Desarrollo y adaptación de un videojuego clásico a las nuevas tecnologías móviles

Universidad de las Palmas de Gran Canaria

Escuela de Ingeniería Informática

Trabajo de Fin de Grado

Autor: Agustín Arizcun Bello

Tutor: Agustín Trujillo Pino

Tutor: Adrián Rivero Pérez

Las Palmas de Gran Canaria, Enero de 2015

Índice

1. ESTADO ACTUAL Y OBJETIVOS

1.1. OBJETIVOS

El objetivo de este Proyecto de Fin de Grado es adentrarme en el desarrollo de videojuegos. Esta aplicación de la ingeniería informática obtiene como resultados productos que llegan a un público cada vez más amplio y transmite sensaciones, conocimientos y experiencias comparables a las generadas mediante el cine. Cara al futuro deseo enfocar mi carrera profesional al diseño y programación de videojuegos.

Para lograr tal objetivo, este Proyecto de Fin de Grado consistirá en el **desarrollo de un videojuego clásico adaptado a dispositivos móviles**. La elección de un videojuego clásico tiene el objeto de obtener una aplicación con mecánicas sencillas, heredadas de los primeros grandes éxitos de la historia de los videojuegos. Esta elección de un videojuego clásico también pretende despertar ciertas sensaciones en los usuarios, ya que una adaptación de un icono histórico siempre evoca recuerdos en las generaciones más adultas, y despierta curiosidad en las más jóvenes.

Para la elección de este videojuego será necesaria la creación de un gran número de **prototipos** que permitan definir con claridad la idea que se está buscando, y una vez obtenida sea posible continuar con las siguientes fases asegurando que el planteamiento ha sido el adecuado.

El desarrollo de software enfocado a creación de videojuegos no solamente viene motivado por el creciente público de este tipo de productos. Este tipo de desarrollo es una actividad **multidisciplinar** que requiere programación, generación de contenido audiovisual, diseño de mecánicas de juego, concepción de niveles, etcétera. Todos estos componentes se aúnan para crear experiencias en el jugador.

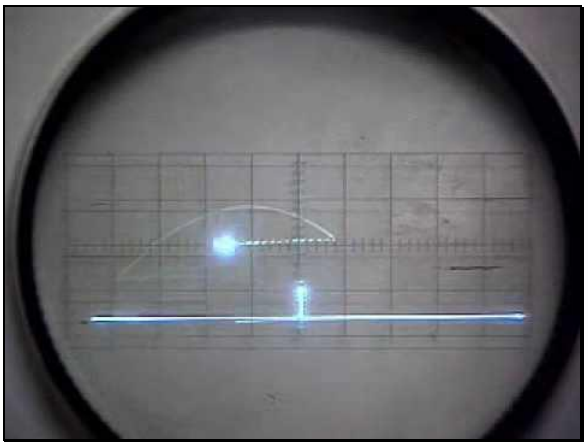
El hecho de que la plataforma de adaptación sea móvil me permitirá explorar entornos de desarrollo completamente desconocidos para mí, y que me enriquecerán de forma técnica. Por otra parte, el público que consume videojuegos móviles es mucho mayor que el de plataformas convencionales, ya que permite al jugador acceder a la aplicación en cualquier momento y lugar.

1.2. HISTORIA DE LOS VIDEOJUEGOS

1.2.1. LOS INICIOS

NOUGHT AND CROSSES

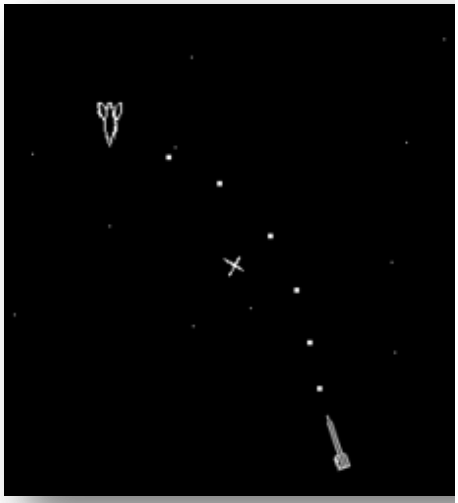
Durante muchos años ha sido difícil determinar cuál fue el primer videojuego existente en la historia. Esta complicación ha surgido debido a las múltiples definiciones de éste que se han ido estableciendo, pero se puede considerar como primer videojuego *Nought and crosses* (también conocido como OXO, Tic-Tac-Toe o Tres en raya). Este videojuego fue desarrollado en **1952** por un estudiante de la Universidad de Cambridge llamado **Alexander S .Douglas**. Este videojuego permitía simular el clásico juego Tres en raya habitualmente practicado sobre papel en una computadora EDSAC. Este videojuego era de un solo jugador, ya que este se enfrentaba directamente a la computadora. Dicho juego no pudo ser popular debido a que el equipo sobre el que se ejecutaba era único,
no existía otro.



TENIS FOR TWO

En **1958** William Higginbotham creó un videojuego para dos jugadores que emulaba el tenis de mesa. Dicho juego se llamaba *Tenis for Two*, y fue creado como como un entretenimiento para los visitantes de la exposición *Brookhaven National Laboratory*. Este videojuego se basaba en un programa de cálculo de trayectorias y un osciloscopio que se

utilizaba como interfaz para mostrar al jugador el estado de la partida. Este fue el primer videojuego que incluyó animaciones en movimiento.

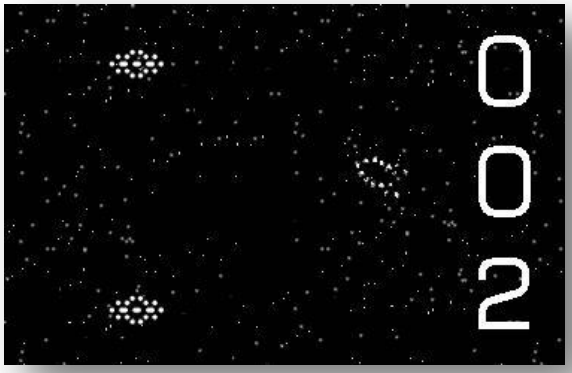


SPACEWAR

En **1961** los ordenadores eran escasos y preciados, pero era posible encontrarlos en los colegios y universidades más prestigiosos, como el MIT. Fue ahí donde surgió el siguiente videojuego, el *Spacewar!*. Fue creado por un estudiante llamado Steve Russel durante seis meses. El juego en cuestión fue desarrollado para equipos DEC PDP-1 y estaba planteado para dos jugadores. Cada uno de estos jugadores debía maniobrar su nave espacial mientras intentaban disparar a la nave contraria. Tanto este como los dos anteriores videojuegos mencionados únicamente fueron disponibles para un reducido número de personas en círculos de investigación. Sin embargo, los que presentaré a continuación ya dejarán de estar reservados para estos grupos académicos.

1.2.2. ECLOSIÓN DE LA INDUSTRIA DE LOS VIDEOJUEGOS

Fue a principios de los 70 cuando dos personas clave, **Ralph Baer** y **Nolan Bushnell**, al fin traerían los videojuegos a las casas y máquinas arcade para que las masas pudieran disfrutarlos. Han sido estos dos visionarios los que dieron el primer paso de lo que hoy conocemos como industria de los videojuegos.



COMPUTER SPACE

Altamente influenciado por *Spacewar*, Nolan comenzó a trabajar en 1969 en recrear *Spacewar* adaptado a un dispositivo de funcionamiento con monedas. A partir del abaratamiento de ordenadores en esas fechas, Nolan decidió fabricar el suyo propio enfocando a soportar únicamente el juego que estaba recreando. Una vez que el primer prototipo estuvo terminado obtuvo apoyo por parte de una empresa de manufacturación, Nutting Associates. Esta compañía ya estaba trabajando en un videojuego trivia sobre ordenadores (*Computer Quiz*), pero apostó por la idea de Nolan y la compraron, renombrándola como ***Computer Space***.

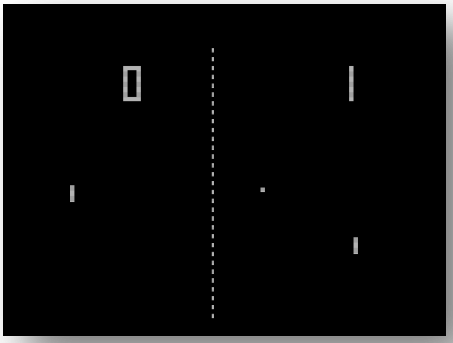
Poco después se habían producido 1.500 equipos con *Computer Space*, pero no tuvo una buena aceptación en el público debido a su complejidad y a lo poco experimentados que eran los jugadores en aquel momento inicial.

