un conjunto de datos más grande, alentamos a repetir este mismo análisis comparativo para comprobar si verdaderamente existen diferencias significativas.

Trabajo Futuro

En cuanto a trabajo futuro, consideramos que existen dos líneas principales por las que poder continuar el estudio llevado a cabo en la presente memoria.

La primera linea está estrechamente relacionada con el experimento de partida, y plantea alcanzar aquellos objetivos que quedaron sin cumplir en este trabajo a causa de la pandemia del COVID-19. En

particular, consideramos las siguientes propuestas:

- Llevar a cabo un experimento presencial con un número sustancial de participantes, empleando los juegos instrumentalizados, y complementando los resultados mediante la administración de tests de inteligencia estandarizados. Esto permitiría aplicar métodos de regresión, modelos CFA y SEM (empleados en el estudio original) y distintas técnicas de aprendizaje automático, para tratar de predecir las diferentes habilidades cognitivas de los participantes en función de su interacción con los juegos.
- Ampliar el conjunto de eventos lanzados por los juegos instrumentalizados para aumentar la capacidad expresiva del sistema de telemetría. Del mismo modo, sería posible ampliar el conjunto de métricas inferidas a partir de estos eventos de cara a extraer nuevas conclusiones en el análisis, o confirmar las conclusiones ya obtenidas.
- Lanzar la plataforma desarrollada para la difusión web de los juegos de forma global de cara a incrementar el número de sujetos y de datos registrados. En este caso, sería necesario encontrar una manera factible de administrar los tests de inteligencia estandarizados vía web, así como potencialmente desarrollar métodos para determinar qué trazas de eventos son verdaderamente válidas.

La segunda linea, por otra parte, considera continuar el análisis específico llevado a cabo en este documento, considerando las siguientes propuestas:

- Encontrar e implementar nuevos videojuegos capaces de medir las habilidades cognitivas consideradas (Gf, Gv y Gs), o incluso habilidades adicionales consideradas en el segundo estrato del modelo CHC. Estos juegos podrían extraerse de la lista considerada en el estudio de referencia (que recordamos contenía 10 títulos distintos), o podrían escogerse siguiendo algún criterio que tenga en cuenta el potencial de los mismos para recoger comportamientos relativos a alguna de estas capacidades. En cualquier caso, un sistema de telemetría semejante al presentado en esta memoria podría utilizarse para registrar nuevos conjuntos de eventos específicos de dichos juegos adicionales, y sería necesario elaborar nuevas métricas que sean lo suficientemente representativas para su análisis posterior.
- Como ya se ha comentado, en un sistema web el comportamiento de los usuarios durante las sesiones puede resultar impredecible. En nuestro caso, los datos obtenidos eran preprocesados y "limpiados a mano" para tratar de descartar muestras no válidas. Una propuesta interesante consistiría en encontrar un método para detectar estos casos de forma automática sin intervención humana.
- El análisis estadístico realizado se ha visto limitado por el tamaño del conjunto de datos. En el caso de disponer de un conjunto más extenso, se recomienda llevar a cabo un análisis más exhaustivo y riguroso, especialmente en la selección del número de factores latentes. Se podrían considerar distintos métodos de extracción o rotación de factores, en lugar de los empleados en este trabajo. Asimismo, sería posible conducir análisis adicionales utilizando las puntuaciones factoriales obtenidas, como por ejemplo la predicción del rendimiento en un juego concreto a partir de los demás. En última instancia, podría añadirse un análisis CFA a fin de confirmar las interpretaciones de los factores resultantes del EFA, como ocurriría en un análisis factorial habitual.
- En cuanto al uso de redes neuronales en modelos de predicción de resultados, con conjuntos de datos suficientemente extensos pueden obtenerse siempre mejores resultados mediante un proceso de

ajuste de hiperparámetros. Aunque el aprendizaje profundo elimina en principio la necesidad de la ingeniería de características, también se podría tratar de transformar los datos en representaciones más fácilmente entendibles para las redes neuronales.

Conclusions and future work

The study conducted in [8] yielded promising results in seeking to establish a correlation between performance in commercial (entertainment) games and general intelligence. Despite the favorable findings in this work, some room for improvement was noted. In particular, the measurements considered were mostly one-dimensional, and the data was collected manually, limiting the possibilities for analysis and making the data recording process more difficult. The present work was developed in order to reproduce the results from the original experiment, albeit on a smaller scale, while trying to improve these weak points and expand the range of possible analysis to be performed on the data obtained. For this purpose, an initial literature review was carried out, aimed at understanding the classic mechanisms used to measure general intelligence, as well as a previous study of the different types of telemetry and game analytics systems used in similar contexts.

The initial study considered three fundamental cognitive abilities defined in the second stratum of the CHC model, namely fluid reasoning (Gf), visuospatial capacity (Gv), and processing speed (Gs). We chose to implement three of the ten titles originally used in the test battery, each of which aimed to measure one of these three cognitive factors. Blek is presented as a natural candidate for the measurement of fluid reasoning, as it incorporates innovative mechanics and a demanding learning process throughout its levels; Edge, on the other hand, corresponds clearly to visuospatial capacity by requiring to navegate in a more or less complex way through a three-dimensional space in an isometric perspective; Finally, Unpossible, despite its relatively simple mechanics, has a more direct relationship with the processing speed, as it requires constant concentration and high reaction speed in irregular time intervals. In this way, these three games are consolidated as the most promising candidates to evaluate each of the aforementioned cognitive abilities.

The main objective of this implementation was to allow for an instrumentalization of the sending process of data related to the players' behavior and their interactions with the games. To this aim, a general telemetry system was developed for the transmission of events to a custom server, taking different commercial products such as Unity Analytics or Google Analytics as reference. The infrastructure corresponding to this system was gradually produced, and consists of three essential components. These are: an event logging library intended for sending generic events from games implemented in Unity, a backend for receiving these events and storing them in a non-relational database, and a frontend to host and distribute the set of titles to the general public with the aim of conducting a web experiment. Once all technical requirements of the project were fulfilled, we proceeded to launch the online experiment. In principle, the population targeted by the platform did not have any major limitation on age or context, so data was eventually obtained from a remarkably heterogeneous group. Once the data from the experiment had been collected, metrics were developed for the analysis phase. For each of the games, not only the original metrics from the reference study, but also a set of additional custom metrics were considered. 62, 59, and 58 participants produced valid event traces for Blek, Edge, and Unpossible, respectively, alt-

