Deformación de modelos 2D casi rígidos

Proyecto de Sistemas Informáticos 2013-2014

Autores:

José Alberto Gayo Hidalgo

Enrique Alberto Gómez Villalobos

Víctor de Nicolás Noblejas

Profesor director:

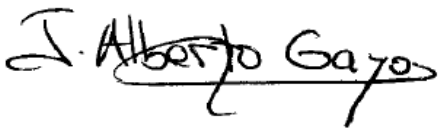
Pedro Jesús Martín de la Calle

Facultad de Informática

Universidad Complutense de Madrid

AUTORIZACIÓN DE DIFUSIÓN Y UTILIZACIÓN

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.



José Alberto Gayo Hidalgo

Enrique Alberto Gómez Villalobos

Víctor de Nicolás Noblejas

AGRADECIMIENTOS

En primer lugar queremos agradecer a nuestros amigos y familiares por el apoyo que nos han brindado a lo largo de toda la carrera.

También agradecer a Pedro J. Martín de la Calle el haber confiado en nosotros para realizar este proyecto de Sistemas Informáticos y por toda la ayuda que nos ha dado durante este año.

Para finalizar, agradecer a todo aquel que revise este proyecto y desear que le sea de ayuda.

ÍNDICE

ÍNDICE DE ILUSTRACIONES

RESUMEN

Deformed Invaders es un juego de plataformas para tablets con sistema operativo Android. Su principal característica es la posibilidad de diseñar personajes usando geometrías deformables en dos dimensiones. Estas geometrías están constituidas por una malla generada a partir del contorno de un polígono simple. A estas geometrías se les pueden aplicar ciertas trasformaciones para deformarlas sin perder su aspecto original. Estas trasformaciones se describen en el algoritmo de deformación *As-Rigid-As-Possible* de Takeo Igarashi.

Palabras clave:

Geometrías, deformación, B-Spline, Delaunay, malla, Android, Tablet.

ABSTRACT

Deformed Invaders is a platform game for Android tablets. Its main feature is the possibility of designing characters, using deformable geometries in two dimensions. These geometries are formed by a mesh generated from the hull of a single polygon. Applying some transformations, these geometries get deformed trying to maintain their original appearance. These transformations are described in the algorithm of *As-Rigid-As-Possible* deformation by Takeo Igarashi.

Keywords:

Geometries, deformation, B-Spline, Delaunay, mesh, Android, tablet.

1. INTRODUCCIÓN

En la actualidad el desarrollo de aplicaciones para dispositivos móviles, es uno de los sectores de la informática con mayor crecimiento. Abarca categorías tan dispares como el entretenimiento o la productividad profesional. Además las tecnologías que incorporan, nos brindan nuevas posibilidades, como por ejemplo la de diseñar interfaces de usuario más atractivas e intuitivas gracias a las pantallas táctiles. Por estas razones, hemos optado por este tipo de dispositivos para desarrollar nuestro proyecto.

Nuestra aplicación implementa el algoritmo *As-Rigid-As-Possible Shape Manipulation* de Takeo Igarashi, que consiste en la deformación de geometrías en dos dimensiones pero intentando mantener su aspecto original. Estas geometrías están constituidas por una malla de triángulos, calculada utilizando el algoritmo de Delanuay sobre el contorno especificado por un polígono simple.

Con el objetivo de buscar un uso práctico a estas geometrías, hemos desarrollado un juego de plataformas en dos dimensiones. Los personajes son diseñados por el jugador, quien además define sus animaciones usando el algoritmo de deformación citado anteriormente.

En la memoria se explica en detalle, las diferentes fases para la construcción de las geometrías deformables, así como el diseño y la implementación de la aplicación. También, se describe el funcionamiento y las características que tiene la aplicación. Es decir, cómo utilizar las herramientas de las que se dispone para la creación de un personaje, y cómo se desarrolla el juego.

Por último, se hace una breve reflexión, de los problemas encontrados durante el trascurso de este proyecto, y sobre ideas acerca de otros posibles usos para estas geometrías deformables.

2. GEOMETRÍAS DEFORMABLES EN 2D

Una geometría deformable está constituida fundamentalmente por una malla de polígonos regulares. Esta malla se genera a partir de un conjunto de puntos ordenados que forman su contorno. Los puntos se obtienen mediante la interacción táctil del usuario con la aplicación.

2.1 Suavizado de contornos

Para poder garantizar la fluidez de la deformación es necesario que la malla tenga un número reducido, pero suficiente, de triángulos. Nuestro objetivo es conseguir que el número de vértices sea constante en el contorno de la figura. La constante concreta se ha elegido al realizar sucesivas pruebas sobre el resultado de la deformación.

Existe la posibilidad de que los puntos ofrecidos por el usuario sean muy numerosos o por lo contrario, que sean escasos. Para solucionar estos problemas usamos el trazado del usuario de forma orientativa, y lo aproximamos con una función b-spline cúbica cerrada.

Se trata de una curva *suave* que ofrece interesantes propiedades, como son el control local de la curva a su paso por los puntos de control (los puntos elegidos por el usuario), la rapidez con que puede construirse, junto con la estabilidad numérica de los métodos que se precisan, y que la curva preserva las transformaciones afines habituales (traslaciones, escalaciones y rotaciones).

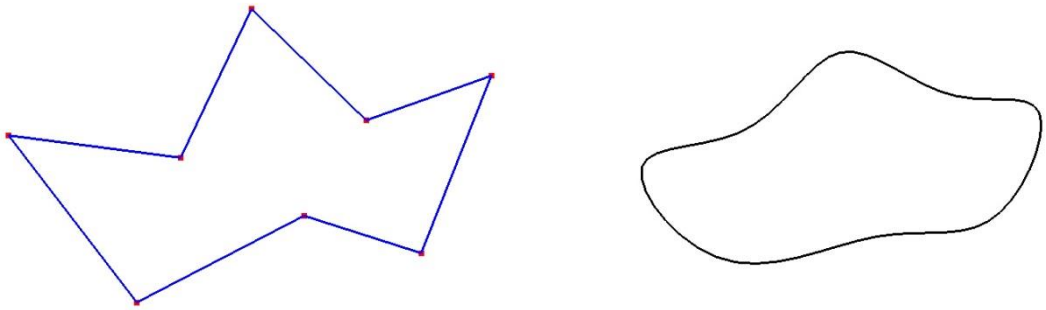


Ilustración 1 - Curva b-spline (derecha) generada a partir de un conjunto pequeño de puntos (izquierda)

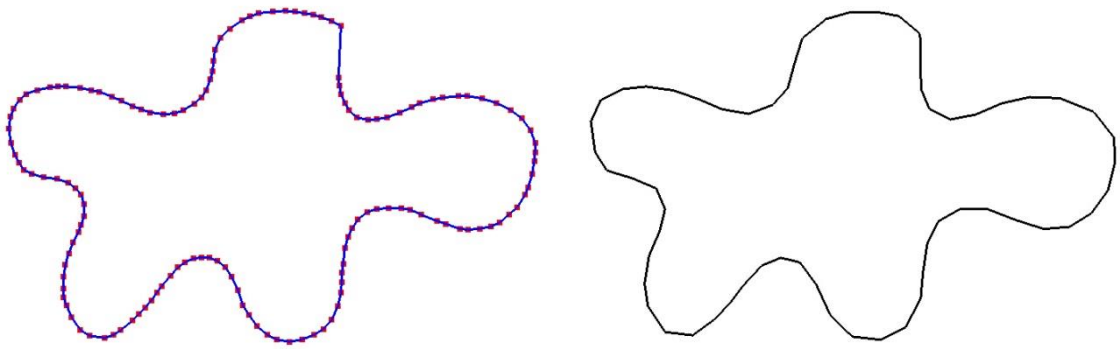


Ilustración 2 – Curva b-spline (derecha) generada a partir de un conjunto grande de puntos (izquierda)

Una b-spline cúbica se define en [1] a partir de una función definida por casos, mediante polinomios de grado 3:

$$N_{0,3}(t) = \begin{cases} u(1-t), & 0 \le t < 1 \\ v(2-t), & 1 \le t < 2 \\ v(t-2), & 2 \le t < 3 \\ u(t-3), & 3 \le t < 4 \\ 0, & en\ otro\ caso \end{cases}$$

donde:

$$u(t) = \frac{1}{6}(1-t)^3, \qquad v(t) = \frac{1}{6}(3t^3 - 6t^2 + 4)$$

Para generar puntos en la curva, se aplica una combinación afín de los puntos de control $P_0, P_1, P_2, \cdots, P_n$:

$$C(t) = \sum_{i=0}^{n} N_{0,3}(t-i)\, P_i$$

usando valores $t \in [3,\ n+1]$.

Para tomar k+1 valores de la b-spline basta con dividir el intervalo [3, n+1] en muestras uniformes de la siguiente forma:

$$t_i = 3 + i\frac{n-2}{k}, \qquad para\ i = 0, 1, \ldots, k$$

Éste método nos permite en consecuencia tanto reducir los puntos del contorno elegidos por el usuario, como crear nuevos puntos de control interpolados para acercarnos al número constante explicado anteriormente.

2.2 Mallado del contorno

Con el contorno de la figura como base, se aplica un algoritmo de triangulación conocido como *triangulación de Delaunay restringida* [2]. Este algoritmo es una generalización del algoritmo de triangulación clásica de Delaunay.

La *triangulación clásica de Delaunay* [3] toma un conjunto de puntos en el plano para los que construye una malla triangular convexa, es decir los triángulos fronterizos describen un polígono convexo. Estos triángulos se caracterizan por maximizar sus ángulos mínimos, esto quiere decir que los ángulos de los triángulos son lo más grandes posibles.

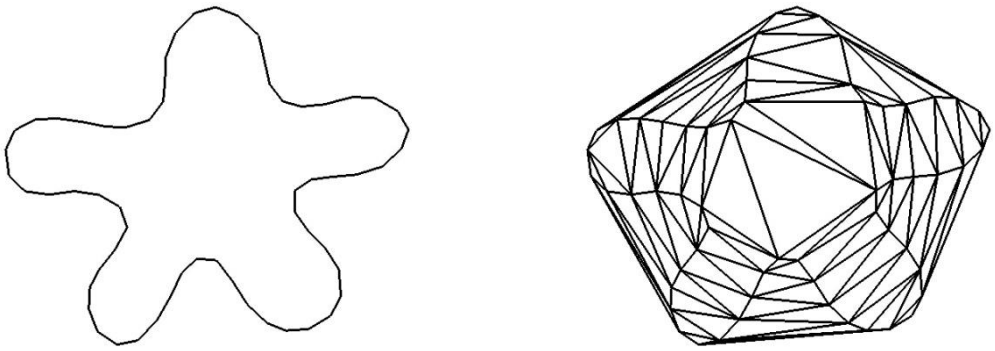


Ilustración 3 - Triangulación Delaunay restringida

La triangulación de Delaunay restringida fuerza que determinados segmentos formen parte de la malla resultante. Siendo estos segmentos, en nuestro caso, el contorno de partida.

A continuación se procede a generar una malla más uniforme añadiendo nuevos puntos a la malla producida en la fase anterior. Estos nuevos puntos serán los baricentros de cada uno de los triángulos incluidos en la malla.

El algoritmo de mallado se muestra a continuación:

```
Malla crearMalla(contorno)

      área original = área de contorno
      vértices = copia de contorno
      triángulos = triangulación Delaunay restringida de vértices

      mientras haya cambios en vértices
            para todos los triángulos
                  si área triángulo > porcentaje de área original
                        añadir baricentro en vértices
                  fsi
            fpara

            triángulos = triangulación Delaunay restringida de
            vértices
      fmientras

      devolver vértices y triángulos
```

Este proceso de triangulación basado en la generación de nuevos puntos, se iterará mientras el área de los triángulos generados sea superior a un porcentaje del área del contorno original. Estas técnicas nos permiten controlar el número de triángulos generados, lo que supondrá mejorar la estabilidad de nuestras mallas frente a la posterior fase de deformación.

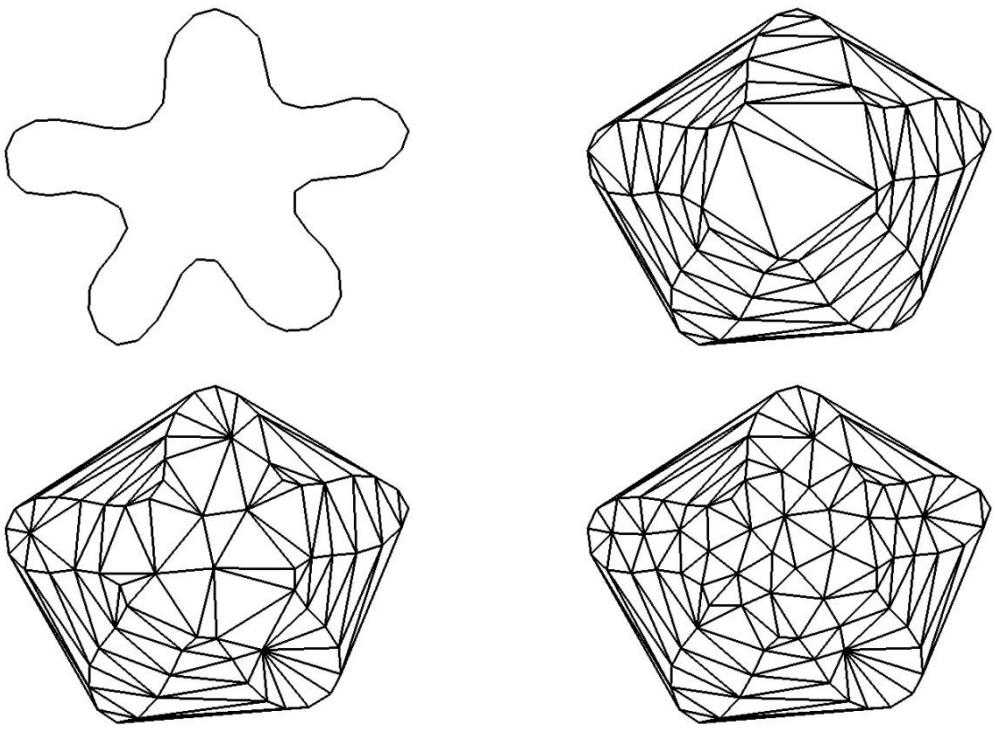


Ilustración 4 – Ejemplo de construcción de una malla mediante tres iteraciones de triangulaciones Delaunay restringidas

Como se ha explicado anteriormente, el algoritmo de triangulación de Delaunay genera una triangulación convexa. Esto quiere decir, que es posible que se añadan triángulos a la malla que no forman parte del interior del contorno de la figura creada por el usuario.