Reflexionando sobre el resultado que tuvo *Computer Space*, Nolan pensó que podría hacer un mejor trabajo de marketing y decidió crear su propia compañía para producir juegos arcade. Esta compañía sería **Atari**.

Desde el nacimiento de Atari, la compañía fue sinónimo de videojuegos. Como proyecto inicial, Nolan decidió combinar las físicas creadas para *Computer Space* con la idea del ping-pong para crear lo que terminaría

siendo un videojuego con un impacto a nivel mundial: ***Pong***. Este fenómeno conocido como el primer videojuego popularmente extendido nació en **1972**.

Tras el *Pong* comenzaron a surgir en los salones recreativos videojuegos como *Space Invaders* (Taito, 1978) o *Asteroids* (Atari, 1979). Durante estos años se realizaron numerosos avances técnicos en los videojuegos, y comenzaron a surgir más compañías interesadas en el nicho de mercado.



PONG (IZQUIERDA) Y SPACE INVADERS (DERECHA)

1.2.3. DÉCADA DE LOS 8 BITS (1980-1989)

Los años 80 comenzaron con un fuerte crecimiento en el sector del videojuego alentado por la popularidad de los salones de máquinas recreativas y los primeros equipos dedicados a videojuegos creados durante la década de los 70.

COMMODORE 64 (IZQUIERDA) Y TURBOGRAFX (DERECHA)

Durante estos años destacan sistemas como *Oddyssey 2* (Phillips), *Intellivision* (Mattel), *Colecovision* (Coleco), *Atari 5200*, *Commodore 64* o *Turbografx* (NEC).

Apartando los avances técnicos de las máquinas previamente mencionadas, el auge de esta generación vino dado por juegos tan icónicos como el famoso ***Pacman*** (Namco), ***Battle Zone*** (Atari), ***Pole Position*** (Namco), ***Tron*** (Midway) o ***Zaxxon*** (Sega).

El negocio asociado a esta nueva industria alcanzó grandes cosas en estos primeros años de los 80. Sin embargo, en 1983 comenzó la denominada crisis del videojuego, la cual afectaba principalmente a Estados Unidos y Canadá y que terminó en 1985. Esta crisis surgió debido a la inmensa producción de videojuegos y consolas de baja calidad, provocando una pérdida de confianza en los clientes que consumían este tipo de productos. Un alto número de empresas de desarrollo de videojuegos cerraron durante este periodo.

Durante esta misma generación Japón apostó por el mundo de las consolas con el éxito de la **Famicom** (deniminada en occidente como *Nintendo Entertainment System*), lanzada por Nintendo en 1983 mientras en Europa se decantaba por microordenadores como *Commodore 64* o el *Spectrum*.

Tras finalizar la crisis de los videojuegos en Estados Unidos, los norteamericanos continuaron con el modelo de los japoneses y adoptaron la NES como principal sistema de videojuegos. A lo largo del tiempo restante de esta década surgieron nuevos sistemas domésticos, como la **Master System** (*Sega*), el **Amiga** (*Commodore*) y el **7800** (*Atari*) con juegos considerados hoy en día como clásicos. Uno de los juegos más populares de este grupo de consolas fue el famoso **Tetris** (Alekséi Pázhitnov, 1984).

TETRIS

Fue a finales de los 80 cuando comenzaron a surgir consolas de 16 bits como la Mega Drive de Sega y los microprocesadores fueron lentamente sustituidos por las computadoras personales basadas en arquitecturas de IBM.

En 1985 apareció ***Super Mario Bros***, que supuso un punto de inflexión en el desarrollo de los videojuegos. Esto fue debido a que los juegos lanzados anteriormente constaban de unas pocas pantallas que se repetían en bucle y los objetivos consistían simplemente en conseguir una puntuación alta. Sin embargo, este juego desarrollado por Nintendo supuso un estallido de creatividad. Por primera vez en la historia se contaba con un objetivo y un final del videojuego. A partir de este lanzamiento numerosas compañías emularon este estilo de juego.

Otra rama interesante de la evolución de los videojuegos ha sido la de las consolas portátiles. Estas comenzaron a principio de los 70, con los primeros juegos completamente electrónicos lanzamos por Mattel, los cuales disponían de escasos elementos que le permitieran ser considerados videojuegos, pero que dieron paso a otras máquinas recreativas o a adictivos minijuegos como Game & Watch (Nintendo). La evolución definitiva de este formato de consolas se produjo en 1989, con el lanzamiento de la Game Boy (Nintendo).

1.2.4. REVOLUCIÓN DE LAS 3D (1990-1999)

A principios de los años 90 las videoconsolas dieron un importante salto técnico gracias a la llamada "generación de 16 bits", compuesta por la **Mega Drive**, la **Super Nintendo Entertainment**, la **PC Engine** (*NEC*) y la **CPS Changer** (*Capcom*).

Esta generación supuso un aumento importante de la cantidad de jugadores, así como la introducción de tecnologías como el CD-ROM, elemento que aportaría numerosos cambios al desarrollo de videojuegos.

DOOM (IZQUIERDA) Y DONKEY KONG COUNRY (DERECHA)

Al principio de esta década diversas compañías habían comenzado a trabajar con entornos tridimensionales, obteniendo resultados como **Doom** o **Alone in Dark**. Algunas de las antiguas consolas, como la *SNES* producirían algunos juegos 3D pre-renderizados de SGI, como es el caso de **Donkey Cong Country** y **Killer Instrinct**.

En poco tiempo los videojuegos 3D fueron ocupando un importante lugar en el mercado, principalmente gracias a la llamada "generación de 32 bits" en videoconsolas como Sony PlayStation y Sega Saturn, y la "generación 64 bits" en videoconsolas como Nintendo 64 y Atari Jaguar. Con respecto a los ordenadores, en esta generación se crearon las aceleradoras 3D.

Por su parte los videojuegos portátiles, producto de las nuevas tecnologías más poderosas, comenzaron su verdadero auge, uniéndose a la **Game Boy** máquinas como **Game Gear** (*Sega*), **Linx** (*Atari*) o la **Neo Geo Pocket** (*SNK*), aunque ninguna pudo hacer frente a la popularidad de la Game Boy.

Hacia finales de la década, la consola más popular era la PlayStation con juegos como **Final Fantasy VII** (*Square*), **Resident Evil** (*Capcom*), **Winning Eleven 4** (*Konami*), **Gran Turismo** (*Polyphony Digital*) y **Metal Gear Solid** (*Konami*).

Mientras tanto, en PC lideraban los videojuegos *FPS* (*First Person Shooter* o juegos de acción en primera persona) como **Quake** (*ID Softare*), **Unreal** (*Epic Megagames*) o **Half-Life** (*Valve*). Otro género que abarcó gran parte del mercado de videojuegos de PC eran los *RTS* (*Real-Time Strategy* o juegos en tiempo real), como **Command & Conquer** (*Westwood*) o **Starcraft** (*Blizzard*). Además, la expansión del acceso a Internet facilitó el juego multijugador, y esto dio lugar a los populares *MMORPG* (*Massively Multiplayer Online Role-Playing Game* o juegos de rol multijugador online) como el **Ultima Online** (*Origin*). Finalmente en 1998 aparecería en Japón la **Dreamcast** (*Sega*) y daría comienzo a la "generación de los 128 bits".

2. ORGANIZACIÓN

Esta memoria está formada por 12 capítulos. El **primero**, es de carácter introductorio. Como ya se ha expuesto, en él se señalan de forma general las características del proyecto, así como su finalidad y su motivación. También se realiza una puesta en contacto con la historia de los videojuegos, imprescindible para comprender la importancia de este tipo de productos que hoy en día generan tanto movimiento.

En este **segundo** capítulo se resume la estructura u organización general que presenta este documento.

En el **tercer** capítulo se indican las competencias que se han cumplido a lo largo del desarrollo de este proyecto, así como una breve explicación de las tareas realizadas para cumplir y asimilar cada una de ellas.

En el **cuarto** capítulo se recogen las diferentes aportaciones, tanto personales como a la sociedad, del ámbito escogido para la realización de este trabajo.