hough only 53 successfully completed all three games. From these data, we conducted two fundamental types of analysis. The first of them focused on searching for underlying latent factors in the recorded indicators by means of an exploratory factor analysis or EFA. On the other hand, the second type of analysis aimed to predict the final result of a given player given truncated event logs where only the data associated with the first minutes of the experiment were considered. In the EFA analysis, we obtained a unifactor model and a bifactor model from the 6 variables considered to represent game performance. The second of them managed to explain $72,427\,\%$ of all the considered indicators' total variance, in contrast to the $55,410\,\%$ obtained by the first model. By interpreting the factor loadings after applying an oblique rotation, we identified the two factors of the bifactor model as the general intelligence factor and a logical-spatial factor, with a high projection on the game Blek. These two factors show a clear mutual correlation (0.463), as the second reflects the first in some way. This result is very positive, confirming the hypothesis that intelligence underlies in good measure the recorded variables. The other main goal of our work was to study whether it is feasible to reduce the duration of the experiment in each game, by predicting the final results from the participants' truncated game logs. In this context, we first converted the recorded data into a sequential representation. Despite the small size of the dataset, we were able to successfully use deep learning techniques, testing both feed-forward and recurring neural network models with different configurations. The results showed no clear preference between both types. In general, the trained models significantly outperform the baseline models, except for Edge, where the margin of error was already quite low. However, these models obtained much worse results on the test data, sometimes showing high variance levels.

We identified this phenomenon as a result of the small size of our dataset. Machine learning models are rarely viewed in the context of small data, where insufficient number of training samples can compromise the learning success. Different studies have shown that traditional machine learning techniques are often better suited to these types of data. As an alternative, we made use of random forests considering its advantages over other algorithms. Using this method, a similar result on the test data was obtained. In conclusion, in our specific case, it is not necessary to resort to deep learning techniques, since traditional shallow learning algorithms such as random forests turn out to be sufficient to achieve the same results. However, if a larger dataset is available, we encourage repeating this same comparative analysis to see if there exists a truly significant difference.

Future Work

Regarding future work, we consider there to be two main lines of work which could be followed in order to continue the study carried out in this document.

The first line is closely related to the initial experiment, and aims to achieve those objectives that were not fulfilled in this work due to the COVID-19 pandemic. In particular, we consider the following proposals:

- Carry out a face-to-face experiment with a substantial number of participants, using the instrumentalized games, and complementing the results by administering standardized intelligence tests. This would allow for the application of regression methods, CFA and SEM models (used in the original study) and different machine learning techniques, to try to predict the participants' different cognitive abilities based on their interaction with the games.
- Expand the set of events logged by instrumentalized games in order to increase the expressive capacity of the telemetry system. Similarly, it would be possible to expand the set of metrics

inferred from these events in order to draw new conclusions in the analysis phase, or to confirm the conclusions already obtained therein.

- Launch the platform developed for the web distribution of the games globally in order to increase the number of subjects and the size of registered data. In this case, it would be necessary to find a feasible way to administer standardized intelligence tests over the internet, as well as potentially developing methods to determine which event traces are truly valid.

The second line, on the other hand, considers continuing the specific analysis carried out in this document, considering the following proposals:

- Find and implement new computer games capable of measuring the differente cognitive skills from the original paper (Gf, Gv and Gs), or even additional skills considered in the second stratum of the CHC model. These games could be selected from the list included in the reference study (which contained 10 different titles), or could be chosen following some criterion taking into account their potential to register behaviors related to some of these abilities. In any case, a telemetry system similar to the one presented in this work could be used to record new sets of events specific to these additional games, and it would be necessary to develop new metrics that are representative enough for further analysis.
- As already mentioned, user behaviour during sessions held on a web system can be rather unpredictable. In our case, the data obtained was thoroughly preprocessed and "cleaned by hand" to try to discard invalid samples. An interesting proposal would be to find a method to detect these cases automatically without the need for human intervention.
- The statistical analysis performed was deeply limited by the size of the data set. In the case of having a more extensive set, it is recommended to carry out a more exhaustive and rigorous analysis, especially regarding the selection of the number of latent factors. Different factor extraction or rotation methods could be considered, instead of those used in this work. Likewise, it would be possible to conduct additional analysis using the factorial scores obtained here, such as predicting performance in a specific game, from performance registered in the rest. Ultimately, a CFA analysis could be added to confirm the interpretations of the factors resulting from the EFA, as would be the case in a typical factor analysis.
- Regarding the use of neural networks in outcome prediction models, with sufficiently large data sets, better results can always be obtained through a hyperparameter adjustment process. Although deep learning in principle eliminates the need for feature engineering, one could also try to transform data into more easily understandable representations for neural networks.

Contribución individual al proyecto

Pablo Gutiérrez Sánchez

Estudiante del Doble Grado en Ingeniería Informática y Matemáticas.

La primera parte del trabajo comprende la investigación previa sobre las distintas técnicas empleadas en la medida de la inteligencia, y las nociones fundamentales sobre sistemas de telemetría en videojuegos. En esta fase, mi trabajo consistió en llevar a cabo la investigación referente a los procesos de *Game Analytics* y telemetría. Inicialmente obtuve información general sobre el proceso de GA a través de distintas fuentes, y prestando especial atención a [25], donde se relata en detalle la metodología general de *Knowledge Discovery*. Este proceso pasó a convertirse en la guía implícita a seguir en el resto del proyecto, como se ha tratado de enfatizar a lo largo de la memoria.

De cara a analizar cómo se seguían estos métodos en casos reales, consulté diferentes papers centrados en estudios previos. Entre estos, [26] (capítulo 14) resulta particularmente interesante, al desarrollar en detalle cuatro casos ilustrativos. En concreto, opté por incluir en el trabajo un resumen del caso correspondiente al análisis de causas de muerte en el título *Tomb Raider: Underworld*, donde se introducían además los mapas de calor, de uso extendido en estos contextos. A continuación, siguiendo la línea de la investigación sobre analíticas y telemetría, centré mi atención en la lectura de artículos y libros más dirigidos a instrumentalización de juegos serios, con el fin de consultar las técnicas y patrones típicos en dicho área. En especial, se realizó una lectura en profundidad de los primeros capítulos de [23], y se extrajeron los puntos más importantes de cara a su incorporación en el trabajo. La razón para estudiar estas fuentes es la naturaleza híbrida de nuestro objetivo. Por un lado, partimos de juegos comerciales destinados al entretenimiento puro, pero nuestra meta resulta más afín a las consideradas habitualmente en juegos serios (evaluación de rendimiento, o de capacidades cognitivas en este caso).