Para solventar este inconveniente se procede a un recorte de dichos triángulos sobrantes. Todos los triángulos cuyo baricentro esté fuera del contorno original serán eliminados de la malla.

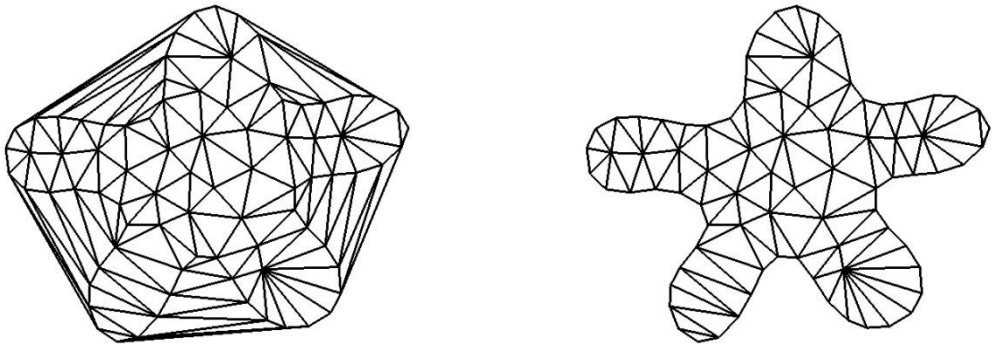


Ilustración 5 – Ejemplo de recorte de los triángulos externos al contorno original

2.3 Deformación de la Malla

Una vez generada una malla que recubra el contorno dado, podemos proceder a deformarla. Para ello usamos el sistema *As-Rigid-As-Posible Shape Manipulation* [4] presentado por Takeo Igarashi, Tomer Moscovich y John F. Hughes.

Este sistema permite la manipulación de una malla en 2D mediante el uso de *handles*. Cada handle (del conjunto C) está asociado a un vértice de la malla, y aquellos que no sean movidos interactivamente preservarán su posición tras el proceso de deformación.

Para que la malla sea deformada, bastará con modificar la posición de uno, o varios handles. En líneas generales, todos los vértices (del conjunto V) de la malla recalcularán su nueva posición para satisfacer las restricciones de los handles, pero intentando que la estructura global intente mantener la forma original. Los nuevos vértices, que forman el conjunto V', deberán minimizar la

distorsión que sufren las aristas de la malla (conjunto E), que se formaliza mediante un problema de mínimos cuadrados. Se trata de encontrar el conjunto de vértices nuevos que minimiza la siguiente expresión:

$$\arg\min_{v' \in V} \left\{ \sum_{\{i,j\} \in E} \left\| (v'_j - v'_i) - (v_j - v_i) \right\|^2 + w \sum_{i \in C} \| (v'_i - C_i) \|^2 \right\} \qquad \text{Ecuación 1}$$

El primer sumando $\sum_{\{i,j\} \in E} \left\| (v'_j - v'_i) - (v_j - v_i) \right\|^2$ representa la distorsión de las aristas. El segundo sumando $w \sum_{i \in C} \| (v'_i - C_i) \|^2$ obliga a los handles a mantener su posición original usando un factor de peso elevado w con valor 1000.

En líneas generales, la solución de este tipo de problemas se conseguiría planteándolos de forma matricial y resolverlos para cada coordenada por separado.

$$\arg\min_{v' \in V} \| Av' - B \|^2 \qquad \text{Ecuación 2}$$

Por ejemplo, para la coordenada x, la matriz A representaría la estructura que forman las aristas y los handles dentro de las malla, mientras que en la matriz B recoge la información de las dimensiones de las aristas y las coordenadas de los handles.

$$\begin{bmatrix} 1 & 0 & -1 & 0 & \\ 0 & -1 & 1 & 0 & \dots \\ & & \vdots & & \\ 0 & 0 & w & 0 & \\ w & 0 & 0 & 0 & \dots \\ & & \vdots & & \end{bmatrix} \begin{bmatrix} v'_{0x} \\ v'_{1x} \\ \vdots \end{bmatrix} - \begin{bmatrix} e_{0x} \\ e_{1x} \\ \vdots \\ wc_{0x} \\ wc_{1x} \\ \vdots \end{bmatrix} = AV'_x - B$$

Las dimensiones de las matrices serán:

A : (n° Aristas + n° Handles) x n° Vértices
B : (n° Aristas + n° Handles) x 1

Para construir la mitad superior de la matriz A se introduce 1 en la columna que corresponda al vértice de inicio de la arista, y -1 en el vértice que corresponda al final. Para la mitad inferior se introduce la constante w en la posición de los vértices en los que estén asociados.

En el caso de la mitad superior de la matriz B se introducen las distancias de las aristas expresadas así:

$$e_{kx} = v_{jx} - v_{ix}$$

$$e_{ky} = v_{jy} - v_{iy}$$

donde el índice i representa el inicio de la arista y el j representa el final. Para la mitad inferior se introducen las coordenadas de los handles multiplicadas por el factor w.

La solución de este tipo de problemas de minimización, se conseguiría entonces resolviendo el siguiente sistema lineal.

$$A^t A v' = A^t B \qquad \text{Ecuación 3}$$

Pero la Ecuación 1, no soporta adecuadamente las rotaciones. En efecto, si los vértices de V rotaran simultáneamente, la geometría global se mantendría pero los sumandos de la izquierda en la Ecuación 1no se anularían (para cada arista la diferencia entre la arista rotada y la original no es cero). Por ello, en [5] se propone un sistema basado en dos fases. Según indican los autores, en una primera fase se tienen en cuanta las rotaciones, traslaciones y escalaciones uniformes, y en una segunda etapa, se ajusta la escala.

2.3.1 Primera fase: Ajuste de traslación y rotación

Supongamos que las aristas originales están rotadas antes de aplicar la Ecuación 1, cada una con su rotación T_{ij}. Esto es, tratamos de solucionar

$$\arg\min_{v' \in V} \left\{ \sum_{\{i,j\} \in E} \left\| (v'_j - v'_i) - T_{ij}(v_j - v_i) \right\|^2 + w \sum_{i \in C} \| (v'_i - C_i) \|^2 \right\} \qquad \text{Ecuación 4}$$

El problema es que las rotaciones deben considerarse nuevas incógnitas que dificultan la solución del problema. En [5] se procede de forma *implícita*, se despeja antes T_{ij} en función de V' primero, y luego se revuelve el problema resultante.

Centrémonos en una arista, la que va de v_i a v_j, y llamémosla:

$$e_k = \begin{pmatrix} v_{jx} - v_{ix} \\ v_{jy} - v_{iy} \end{pmatrix}$$

Se definen sus dos vértices vecinos (l, r), que serán los vértices restantes que forman los dos triángulos que tienen en común la arista e_k. En el caso de las aristas fronterizas, solo existirá un vértice vecino. Por consiguiente todos los datos referentes a este vértice ausente no serán tenidos en cuenta en los cálculos.

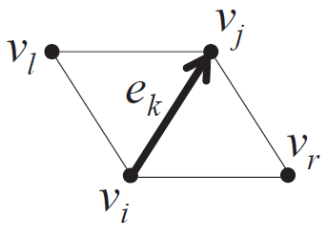


Ilustración 6 – Representación de los cuatro vértices asociados a una arista.

Sea T_k la trasformación afín definida por la matriz

$$T_k = \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$$

se trata de una rotación más una escalación uniforme. Si suponemos que V' son vértices conocidos, podemos calcular T_k resolviendo la siguiente ecuación:

$$T_k = \arg\min_{T_k} \sum_{v \in N(e_k)} \| T_k v - v' \|^2, \qquad N(e_k) = \{v_i, v_j, v_l, v_r\}$$

Derivando la transformación T_k:

$$\{c_k, s_k\} = \arg\min_{c_k, s_k} \sum_{v \in N(e_k)} \left\| \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} - \begin{bmatrix} v'_x \\ v'_y \end{bmatrix} \right\|^2$$

$$= \arg\min_{c_k, s_k} \sum_{v \in N(e_k)} \left\| \begin{bmatrix} v_x & v_y \\ v_y & -v_x \end{bmatrix} \begin{bmatrix} c_k \\ s_k \end{bmatrix} - \begin{bmatrix} v'_x \\ v'_y \end{bmatrix} \right\|^2$$

$$= \arg\min_{c_k, s_k} \left\| \begin{bmatrix} v_{ix} & v_{iy} \\ v_{iy} & -v_{ix} \\ v_{jx} & v_{jy} \\ v_{jy} & -v_{jx} \\ v_{lx} & v_{ly} \\ v_{ly} & -v_{lx} \\ v_{rx} & v_{ry} \\ v_{ry} & -v_{rx} \end{bmatrix} \begin{bmatrix} c_k \\ s_k \end{bmatrix} - \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ v'_{jy} \\ v'_{lx} \\ v'_{ly} \\ v'_{rx} \\ v'_{ry} \end{bmatrix} \right\|^2$$

La primera matriz de la ecuación anterior la llamaremos G_k y será utilizada posteriormente en varios cálculos. Esta matriz se calcula para cada arista de la malla.

$$G_k = \begin{bmatrix} v_{ix} & v_{iy} \\ v_{iy} & -v_{ix} \\ v_{jx} & v_{jy} \\ v_{jy} & -v_{jx} \\ v_{lx} & v_{ly} \\ v_{ly} & -v_{lx} \\ v_{rx} & v_{ry} \\ v_{ry} & -v_{rx} \end{bmatrix}$$

El cálculo de la transformación anterior, puede resolverse como un problema de mínimos cuadrados de la siguiente forma:

$$\begin{bmatrix} c_k \\ s_k \end{bmatrix} = (G_k^t G_k)^{-1} G_k^t \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ v'_{jy} \\ v'_{lx} \\ v'_{ly} \\ v'_{rx} \\ v'_{ry} \end{bmatrix}$$

Ya sabemos cómo calcular T_k en función de V'. Ahora sustituyamos T_k en el término $(v'_j - v'_i) - T_k(v_j - v_i)$ que aparece en la Ecuación 4 ($T_k = T_{ij}$), obtenemos:

$$(v'_j - v'_i) - T_k(v_j - v_i) = (v'_j - v'_i) - \begin{bmatrix} e_{kx} & e_{ky} \\ e_{ky} & -e_{kx} \end{bmatrix} \begin{bmatrix} c_k \\ s_k \end{bmatrix}$$

$$= (v'_j - v'_i) - \begin{bmatrix} e_{kx} & e_{ky} \\ e_{ky} & -e_{kx} \end{bmatrix} (G_k^t G_k)^{-1} G_k^t \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ v'_{jy} \\ v'_{lx} \\ v'_{ly} \\ v'_{rx} \\ v'_{ry} \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} e_{kx} & e_{ky} \\ e_{ky} & -e_{kx} \end{bmatrix} (G_k^t G_k)^{-1} G_k^t \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ v'_{jy} \\ v'_{lx} \\ v'_{ly} \\ v'_{rx} \\ v'_{ry} \end{bmatrix}$$

$$= \begin{bmatrix} h_{k00} & h_{k10} & h_{k20} & h_{k30} & h_{k40} & h_{k50} & h_{k60} & h_{k70} \\ h_{k01} & h_{k11} & h_{k21} & h_{k31} & h_{k41} & h_{k51} & h_{k61} & h_{k71} \end{bmatrix} \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ v'_{jy} \\ v'_{lx} \\ v'_{ly} \\ v'_{rx} \\ v'_{ry} \end{bmatrix}$$

Esta última matriz de coeficientes H_k, será utilizada para calcular los valores de los vértices modificados. Deberá ser calculada para cada arista de la malla.

$$H_k = \begin{bmatrix} h_{k00} & h_{k10} & h_{k20} & h_{k30} & h_{k40} & h_{k50} & h_{k60} & h_{k70} \\ h_{k01} & h_{k11} & h_{k21} & h_{k31} & h_{k41} & h_{k51} & h_{k61} & h_{k71} \end{bmatrix} =$$

$$= \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} e_{kx} & e_{ky} \\ e_{ky} & -e_{kx} \end{bmatrix} (G_k^t G_k)^{-1} G_k^t$$

Cada arista da lugar a una matriz H_k. Juntándolas todas en una única matriz A_1, y añadiendo la parte de los handles, la Ecuación 4 puede expresarse como

$$\arg\min_{v' \in V} \|A_1 v' - B_1\|^2 \qquad \text{Ecuación 5}$$

donde A_1 y B_1 se muestran en la siguiente expresión:

$$\begin{bmatrix} h_{020} & h_{030} & 0 & 0 & h_{000} & h_{010} & h_{060} & h_{070} & 0 & 0 & h_{040} & h_{050} \\ h_{021} & h_{031} & 0 & 0 & h_{001} & h_{011} & h_{061} & h_{071} & 0 & 0 & h_{041} & h_{051} \\ 0 & 0 & h_{100} & h_{110} & h_{120} & h_{130} & 0 & 0 & h_{140} & h_{150} & h_{160} & h_{170} \\ 0 & 0 & h_{101} & h_{111} & h_{121} & h_{131} & 0 & 0 & h_{141} & h_{151} & h_{161} & h_{171} \\ & & & & & & \vdots & & & & & \\ 0 & 0 & 0 & 0 & w & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & w & 0 & 0 & 0 & 0 & 0 & 0 \\ w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & w & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & & & & \vdots & & & & & \end{bmatrix} \dots \begin{bmatrix} v'_{0x} \\ v'_{0y} \\ v'_{1x} \\ v'_{1y} \\ \vdots \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ wc_{0x} \\ wc_{0y} \\ wc_{1x} \\ wc_{1y} \\ \vdots \end{bmatrix}$$

$$= A_1 V' - B_1$$

Las dimensiones de las matrices serán:

A_1 : (2 * n° Aristas + 2 * n° Handles) x 2 * n° Vértices
B_1 : (2 * n° Aristas + 2 * n° Handles) x 1

Para construir la matriz A, se introducen los 2x8 coeficientes h en la posición de los vértices y vértices vecinos de cada arista. Las siguientes se ocupan de los handles, cada uno requiere dos filas consecutivas que exhiben un único valor (w) distinto de 0.

Para la matriz B solo se introducen las coordenadas x e y consecutivas de los handles multiplicadas por el factor de peso w.

Finalmente, la Ecuación 5 puede resolverse a través del siguiente sistema lineal:

$$A_1^t A_1 v' = A_1^t B_1 \qquad \text{Ecuación 6}$$

2.3.2 Segunda fase: Ajuste de escalación

Tras el primer ajuste obtenemos unos valores aproximados, pero que tienen un problema de escalación cuando un handle es movido a posiciones distantes de su posición original.