El **quinto** capítulo se realiza una introducción a la Normativa y Legislación vigente a nivel nacional, así como diferentes conceptos y clasificaciones importantes a tener en cuenta en este ámbito de acción.

El **sexto** capítulo recoge la planificación seguida para la realización del proyecto, dividiendo la carga de trabajo en tareas. Asimismo, estas tareas han sido temporalizadas de acuerdo al tiempo real consumido durante la realización del trabajo.

El **séptimo** capítulo recoge una serie de análisis y tomas de decisiones de diferente índole, véase la elección del entorno de desarrollo, plataforma de lanzamiento final y videojuegos a prototipar. Además, cada uno de estos análisis va seguido de una conclusión, en la cual se explican los motivos de cada decisión tomada.

El **octavo** capítulo recoge la fase de prototipado, en la cual se detallan las diferentes versiones por las que pasó cada uno de los prototipos, así como las decisiones tomadas una vez fueron finalizados.

En el **noveno** capítulo se detallan todos los pasos seguidos durante la etapa de desarrollo. En este capítulo se mencionan los detalles de implementación más destacables, así como un breve análisis del juego original finalmente adaptado.

En el **décimo** capítulo se aporta una conclusión del trabajo realizado y lo que ha supuesto en diferentes aspectos.

En el **undécimo** capítulo se referencian las fuentes más importantes de las que se han hecho uso durante la realización de todo el desarrollo del proyecto.

Finalmente, en el **duodécimo** capítulo se adjuntan algunos apéndices cuya aportación es interesante, aunque no imprescindible para la memoria.

3. COMPETENCIAS

A continuación se hará referencia a diferentes competencias que se han cubierto durante la realización de este Proyecto de Fin de Grado, así como una breve explicación de las mismas.

CII01 – Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

Durante cada punto de la realización de este proyecto se ha conseguido esta competencia, ya que desde las fases iniciales de prototipado hasta la implementación final, pasando por el diseño, se ha prestado especial atención a los aspectos de calidad y fiabilidad del videojuego resultante.

CII02 – Capacidad para planificar, concebir, desplegar y dirigir proyectos, servicios y sistemas informáticos en todos los ámbitos, liderando su puesta en marcha y su mejora continua y valorando su impacto económico y social.

Desde el planteamiento de este proyecto se ha tenido en cuenta el impacto social que este produciría, así como su utilidad final. En los capítulos de prototipado e implementación se detalla cada uno de los aspectos de la planificación realizada para poder organizar de forma consistente el desarrollo.

CII18 – Conocimiento de la normativa y la regulación de la informática en los ámbitos nacional, europeo e internacional.

En el capítulo dedicado a Normativa y Legislación se cubre esta competencia, analizando la normativa y la regulación de la informática en los ámbitos nacional, europeo e internacional.

TFG01 – Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas.

La realización completa de este Proyecto de Fin de Grado cubre esta competencia, puesto que ha sido necesario aplicar un gran conjunto de los conocimientos adquiridos durante la carrera para cumplir de forma profesional todos los retos que ha supuesto este desarrollo.

4. APORTACIONES

El desarrollo de este Proyecto de Fin de Grado me ha servido para mejorar mis conocimientos en la planificación, dirección y realización de proyectos. Uno de los objetivos principales que he cumplido ha sido conseguir una visión global tanto del desarrollo de videojuegos como de la realización de un proyecto genérico, desde la propia concepción de la idea hasta la redacción final de la memoria. Este proceso aporta al estudiante infinidad de conocimientos relativos al ámbito de desarrollo escogido, y permite descubrir e investigar tecnologías muy dispares, así como llevar a cabo estudios en los que es necesario realizar búsquedas intensivas de información y aplicar un cierto criterio cara a las conclusiones.

Desde los comienzos de la historia del ser humano, el videojuego siempre ha sido una importante herramienta de aprendizaje de conductas, actitudes sociales y prácticas. Este es uno de los métodos de transmisión de conocimiento más eficaces conocidos hasta el momento. No solamente los seres humanos lo utilizamos para transmitir experiencias y asentarlas de forma eficaz en otros individuos, también los animales entrenan a sus crías desde temprana edad con simulaciones de caza mediante juegos.

El juego ofrece la ventaja de asimilar conceptos mediante la propia experiencia, de acciones propias y de otros. Y, los videojuegos, no son más que una versión actual de estos juegos llevados a cabo desde el comienzo de la historia.

Son, además, un instrumento tecnológico perfectamente asentado en nuestra sociedad actual, sirviendo como importante medio de difusión de la cultura. Prácticamente todos los habitantes de los países avanzados disponen de dispositivos en los que pueden ejecutar diferentes tipos de videojuegos. Esto hace que se vuelva una herramienta de difusión de experiencias y conocimiento extremadamente potente y eficaz.

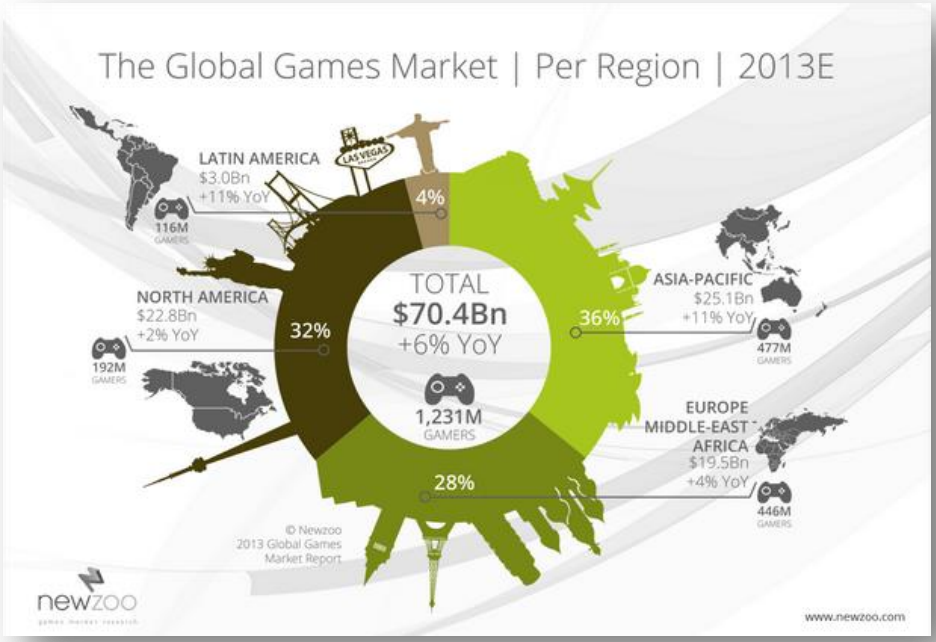
La industria del videojuego ha mantenido en los últimos años su posición como principal industria de ocio audiovisual e interactivo, con una cuota de mercado muy superior a la del cine y la música. A pesar del contexto económico y de su efecto en el consumo, el sector del videojuego ha mantenido su apuesta por la innovación, adaptándose a las nuevas tendencias de consumo y encontrando nuevos segmentos de mercado para favorecer el crecimiento de negocio.

Por este motivo, los principales estudios de mercado la sitúan como la industria tecnológica con mayor proyección de crecimiento. Los analistas aseguran que el sector será uno de los protagonistas en la denominada “economía de la innovación” gracias principalmente al desarrollo del canal online de ventas y la multiplicación de las funcionalidades del videojuego, que comienza ya a alcanzar nuevos ámbitos más allá del ocio.

El *Strategic Research Center* de *EAE Business School* presenta el estudio “El gasto en videojuegos en España 2012”14, uno de los muchos estudios de la situación del mercado de videojuegos en España, tanto del total del país como a nivel autonómico, y a escala internacional, observando además la tendencia de este mercado de ocio para el periodo 2013

– 2016. En él se pueden analizar muchos de los números que genera este mercado, a continuación se comentan los más interesantes.

El tamaño del mercado mundial de videojuegos en el año 2012 es de 31.909 millones de euros. Desde 2007 ha crecido todos los años (excepto en 2009, cuando cayó más de un 8%), mostrando porcentajes de crecimiento en torno al 10% en 2010, 2011 y 2012. Las previsiones para el periodo 2013 – 2016 muestran que el mercado global de videojuegos pasará de los 31.909 millones de euros de 2012 a los 47.024 millones en 2016, un crecimiento del 47% en apenas cuatro años y un crecimiento medio anual superior al 10%. Según el estudio de EAE, en 2016 los principales mercados serán EEUU, Japón y Alemania con 15.000, 4.200 y 3.300 millones de euros de gasto respectivamente.