De forma simultánea, se procedió a implementar "a mano" los tres juegos ya presentados en la memoria por parte de los integrantes del equipo, uno por cada una de las capacidades cognitivas consideradas en el estudio de referencia. Cada uno de nosotros comenzó pues a desarrollar una versión "instrumentalizable" del juego correspondiente. En mi caso particular, fui el responsable de *Blek*, el juego escogido para la evaluación del razonamiento fluido, o la capacidad de un individuo para resolver problemas nuevos sin conocimiento previo. Se comenzó produciendo un primer prototipo para verificar el funcionamiento de las mecánicas principales del juego, prestando especial atención al control de la traza, la gestión del estado del jugador, y la implementación de los distintos tipos de *Prefabs* para poder generar un conjunto reducido de niveles de prueba. Una vez tuve el visto bueno de los tutores del trabajo, pasé a añadir las mecánicas restantes, como las transiciones entre niveles, los menús de control e interfaces, y nuevos elementos como las bolas con proyectiles. En última instancia procedí a la inclusión de un total de 35 niveles procedentes del juego original: 8 a modo de "tutorial" y 27 a modo de "sesión". Durante todo el

proceso de desarrollo se tuvieron en cuenta los patrones de diseño en videojuegos detallados en [36].

Una vez completada la fase de investigación sobre telemetría y analíticas, procedí a buscar diferentes alternativas para instrumentalizar los distintos juegos que estábamos desarrollando. Unity Analytics [22] y Google Analytics [20] se presentaban como opciones prometedoras, pero tras experimentar con ellas en distintos proyectos de prueba llegué a la conclusión de que no eran las más apropiadas para nuestros intereses. Como consecuencia, pasé a implementar la infraestructura de un sistema muy básico de envío de eventos, compuesto por un servidor escrito en Node.js utilizando Express como framework y MongoDB como base de datos subyacente, y una librería en Unity que pretendía imitar, en la medida de lo posible, la estructura de logging seguida por Unity Analytics. Una vez terminé de desarrollar este módulo, lo convertí en un fichero `.dll` que distribuí entre los demás integrantes del equipo para que pudieran incorporarlo en sus respectivos proyectos.

Una vez completada la primera instrumentalización de los juegos, y terminada una serie de pruebas preliminares para comprobar el correcto envío de eventos al servidor, procedimos de manera conjunta a redactar un protocolo de actuación en vistas a conducir un experimento físico en la facultad de informática. Se especificaron, asimismo, los eventos que estábamos interesados en registrar desde los juegos y se añadieron al código de cada juego.

Sin embargo, las condiciones inusuales debidas a la pandemia del COVID-19 llevaron a una situación en la que ya no resultaba factible realizar sesiones presenciales. Una vez se decidió migrar los juegos a una plataforma web de cara a su distribución online, comencé a realizar los cambios desarrollados a comienzos del capítulo 5. Esto suponía la inclusión de un sistema básico de gestión de usuarios en el lado del backend, y el desarrollo de la propia plataforma web. Por la familiaridad previa con el framework, opté por utilizar React para crear la aplicación web, y Heroku para hospedarla. De manera simultánea al desarrollo de la plataforma, los tres integrantes del equipo realizamos los cambios pertinentes para adaptar los juegos a versiones de WebGL, las cuales integré en la página por medio de una librería externa [50]. En el proceso de migración, sin embargo, detecté diversos problemas a la hora de incorporar las credenciales de usuario en las peticiones `HTTP` lanzadas desde el motor de Unity WebGL en el navegador, lo que me llevó a descartar la librería de envío de eventos que habíamos incorporado en las versiones de los juegos para PC y Android, en favor del método descrito en la sección 5.2.

Finalizada la migración web, dimos comienzo al experimento online, pasando a distribuir la aplicación entre distintos contactos. Durante el trascurso del experimento, fui el encargado de gestionar la corrección de los errores y problemas encontrados por los distintos usuarios, así como de realizar las modificaciones correspondientes en respuesta al feedback obtenido. A finales del experimento realicé una primera limpieza de eventos en la base de datos, eliminando las partidas no válidas reportadas por usuarios e integrantes del equipo.

Una vez hubimos descargado los datos recolectados durante aproximadamente tres semanas de experimento, dimos pie a la fase de análisis de datos. En esta etapa, se nos instó a realizar una lectura en detalle de ciertas secciones de interés en [60], con el fin de familiarizarnos con algunas técnicas de procesamiento de secuencias por medio de redes neuronales. Consideré algunos modelos previos siguiendo las estrategias ahí descritas, desde un uso básico de redes recursivas sobre versiones transformadas de las trazas de eventos, hasta un intento de procesar secuencias empleando estrategias habitualmente usadas en textos. No obstante, el modelo final considerado en el trabajo se debe en su mayoría a Mingxiao, que acabó haciéndose cargo de la mayoría del esfuerzo de análisis detallado en la memoria, produciendo modelos muy prometedores.

En cuanto a trabajo de redacción y de estructura de la memoria, fui el encargado de redactar los capítulos 2, 4, la sección correspondiente a Blek del capítulo 3 y las secciones correspondientes a modifi-

caciones sobre la adquisición de datos del capítulo 5. En este último capítulo, tras una primera revisión por parte de nuestros tutores, tuvimos que realizar una labor de reestructuración sobre la base inicialmente redactada por Alejandro, terminando este por ser un módulo finalmente escrito de forma conjunta entre los tres. Las partes de resumen, introducción y conclusiones fueron redactadas conjuntamente con Mingxiao.

Alejandro Ortega Álvarez

Estudiante del Grado en Desarrollo de Videojuegos.

Mis tareas están divididas en (1) tareas comunes que nos afectaron a los 3 miembros del proyecto; (2) tareas en las que ayudé parcialmente pero en menor grado (en este caso menciono el miembro del grupo que realizó la mayor parte de la tarea); y (3) tareas mías exclusivas, que serían las excluyentes a las otras dos.

Como primer paso se inició el estudio e investigación común para los tres miembros referido a la medida de la inteligencia y la evaluación de factores psicológicos directamente relacionados con el tema propuesto para el proyecto. A su vez, debimos de forma común analizar cada uno de los juegos propuestos por el tutor mencionados en la sección 3, y estudiar las posibles tecnologías y metodologías a emplear para tomar la decisión. Desde el comienzo del proyecto, realizábamos tutorías con los profesores del TFG y con la profesora en psicología María Ángeles Quiroga, cada 2-3 semanas aproximadamente, para que estuvieran al tanto de todas las novedades y decisiones en cuanto a la investigación del proyecto, y para resolver las posibles dudas de implementación que ocurrieran.