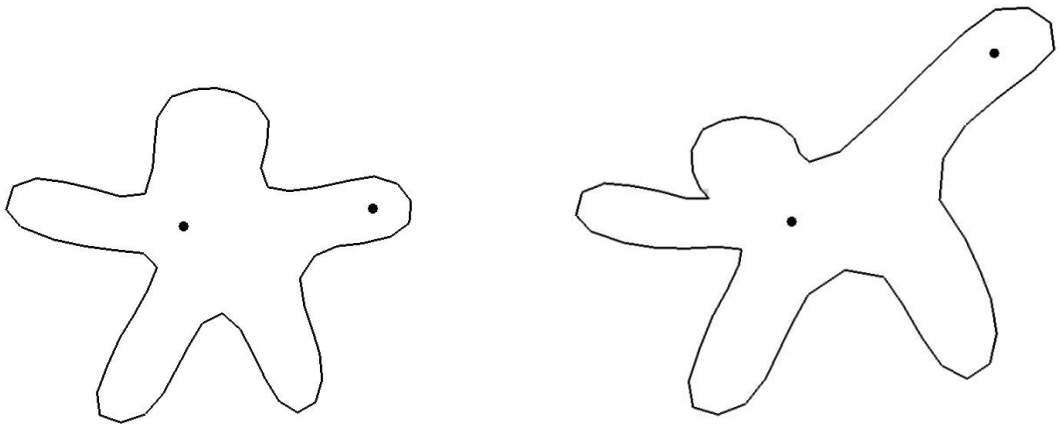


Ilustración 7 – Ejemplo de deformación de una malla sin ajuste de escala

Por ello en esta etapa se ajusta el cálculo de la transformación T_k para evitar el problema de las escalas. Básicamente, se trata de calcular T_k usando los valores de V' calculados en la etapa anterior.

$$T'_k = \frac{1}{c_k^2 + s_k^2}\begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}, \qquad \begin{bmatrix} c_k \\ s_k \end{bmatrix} = (G_k^t G_k)^{-1} G_k^t \begin{bmatrix} v'_{ix} \\ v'_{iy} \\ v'_{jx} \\ v'_{jy} \\ v'_{lx} \\ v'_{ly} \\ v'_{rx} \\ v'_{ry} \end{bmatrix}$$

Con estos valores de T'_k volvemos a plantear la Ecuación 4, usamos V'' para denotar los vértices que resultan en esta etapa:

$$\arg\min_{v''\in V}\left\{\sum_{\{i,j\}\in E}\left\|\left(v''_j - v''_i\right) - T'_{ij}(v_j - v_i)\right\|^2 + w\sum_{i\in C}\|(v''_i - C_i)\|^2\right\} \quad \text{Ecuación 7}$$

Usando notación matricial, se trata de resolver para cada coordenada

$$\arg\min_{v''\in V}\|A_2 v'' - B_2\|^2 \quad \text{Ecuación 8}$$

donde A_2 y B_2 se muestran para la coordenada x en la siguiente expresión:

$$\begin{bmatrix} 1 & 0 & -1 & 0 & \\ 0 & -1 & 1 & 0 & \dots \\ & & \vdots & & \\ 0 & 0 & w & 0 & \\ w & 0 & 0 & 0 & \dots \\ & & \vdots & & \end{bmatrix}\begin{bmatrix} v''_{0x} \\ v''_{1x} \\ \vdots \end{bmatrix} - \begin{bmatrix} T'_0 e_0|_x \\ T'_1 e_1|_x \\ \vdots \\ wc_{0x} \\ wc_{1x} \\ \vdots \end{bmatrix} = A_2 V''_x - B_2$$

Siendo $T'_k e_k|_x = c_k e_x + s_k e_y$.

Las dimensiones de las matrices serán:

A_2 : (n° Aristas + n° Handles) x n° Vértices
B_2 : (n° Aristas + n° Handles) x 1

Para construir la matriz A_2, se introduce 1 en la posición del vértice que corresponda al inicio de la arista, y -1 en el vértice que corresponda al final. También se introduce la constante w en los vértices que tengan un handle.

Para la matriz B_2 se calcula el producto matricial de la transformación y la dimensión de la arista para la mitad superior. Para la mitad inferior se introducen las coordenadas de los handles multiplicadas por el factor de peso w.

Para las coordenadas y de los vértices de la malla, las matrices A_2 y B_2 se calculan de la misma forma. El problema puede resolverse a través del sistema lineal

$$A_2^t A_2 v'' = A_2^t B_2$$ Ecuación 9

para obtener por separado las coordenadas x e y de los vértices resultantes. Estos valores permiten obtener un escalado correcto, como puede comprobarse en la siguiente ilustración.

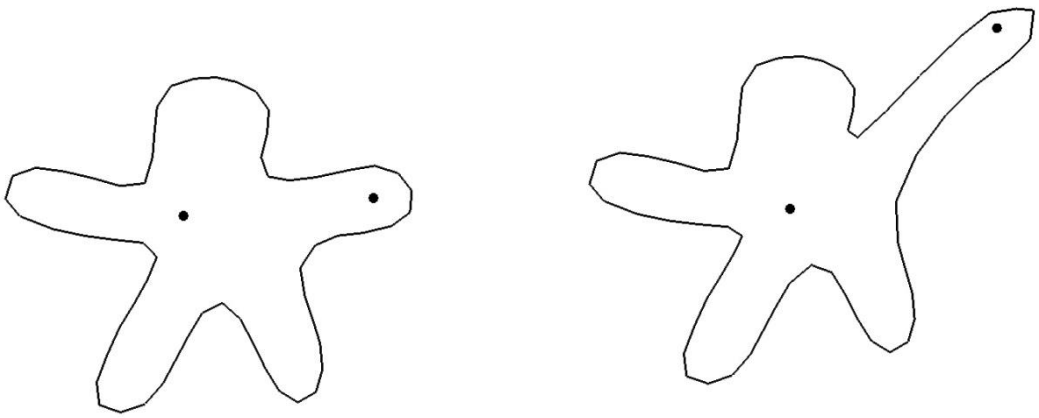


Ilustración 8 – Ejemplo de deformación de una malla con ajuste de escala

2.4 Representación baricéntrica de handles

Una de las mejoras que se proponen en el paper de *As-Rigid-As-Posible* [5] es la posibilidad de poder colocar los handles en posiciones arbitrarias dentro de la malla. Esto modifica la asociación original de los handles con vértices de la malla, por una representación basada en *coordenadas baricéntricas*.

Dado un punto $P = (x, y)$, dentro del triángulo $\widehat{ABC}$, se definen las coordenadas baricéntricas de P como:

$$alfa = \frac{(\mathrm{bY} - \mathrm{cY}) * (\mathrm{x} - \mathrm{cX}) + (\mathrm{cX} - \mathrm{bX}) * (\mathrm{y} - \mathrm{cY})}{(\mathrm{bY} - \mathrm{cY}) * (\mathrm{aX} - \mathrm{cX}) + (\mathrm{cX} - \mathrm{bX}) * (\mathrm{aY} - \mathrm{cY})}$$

$$beta = \frac{(\mathrm{cY} - \mathrm{aY}) * (\mathrm{x} - \mathrm{cX}) + (\mathrm{aX} - \mathrm{cX}) * (\mathrm{y} - \mathrm{cY})}{(\mathrm{bY} - \mathrm{cY}) * (\mathrm{aX} - \mathrm{cX}) + (\mathrm{cX} - \mathrm{bX}) * (\mathrm{aY} - \mathrm{cY})}$$

$$gamma = 1 - alfa - beta$$

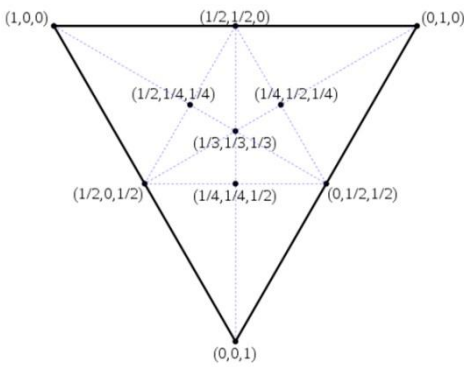


Ilustración 9 – Ejemplos de coordenadas baricéntricas en un triángulo equilátero

Estas coordenadas baricéntricas resultan al resolver el siguiente sistema:

$$x = alfa * aX + beta * bX + gamma * cX$$

$$y = alfa * aY + beta * bY + gamma * cY$$

$$alfa + beta + gamma = 1$$

Además para que el punto esté dentro del triángulo se debe cumplir que:

$$0 \leq alfa, beta, gamma \leq 1$$

2.5 Búsqueda de triángulos en QuadTree

Permitir que los handles se puedan colocar en cualquier punto de la malla requiere identificar qué triángulo contiene el handle, para así proceder a calcular sus coordenadas baricéntricas.

Para mejorar la eficiencia de esta búsqueda hemos utilizado una representación auxiliar de la malla en forma de *QuadTree*. Un QuadTree es una estructura de datos jerárquica que divide el espacio de forma recursiva en cuadrantes.

```
public class QuadTree<T>
{
	private Rectangle region;

	private T[] elements;
	private int numElements;

	private QuadTree<T>[] childrens;
	private boolean subdivided;

}
```

Una implementación de un QuadTree debe tener al menos los siguientes elementos:

- La región del espacio que delimita el nodo.
- Los elementos que incluye la región si es un nodo hoja.
- Las subdivisiones de la región si es un nodo interior.

Si al insertar un elemento en un nodo hoja se excede el número máximo de elementos que puede contener, este nodo se subdivide en 4 regiones nuevas y sus elementos se transfieren a los nodos hijos resultantes.

Este tipo de estructuras son comúnmente utilizadas para almacenar puntos del espacio. Tomando como referencia [6], hemos implementado una versión adaptada de Quadtree que almacena referencias a polígonos triángulos. A continuación se incluye un ejemplo de representación de una malla con diez elementos.

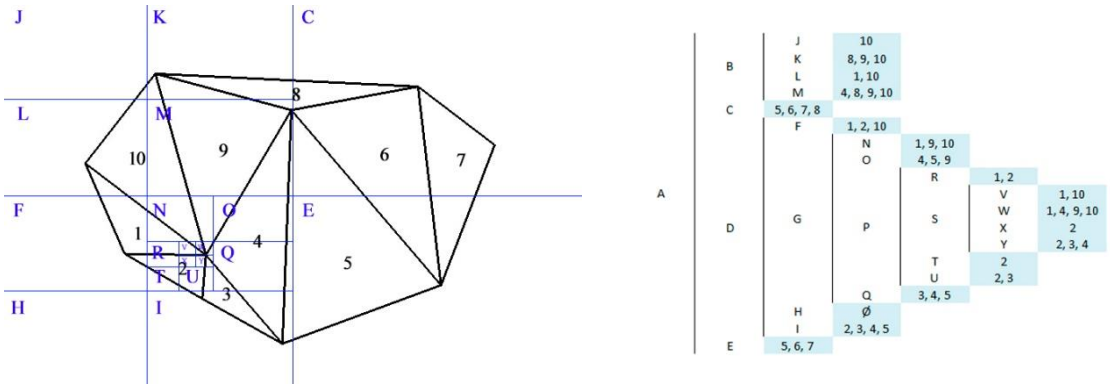


Ilustración 10 – Representación gráfica y esquemática de un Quadtree

A diferencia de lo que sucede con los puntos, un polígono puede pertenecer a varias regiones simultáneamente. Para ilustrar el funcionamiento de muestra implementación, indicamos el comportamiento de los dos principales métodos que soporta esta estructura de datos: inserción y búsqueda.

- Inserción de un triángulo en el Quadtree.

```
boolean insertar(triángulo)

      si el nodo no corta al triángulo y el nodo no lo contiene
            no añadir el triángulo
            devolver falso
      sino

            si el nodo no es hoja
                  si hay espacio para otro triángulo
                        añadir el triángulo
                        devolver cierto
                  sino
                        subdividir la región del nodo
                  fsi
            fsi

            para todas las subdivisiones del nodo
                  subdivisión.insertar(triángulo)
            fpara

            devolver cierto
      fsi


void subdividir()

      para el máximo de subdivisiones de un nodo
            construir subdivision
      fpara

      para todos los triángulos del nodo
            para todas las subdivisiones del nodo
                  subdivision.insertar(triángulo)
            fpara
      fpara
```

- Búsqueda de un triángulo en el Quadtree, a partir de un punto en coordenadas globales.

```
triángulo buscar(punto)

      si el nodo no contiene al punto
            devolver error
      sino

            si la región no está subdividida

                  si algún triángulo del nodo contiene al punto
                        devolver el triángulo
                  sino
                        devolver error
                  fsi
            sino

                  para todas las subdivisiones del nodo
                        devolver subdivisión.buscar(punto) si está
                        en la subdivisión
                  fpara

                  devolver error
            fsi
      fsi
```

3. APLICACIÓN DE LAS GEOMETRÍAS DEFORMABLES

La aplicación que hemos desarrollado, se basa en la creación y animación de personajes en 2D. Así como un juego de plataformas en el que poner en marcha nuestras creaciones.

Por tanto en nuestra aplicación hay dos partes claramente diferenciadas. La creación y animación de personajes, en la que creamos y manipulamos las geometrías deformables. Y la otra, es el juego propiamente dicho, con el que hemos dado una utilidad a estas geometrías.

Ilustración 11 - Pantalla inicial de la aplicación

Cuando iniciamos por primera vez la aplicación, el primer paso será la creación de un personaje basado en una geometría deformable. Para ello se pasa por tres fases en las que diseñamos su forma primero, lo decoramos a nuestro gusto después, y por último, grabamos las animaciones de los movimientos que reproducirá durante el juego. En esta última fase entra en juego el algoritmo de deformación.

3.1 Creación de un personaje

La creación está estructurada en estas tres fases, que se explican a continuación.

3.1.1 Fase de diseño

En esta fase, podemos trazar el contorno del personaje desplazando el dedo sobre la pantalla del dispositivo móvil. El requisito para pasar a la siguiente fase (la fase de pintura), es haber diseñado un polígono simple (sus lados no deben cruzarse entre sí). La aplicación une automáticamente el primer punto dibujado de nuestro polígono con el último para cerrar el contorno. Con estos puntos se calcula una curva b-spline, que regula el número de ellos y suaviza la figura.

En el momento en el que levantamos el dedo, se aplica el algoritmo de mallado al personaje explicado en el capítulo anterior. Antes de pasar a la siguiente fase debemos ubicar nuestro personaje en un rectángulo sombreado como el que aparece en la Ilustración 12, pudiendo para ello trasladar, rotar y/o escalar interactivamente la malla resultante.