En España, el 91,7% de los videojuegos que se venden se han desarrollado para videoconsolas, siendo el restante 8,3% del mercado para videojuegos desarrollados para ordenador, bien sea PC o para Mac. En lo que se refiere a la distribución de estos productos, el estudio de EAE pone de manifiesto que se encuentra muy concentrada en tres canales: distribuidores de electrónica (con un 56,4% del mercado), distribuidores de música, video y libros (con un 19,3%) y supermercados e hipermercados (con un 13,8%). Se prevé que el mercado español de videojuegos para los años 2013 – 2016 sea de 824 millones de euros, un crecimiento del 20,27% respecto la cifra actual.

Teniendo en cuenta el gran impacto de este tipo de producto y la importante cantidad de herramientas gratuitas de desarrollo convierten la programación y diseño orientado a la creación de videojuegos en una opción profesional realmente interesante a tener en cuenta.

5. NORMATIVA Y LEGISLACIÓN

A continuación se reflejan aspectos de la normativa y legislación principalmente a nivel nacional. Además se incluye un apartado de clasificaciones de los diferentes tipos de delitos informáticos reconocidos.

5.1. DELITOS INFORMÁTICOS

El **delito informático** implica actividades criminales que los países han tratado de encuadrar en figuras típicas de carácter tradicional, tales como robos, hurtos, fraudes, falsificaciones, perjuicios, estafas, sabotajes. Sin embargo, debe destacarse que el uso de las técnicas informáticas ha creado nuevas posibilidades del uso indebido de las computadoras lo que ha creado la necesidad de regulación por parte del derecho.

Se considera que no existe una definición formal y universal de delito informático pero se han formulado conceptos respondiendo a realidades nacionales concretas: "no es labor fácil dar un concepto sobre delitos informáticos, en razón de que su misma denominación alude a una situación muy especial, ya que para hablar de "delitos" en el sentido de acciones típicas, es decir tipificadas o contempladas en textos jurídicos penales, se requiere que la expresión "delitos informáticos" esté consignada en los códigos penales, lo cual en nuestro país, al igual que en otros muchos no han sido objeto de tipificación aún."

En 1983, la Organización de Cooperación y Desarrollo Económico (OCDE) inicio un estudio de las posibilidades de aplicar y armonizar en el plano internacional las leyes penales a fin e luchar contra el problema del uso indebido de los programas computacionales.

En 1992 la Asociación Internacional de Derecho Penal, durante el coloquio celebrado en Wurzburgo (Alemania), adoptó diversas recomendaciones respecto a los delitos informáticos, entre ellas que, en la medida que el Derecho Penal no sea suficiente, deberá promoverse la modificación de la definición de los delitos existentes o la creación de otros nuevos, si no basta con la adopción de otras medidas como por ejemplo el "principio de subsidiariedad".

Se entiende Delito como: "acción penada por las leyes por realizarse en perjuicio de algo o alguien, o por ser contraria a lo establecido por aquéllas".

Finalmente la OCDE publicó un estudio sobre delitos informáticos y el análisis de la normativa jurídica en donde se reseñan las normas legislativas vigentes y se define **Delito Informático** como "cualquier comportamiento antijurídico, no ético o no autorizado, relacionado con el procesado automático de datos y/o transmisiones de datos."

"Los delitos informáticos se realizan necesariamente con la ayuda de los sistemas informáticos, pero tienen como objeto del injusto la información en sí misma".

Adicionalmente, la OCDE elaboró un conjunto de normas para la seguridad de los sistemas de información, con la intención de ofrecer las bases para que los distintos países pudieran erigir un marco de seguridad para los sistemas informáticos.

1. En esta delincuencia se trata con especialistas capaces de efectuar el crimen y borrar toda huella de los hechos, resultando, muchas veces, imposible de deducir como es como se realizó dicho delito. La Informática reúne características que la convierten en un medio idóneo para la comisión de nuevos tipos de delitos que en gran parte del mundo ni siquiera han podido ser catalogados.

2. La legislación sobre sistemas informáticos debería perseguir acercarse lo más posible a los distintos medios de protección ya existentes, pero creando una nueva regulación basada en los aspectos del objeto a proteger: la información.

En este punto debe hacerse un punto y notar lo siguiente:

- No es la computadora la que atenta contra el hombre, es el hombre el que encontró una nueva herramienta, quizás la más poderosa hasta el momento, para delinquir.
- No es la computadora la que afecta nuestra vida privada, sino el aprovechamiento que hacen ciertos individuos de los datos que ellas contienen.
- La humanidad no está frente al peligro de la informática sino frente a individuos sin escrúpulos con aspiraciones de obtener el poder que significa el conocimiento.
- Por eso la amenaza futura será directamente proporcional a los adelantos de las tecnologías informáticas.
- La protección de los sistemas informáticos puede abordarse desde distintos perspectivas: civil, comercial o administrativa.

Lo que se deberá intentar es que ninguna de ellas sea excluyente con las demás y, todo lo contrario, lograr una protección global desde los distintos sectores para alcanzar cierta eficiencia en la defensa de estos sistemas informáticos.

Julio Téllez Valdez clasifica a los delitos informáticos en base a dos criterios:

1. Como instrumento o medio: se tienen a las conductas criminales que se valen de las computadoras como método, medio o símbolo en la comisión del ilícito.

   Ejemplos:

   - Falsificación de documentos vía computarizada: tarjetas de créditos, cheques, etc.
   - Variación de la situación contable.
   - Planeamiento y simulación de delitos convencionales como robo, homicidio y fraude.
   - Alteración el funcionamiento normal de un sistema mediante la introducción de código extraño al mismo: virus, bombas lógicas, etc.

- Intervención de líneas de comunicación de datos o teleprocesos.

2. Como fin u objetivo: se enmarcan las conductas criminales que van dirigidas en contra de la computadora, accesorios o programas como entidad física.

Ejemplos:

- Instrucciones que producen un bloqueo parcial o total del sistema.
- Destrucción de programas por cualquier método.
- Atentado físico contra la computadora, sus accesorios o sus medios de comunicación.
- Secuestro de soportes magnéticos con información valiosa, para ser utilizada con fines delictivos.

Este mismo autor sostiene que las acciones delictivas informáticas presentan las siguientes características:

1. Sólo una determinada cantidad de personas (con conocimientos técnicos por encima de lo normal) pueden llegar a cometerlos.
2. Son conductas criminales del tipo "cuello blanco": no de acuerdo al interés protegido (como en los delitos convencionales) sino de acuerdo al sujeto que los comete. Generalmente este sujeto tiene cierto status socioeconómico y la comisión del delito no puede explicarse por pobreza, carencia de recursos, baja educación, poca inteligencia, ni por inestabilidad emocional.
3. Son acciones ocupacionales, ya que generalmente se realizan cuando el sujeto atacado se encuentra trabajando.
4. Son acciones de oportunidad, ya que se aprovecha una ocasión creada por el atacante.
5. Provocan pérdidas económicas.
6. Ofrecen posibilidades de tiempo y espacio.
7. Son muchos los casos y pocas las denuncias, y todo ello por la falta de regulación y por miedo al descrédito de la organización atacada.
8. Presentan grandes dificultades para su comprobación, por su carácter técnico.
9. Tienden a proliferar, por lo se requiere su urgente regulación legal.

María Luz Lima, por su parte, presenta la siguiente clasificación de "delitos electrónicos":

1. Como Método: conductas criminales en donde los individuos utilizan métodos electrónicos para llegar a un resultado ilícito.
2. Como Medio: conductas criminales en donde para realizar un delito utilizan una computadora como medio o símbolo.

3. Como Fin: conductas criminales dirigidas contra la entidad física del objeto o máquina electrónica o su material con objeto de dañarla.

5.2. LEGISLACIÓN NACIONAL

Este país quizás sea el que mayor experiencia ha obtenido en casos de delitos informáticos, en Europa.

Su actual Ley Orgánica de Protección de Datos de Carácter Personal (LOPDCP) aprobada el 15 de diciembre de 1999, la cual reemplaza una veintena de leyes anteriores de la misma índole, contempla la mayor cantidad de acciones lesivas sobre la información.