Posteriormente y como trabajo de más peso en cuanto a tiempo empleado, tuvo lugar el desarrollo e implementación del juego elegido para cada uno. En mi caso fue el juego de Edge, el juego escogido para la evaluación de la habilidad visoespacial. La implementación y estructura correspondiente se detalla en el apartado 3.3, y el código del proyecto en Unity se puede encontrar en el repositorio [37]. Para este juego, se comenzó implementando el movimiento característico del jugador junto con la cámara y poco a poco añadiendo los demás elementos del juego como *prefabs* (activadores, prismas, plataforma final, etc) junto con los gestores o *managers*. Las plataformas movibles y las simulación parcial de físicas se añadió posteriormente y fue lo más costoso en cuanto a dificultad. Una vez hecho los primeros niveles, los siguientes se construían a partir de elementos de otros anteriores, aunque casi siempre iban apareciendo nuevos elementos que añadir a los nuevos niveles. Se desarrollaron en un principio 6 niveles para el experimento que incluían 3 niveles de tutorial y 3 de sesión. Pero debido a un error en uno de los anexos adjuntados del experimento antiguo, se desarrollaron erróneamente los 3 últimos niveles y posteriormente se tuvieron que desarrollar otros 6 niveles nuevos que eran los que en verdad coincidían con los que se jugaron en el estudio de referencia. Esto hizo que la tarea particular de desarrollo de Edge se retrasara un poco más de lo esperado. Afortunadamente, los otros 3 niveles erróneos no se desaprovecharon y decidí colocarlos justo al final de los otros 6, que junto con los 3 niveles de tutorial hicieron un total de 12 niveles para la experiencia del jugador. Se estudiaron las variables propuestas para el juego y se propusieron algunas nuevas para la posible mejora en el análisis (las mencionadas en la sección 5.4). A la par que se iba desarrollando el juego, se fue incluyendo el sistema de telemetría diseñado por Pablo mediante una .dll, e incluyendo las variables a registrar particulares del juego.

Al llegar al punto en el que prácticamente estaban los 3 juegos acabados, relizamos una pequeña fase de testeo entre los 3 miembros del proyecto para probar los otros juegos y tratar de sacar errores no detectados en la fase de desarrollo. Esto conllevó unos pocos arreglos que solucionar en Edge antes de

poder lanzar la versión jugable del experimento. Mientras ocurría esta fase de testeo y arreglo de fallos, se iba desarrollando y estructurando de forma común el protocolo del experimento que marcaba las pautas y los procedimientos a seguir para el experimento físico que pensamos en un principio que iba a tener lugar. Poco después ocurrió el incidente del COVID-19 que nos obligó a cambiar los planes y realizar otro enfoque distinto del experimento. Mientras aún no se sabía si se prolongaría demasiado o no el estado de alarma, realicé una lista de ventajas e inconvenientes que comparaba las diferencias entre realizar el experimento de manera presencial, o comenzar a desarrollar un sistema para realizar experimentos de manera online, que nos requería trabajo extra. Esta lista puede verse en la sección referida al experimento 5.3. Pronto vimos que iba a resultar inviable el realizar experimentos presenciales por lo que procedimos a la preparación de los experimentos de manera online mediante una página web. Para ello se tuvieron que añadir al juego de Edge todos los cambios necesarios para su cambio de plataforma. También tuvieron que corregirse algunos errores puntuales más en el juego que no sucedían en la versión de escritorio y sí en la versión para WebGL.

En cuanto tuvimos los juegos listos para los experimentos online, nuestra tarea fue conseguir participantes entre amigos, familia y redes sociales. Tuvimos pues un primer acercamiento con usuarios de confianza, los cuales nos dieron un feedback muy relevante. Con ello tuvimos que tomar algunas medidas de forma común para reforzar el experimento: en definitiva cambios que ayudaran y facilitaran a los usuarios y mejorara su experiencia de juego. Esto se resumió en algunos detalles dentro de los juegos y, sobre todo, el resumir la cantidad de texto de instrucciones tanto de la propia página web como de cada uno de los juegos. Una vez estos errores fueron solucionados, tan sólo tuvimos que esperar a la recopilación de datos, observando cómo iban siendo los resultados y eliminando algunos que no cumplieran con la información mínima necesaria o que tuvieran algún error esporádico en el registro de eventos.

Una vez obtenidos los datos de los juegos, se estudió comúnmente varios métodos que pudieran resultar eficaces para analizar los resultados teniendo en cuenta la poca cantidad de información obtenida. Como primera idea planteé en Python para Edge un sistema de redes neuronales para la predicción de posibles datos mediante regresión. Con este sistema se pretendía determinar hasta dónde sería capaz de llegar un usuario obteniendo la información de sus primeros minutos de juego. Entre esta información se incluían, por cada nivel, los prismas recogidos, las muertes producidas, el número de movimientos y el tiempo empleado en el nivel; y con todo ello el propósito era predecir a qué nivel sería capaz de llegar el usuario y cuántos prismas en total recogería. Aunque éste código no fue del todo eficaz y se empleó otro distinto pero semejante implementado en mayor medida por Ming para el análisis final.

En cuanto a aportaciones en la memoria, me corresponden como investigación el capítulo de Medida de la Inteligencia 1; como implementación, el apartado de desarrollo del juego de Edge 3.3; secciones comunes como introducción, resúmenes, palabras clave y conclusiones; el protocolo del experimento; y el capítulo que explica el desarrollo del experimento 5. A éste último aportaron posteriormente tanto Ming como Pablo partes que enlazaban directamente con información de sus secciones.

Mingxiao Guo

Estudiante del Doble Grado en Ingeniería Informática y Matemáticas.

La primera fase del trabajo comprende un estudio previo sobre las investigaciones anteriores de las que partimos en el trabajo actual. Esta tarea es realizada conjuntamente por todos los integrantes del grupo. Dentro de dicho estudio, mi trabajo fue indagar más en los resultados obtenidos en diferentes análisis sobre la relación entre videojuegos y la inteligencia, así como los detalles técnicos desarrollados,

de cara a comprender mejor el proceso de un análisis estadístico posterior. Se realizó a posteriori una comparativa a alto nivel entre distintos resultados, con el fin de redactar el estudio de antecedentes en la introducción de la memoria.

Tras examinar todos los juegos incluidos en la batería de test del artículo de partida, cada integrante eligió un juego que trata cada una de las tres habilidades cognitivas estudiadas. En mi caso, me correspondió el juego *Unpossible*, un candidato especialmente interesante para medir la velocidad de procesamiento. Para ello, debido a la exigencia de los modelos 3D, realizé previamente un estudio sobre herramientas de modelado 3D y librerías disponibles en Unity para crear objetos curvados. Entre todas las disponibles, se optó por Blender y la librería Bezier Path Creator. También resultó necesario un proceso de aprendizaje sobre la representación de las rotaciones en este motor, ya que son imprescindibles para el juego a implementar. Se elaboró un primer prototipo con los obstáculos primitivos creados, para poner a prueba la mecánica del juego. Una vez se consiguió que esta sea más robusta y madura tras varias iteraciones, incorporé la distribución de todos los obstáculos que corresponden a los niveles requeridos para el experimento. Adicionalmente, se introdujeron mejoras en el aspecto visual del juego, utilizando shaders en Unity.

Al mismo tiempo de la implementación de los tres juegos, se desarrolló por parte de Pablo el sistema de la recogida de eventos, incluyendo tanto una librería a incorporar en cada juego, como un servidor y una base de datos subyacente. Por parte del resto de integrantes, necesitamos únicamente incluir dicha librería y comprobar el registro correcto de los eventos, realizado en varias iteraciones.