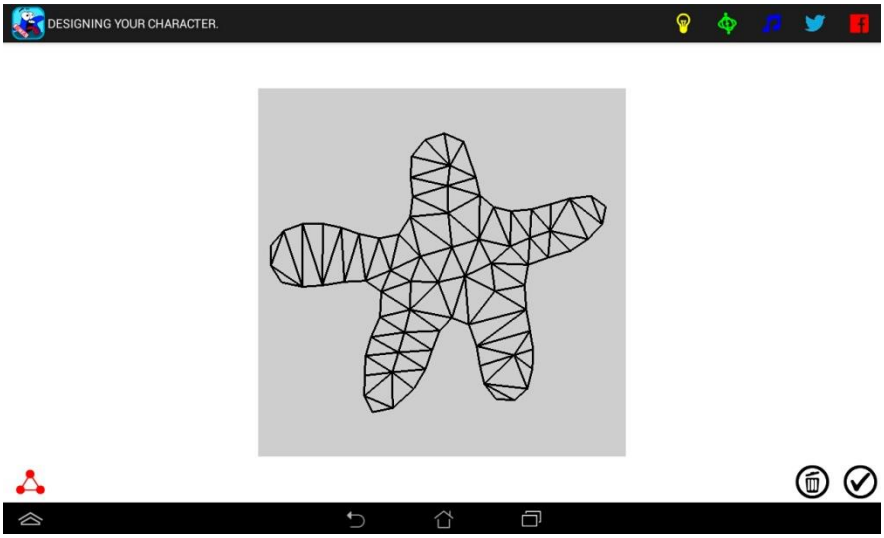


Ilustración 12 – Ejemplo de malla de un personaje

Para trasladar, rotar o escalar, hay que hacer un determinado gesto táctil (colocar y desplazar los dedos de una forma concreta). Explicados en la sección 5.2.1.

El objetivo de situar la malla dentro de este rectángulo radica en que ese área será la utilizada para capturar la textura en la fase posterior.

3.1.2 Fase de pintura

En esta fase nos encontramos con múltiples opciones para caracterizar nuestro personaje.

Con el *lápiz* podemos colorear y dibujar detalles. Admite 3 posibles tamaños, y puede elegirse su color con la paleta de colores incluida en la barra de herramientas.

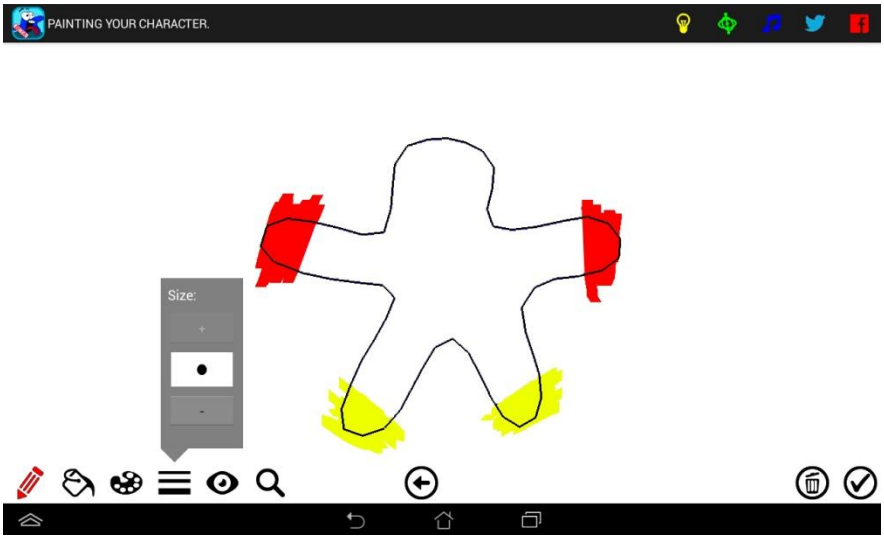


Ilustración 13 – Herramienta lápiz

Las líneas que dibujemos con el lápiz y que sobresalgan del contorno del personaje, no formarán parte de la textura final del mismo. A la hora de rasterizar la textura, todos los pixeles que queden fuera del contorno de la figura no se lanzarán a la tubería gráfica, por consiguiente aunque estén presentes en el bmp, no serán pintados.

El color de la paleta, también afecta al *cubo de pintura*, herramienta para colorear por completo el interior del personaje.

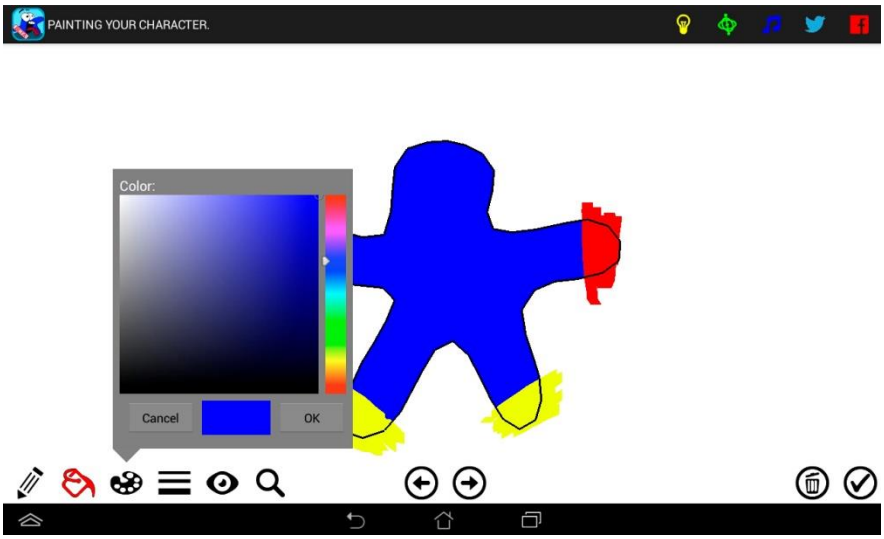


Ilustración 14 – Herramienta paleta de colores

La última herramienta de caracterización, consiste en añadir *pegatinas*, que pueden ser de 5 tipos: accesorios, cascos, ojos, bocas y armas. No todas las pegatinas están disponibles desde un principio, pero podemos conseguirlas a medida que superemos niveles en el juego. El juego y los niveles de los que consta están explicados más adelante.

Ilustración 15 – Ejemplo de pegatinas

Sólo se puede poner una pegatina de cada tipo, pero éstas se pueden editar, para cambiar su tamaño o su orientación de forma interactiva, a través de los gestos táctiles. Se pueden situar en cualquier punto del personaje, haciendo uso de las mismas coordenadas baricéntricas que usamos con los handles.

En la barra de herramientas también encontramos la opción de deshacer los últimos cambios en el personaje (*retroceder*) o volver hacia delante en el caso de querer recuperar las ediciones descartadas (*avanzar*). También podemos borrar todo y empezar de nuevo la fase de pintura pulsando en el *cubo de la basura*.

Y con el objetivo de ver nuestro diseño con más detalle o desde otros ángulos, podemos activar la *lupa* para acercar, alejar y mover el personaje mediante los diferentes gestos táctiles.

Éstas son todas las herramientas y opciones de diseño incluidas en esta fase. No es necesario utilizar todas las herramientas ya que en cualquier momento podemos pasar a la última fase, la de animación.

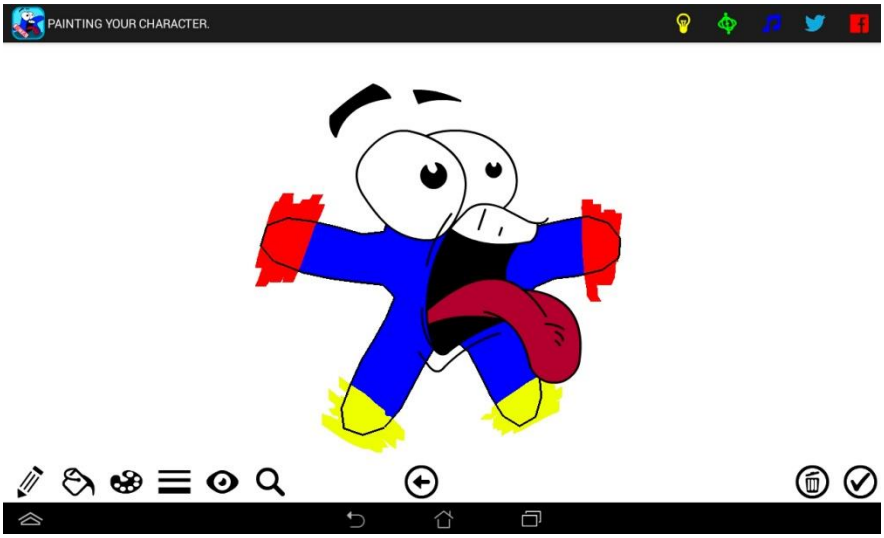


Ilustración 16 – Ejemplo de diseño de un personaje

3.1.3 Fase de animación

Durante la fase de animación se definen los cuatro movimientos que el personaje reproducirá durante el juego: correr, saltar, agacharse y atacar.

Ahora en la barra inferior, se nos presentan nuevas opciones. Podemos poner nuevos handles, quitarlos, moverlos y grabar la animación deseada. Una vez grabada una animación para uno de los cuatro movimientos, tenemos la posibilidad de reproducirla y decidir así, si nos gusta o por el contrario queremos descartarla y grabarla de nuevo.

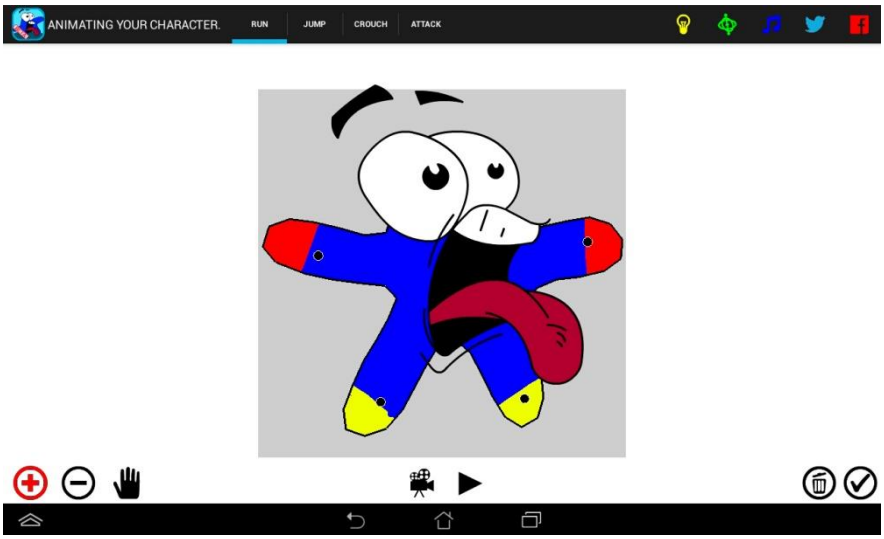


Ilustración 17 – Pantalla de fase de animación.

En la Ilustración 17, podemos observar al personaje con 4 handles situados en lo que serían sus extremidades.

Con la herramienta *mano* activada, podemos mover uno o varios handles simultáneamente, arrastrándolos con los dedos. Los handles que no toquemos se quedarán fijos, y el personaje se deformará intentando guardar la forma original. Todo esto se consigue con el algoritmo de deformación descrito en el capítulo anterior.

Es necesario grabar las animaciones de los cuatro movimientos para poder terminar la creación del personaje. En caso afirmativo esta etapa concluye otorgando un nombre a nuestra creación. A continuación la aplicación volverá a la pantalla principal.

Ahora que ya hay al menos un personaje creado, se puede acceder a un menú donde se muestran todos los personajes existentes.

3.2 Selección de un personaje

En el menú de selección de personaje podemos editar, compartir o seleccionar a uno de ellos para jugar. También es posible reproducir los cuatro movimientos que le hemos definido anteriormente, tocando los botones destinados a cada una de estas funciones, que se encuentran en la parte inferior izquierda de la pantalla.

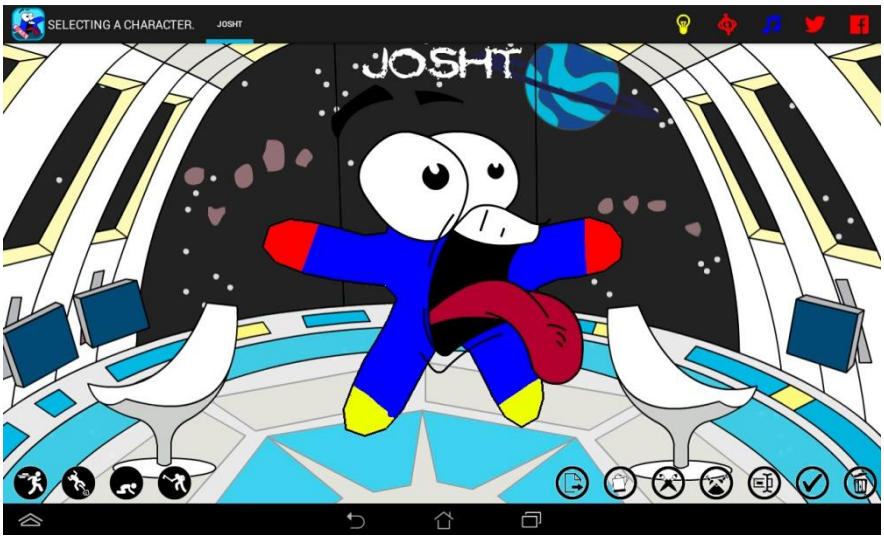


Ilustración 18 – Pantalla de selección de personajes

En lo referente a la edición de un personaje, se dispone de tres opciones:

- Modificar su diseño: se puede volver a editar el estilo del personaje en la fase de pintura (3.1.2).
- Cambiar sus movimientos: se puede volver a grabar cualquiera de sus animaciones en la fase de animación (3.1.3).
- Renombrarlo.

Ilustración 19 – Iconos de botones de la fase de selección de personajes
de izquierda a derecha: a) Exportar, b) Compartir, c) Repintar, d) Reanimar, f) Renombrar

Para compartirlo, hay que tomar una instantánea (Ilustración 19.b) en la que podemos cambiar la posición y el tamaño de nuestro personaje mediante diferentes gestos táctiles. Cuando hagamos la foto, tenemos la opción de

acompañarla con algún comentario. El resultado se publicará en Twitter y/o en Facebook si hemos iniciado sesión en alguna de estas dos redes sociales a través de nuestra aplicación.

Otra forma de compartir un personaje, es guardándolo en un fichero externo (Ilustración 19.a), el formato de estos ficheros se explicarán posteriormente. Y así, enviarlo por ejemplo adjunto en un e-mail a un amigo. El cual puede cargar el fichero desde la pantalla principal de la aplicación, para poder tener el personaje que nosotros hemos creado.