Se sanciona en forma detallada la obtención o violación de secretos, el espionaje, la divulgación de datos privados, las estafas electrónicas, el hacking maligno o militar, el *phreacking*, la introducción de virus, etc.; aplicando pena de prisión y multa, agravándolas cuando existe una intención dolosa o cuando el hecho es cometido por parte de funcionarios públicos.

Así mismo su nuevo Código Penal establece castigos de prisión y multas "a quien por cualquier medio destruya, altere, inutilice o de cualquier otro modo dañe los datos, programas o documentos electrónicos ajenos contenidos en redes, soportes o sistemas informáticos".

5.3. TIPOS DE DELITOS INFORMÁTICOS

La Organización de Naciones Unidas (ONU) reconocen los siguientes tipos de delitos informáticos:

1. **Fraudes cometidos mediante manipulación de computadoras**

   - **Manipulación de los datos de entrada**: este tipo de fraude informático conocido también como sustracción de datos, representa el delito informático más común ya que es fácil de cometer y difícil de descubrir.

   - **La manipulación de programas**: consiste en modificar los programas existentes en el sistema o en insertar nuevos programas o rutinas. Es muy difícil de descubrir y a menudo pasa inadvertida debido a que el delincuente tiene conocimientos técnicos concretos de informática y programación.

   - **Manipulación de los datos de salida**: se efectúa fijando un objetivo al funcionamiento del sistema informático. El ejemplo más común es el fraude del que se hace objeto a los cajeros automáticos mediante la falsificación de instrucciones para la computadora en la fase de adquisición de datos.

   - **Fraude efectuado por manipulación informática**: aprovecha las repeticiones automáticas de los procesos de cómputo. Es una técnica especializada que se

denomina "técnica del salchichón" en la que "rodajas muy finas" apenas perceptibles, de transacciones financieras, se van sacando repetidamente de una cuenta y se transfieren a otra. Se basa en el principio de que 10,66 es igual a 10,65 pasando 0,01 centavos a la cuenta del ladrón n veces.

2. **Manipulación de los datos de entrada**

   - **Como objeto**: cuando se alteran datos de los documentos almacenados en forma computarizada.

   - **Como instrumento**: las computadoras pueden utilizarse también para efectuar falsificaciones de documentos de uso comercial.

3. **Daños o modificaciones de programas o datos computarizados**

   - **Sabotaje informático**: es el acto de borrar, suprimir o modificar sin autorización funciones o datos de computadora con intención de obstaculizar el funcionamiento normal del sistema.

   - **Acceso no autorizado a servicios y sistemas informáticos**: estos accesos se pueden realizar por diversos motivos, desde la simple curiosidad hasta el sabotaje o espionaje informático.

   - **Reproducción no autorizada de programas informáticos de protección legal**: esta puede entrañar una pérdida económica sustancial para los propietarios legítimos. Algunas jurisdicciones han tipificado como delito esta clase de actividad y la han sometido a sanciones penales. El problema ha alcanzado dimensiones transnacionales con el tráfico de esas reproducciones no autorizadas a través de las redes de telecomunicaciones modernas. Al respecto, se considera, que la reproducción no autorizada de programas informáticos no es un delito informático debido a que el bien jurídico a tutelar es la propiedad intelectual.

Adicionalmente a estos tipos de delitos reconocidos, el XV Congreso Internacional de Derecho ha propuesto todas las formas de conductas lesivas de la que puede ser objeto la información.

Ellas son:

- "Fraude en el campo de la informática.
- Falsificación en materia informática.
- Sabotaje informático y daños a datos computarizados o programas informáticos.
- Acceso no autorizado.
- Intercepción sin autorización.
- Reproducción no autorizada de un programa informático protegido.
- Espionaje informático.

- Uso no autorizado de una computadora.
- Tráfico de claves informáticas obtenidas por medio ilícito.
- Distribución de virus o programas delictivos."

6. PLAN DEL PROYECTO

A continuación se indicará el listado de tareas que se ha realizado durante este Proyecto de Fin de Grado, así como el coste temporal vinculado a cada una de ellas.

6.1. TAREAS

El curso de mi trabajo se he dividido en los siguientes bloques de actividades generales:

Tarea	**Descripción**
Tarea 1	Estudio individual de motores gráficos.
Tarea 2	Aprendizaje en profundidad de herramientas de desarrollo del entorno seleccionado.
Tarea 3	Estudio y aprendizaje de técnicas de diseño de videojuegos
Tarea 4	Diseño e implementación inicial de mecánicas. Esta tarea se convirtió en la fase de prototipado.
Tarea 5	Búsqueda y creación de recursos para el desarrollo.
Tarea 6	Evaluación de los prototipos.
Tarea 7	Diseño de la arquitectura del programa.
Tarea 8	Implementación de mecánicas. Incluye pruebas y gestión de animaciones.
Tarea 9	Desarrollo de la interfaz. Incluye pruebas y diseño gráfico.
Tarea 10	Diseño del nivel. Incluye pruebas y diseño gráfico.
Tarea 11	Redacción de la memoria.

6.2. PLANIFICACIÓN TEMPORAL

A continuación indico el tiempo aproximado que me tomó la realización de cada una de las tareas indicadas en el anterior punto:

Tarea	**Duración (horas)**
Tarea 1	10
Tarea 2	40
Tarea 3	20
Tarea 4	80
Tarea 5	15
Tarea 6	15
Tarea 7	10
Tarea 8	40
Tarea 9	20
Tarea 10	30
Tarea 11	20

La realización de todas estas tareas me ha supuesto una inversión de un total de 300 horas en la realización del Proyecto de Fin de Grado.

7. ELECCIONES PREVIAS AL DESARROLLO

7.1. ELECCIÓN DEL ENTORNO DE DESARROLLO

Uno de los primeros aspectos que valorar en la realización de este proyecto fue el entorno de desarrollo que más beneficios pudiera aportar durante la etapa de implementación del mismo. A continuación se indicarán las opciones que se han tenido en cuenta antes de realizar la elección:

UNREAL DEVELOPMENT KIT

Unreal Development Kit es la edición gratuita de Unreal Engine 3, desarrollada por Epic Games, la cual proporciona acceso a un conjunto de herramientas de desarrollo de videojuegos 3D de alta calidad, utilizadas en una gran variedad de productos, como videojuegos móviles, películas digitales, renderizado 3D, grandes producciones de videojuegos, etcétera.

Este entorno está enfocado para desarrolladores de videojuegos, investigadores, estudios de televisión, artistas e incluso estudiantes, y permite generar aplicaciones para PC, iOS y Mac.

UNITY3D

Unity es un entorno de desarrollo que incluye su propio motor de renderizado totalmente integrado con un conjunto completo de herramientas intuitivas y flujos de trabajo rápidos para crear contenido 3D interactivo; publicación multiplataforma sencilla y posibilidad de publicar y conseguir un gran número de complementos en la Comunidad.

Las comodidades que aporta el entorno de Unity elimina muchas barreras de tiempo y de organización, permitiendo realizar muchas tareas de forma sencilla y rápida.

Existen dos modalidades de este entorno de desarrollo:

Unity Free: esta versión gratuita está enfocada a desarrolladores independientes que quieran crear videojuegos sin necesidad de utilizar determinadas funciones Premium que aporta la otra versión. Todo el contenido generado con esta versión se puede publicar sin ningún tipo de problema de licencia. Es posible generar aplicaciones tanto para PC, Mac, versión web, Android, iOS y Windows Phone.

Unity Pro: esta versión de pago comparte las mismas características mencionadas en la anterior versión, pero además aporta otras muchas funcionalidades como el renderizado de texturas, *occlusion culling*, iluminación global, efectos post-procesado o actualización de mallas de navegación. El precio de esta versión asciende a 1.500€.

CONSTRUCT2

Construct2 es un entorno de desarrollo centrado únicamente en desarrollo de videojuegos en 2D. Permite crear videojuegos en HTML5 con una carga mínima de código, siendo gran parte del desarrollo guiada por herramientas visuales.

También existen varias modalidades de este entorno. Se dispone de la versión gratuita, con las herramientas generales necesarias para el desarrollo de cualquier videojuego. Además, existen 3 tipos de versiones de pago, dependiendo del uso que se vaya a dar al entorno: **Personal License** (99€) y **Business License** (329.99€).

Para la realización de este proyecto he escogido como entorno de desarrollo **Unity3D**, concretamente la versión gratuita.