Una vez implementados los requisitos técnicos, pasamos a especificar el entorno del experimento inspirado en el realizado originalmente en el estudio de referencia. Redacté el esquema del protocolo a seguir, de cara a preparar el experimento físico previsto. Sin embargo, debido a la situación de COVID-19, se abandonó la realización de recogida de datos de forma presencial y pasamos a plantear nuevas posibilidades: distribuir los juegos de manera online. Incorporé cambios necesarios de plataforma en el juego Unpossible, ya que estaba pensado para dispositivos móviles.

A la par del desarrollo del experimento, realicé un estudio previo para la fase de análisis desde un enfoque estadístico. En concreto, el estudio fue sobre el análisis factorial confirmatorio, la técnica empleada originalmente para llegar a una conclusión sobre el factor de inteligencia subyacente. Para ello, empecé con la instalación de las herramientas necesarias como el lenguaje R y el entorno R Studio. Completé cursos online y lecturas de las bibliografías correspondientes, ya que no disponía de ningún conocimiento sobre análisis estadístico. Posteriormente, seguí los tutoriales recomendados por la investigadora Quiroga, que siempre nos ha asesorado durante el desarrollo de todo el trabajo. Para comprobar el entendimiento correcto de todo el proceso, reproduje con éxito el proceso de análisis realizado en el estudio de referencia, utilizando el mismo conjunto de datos.

Una vez terminado de llevarse a cabo el experimento online, me encargué de realizar el proceso del análisis de los datos recogidos, así como el preprocesamiento de los datos y el desarrollo de las métricas de manera previa. Cabe mencionar que el análisis confirmatorio no fue incluido al final en el trabajo, debido a las características de los datos conseguidos del experimento. Por esta razón, repetí el proceso de estudio explicado anteriormente para otro tipo de análisis factorial más adecuado, el EFA. De este modo, un nuevo programa SPSS de IBM y nuevas referencias entraron en proceso.

Los aspectos técnicos del segundo enfoque del análisis cuentan con la aplicación de algoritmos de aprendizaje automático, en concreto las redes neuronales y los *random forests*. Siguiendo la propuesta de los directores, todos los integrantes leímos un libro de referencia para entender las técnicas del aprendizaje profundo. Después de esta lectura, intenté realizar un análisis exhaustivo para el juego Unpossible, plasmando las explicaciones e interpretaciones de todos los resultados obtenidos en un borrador, que

pretendía ser la base de la sección correspondiente de la memoria. Tras el visto bueno de los directores, empecé la investigación de otras técnicas tradicionales de aprendizaje automático como alternativa, ya que los resultados obtenidos por las redes neuronales en conjuntos de prueba no eran prometedores. Si bien la idea original era que cada integrante realizara el análisis correspondiente a su juego, se vio afectado por modificaciones. Tras descubrir que el análisis de otros juegos estaba inconsistente o haber adoptado un enfoque erróneo, se decidió unificar los criterios y realizar un análisis más exhaustivo para ellos, siguiendo la misma metodología que el caso de Unpossible. Por lo tanto, me encargué de repetir este análisis para los dos juegos restantes, a la par de la redacción correspondiente de la memoria. Finalmente, incluí un análisis detallado sobre la aplicación de modelos de *random forests*.

En el desarrollo de la memoria, me corresponde la redacción completa del capítulo 6 (Análisis de datos) y la sección de la implementación de Unpossible en el capítulo 3. Por su parte, redactamos el resumen, el capítulo de introducción y el capítulo de conclusiones conjuntamente entre Pablo y yo. Adicionalmente, aportamos una reestructura y parte de reescritura del capítulo 5 (Desarrollo de experimento), tras recibir feedback de los directores.

Apéndice A

Protocolo de Experimento Presencial

Requisitos y selección de participantes

Intentaremos seleccionar tantos participantes como sea posible para la realización del experimento, si bien los resultados comenzarían a ser especialmente relevantes entre 50 y 100. Los participantes deben ser personas que saben manejar ordenadores y móviles/tablets.

En principio buscaremos participantes entre los alumnos de la facultad de informática y/o psicología, si no resultaría un grupo demasiado homogéneo para el experimento.

Entorno de evaluación y material

El experimento se realizará presencialmente en la facultad de informática. Estableceremos distintas sesiones distribuidas entre distintos días y horas para adaptarnos a las necesidades de los participantes (quizá haremos una encuesta previa para recoger las disponibilidades). Idealmente se deberá distribuir a los participantes en grupos que no tengan en principio contacto entre ellos para evitar intercambios de comunicación entre sujetos, o bien sincronizar las sesiones para que estas tengan lugar inmediatamente después una de otra entre grupos con contacto mutuo. Pedirles que apaguen los móviles y los dejen fuera de su vista.

En cada sesión, reuniremos a los participantes en un laboratorio o aula con:

- Un(os) moderador(es) (Alex, Pablo y Ming).
- Mesas para que puedan realizar pruebas escritas.
- Ordenadores/portátiles para el juego de Edge.
- Móviles y/o Tablets para los juegos de Unpossible y Blek.

Los materiales que hay que preparar para cada sesión son:

- La tabla prehecha de los IDs de usuario de cada participante, para recortar y repartir.
- Una hoja A4 con una tabla de dos columnas, la de IDs de usuario y la de Nicknames. Esta hoja es la que se pasará a todos los participantes para que rellenen con un nickname que deseen, en el caso

de que pierdan su ID o quieran consultar su resultado del experimento con facilidad. En principio el nickname pretende resolver el problema de que un usuario olvide su ID después del experimento, permitiendo que recupere sus datos por medio de un identificador más fácilmente recordable.

- Un consentimiento y autorización por participante.
- Un cuestionario de hábito de videojuegos por participante.
- Un test de inteligencia por participante.

Tareas a tener en cuenta a la hora del experimento:

- Comprobar disponibilidad de laboratorios/aulas.
- Comprobar disponibilidad de los dispositivos necesitados. En caso de usar ordenadores de laboratorios, ¿cómo haríamos con las cuentas?
- ¿Tenemos que preparar bolígrafos nosotros?
- ¿Cuántas fotocopias de todo material a papel que necesitamos?

Moderador

Antes de empezar la sesión

Antes de sentar a los participantes en sus sitios, el moderador debería comprobar que todos los dispositivos tienen acceso a internet para poder conectar con el servidor. Para el caso de usar móviles/tabletas, debería asegurarse de que las notificaciones estén silenciadas y que no se puedan recibir llamadas, para no interrumpir el experimento. Los móviles de los participantes también deberían estar apagados o en silencio. Debería comprobar que el papel de los IDs de usuario esté recortado para repartir.

El moderador deberá preguntar si algún participante necesita ir al baño ya que no podrá a lo largo del experimento.