3.3 El juego

Si hay algún personaje creado o importado, y ha sido seleccionado, éste aparecerá en la pantalla principal de la aplicación. Además estará disponible la opción de jugar. Tocando la pantalla, se reproducirá aleatoriamente, uno de los cuatro movimientos grabados del personaje.

El juego se divide en 5 niveles. Cada uno con dos fases en las que cambia la mecánica de juego. Al seleccionar jugar desde el menú principal, pasamos a la selección de niveles.

Hemos realizado un video de presentación, que introduce al usuario en el juego con una pequeña historia. El video cuenta con varias escenas en las que intervienen las geometrías deformables y en las que podemos interactuar pulsando sobre algunos de los elementos que aparecen.

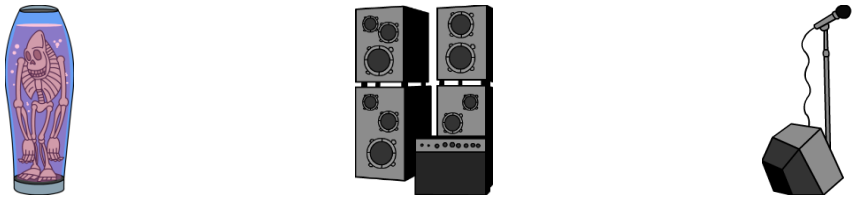


Ilustración 20 – Ejemplo de objetos animados.

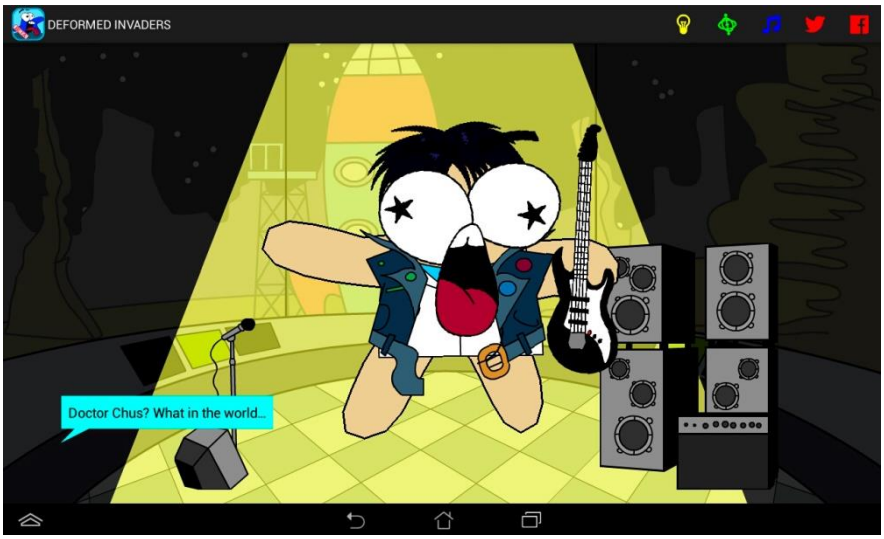


Ilustración 21 - Pantalla del vídeo de introducción.

3.3.1 Selección de niveles

En la selección de niveles, no sólo se elige el nivel en el que jugar, sino que se muestra información acerca de los logros conseguidos en ese nivel y si éste está bloqueado o no. Si está bloqueado, no podremos jugar hasta que no completemos los niveles previos. Se puede navegar entre los diferentes niveles deslizando el dedo por la pantalla hacia la derecha o hacia la izquierda.

Ilustración 22 - Pantalla para la selección de niveles

Cada nivel está ambientado en una época diferente, representando todos ellos, localizaciones importantes de la historia. En estos, aparecen obstáculos y enemigos que intentaran acabar con nosotros. Los diseños de los enemigos, así como sus animaciones, han sido realizados con la propia aplicación, caracterizándolos de acuerdo con la época en la que aparecen. También los obstáculos están personalizados para cada nivel.

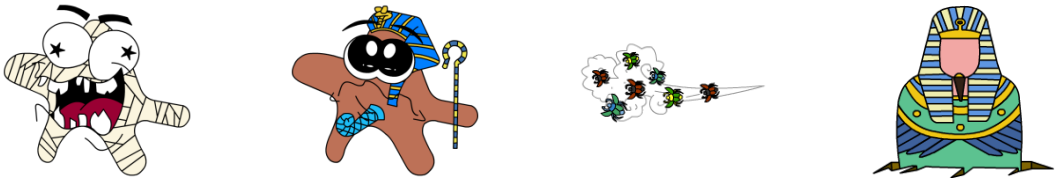


Ilustración 23 – Ejemplo de enemigos y obstáculos

Las posiciones en las que aparecen los enemigos y los obstáculos se generan aleatoriamente cada vez que entramos en un nivel.

3.3.2 Fases del Juego

Cada nivel del juego se divide en dos fases: la fase de enemigos y la fase del jefe final. En cada una la mecánica de juego es diferente.

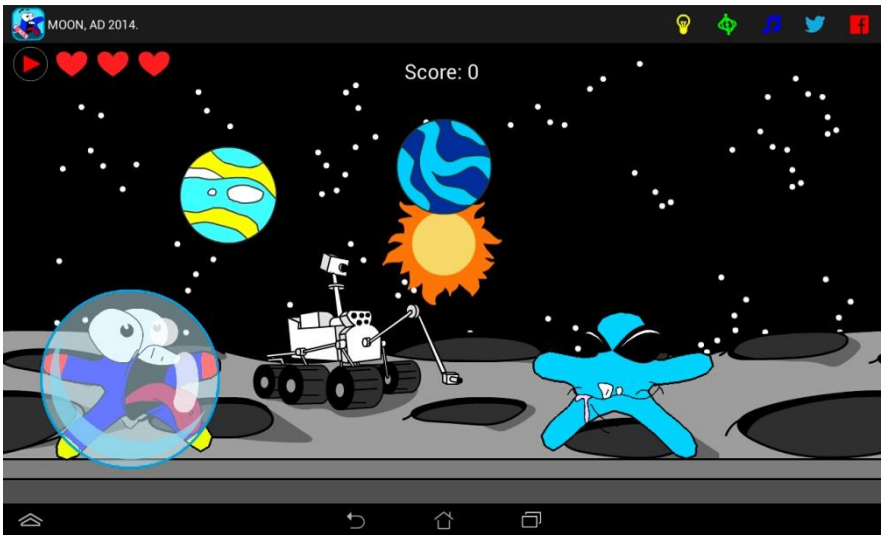


Ilustración 24 – Fase de enemigos iniciales

La primera fase consiste en avanzar, matando los enemigos y evitando los obstáculos que nuestro avatar se encuentre por el camino. El jugador dispone de tres movimientos, cada uno de ellos asociado a un gesto táctil: atacar, tocando una vez la pantalla; saltar, deslizando el dedo hacia arriba; y agacharse, deslizando el dedo hacia abajo. La velocidad con la que llegan los enemigos y los obstáculos se va incrementando cuanto más nos acercamos a la fase del jefe final.

El personaje está protegido con una burbuja que le otorga tres vidas, con cada impacto con un enemigo o el choque contra un obstáculo se pierde una vida.

Ilustración 25 – Evolución de la burbuja al perder vidas

Al terminar la fase de enemigos, llegamos al jefe final. En este momento nuestro avatar cuenta con una plataforma voladora, con la que puede moverse verticalmente (deslizando el dedo hacia arriba o hacia abajo, o inclinando la tableta en caso de poseer y tener activado los sensores) y atacar disparando un rayo (tocando la pantalla).

El jefe final dispone de una plataforma similar, además de tener tres vidas. Nuestro objetivo será derrotarle, alcanzándolo hasta cuatro veces con un rayo.

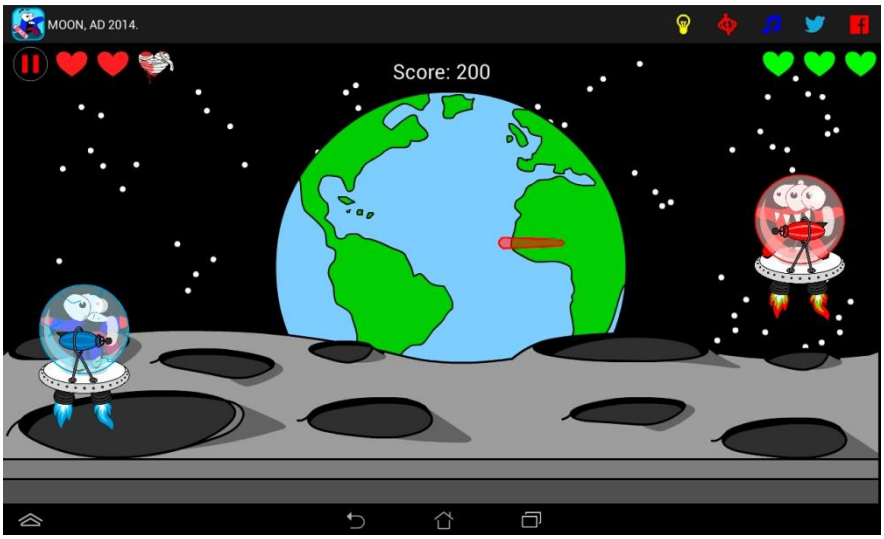


Ilustración 26 – Fase del jefe final

Si logramos acabar con él, habremos completado el nivel y podemos obtener hasta tres *trofeos*, dependiendo de si se han cumplido las siguientes condiciones:

- Trofeo de nivel completado: Se obtiene al completar el nivel.
- Trofeo de nivel perfecto: Si completamos el nivel llegando con las tres vidas intactas hasta la fase del jefe final.
- Trofeo de nivel dominado: Al terminar ambas fases del nivel sin perder ninguna vida.


Ilustración 27 – Iconos correspondientes a los trofeos

Al conseguir los tres trofeos de un nivel, tendremos disponibles en la fase de pintura, las pegatinas de los enemigos que aparecen en ese nivel.

4. TECNOLOGÍAS

4.1 Android y tecnología táctil

Actualmente, en el ámbito del desarrollo de aplicaciones móviles está dividido en tres géneros:

- Aplicaciones web optimizadas para móviles (*app web*).
- Aplicaciones móviles nativas (*app nativa*).
- Aplicaciones híbridas.

Las *app web* son páginas web optimizadas para dispositivos móviles o tablets que utilizan tecnologías como HTML5, CSS y JavaScript. Estas aplicaciones son accedidas desde el navegador, por ello serán compatibles con la mayoría de los dispositivos.

Esta aplicación se ha programado como una *app nativa* para el sistema operativo **Android**. Este tipo de aplicaciones ofrecen la ventaja de poder utilizar funcionalidades nativas, como la cámara o multitud de sensores integrados, además de mejorar el rendimiento y poderse utilizar sin necesidad de conexión a Internet.

Sin embargo el desarrollo nativo restringe la aplicación a un sistema operativo concreto. Para Android se desarrollan usando el lenguaje de programación **Java**, junto con las herramientas ofrecidas por el **SDK para Android** (Software Development Kit) [7].

Programar app nativas para este tipo de dispositivos móviles nos ofrece la capacidad de diseñar aplicaciones con un manejo más intuitivo y dinámico, gracias a la **interacción táctil** que ofrecen.

Esta interacción está incluida dentro del paquete **android.view** del SDK de Android. Esta API nos permite responder a eventos de movimiento (*MotionEvent*) generados tanto por ratones, dedos o lápices táctiles. Su uso principal es el de distinguir entre acciones como pulsar, mover o soltar, incluso en dispositivos que reconozcan múltiples puntos de contacto (*multitouch*).

Para poder visualizar las geometrías deformables y el resto de elementos del juego, hemos utilizado la tecnología **OpenGL ES 1.0** [8]. OpenGL ES es una versión reducida de la tecnología OpenGL, y está diseñada especialmente para dispositivos integrados. Entre la eliminación de muchas funcionalidades originales cabe destacar dos cambios importantes:

- La eliminación de la semántica *glBegin – glEnd*.
- La limitación a renderizar vértices utilizando buffers.

El uso de esta tecnología permite mantener un código multilenguaje y multiplataforma.

4.2 API's de cálculo

Como ha podido comprobarse, las geometrías deformables requieren una importante carga de cálculos geométricos y algebraicos.

Muchos de los cálculos geométricos necesarios para el proceso de mallado los hemos encontrado implementados en el *framework* de desarrollo de videojuegos **Libgdx** [9].

Libgdx también da soporte para audio, gráficos 2D y 3D, gestión de ficheros y gran cantidad de simulaciones físicas como detectores de colisiones. Esta tecnología nos ha facilitado desde el principio la organización de los datos necesarios en diversos buffers, que se detallarán más adelante.

Por otra parte, en lo relativo al algoritmo de deformación, hemos necesitado resolver ecuaciones matriciales. Este tipo de operaciones han sido implementadas gracias a **JAMA** [10], un paquete de algebra lineal básica implementado en Java. JAMA ofrece múltiples factorizaciones de matrices como LU, QR, Cholesky, entre otras.

4.3 Elementos multimedia

Además de los cálculos y algoritmos para gestionar las geometrías deformables, en nuestro proyecto hemos desarrollado un juego que usa estas geometrías. Todas las **texturas** incluidas han sido diseñadas por nosotros, tomando como referencia imágenes procedentes de internet.

Para ello se ha utilizado la herramienta de tratamiento de gráficos vectoriales **Adobe Flash**. Esta herramienta nos ha permitido disponer de una estética común en todas las texturas, similar a comics o cartoons.

Usando esta misma técnica hemos incluido videos de consejos para el usuario, a modo de ayuda, sobre los aspectos y funcionalidades de algunas fases de la aplicación. Estos vídeos han sido diseñados con la tecnología Flash y posteriormente convertidos usando la herramienta online **Flabaco** (Flash Banner Converter) [11]. Flabaco convierte ficheros *swf* de Flash en formatos de vídeo compatibles con HTML5 (*webm*, *mp4* y *ogg*).

Además se han incluido una variedad de pistas musicales y efectos de sonido. Todos estos archivos de audio han sido obtenidos de páginas web que trabajan bajo la licencia *Creative Commons* y que se listan a continuación: Incompetech [12], SoundBible [13], Audionautix [14], GRSites [15] y Freesound [16].

Para reproducir estos elementos multimedia hemos utilizado la API del SDK de Android incluida en el paquete **android.media**. En este paquete se incluyen herramientas capaces de grabar y reproducir gran variedad de archivos audio, vídeo e incluso distribuciones en *streaming*.

5. IMPLEMENTACIÓN DE LA APLICACIÓN

5.1 Arquitectura

Para la implementación de nuestro proyecto, hemos usado un patrón de diseño conocido como *Modelo-Vista-Controlador*. El patrón MVC es una arquitectura de propósito general que divide la aplicación en tres componentes: modelo, vista y controlador.