El motivo de esta decisión es la extendida **documentación** que existe sobre las diferentes herramientas que proporciona el entorno tanto en su propia comunidad como en webs externas. Así mismo, este entorno es propicio para que un desarrollador de videojuegos junior cree su primer videojuego, manteniendo el equilibrio entre herramientas visuales y carga programática, sin añadir excesiva complejidad innecesaria a tareas repetitivas durante las diferentes etapas de la realización.

Otro aspecto que inclinó la balanza sobre esta alternativa fue el lanzamiento de la **beta de Unity 4.6** en una fecha cercana al comienzo del desarrollo, siendo este proyecto uno de los primeros generados con esta nueva versión de Unity, la cual mejora un gran número de herramientas, especialmente las relacionadas con la GUI.

7.2. ELECCIÓN DE LA PLATAFORMA DE LANZAMIENTO

La siguiente decisión que fue necesario plantearse fue a qué tipo de dispositivo se centraría el desarrollo.

Si bien ya se había determinado que este proyecto sería un desarrollo para dispositivos móviles, era necesario determinar una plataforma inicial de lanzamiento.

Las opciones a tener en cuenta fueron:

ANDROID

El sistema operativo Android está basado en Linux y desarrollado por Google. Actualmente es el sistema operativo móvil más utilizado a nivel mundial. Dispone de la ventaja de estar instalado en una gran variedad de dispositivos de diferentes características, no estando limitado a una gama específica de teléfonos móviles. El código de este sistema operativo es *open source*. La plataforma de lanzamiento de aplicaciones, Play Store, minimiza las complicaciones existentes para publicar una aplicación en ella, y este es uno de los motivos por los que el 71% de desarrolladores de aplicaciones móviles desarrollen para Android.

iOS

El otro gigante de los sistemas operativos móviles, iOS, desarrollado por Apple Inc. A diferencia de Android, no permite instalar el sistema operativo en hardware de terceros. El desarrollo para iOS requiere una licencia de 99€ al año y su plataforma de lanzamiento de aplicaciones impone requisitos más estrictos para publicar nuevas apps.

Ante estas alternativas, la opción elegida ha sido **Android**. El motivo más importante es el dinero, ya que para la realización de este proyecto no dispongo de ningún tipo de ayuda económica. Omitiendo este factor, Android es el sistema operativo móvil más extendido mundialmente, y las aplicaciones desarrolladas para esta plataforma dispondrán de un mayor público al que llegar.

7.3. ELECCIÓN DE JUEGOS CANDIDATOS

El objetivo de este proyecto es recrear un videojuego clásico a dispositivos móviles, y para lograr un resultado interesante a ojos de los jugadores es necesario coger un juego icónico, que tenga una gran acogida por parte del público que lo juegue por primera vez o de nuevo después de muchos años. Este apego emocional hacia determinados títulos clásicos es un factor muy importante a tener en cuenta, ya que parte del éxito de una adaptación depende del cariño que desprendiera y el impacto que tuviera el título original.

Por lo tanto, cara a la elección del videojuego finalmente adaptado, se realizó un listado inicial con diferentes videojuegos clásicos candidatos:

FROGGER

Frogger es un videojuego arcade publicado originalmente en 1981 por Sega y desarrollado por **Konami**. Este videojuego es un clásico de la época de los salones recreativos, y ha sido reversionado múltiples veces en diferentes formatos.

El videojuego consiste en una pequeña rana que debe cruzar desde una carretera hasta su hogar al otro lado de un río evitando diferentes obstáculos a lo largo de su camino.

Dispone de diferentes niveles de dificultad y es posible jugarlo en modo multijugador, alternando entre J1 y J2 al perder una vida.

Ventajas de esta opción:

- Jugabilidad muy sencilla y fácilmente adaptable a dispositivos móviles táctiles.
- Objetivo sencillo y asequible para un público muy amplio.
- Dificultad fácilmente incrementable para usuarios más experimentados (aumentando la velocidad del entorno).
- Es predecible (el jugador puede mentalizarse de los movimientos y coordinarse con tiempo).
- Aparentemente sencillo de programar.

Desventajas de esta opción:

- Dispone de un solo nivel.
- No dispone de ningún tipo de IA.

CHEEKY MOUSE

Videojuego lanzado en **1980** por **Universal Games**. El objetivo del juego es defender unos trozos de queso en una cocina de ratones que intentan llevárselo. Para ello, el jugador dispone de un martillo y se puede mover lateralmente y golpear con dicho martillo a los ratones cercanos.

A medida que se completan niveles en el juego aumenta la dificultad (mayor número de ratones atacando simultáneamente y ventanas ocultando las trayectorias).

Ventajas de esta opción:

- El movimiento de los ratones es impredecible, y el jugador tiene que tomar decisiones rápidamente, lo que añade diversión al juego.

- Dispone de IA (a medida que se avanza de nivel los ratones avanzan y retroceden de forma cada vez más evasiva para alejarse del jugador).
- Dispone de diferentes niveles.

Desventajas de esta opción:

- Enfocado a jugadores ágiles que sean capaces de desplazarse y atacar simultáneamente de forma coordinada y rápida (público más reducido).

TRON

Videojuego lanzado en **1982** (al mismo tiempo que la película homónima) por **Bally Midway**. El objetivo del juego consiste en resolver 4 pequeños juegos sacados de la película a la que pertenece la trama.

En 1982 se vendieron 800 cabinas de Tron. Un año más tarde, en 1983, se estima que fueron vendidas un total de 10.000 cabinas con el juego, generando unos beneficios de más de 30.000.000$.

Ventajas de esta opción:

- Jugabilidad variada en un mismo videojuego.
- Relativamente conocido por referencias en otros juegos más populares y por las películas propias.

Desventajas de esta opción:

- El jugador debe adaptarse a 4 minijuegos completamente diferentes
- Es necesario desarrollar 4 tipos de juego diferentes, ya que ninguno de los minijuegos de forma independiente es lo suficientemente entretenido como para ser el único atractivo del juego.

- Algunos de los controles de los minijuegos son demasiado complejos para un control desde una pantalla táctil.

PACMAN

Videojuego lanzado en 1982 por **Bally Midway**. Este es el cuarto juego de la serie **Pac-Man**. El objetivo que debe alcanzar el jugador es comer todos los puntos repartidos por el nivel mientras evita diferentes fantasmas que le persiguen. Existen varios tipos de bolas en cada nivel que permite al jugador comer también a los fantasmas del nivel temporalmente.

Este videojuego se ha convertido en uno de los mayores iconos de la época de los salones recreativos.

Ventajas de esta opción:

- Es uno de los juegos arcade más populares y conocido por prácticamente todas las generaciones, incluso por personas que no han llegado a ser jugadores.
- Cuenta con IA para los fantasmas.
- Creación de niveles relativamente fácil una vez esté implementada la jugabilidad básica.

Desventajas de esta opción:

- Demasiado utilizado para las referencias arcade y altamente reversionado.

- Complejidad de la IA, ya que cada uno de los fantasmas dispone de una particular.

Partiendo de este listado de opciones iniciales fue necesario reducir la lista debido a restricciones de tiempo cara a la fase de prototipado. Tras la distribución temporal de las tareas, el listado de opciones que llegaría a la fase de prototipado sería el siguiente:

- **Frogger**
- **Cheeky Mouse**
- **Pacman**

Con estas opciones definidas, comencé a realizar prototipos para evaluar la calidad del posible resultado.

8. PROTOTIPADO

8.1. INTRODUCCIÓN

Todos los desarrollos que tienen como resultado final un producto requieren de una fase de prototipado en la que se pueda experimentar con diferentes conceptos o elementos que conformarán finalmente el resultado con cierto margen de error y en un entorno enfocado a pruebas y feedback.

Una vez se ha concebido un idea para llevar a cabo el desarrollo, es necesario definirla y concretarla. Para ello existen los prototipos: permiten testar una idea antes de que entre en producción, detectar errores, deficiencias, etcétera. Solamente cuando el prototipo está lo suficientemente perfeccionado en todos los sentidos se puede dar paso al proceso de producción.

Cara al desarrollo de videojuegos, el pilar fundamental de cualquier buen producto es su mecánica. Entiéndase por mecánica la jugabilidad básica de un videojuego, las acciones más primarias que se pueden realizar en él, como el movimiento o interacciones básicas. Si esta mecánica no es divertida y agradable para el jugador, el juego final no será un buen producto, ya que se formará en base a esta. Este es el motivo de que la etapa de prototipado sea tan importante.