En el caso del juego de ordenador, el moderador debería dejar el juego preparado en cada uno de los sitios mostrando la pantalla de inicio de sesión.

A continuación, pasarán los participantes y el moderador les repartirá sus IDs de usuario y la hoja de consentimiento. En principio los asientos no son preasignados. Los participantes pueden sentarse donde quieran mientras que mantengan un espacio adecuado entre ellos (un asiento libre en las aulas). Cuando estén preparados los participantes, éste dará un discurso de bienvenida, con un resumen de la finalidad de este experimento y explicará el proceso de la realización del mismo. Pasará la tabla de nicknames para que los participantes la rellene y aquellos que lo hayan hecho pueden empezar su evaluación.

Posible discurso del moderador (anexo), cuando se tenga toda la información necesaria.

Durante la sesión

El moderador debería asegurarse de que los participantes no puedan comunicarse entre sí ni recibir ninguna ayuda externa durante cualquier fase del experimento. Debería estar en todo momento pendiente del estado de los participantes. Si un participante:

- Tiene duda sobre el juego: Las dudas relacionadas con los juegos solo están permitidas durante la fase de tutorial de cada juego. Y se pueden aclarar.

- Pide una pausa: no se debería dejar interrumpir la sesión de evaluación a no ser por fuerza mayor.

Después de la sesión

Cuando un participante termina su evaluación con los videojuegos, el moderador le pasará el test de inteligencia. Al completarlo, recibirá el cuestionario de experiencia con videojuegos. El moderador comprobará que ha recogido los 3 documentos, que el participante no haya dejado ningún hueco por rellenar y da la sesión por completada. Posibilidad de dar los resultados de evaluación a los participantes, tanto de los juegos como de los test.

Fases del Experimento

a) Experimento con los videojuegos

Todos los juegos tienen un tutorial para que los participantes aprendan la mecánica del juego antes de empezar el experimento en sí. Durante el tutorial pueden preguntar todas las dudas que quieran.

- Blek: 5 niveles del 3 al 7, sin límite de tiempo
- Edge: 3 niveles del 1 al 3, sin límite de tiempo
- Unpossible: 2 minutos del nivel Daily Simplicity

Para realizar el experimento, los participantes disponen de:

- Blek: 10 minutos desde el nivel 8
- Edge: 12 minutos desde el nivel 10
- Unpossible: 5 minutos del nivel Simplicity

Nuestra propuesta es separar en grupos con ordenador, grupos con móviles, hacer las sesiones en tandas de dos grupos de 6 personas y distribuirnos en dos salas separadas, haciendo un intercambio de grupo cuando todos acaben. Dependiendo del número de dispositivos que tengamos, los participantes podrán empezar a la vez todos los juegos o podrán ir a su ritmo.

b) Test de inteligencia en papel

Tareas a tener en cuenta a la hora del experimento:

- Determinar la duración del test.
- Conseguir el documento.
- ¿Se hará con nosotros o con María Ángeles?
- Se dará el resultado al finalizar.

c) Cuestionario de hábito de videojuegos

Tan solo hay que modificar el cuestionario para incluir los datos demográficos del participante (edad, sexo, etc).

Discurso del moderador

El moderador ha de:

- Explicar todas las fases del experimento.
- Indicar la duración total del experimento y de cada fase.
- Indicar que tienen que levantar la mano en caso de duda/lo que sea.
- Informar de que no pueden comunicarse entre sí.
- Explicar los tiempos que disponen para el tutorial y el experimento en el juego concreto. En caso de no tener tiempo límite, especificar cuántos niveles.

Respecto a las instrucciones de los juegos particulares:

- **Blek**: Para jugar tienes que coger el Ipad de forma horizontal y usar el lápiz. Tu objetivo es explotar los círculos de colores sin tocar los negros. Para ello tendrás que dibujar una línea de cualquier forma y tamaño. Cuando termines, cobrará vida y avanzará repitiendo el movimiento que dibujaste. Si fallas puedes pulsar brevemente sobre la pantalla para reiniciar el nivel. Primero haremos el tutorial para que te habitúes a los movimientos. Después cambiaremos el nivel y tendrás 10 minutos para resolver todos los niveles que puedas. Puedes repetir cada nivel tantas veces como necesites.
- **Edge**: Este juego es de ordenador. Tu objetivo es conseguir el mayor número de prismas. Para ello, tendrás que moverte por toda la plataforma hasta llegar a la meta (dos cuadrados de colores concéntricos). Primero haremos el tutorial para que te habitúes a los comandos (WASD o flechas). Después cambiaremos el nivel y tendrás 12 minutos para resolver todos los niveles que puedas.
- **Unpossible**: Para jugar tienes que coger el Ipad de forma horizontal y poner tus pulgares en los bordes de la pantalla. Tu objetivo es aguantar el mayor tiempo posible sobre el tubo. Ten en cuenta que a lo largo del camino irás topándote con distintos obstáculos. Para esquivarlos tendrás que pulsar sobre la pantalla con el pulgar derecho (para girar el tubo hacia la derecha) o con el izquierdo (para girar el tubo hacia la izquierda). Primero haremos el tutorial para que te habitúes a los movimientos. Después cambiaremos el nivel y tendrás 5 minutos para recorrer la mayor distancia que puedas. si te chocas con un obstáculo, puedes repetir volver a empezar tantas veces como necesites.

Bibliografía

[1] M. B. Jones, W. P. Dunlap e I. M. Bilodeau, «Comparison of video game and conventional test performance», *Simulation & Games*, vol. 17, n.º 4, págs. 435-446, 1986.

[2] P. Rabbitt, N. Banerji y A. Szymanski, «Space Fortress as an IQ test? Predictions of learning and of practised performance in a complex interactive video-game», *Acta Psychologica*, vol. 71, n.º 1-3, págs. 243-257, 1989.

[3] K. Squire, «Video games in education», *Int. J. Intell. Games & Simulation*, vol. 2, n.º 1, págs. 49-62, 2003.

[4] R. Rosas, M. Nussbaum, P. Cumsille, V. Marianov, M. Correa, P. Flores, V. Grau, F. Lagos, X. López, V. López y col., «Beyond Nintendo: design and assessment of educational video games for first and second grade students», *Computers & Education*, vol. 40, n.º 1, págs. 71-94, 2003.

[5] D. Gagnon, «Videogames and spatial skills: An exploratory study», *ECTJ*, vol. 33, n.º 4, págs. 263-275, 1985.

[6] M. Quiroga, M. Herranz, M. Gómez-Abad, M. Kebir, J. Ruiz y R. Colom, «Video-games: Do they require general intelligence?», *Computers & Education*, vol. 53, n.º 2, págs. 414-418, 2009.