En el siguiente diagrama de colaboración representamos estos elementos integrados en nuestro proyecto. *Game Core* representa el modelo, *View Activity* la vista y *Game Controller* el controlador.

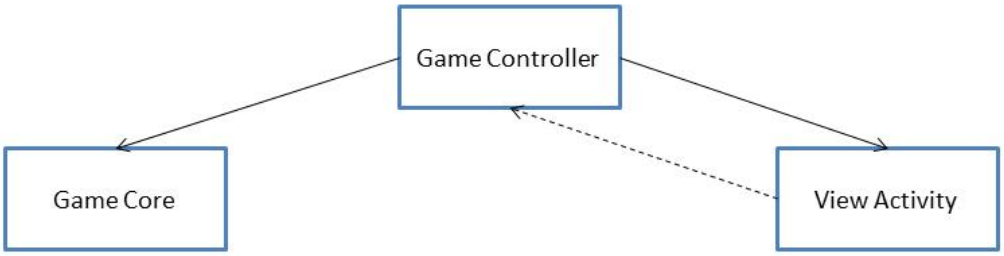


Ilustración 28 - Estructura del patrón Modelo-Vista-Controlador

A continuación explicamos cómo hemos adaptado este patrón a nuestra aplicación.

5.1.1 Controlador

El controlador es el encargado de mediar entre el modelo y los eventos generados por las vistas. Además de esta funcionalidad, hemos añadido la capacidad de gestionar el estado de la aplicación mediante un autómata.

Ilustración 29 - Bottom homescreen bar

Debido al diseño propio de las aplicaciones desarrolladas para Android, existe la posibilidad de mantener una pila de vistas (*backstack*) para deshacer los pasos efectuados por el usuario. Esta funcionalidad se la hemos otorgado también al controlador, que será capaz de mantener la información de dicho historial.

A continuación se representa el autómata que consta de 9 estados. Cada estado encapsula las diferentes fases y funcionalidades que están disponibles en un momento especifico durante la ejecución de la aplicación. En azul se representan las transiciones generadas por eventos del usuario. En rojo, las transiciones permitidas para el uso del botón *backstack*.

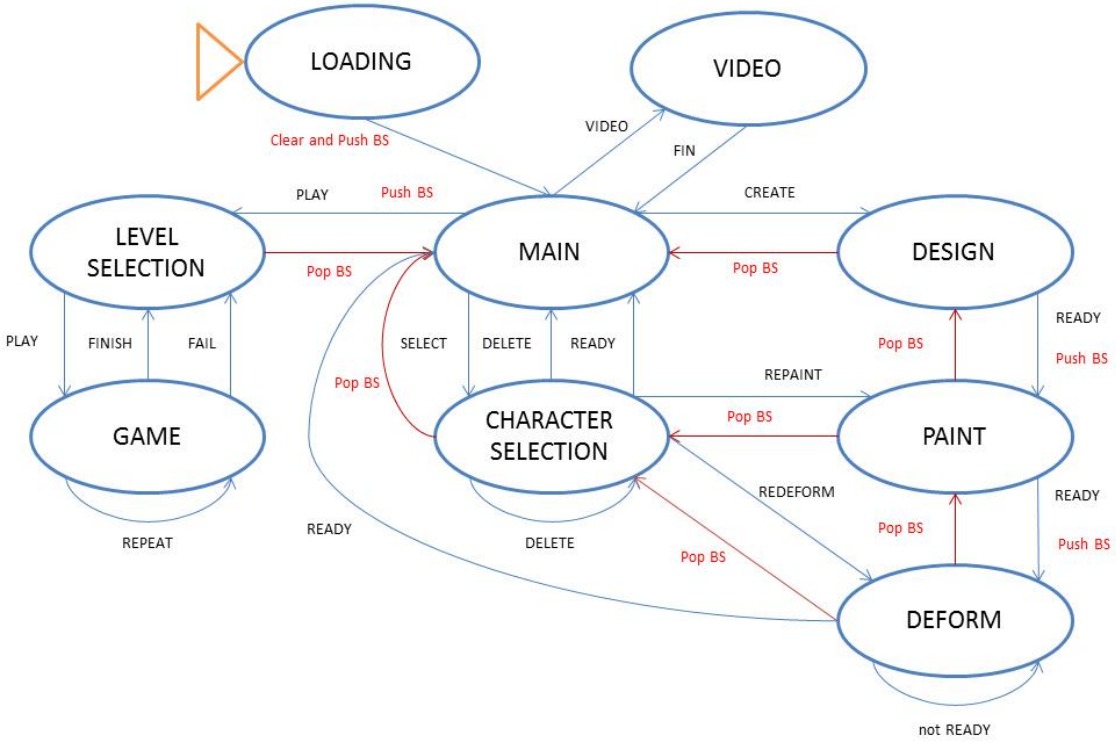


Ilustración 30 – Autómata de los estados de la aplicación

Durante el estado *Loading*, la aplicación recupera todos los personajes creados por el usuario, así como las estadísticas de los niveles que haya completado. También en este estado se generan los elementos estáticos de los niveles y del vídeo de introducción, que explicaremos más adelante. Una vez terminado este proceso se carga la pantalla principal de la aplicación.

El estado *Main* es un estado de espera en el que se representa el personaje seleccionado (si lo hay), y se espera del usuario un cambio de estado a la fase de creación, selección de personaje, selección de nivel o reproducción del vídeo de introducción.

Los estados que conforman la fase de creación de personaje son *Design*, *Paint* y *Deform*. En el estado *Design* se encuentra la funcionalidad responsable de la construcción del esqueleto del personaje a partir de una geometría deformable. Se espera del usuario que dibuje un contorno para, a partir de él, construir la geometría.

Durante el estado *Paint* el usuario podrá decorar el diseño del personaje creado anteriormente. En este estado se construye la textura de las geometrías a partir de las pegatinas añadidas y el bitmap generado por el color de fondo y los detalles dibujados con el lápiz. Por último en el estado *Deform*, el usuario podrá crear las animaciones del personaje utilizando el algoritmo de deformación. Una vez terminado este proceso se volverá al estado Main tras darle un nombre al personaje y guardarlo en el sistema de ficheros.

En el estado *Character Selection* se representará la lista de personajes existente y se ofrece la posibilidad de realizar modificaciones sobre algún personaje, así como visualizar las animaciones asociadas a cada personaje.

La fase del juego está formada por dos estados: *Level Selection* y *Game*. En el estado *Level Selection* se muestra la lista de niveles que conforman el juego. El usuario podrá elegir en qué nivel quiere jugar, pasando así al estado *Game*. Durante el estado *Game* se desarrolla la mecánica del juego asociada a un nivel. Tras finalizar el nivel se dará la opción de volver a jugarlo o de regresar al estado *Level Selection*.

Por último el estado *Video* reproduce el video de introducción del juego, el usuario podrá interactuar con algunos objetos de las escenas durante la reproducción del vídeo. Una vez finalizado el vídeo se cargará de nuevo el estado *Main*.

5.1.2 Modelo

El modelo contiene la funcionalidad básica y los datos de la aplicación. Encontramos también los módulos encargados de interactuar con el sistema operativo, como el almacenamiento de datos, la gestión de ficheros, audio y la conexión social. Cada uno de estos módulos los explicaremos a continuación.

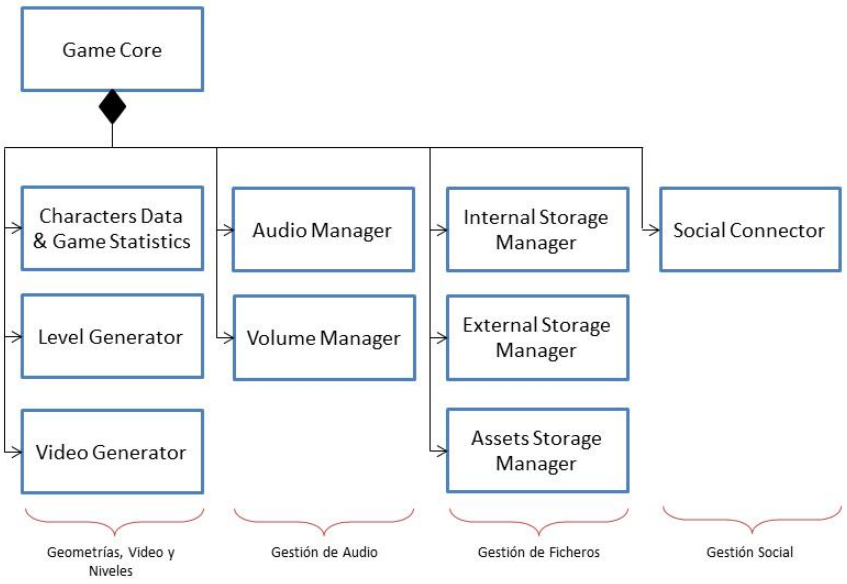


Ilustración 31 - Componentes del Modelo

5.1.2.1 Representación de las geometrías

La mayor parte de la información generada por la aplicación se encapsula en *entidades*. Estas entidades tienen como objetivo representar en la tubería gráfica diferentes elementos de la escena.

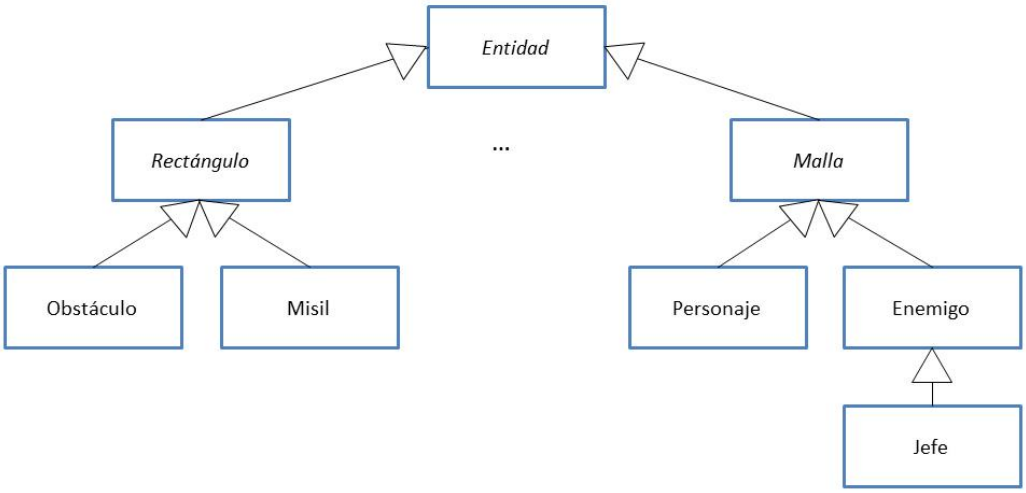


Ilustración 32 – Jerarquía de entidades

Los elementos a representar más importantes se dividen en rectángulos y mallas. Los rectángulos son aquellas entidades que se representan como una textura bmp rectangular. Por el contrario, las mallas implementan las geometrías deformables.

Las mallas están compuestas por un *esqueleto*, una *textura* y una lista de *movimientos*.

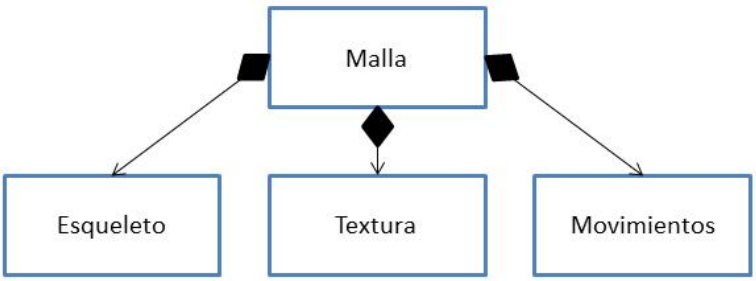


Ilustración 33 – Estructura interna de una malla

El esqueleto de una malla almacena toda la información geométrica recogida en la fase de diseño y está compuesto por la lista de vértices y triángulos que forman la malla y por el conjunto de vértices que forma su contorno.

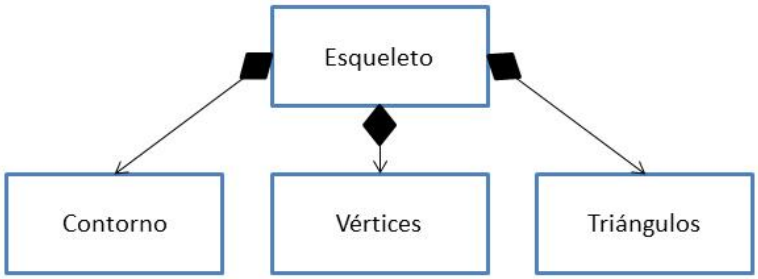


Ilustración 34 – Estructura interna del esqueleto de una malla

La textura se compone tanto por el mapa de bits y las pegatinas obtenido en la fase de pintura, así como las coordenadas de textura. Las coordenadas de textura se usan para representar bitmaps en OpenGL ES, representan las coordenadas de los vértices de la malla dentro del bitmap.

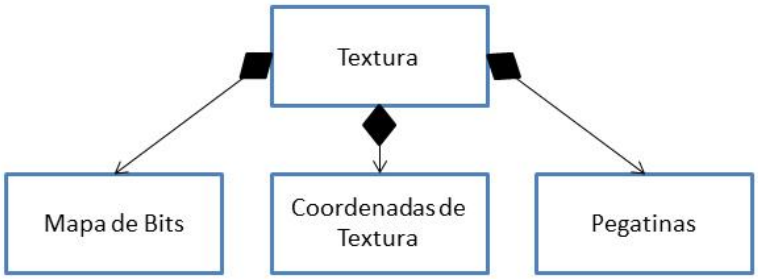


Ilustración 35 – Estructura interna de la textura de una malla

Por último la lista de movimientos está formada por los valores de las coordenadas de cada vértice durante el conjunto de fotogramas que contiene la animación.

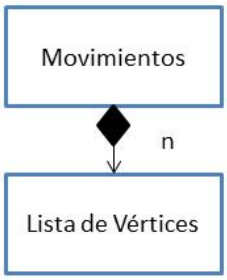


Ilustración 36 – Estructura interna de los movimientos de una malla

Además de estas dos entidades principales, existen otras que son utilizadas sólo durante el juego y el vídeo de introducción, y que sirven para modelar diferentes elementos de la animación, como la burbuja protectora, la plataforma voladora o los objetos animados del escenario del vídeo.

5.1.2.2 Creación del vídeo y los niveles

Además de las entidades anteriores, también tenemos que representar otros elementos como son los niveles y el vídeo.