Durante la fase de prototipado de un videojuego se realizan diferentes tipos de pruebas centradas en los elementos que dan vida al videojuego:

1º Controles

2º Personaje

3º Cámara

Las iteraciones dentro de la fase de prototipado generalmente se realizan en ese orden, debido a que es imperativo conseguir una mecánica agradable para poder continuar con las demás fases. Si este primer punto no se cumple y es imposible encontrarle una solución, es recomendable abortar el proyecto, ya que no dispondrá de ese pilar fundamental en el que apoyarse.

Una vez se ha definido una jugabilidad adecuada se procede a realizar diferentes prototipos de cómo va a ser el personaje protagonista del videojuego. Este será el que reciba gran parte de la atención del jugador durante todo el tiempo, por lo que debe ser característico de alguna forma, agradable a la vista y debe transmitir algún tipo de sensación al usuario acorde a la atmósfera del videojuego. En esta fase inicial aún no se dispone del diseño visual final, pero sí de una aproximación en cuanto a formas y colores.

Por último, la elección del tipo de cámara que se aplicará, ya que será la perspectiva del jugador a lo largo de la historia o sensación que queramos transmitir.

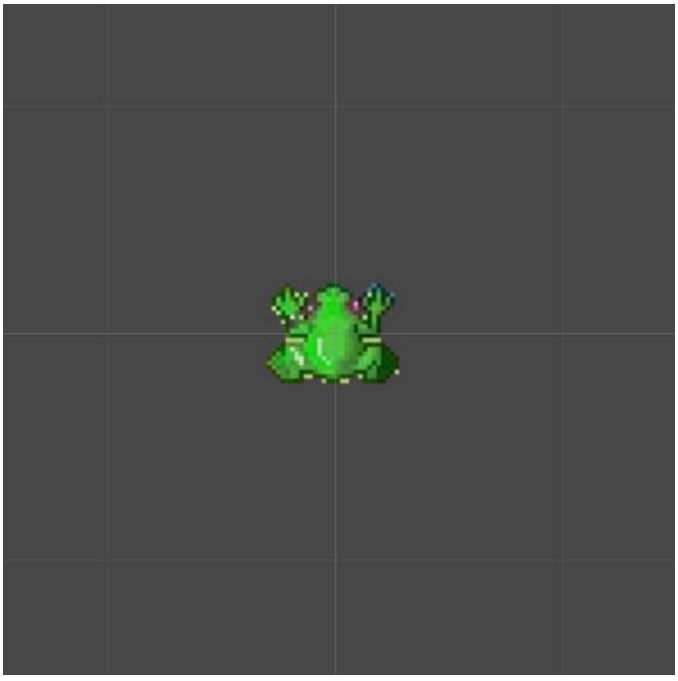
A continuación, para cada uno de los juegos seleccionados comenzaré a explicar el proceso de prototipado que he seguido.

8.2. FROGGER

La realización de estos primeros prototipos fueron mis primeras tomas de contacto con el entorno Unity3D, por lo que el tiempo de realización de cada uno de ellos va en decremento a partir del primero. Este ejercicio no solamente me ha permitido definir una mecánica agradable para el jugador, sino que además me ha aportado experiencia con el uso de las diferentes herramientas que trae Unity3D cara a la realización de los demás prototipos y al desarrollo final.

ALPHA 0.1

La primera versión de la fase de prototipado de Frogger consistió en un jugador que se pudiera mover vertical y horizontalmente en un plano 2D.

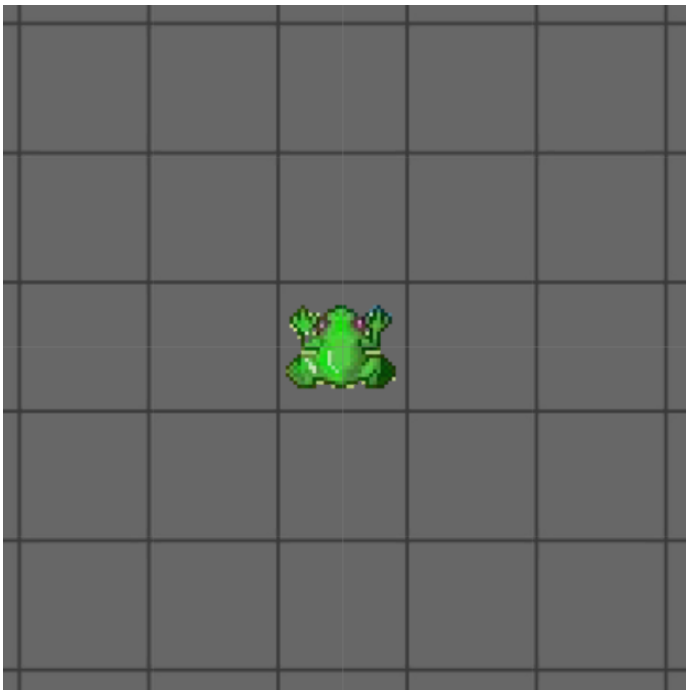


Inicialmente este movimiento se realizaba mediente las teclas “WASD” del teclado, y dicho movimiento era un movimiento fluido y sin restricciones lateral y verticalmente.

Este prototipo sirvió para probar la sensibilidad de las físicas del motor de Unity3D con el movimiento del jugador aplicando diferentes fuerzas sobre el personaje en función del tiempo que este estuviera en movimiento, cambios de movimientos radicales, etcétera.

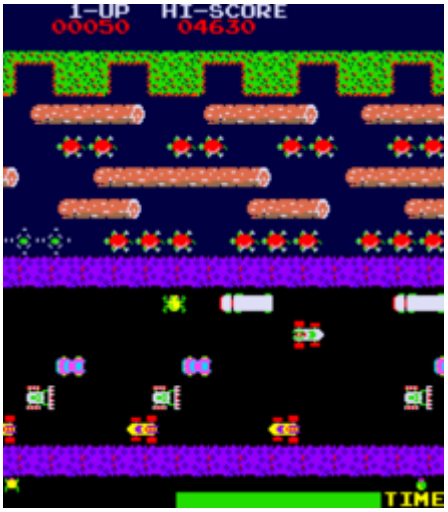
ALPHA 0.2

Una vez implementada la interfaz básica del juego (controles) pasé a definir una matriz de celdas que conformarían el espacio de juego sobre el cual se pudiera desplazar la rana de forma fija.



El motivo de esta matriz de movimiento es la emulación lo más fiel posible del juego original, ya que en este la rana se desplazaba siempre una distancia fija en los 4 sentidos previamente indicados. En esta segunda versión, la rana avanzaba por el tablero vertical y horizontalmente ocupando la casilla contigua cada vez.

El problema de este sistema de movimiento basado de una matriz estaba en que en el juego original, la segunda mitad del escenario consiste en un río con superficies con una velocidad constante. Es decir, si conservo este sistema de movimiento, cara a la versión final, la rana nunca podría estar situada entre dos celdas, y el juego original así lo requiere a lo largo de esa región de movimiento constante lateral.



En la imagen superior se muestran ambas partes del nivel, y se puede observar cómo en la segunda mitad la posición de las superficies no está regida por las matriz de movimiento de la rana.

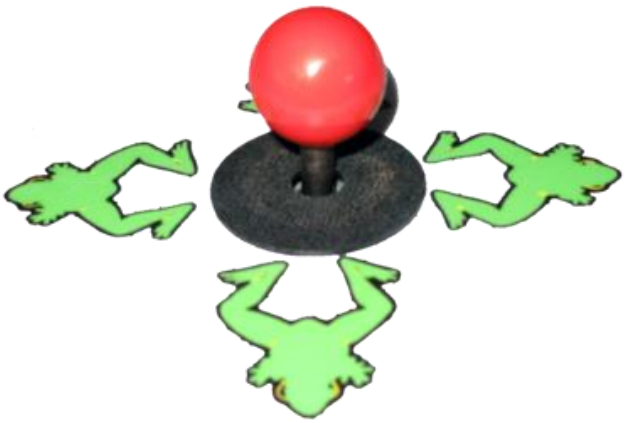
ALPHA 0.3

Ante el problema planteado en la Alpha 0.2 nace esta versión, la cual utiliza como traducción de los input de movimiento un desplazamiento fijo en una dirección, pero sin estar ligado a la posición de las cuadrículas.