[7] M. Á. Quiroga, S. Escorial, F. J. Román, D. Morillo, A. Jarabo, J. Privado, M. Hernández, B. Gallego y R. Colom, «Can we reliably measure the general factor of intelligence (g) through commercial video games? Yes, we can!», *Intelligence*, vol. 53, págs. 1-7, 2015.

[8] M. Quiroga, A. Diaz, F. Román, J. Privado y R. Colom, «Intelligence and video games: Beyond "brain-games"», *Intelligence*, vol. 75, págs. 85-94, 2019.

[9] K. Martínez y R. Colom Marañón, *Artículo Imaging the Intelligence of Humans*, en. dirección: `https://www.researchgate.net/publication/341867483_Imaging_the_intelligence_of_humans` (visitado 13-06-2020).

[10] R. Colom Marañón, *Manual de psicología diferencial: métodos, modelos y aplicaciones*, Spanish. Madrid: Pirámide, 2018, OCLC: 1053625118, ISBN: 9788436839630.

[11] «Artículo Definición de Psicometría», en, dirección: `https://definicion.de/psicometria/` (visitado 13-06-2020).

[12] *Artículo Procesos Cognitivos*, en. dirección: `https://infotiti.com/2017/03/procesos-cognitivos/` (visitado 13-06-2020).

[13] *Artículo Modelos Cognitivos*, en. dirección: `http://www.psicologo-barcelona.cat/articulos/que-es-la-inteligencia/modelos-cognitivos/` (visitado 13-06-2020).

[14] K. S. McGrew y B. J. Wendling, «Cattell–Horn–Carroll cognitive-achievement relations: What we have learned from the past 20 years of research», *Psychology in the Schools*, vol. 47, n.º 7, págs. 651-675, 2010.

[15] M. Á. Quiroga y R. Colom Marañón, «26 Intelligence and Video Games», en, dirección: https://www.researchgate.net/publication/334225070_26_Intelligence_and_Video_Games (visitado 12-06-2020).

[16] «Curva de Dificultad en Videojuegos», en, dirección: https://es.ign.com/reportaje/92847/feature/la-curva-de-dificultad-en-videojuegos (visitado 21-06-2020).

[17] *Big Brain Academy*, en. dirección: https://es.wikipedia.org/wiki/Big_Brain_Academy (visitado 14-06-2020).

[18] C. Spearman, «"General Intelligence". Objectively Determined and Measured.», 1961.

[19] *Psicología Diferencial UCM*, en. dirección: https://www.studocu.com/es/document/universidad-complutense-madrid/psicologia-diferencial/apuntes/psicologia-diferencial-tema-3/2936456/view (visitado 14-06-2020).

[20] *Google Analytics*. dirección: https://analytics.google.com/analytics/web/ (visitado 11-06-2020).

[21] *Add Firebase to your Unity project*, en. dirección: https://firebase.google.com/docs/unity/setup (visitado 15-06-2020).

[22] *Services - Unity Analytics*, en. dirección: https://unity3d.com/unity/features/analytics (visitado 11-06-2020).

[23] C. S. Loh, Y. Sheng y D. Ifenthaler, eds., *Serious games analytics: methodologies for performance measurement, assessment, and improvement*, eng, ép. Advances in Game-Based Learning. Cham: Springer, 2015, OCLC: 927940958, ISBN: 9783319058337.

[24] L. Mellon, *Applying metrics driven development to MMO costs and risks*, English. 2009. dirección: http://maggotranch.com/MMO_Metrics.pdf.

[25] M. S. El-Nasr, *Game analytics: maximizing the value of player data*. New York: Springer, 2013, ISBN: 9781447147688.

[26] S. Metoyer, S. Miller, J. Mount y S. Westmoreland, «Examples From the Trenches: Improving Student Learning in the Sciences Using Team-Based Learning», en, *Journal of College Science Teaching*, vol. 043, n.º 05, 2014, ISSN: 0047-231X. DOI: 10.2505/4/jcst14_043_05_40. dirección: https://www.nsta.org/store/product_detail.aspx?id=10.2505/4/jcst14_043_05_40 (visitado 24-06-2020).

[27] *Blek*, en. dirección: http://www.blekgame.com/ (visitado 11-06-2020).

[28] *Rail Maze*, en. dirección: https://apps.apple.com/es/app/rail-maze-train-puzzler/id445853367 (visitado 12-06-2020).

[29] *Ski Jumping Pro*, en. dirección: https://apps.apple.com/us/app/ski-jumping-pro/id585599497 (visitado 12-06-2020).

[30] *Unpossible*, en. dirección: https://apps.apple.com/us/app/unpossible/id583577503 (visitado 12-06-2020).

[31] *Art Of Balance*, en. dirección: https://www.nintenderos.com/2014/10/analisis-art-of-balance-eshop-wii-u/ (visitado 12-06-2020).

[32] *Crazy Pool*, en. dirección: https://apps.apple.com/us/app/crazy-pool-3d-free/id391267734 (visitado 12-06-2020).

[33] *Edge*, en. dirección: https://en.wikipedia.org/wiki/Edge_(video_game) (visitado 12-06-2020).

[34] U. Technologies, *Unity Official Website*, en. dirección: https://unity.com/es (visitado 23-06-2020).

[35] gutierrpdev, *gutierrpdev/blek_ intelligence*, original-date: 2019-11-10T12:17:50Z, jun. de 2020. dirección: `https://github.com/gutierrpdev/blek_intelligence` (visitado 23-06-2020).

[36] R. Nystrom, *Game programming patterns*, eng. s.l.: genever benning, 2014, OCLC: 897177209, ISBN: 9780990582908.

[37] *Edge Game Repository*. dirección: `https://github.com/alex97ortega/Edge` (visitado 23-06-2020).

[38] M. Guo, *mingxguo/Unpossible*, original-date: 2019-09-28T11:21:55Z, mayo de 2020. dirección: `https://github.com/mingxguo/Unpossible` (visitado 26-06-2020).

[39] *Bezier Path Creator | Unity Asset Store*, en. dirección: `https://assetstore.unity.com/packages/tools/utilities/b-zier-path-creator-136082` (visitado 23-06-2020).

[40] *About Shader Graph | Shader Graph | 6.9.2*. dirección: `https://docs.unity3d.com/Packages/com.unity.shadergraph@6.9/manual/index.html` (visitado 23-06-2020).

[41] *About the Lightweight Render Pipeline | Package Manager UI website*. dirección: `https://docs.unity3d.com/Packages/com.unity.render-pipelines.lightweight@5.10/manual/index.html` (visitado 23-06-2020).

[42] U. Technologies, *Unity - Scripting API: Transform.RotateAround*, en. dirección: `https://docs.unity3d.com/ScriptReference/Transform.RotateAround.html` (visitado 23-06-2020).

[43] gutierrpdev, *gutierrpdev/server-event-manager*, original-date: 2020-03-09T16:53:08Z, mar. de 2020. dirección: `https://github.com/gutierrpdev/server-event-manager` (visitado 25-06-2020).