Cada nivel encapsula la información que le corresponde en la fase del juego. Contiene una lista con todos los enemigos a derrotar, una cola con instancias de dichos enemigos que definirán su posición concreta en la escena. También incluye las texturas de los fondos que van ciclando mientras dure el tránsito por el nivel.

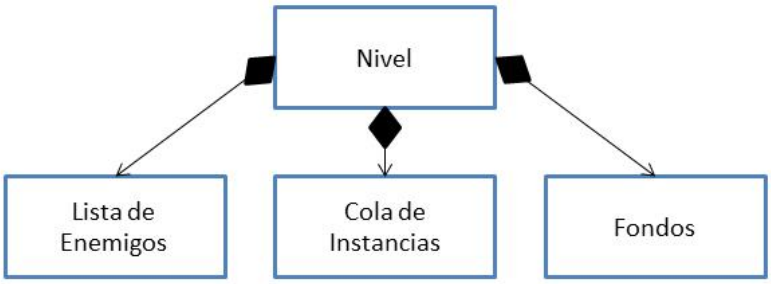


Ilustración 37 – Estructura de un nivel del juego

Durante el proceso de carga de la aplicación, se construye la lista de enemigos y de fondos. Estos elementos serán estáticos durante toda la aplicación. La cola de instancias se construirá aleatoriamente cada vez que el usuario decida jugar un nivel en concreto.

El vídeo tiene una estructura similar a un nivel. Incluye una lista con los actores y los objetos animados que intervienen, junto con los mensajes que conforman el guion del vídeo. El vídeo no está grabado, es procedimental y se reproduce como un nivel más.

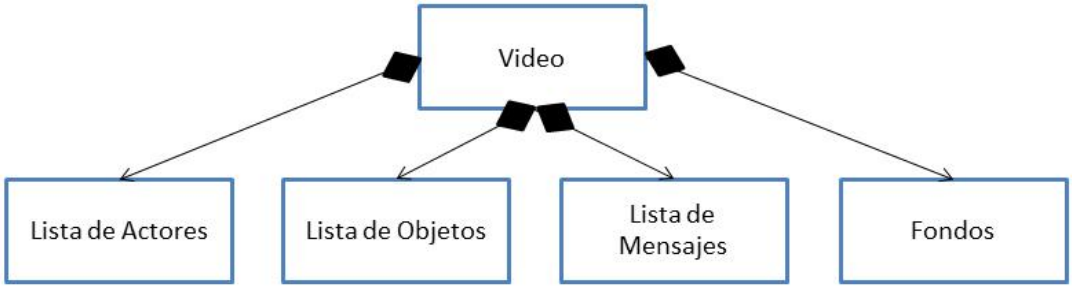


Ilustración 38 – Estructura del video de introducción

Durante el vídeo aparecen tres tipos de objetos:

- Objetos inanimados: son aquellos objetos que solo tienen una textura y no responden a ningún evento al ser pulsados.
- Objetos animados: son aquellos objetos que tienen una serie de texturas que reproducen una animación al intercambiarse en el tiempo. Esta animación puede ser cíclica o reproducirse sólo cuando se pulsa sobre él.
- Objetos móviles: son aquellos objetos animados que además se desplazan por la escena.

Ilustración 39 – Representación de los objetos del vídeo

Todos los elementos del vídeo se construyen también en la fase de cargado y no serán modificados durante la ejecución del vídeo.

5.1.2.3 Gestión de ficheros

Toda la información generada por la aplicación debe ser almacenada en ficheros para poder recuperarla posteriormente. Hemos implementado tres gestores que se encargan de interactuar con diferentes parcelas de memoria dentro del dispositivo.

El *gestor de almacenamiento interno*, almacena los personajes creados por el usuario, las estadísticas de los niveles completados y las opciones seleccionadas, tales como la activación de música o los consejos, entre otros. Estos datos se almacenan en un espacio reservado por el sistema operativo para cada aplicación.

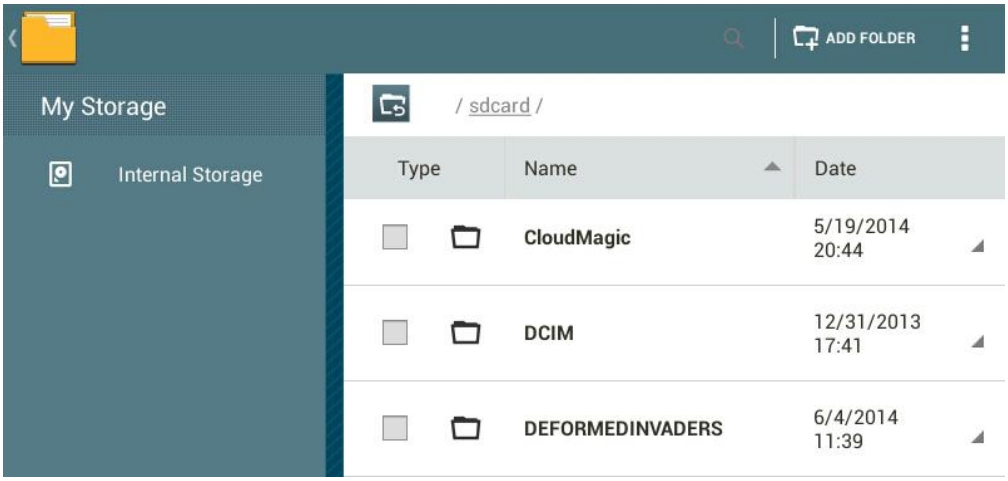


Ilustración 40 – Directorio de la aplicación en el dispositivo

El *gestor de almacenamiento externo* permite la exportación e importación de personajes desde el directorio de *Deformed Invaders* localizado en el disco o la tarjeta de memoria del dispositivo. Estos ficheros tienen una extensión .cdi (*character deformed invaders*) y su estructura es la representada en la siguiente ilustración:

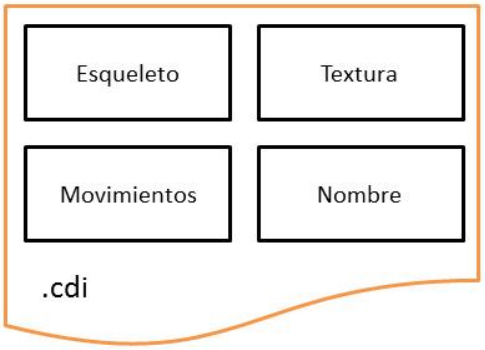


Ilustración 41 – Estructura interna de un fichero .cdi

Este tipo de ficheros contiene los componentes del personaje (esqueleto, textura y movimientos explicados anteriormente) serializados en una secuencia de bytes.

El último gestor de almacenamiento permite gestionar algunos ficheros incluidos en la propia aplicación, como los ficheros de música, vídeo, fuentes de texto y el catálogo de enemigos diseñados con la propia aplicación. Los ficheros de los enemigos tienen la extensión .edi (*enemy deformed invaders*) y su estructura es similar al de los personajes creados por el usuario.

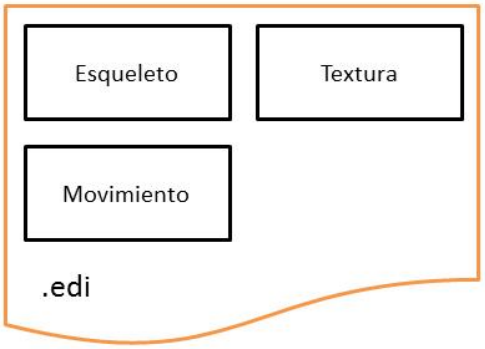


Ilustración 42 – Estructura interna de un fichero .edi

5.1.2.4 Gestión de audio

Durante la ejecución de la aplicación se reproduce en todo momento una música de fondo que irá cambiando dependiendo de la fase en la que nos encontremos. Se incluyen también ciertos efectos sonoros para el juego y el vídeo de introducción.

Para ello se disponen los siguientes tres reproductores de sonido, que podrán reproducirse simultáneamente:

- Reproductor de la música de fondo.
- Reproductor de efectos de audio, como disparos, saltos o ruidos generados por los objetos animados del vídeo de introducción.
- Reproductor de voces de los actores del vídeo de introducción.

Toda esta funcionalidad está disponible gracias a los gestores de volumen y reproducción de audio.

5.1.2.5 Conexión social

Se ha implementado una integración con las dos principales redes sociales: Facebook y Twitter. Esto nos permite publicar diferentes acciones llevadas a cabo mientras se ejecuta la aplicación.

Podemos compartir el diseño de los personajes que hemos creado. También al completar los diferentes niveles, se publicará automáticamente una imagen con el trofeo y la puntuación que hayamos conseguido.

A continuación se muestran dos ejemplos de este tipo de publicaciones:

Ilustración 43 – Ejemplos de publicaciones en Twitter.

5.1.3 Vista

Las vistas muestran los datos y el modelo, y recogen la interacción del usuario. Nuestra aplicación utiliza para la representación gráfica la tecnología OpenGL ES. Debido a ello, la gran mayoría de las vistas mantienen una estructura común.

5.1.3.1 Interfaz de usuario

En Android una aplicación se representa mediante una *actividad*. Esta actividad es la encargada de gestionar la ventana que representará la interfaz de usuario.

Las implementaciones de actividades pueden hacer uso de *fragmentos*. Un fragmento representa un comportamiento o una parte de la interfaz de usuario. Se pueden reemplazar o combinar múltiples fragmentos para diseñar interfaces de usuario dinámicas y flexibles.

En nuestra aplicación, cada fragmento representa una vista diferente. Tienen asociado un fichero xml donde se representa su contenido (*views*), tales como imágenes, cuadros de textos y demás elementos propios de la interfaz.

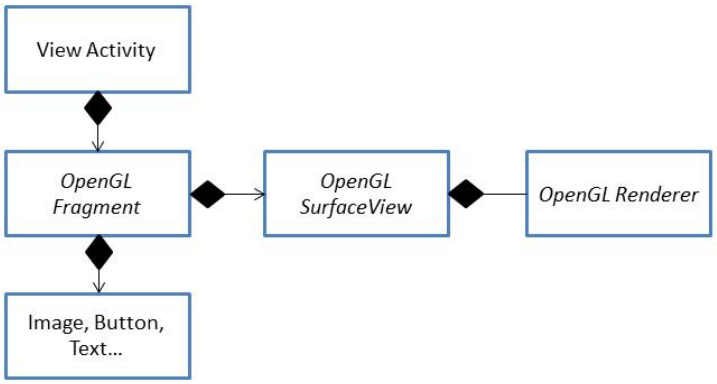


Ilustración 44 - Representación genérica de una vista

En algunas vistas se dispone de un diseño compuesto para representar múltiples vistas a la vez, tales como la lista de personajes o los niveles del juego. Para ello se usa un tipo especial de fragmentos (*swipeable fragments*) que permiten desplazarse por ellos mediante un gesto táctil, como si fueran las hojas de un libro.

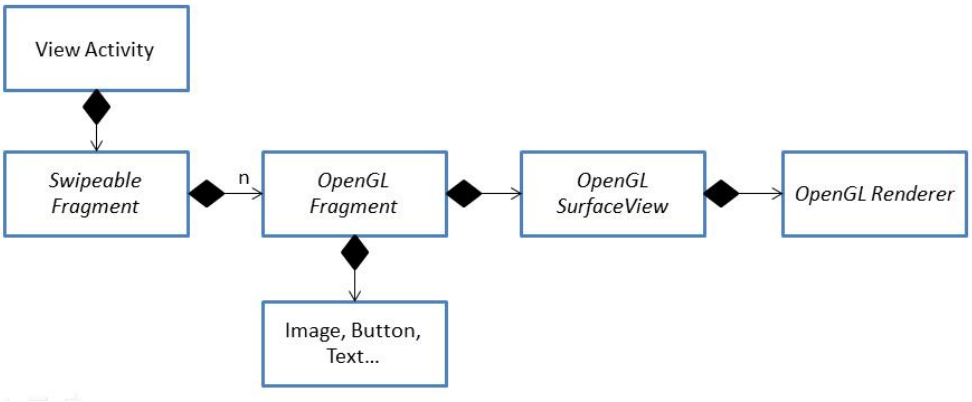


Ilustración 45 – Representación genérica de una vista compuesta

Para ilustrar mejor este diseño jerárquico de las vistas vamos a explicar un ejemplo concreto de una vista.

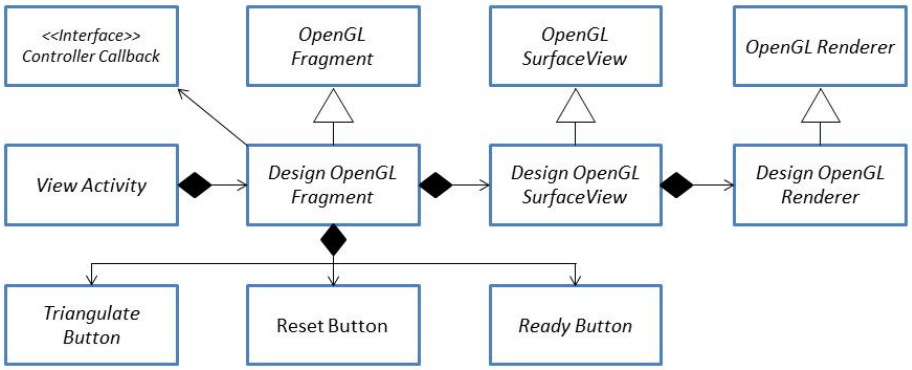


Ilustración 46 – Representación de la interfaz en la fase de diseño

En la fase de diseño del personaje, la actividad representará un fragmento llamado *Design OpenGL Fragment*. Este fragmento heredará de OpenGL Fragment cierta funcionalidad común como los oyentes de la pantalla táctil. Estará compuesto por tres botones (Triangulate Button, Reset Button y Ready Button), para que el usuario pueda seleccionar determinadas funcionalidades de esta fase.

5.1.3.2 Gráficos en OpenGL ES

Para incorporar la tecnología OpenGL ES en Android, se dispone de un elemento especial denominado *GLSurfaceView*. Este elemento realiza la configuración necesaria para incorporar un *renderer* de OpenGL ES y gestiona los eventos producidos tanto por la pantalla como por los sensores que hemos utilizado y que explicaremos posteriormente.

La clase *OpenGLRenderer* es la encargada de lanzar a la tubería gráfica las diferentes entidades que compongan la escena. También dispone de técnicas para representar fondos estáticos y animados (por desplazamiento horizontal o por intercambio).

Para poder visualizar las geometrías deformables, hemos utilizado una serie de buffers que almacenan toda la información necesaria. Son los siguientes:

- VertexArray. Contiene una colección de las coordenadas de los vértices de la malla. Cada vértice está compuesto por:

  Float x, y: coordenadas de los puntos.

X0	Y0	X1	Y1	X2	Y2	X3	Y3	…

- TriangleArray. Contiene una lista con los índices de los tres vértices que forman cada triángulo de la malla.

  Short indexA, indexB, indexC: índice de los tres vértices que forman el triángulo.

A1	B1	C1	A2	B2	C2	A3	B3	C3	…

- HullArray. Contiene una lista con los índices de los vértices que conforman el contorno de la malla.

  Short indexV: índice de los vértices que forman el contorno de la malla.

V1	V2	V3	V4	V5	V6	V7	V8	…

- StickerArray. Contiene la información de las pegatinas que se han colocado sobre la malla, incluyendo sus datos de posición, escalación y rotación. Cada pegatina consta de los siguientes datos:

  Integer id: identificador de la pegatina.
  Short indexT: índice del triángulo sobre el que está colocado el centro de la pegatina.
  Float alfa, beta, gamma: coordenadas baricéntricas dentro del triángulo.
  Float iota: ángulo que forma el eje Y con el vector PA, donde A es el primer vértice del triángulo y P la posición del centro la pegatina.
  Float kappa: ángulo iota en el instante inicial, calculado al añadir la pegatina.
  Float theta: ángulo de rotación inicial de la pegatina dado por el usuario.
  Float factor: factor de escalado inicial de la pegatina dado por el usuario.

Id1	T1	α1	β1	γ1	ι1	κ1	θ1	F1	Id2	…

Las pegatinas, como la mayoría de las texturas que se incluyen en la aplicación se representan en OpenGL ES utilizando un rectángulo que cubre al bitmap. Una pegatina estará asociada a un triángulo de la malla, cuando el centro de este rectángulo esté incluido en el área de dicho triángulo, como se muestra en la siguiente ilustración.

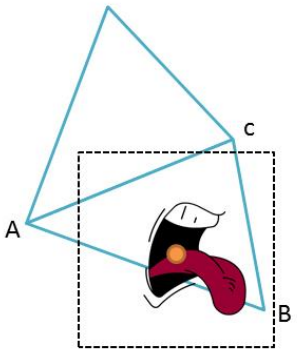


Ilustración 47 – Representación de una pegatina como un bitmap

Al añadir una pegatina se le da al usuario la opción de modificar su escalación y orientación (ángulo theta) mediante gestos táctiles que se explicarán más adelante.

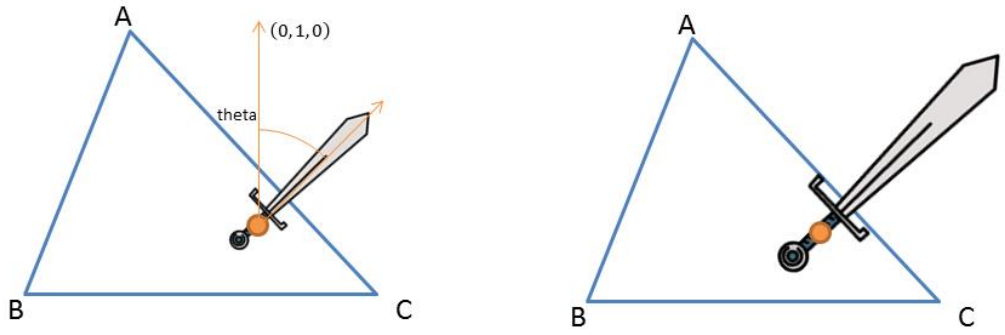


Ilustración 48 – Representación del ángulo theta (izquierda) y factor de escalado (derecha)

Para que la animación sea más realista, se necesita que las pegatinas se orienten manteniendo el mismo movimiento que produzca el algoritmo de deformación. Para ello utilizamos los ángulos iota y kappa que explicamos a continuación.

Llamaremos kappa al ángulo que forma el vector $\overrightarrow{PA}$ con el vector paralelo al eje Y $(0,1,0)$. Este ángulo se conservará durante toda la animación.

Durante el proceso de deformación, el ángulo kappa variará según se modifica el triángulo al que esté asociado. A este nuevo ángulo modificado lo llamaremos ángulo iota.

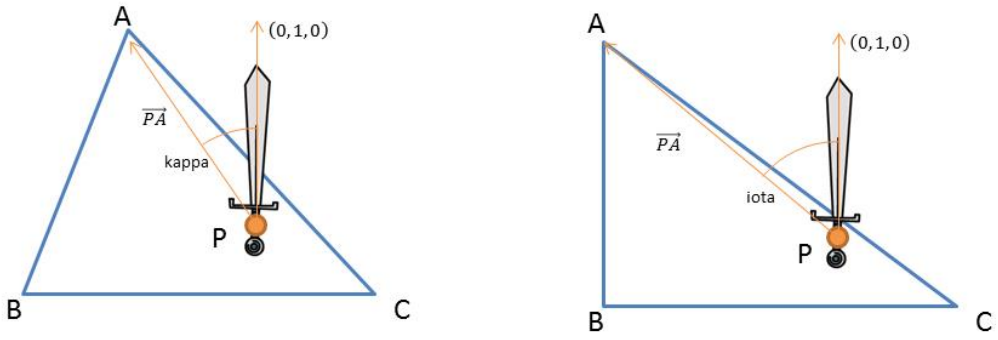


Ilustración 49 – Representación del ángulo kappa (izquierda) e iota (derecha)

Para producir el efecto anteriormente nombrado a la hora de pintar la pegatina, es necesario deshacer el ángulo iota para recuperar la dirección del vector $\overrightarrow{PA}$. Una vez recuperada esta dirección se realiza el giro del ángulo kappa, manteniendo así este ángulo siempre constante respecto al vector $\overrightarrow{PA}$, como se muestra en la siguiente ilustración.

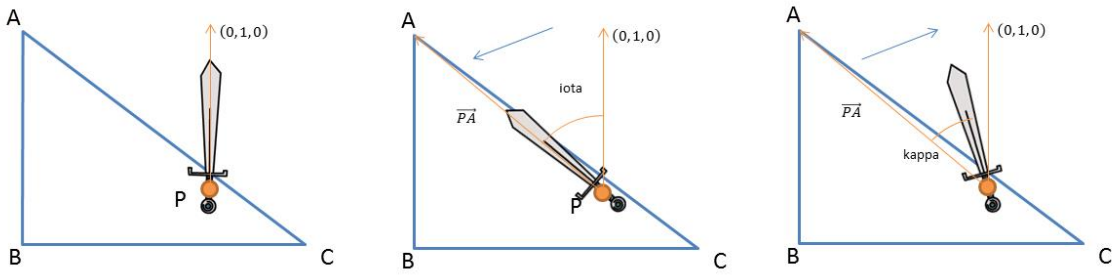


Ilustración 50 – Proceso de animación de las pegatinas
posición inicial (izquierda), giro ángulo iota (centro), giro ángulo kappa (derecha)

Por consiguiente la orientación final vendrá dada por el siguiente ángulo:

$$\varphi = -\iota + \kappa + \theta$$

- HandleArray: Contiene la lista de los handles, incluyendo marcas para los handles seleccionados. Cada handle incluye la siguiente información:

  Float x, y: coordenadas de la posición del handle.
  Short indexT: índice del triángulo al que pertenece el handle.
  Float alfa, beta, gamma: coordenadas baricéntricas dentro del triángulo.
  Boolean selected: si está el handle seleccionado.

X1	Y1	T1	α1	β1	γ1	S1	X2	Y2	T2	…

- EdgeArray: Contiene el conjunto de aristas que forman la malla. Su uso se precisa en el algoritmo de deformación. Cada arista está formada por:

  Short a, b: índice de los vértices que forman la arista.
  Short l, r: índice de los vértices vecinos izquierdo y derecho de los triángulos que comparten la arista. Para las aristas fronterizas -1 en el vértice ausente.

A1	B1	L1	R1	A2	B2	L2	R2	A3	B3	…

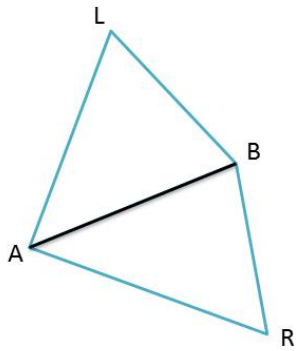


Ilustración 51 - Representación de los vértices vecinos asociados a una arista

5.2 Interacción

Como toda aplicación diseñada para un dispositivo táctil, la interacción con el usuario será fundamentalmente a través de la superficie de la pantalla.

5.2.1 Pantalla Multitouch

La aplicación está pensada para pantallas que dispongan de soporte para la tecnología de múltiples puntos de contacto llamada *multitouch*. Con este objetivo hemos implementado los *callbacks* correspondientes a los siguientes gestos:

- Desplazamiento usando un solo dedo pulsado sobre la superficie de la pantalla.

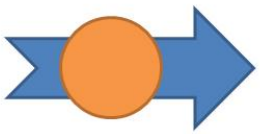


Ilustración 52 - Gesto de desplazamiento

- Ampliación o reducción (*zoom*) usando dos dedos pulsados simultáneamente.

Ilustración 53 - Gesto de ampliación (izquierda) y reducción (derecha)

- Rotación, manteniendo un dedo pulsado fijo y rotando otro dedo alrededor suyo.

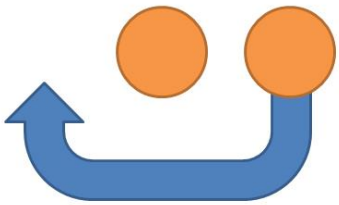


Ilustración 54 - Gesto de rotación

- Restauración usando una doble pulsación rápida.

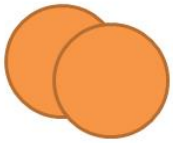


Ilustración 55 - Gesto de restauración

5.2.2 Sensores

Además para aquellos dispositivos que dispongan de sensores, la aplicación será capaz de detectar giros en el eje X. Para ello es necesario que el dispositivo disponga de un sensor de aceleración (acelerómetro) y una brújula (magnetómetro) con los que poder obtener los datos necesarios para realizar estos cálculos.

Durante el cambio de la fase de enemigos y la fase del jefe final del juego, se calibrará el detector tomando como referencia la orientación en ese momento.

Una vez calibrado el detector tomará muestras según se detecte alguna variación de la orientación del dispositivo.

Este cálculo se realiza utilizando las funciones ofrecidas por el paquete android.hardware, dando como resultado el ángulo de rotación respecto al eje X a partir de la orientación calibrada inicialmente.

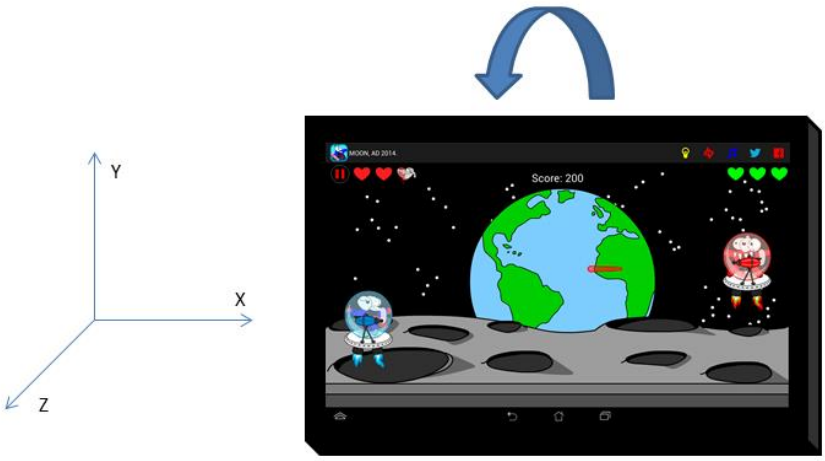


Ilustración 56 – Sensor de giro de la pantalla

6. CONCLUSIONES

Con la realización de este proyecto, nos hemos introducido en el desarrollo de aplicaciones para dispositivos móviles. Un sector de la informática en auge, y que nos brinda muchas posibilidades para el futuro.

El desarrollo de estas aplicaciones implica un reto debido a las limitaciones técnicas y la variedad de este tipo de dispositivos. Hemos tenido que prestar especial atención al uso de la memoria, así como ajustar la aplicación para que funcione adecuadamente con pantallas de diferentes características.

Además de haber aprendido nuevas tecnologías, hemos consolidado nuestros conocimientos en el área de la informática gráfica y en el desarrollo de software en equipo, temas introducidos en la carrera. Por todo ello, nos sentimos especialmente orgullosos de desarrollado con éxito este proyecto.

Creemos que nuestra implementación de las geometrías deformables podría utilizarse integrarse como una API en otro tipo de aplicaciones, como por ejemplo para la realización y edición de videos y animaciones, en la línea de los comics o cartoons.

BIBLIOGRAFÍA

[1] J. F. S. Hill, Computer Graphics using OpenGL, 2ª ed., Upper Sddle River, N.J: Prentice Hall, 2000.

[2] L. P. Chew., «Constrainded Delaunay Triangulations,» *Algorithmica,* vol. 4, pp. 97-108, 1989.

[3] M. d. Berg, O. Cheong, M. v. Kreveld y M. Overmars, Computacional Geometry, Berlin: Springer, 2008.

[4] T. Igarashi, T. Moscovich y J. F. Hughes, «As-Rigid-As-Possible Shape Manipulation,» *ACM Transactions on Computer Graphics,* vol. 24, nº 3, 2005.

[5] T. Igarashi y Y. Igarashi, «Implementing As-Rigid-As-Possible Shape Manipulation and Surface Flattening,» *Journal of Graphics, GPU, and Game Tools,* vol. 14, nº 1, 2009.

[6] R. E. W. Hanan Samet, «Storing a Collection of Polygons using Quadtrees,» *ACM Transactions on Graphics,* vol. 4, nº 3, 1985.

[7] «Android Software Development Kit,» [En línea]. Available: http://developer.android.com/sdk/.

[8] «OpenGL ES,» [En línea]. Available: http://www.khronos.org/opengles/1_X.

[9] «Libgdx,» [En línea]. Available: http://libgdx.badlogicgames.com/.

[10] «JAMA,» [En línea]. Available: http://math.nist.gov/javanumerics/jama/.

[11] «Flash Banner Converter,» [En línea]. Available: https://www.flash-banner-converter.com/.

[12] «Incompetech,» [En línea]. Available: http://incompetech.com/.

[13] «SoundBible,» [En línea]. Available: http://soundbible.com/.

[14] «Audionautix,» [En línea]. Available: http://audionautix.com/.

[15] «GRSites,» [En línea]. Available: http://www.grsites.com/archive/sounds/.

[16] «Freesound,» [En línea]. Available: https://www.freesound.org/.