[44] *Managed MongoDB Hosting | Database-as-a-Service*, en-us. dirección: `https://www.mongodb.com/cloud/atlas` (visitado 11-06-2020).

[45] *Express - Node.js web application framework*, en. dirección: `https://expressjs.com/` (visitado 11-06-2020).

[46] *Mongoose v5.9.18: Schemas*. dirección: `https://mongoosejs.com/docs/guide.html` (visitado 11-06-2020).

[47] auth0.com, *JWT.IO - JSON Web Tokens Introduction*, en. dirección: `http://jwt.io/` (visitado 11-06-2020).

[48] gutierrpdev, *gutierrpdev/intelligence-server*, original-date: 2020-04-27T19:43:24Z, mayo de 2020. dirección: `https://github.com/gutierrpdev/intelligence-server` (visitado 26-06-2020).

[49] U. Technologies, *Unity - Manual: WebGL: Interacting with browser scripting*, en. dirección: `https://docs.unity3d.com/Manual/webgl-interactingwithbrowserscripting.html` (visitado 11-06-2020).

[50] *elraccoone/react-unity-webgl*, en. dirección: `https://github.com/elraccoone/react-unity-webgl` (visitado 11-06-2020).

[51] *React – A JavaScript library for building user interfaces*, en. dirección: `https://reactjs.org/` (visitado 11-06-2020).

[52] *Intelligence Assessment Games Web App*. dirección: `https://juegos-tfg.herokuapp.com/` (visitado 11-06-2020).

[53] gutierrpdev, *gutierrpdev/intelligence-client*, original-date: 2020-05-02T17:16:30Z, mayo de 2020. dirección: `https://github.com/gutierrpdev/intelligence-client` (visitado 26-06-2020).

[54] J. F. Hair, W. C. Black, B. J. Babin, R. E. Anderson, R. L. Tatham y col., *Multivariate data analysis (Vol. 6)*. Upper Saddle River, NJ: Pearson Prentice Hall, 2006.

[55] R. C. MacCallum, K. F. Widaman, K. J. Preacher y S. Hong, «Sample size in factor analysis: The role of model error», *Multivariate Behavioral Research*, vol. 36, n.º 4, págs. 611-637, 2001.

[56] T. Brown, *Confirmatory Factor Analysis for Applied Research, Second Edition*, ép. Methodology in the Social Sciences. Guilford Publications, 2015, ISBN: 9781462515363. dirección: https://books.google.es/books?id=ITL2BQAAQBAJ.

[57] M. Tavakol y R. Dennick, «Making sense of Cronbach's alpha», *International Journal of Medical Education*, vol. 2, págs. 53-55, jun. de 2011, ISSN: 2042-6372. DOI: 10.5116/ijme.4dfb.8dfd. dirección: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4205511/ (visitado 14-06-2020).

[58] J. M. Bland y D. G. Altman, «Statistics notes: Cronbach's alpha», *Bmj*, vol. 314, n.º 7080, pág. 572, 1997.

[59] N. Mohd Razali y B. Yap, «Power Comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling Tests», *J. Stat. Model. Analytics*, vol. 2, ene. de 2011.

[60] F. Chollet, *Deep Learning with Python*. Manning Publications Company, 2017, ISBN: 9781617294433. dirección: https://books.google.es/books?id=Yo3CAQAACAAJ.

[61] B. C. Csáji y col., «Approximation with artificial neural networks», *Faculty of Sciences, Etvs Lornd University, Hungary*, vol. 24, n.º 48, pág. 7, 2001.

[62] G. Cybenko, «Approximations by superpositions of a sigmoidal function», *Mathematics of Control, Signals and Systems*, vol. 2, págs. 183-192, 1989.

[63] K. Team, *Keras documentation: Keras API reference*, en. dirección: https://keras.io/api/ (visitado 14-06-2020).

[64] I. Goodfellow, Y. Bengio y A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[65] P. Ramachandran, B. Zoph y Q. V. Le, «Searching for Activation Functions», 2017. arXiv: 1710.05941 [cs.NE].

[66] J. Desmond, *Machine Learning Benchmarks and AI Self-Driving Cars*, en-US, jun. de 2018. dirección: https://www.aitrends.com/ai-insider/machine-learning-benchmarks-and-ai-self-driving-cars/ (visitado 25-06-2020).

[67] I. S. Markham y T. R. Rakes, «The effect of sample size and variability of data on the comparative performance of artificial neural networks and regression», *Computers & Operations Research*, vol. 25, n.º 4, págs. 251-263, 1998, ISSN: 0305-0548. DOI: https://doi.org/10.1016/S0305-0548(97)00074-9. dirección: http://www.sciencedirect.com/science/article/pii/S0305054897000749.

[68] A. Alwosheel, S. [Cranenburgh] y C. G. Chorus, «Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis», *Journal of Choice Modelling*, vol. 28, págs. 167-182, 2018, ISSN: 1755-5345. DOI: https://doi.org/10.1016/j.jocm.2018.07.002. dirección: http://www.sciencedirect.com/science/article/pii/S1755534518300058.

[69] K. Pasupa y W. Sunhem, «A comparison between shallow and deep architecture classifiers on small dataset», págs. 1-6, 2016.

[70] S. Feng, H. Zhou y H. Dong, «Using deep neural network with small dataset to predict material defects», *Materials & Design*, vol. 162, págs. 300-310, 2019, ISSN: 0264-1275. DOI: https://doi.org/10.1016/j.matdes.2018.11.060. dirección: http://www.sciencedirect.com/science/article/pii/S0264127518308682.

[71] A. Pasini, «Artificial neural networks for small dataset analysis», *Journal of Thoracic Disease*, vol. 7, n.º 5, págs. 953-960, mayo de 2015, ISSN: 2072-1439. DOI: 10.3978/j.issn.2072-1439.2015.04.61. dirección: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4454870/ (visitado 11-06-2020).

[72] L. Breiman, «Random forests», *Machine learning*, vol. 45, n.º 1, págs. 5-32, 2001.

[73] J. R. Quinlan, «Induction of decision trees», *Machine learning*, vol. 1, n.º 1, págs. 81-106, 1986.

[74] L. Breiman, «Bagging predictors», *Machine learning*, vol. 24, n.º 2, págs. 123-140, 1996.

[75] C. Strobl, J. Malley y G. Tutz, «An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests.», *Psychological methods*, vol. 14, n.º 4, pág. 323, 2009.

[76] N. Horning, «Introduction to decision trees and random forests», *Am. Mus. Nat. Hist*, vol. 2, págs. 1-27, 2013.

[77] *API Reference scikit-learn 0.23.1 documentation*. dirección: https://scikit-learn.org/stable/modules/classes.html# (visitado 13-06-2020).