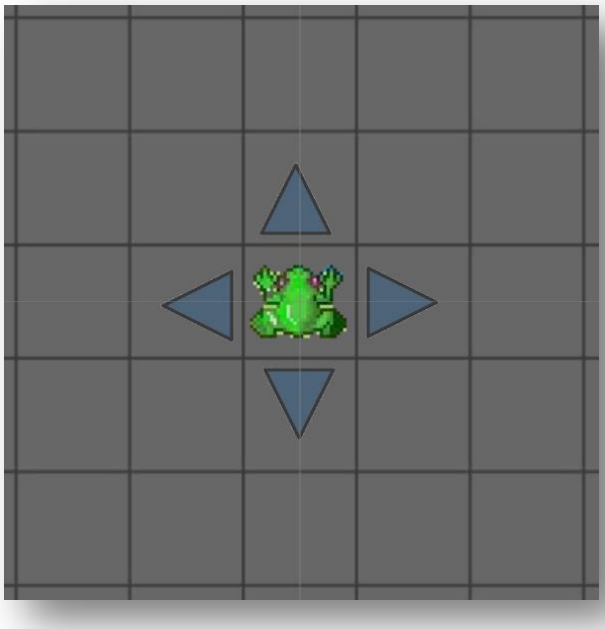
Esto permite simular el movimiento en cuadrícula en la zona inicial del nivel y realizar movimientos laterales constantes al descansar sobre una superficie móvil, pudiendo continuar con los movimientos verticales y horizontales sin ningún tipo de inconveniente.

ALPHA 0.4 – MECÁNICA 1

Una vez definido el tipo de movimiento que realiza el protagonista del videojuego pasé a definir la primera forma de jugar esta adaptación.

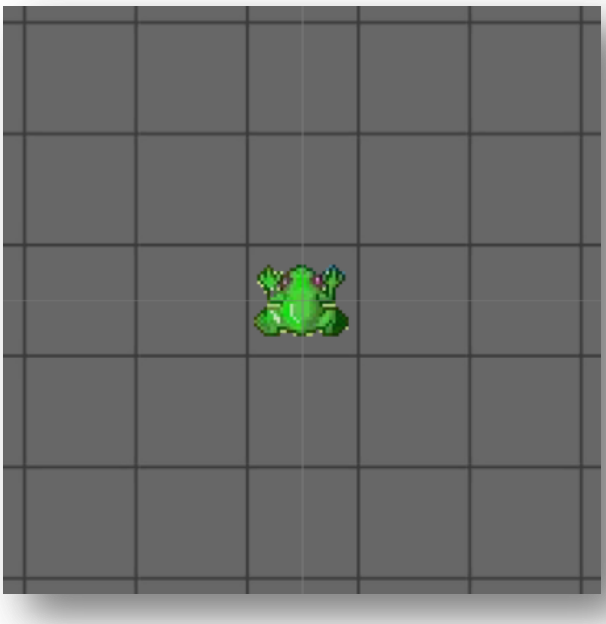


En el videojuego original se disponía de una palanca de 2 ejes con la que se podían realizar los movimientos verticales y horizontales. La primera aproximación para dispositivos móviles que planteé fue unas regiones de control de movimiento con forma de flechas ubicadas alrededor del personaje, de forma que al tocar una de estas flechas la rana se movería en la dirección indicada.



ALPHA 0.5 – MECÁNICA 2

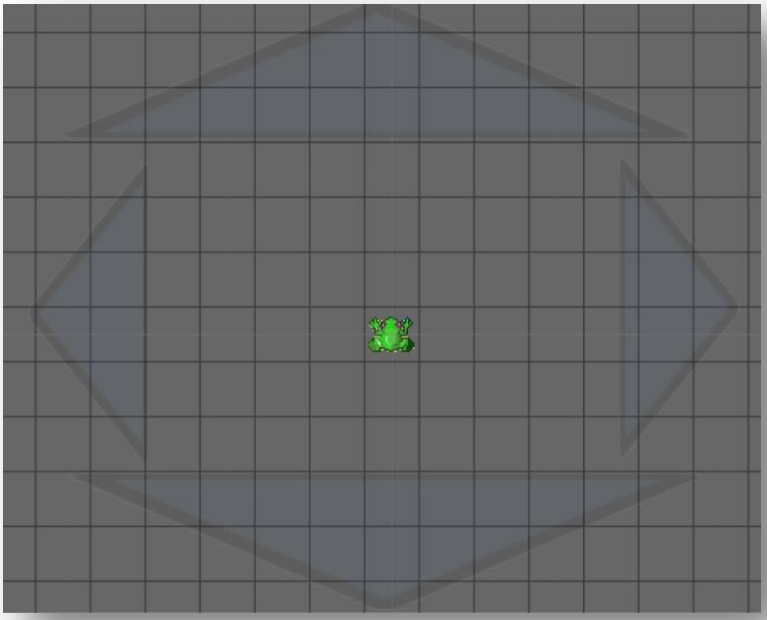
En contraposición con la anterior jugabilidad que emula botones de dirección clásicos, la segunda mecánica que implementé se basa en características más particulares de las pantallas táctiles, como es el *swipe* (deslizamiento) sobre la superficie del teléfono.



En esta versión del prototipo, el jugador puede indicar a la rana que realice movimientos unitarios (conservando las distancias fijas establecidas previamente) con un desplazamiento del dedo sobre la superficie en la dirección deseada.

ALPHA 0.6 – MECÁNICA 3

Otro tipo de jugabilidad planteada hace uso de las dimensiones de la pantalla del dispositivo móvil donde se esté ejecutando el juego. Concretamente, utilizando los límites de esta, es decir, los bordes de la pantalla.



De esta forma, el jugador indica el movimiento deseado tocando un borde u otro de la pantalla. Así, los cuatro bordes del área de juego se convierten en elementos de control de la propia rana.

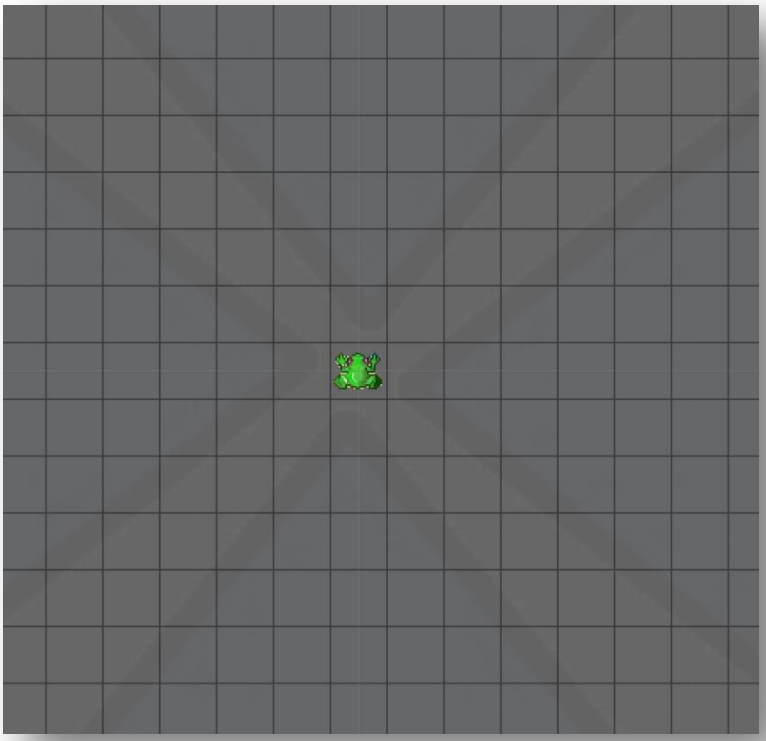
ALPHA 0.7 – MECÁNICA 4

La anterior mecánica planteaba un problema a nivel de intuición, y es que ciertos movimientos resultaban antinaturales. Un caso es el siguiente:

Si la rana estaba posicionada en el punto más inferior de la pantalla, los jugadores que probaron el juego tenían a tocar en una región superior a esta, pero al mismo tiempo cercano, no llegaban a tocar la zona superior de la pantalla. Al suceder esto, técnicamente están tocando en la mitad inferior de la pantalla, aunque sea encima de la rana, por tanto se interpretaba como movimiento hacia abajo, y esto resultaba incómodo y antinatural.

Este tipo de observaciones las obtuve como feedback de pruebas de los prototipos con diferentes personas de mi entorno en cada iteración.

Como mejora a la anterior mecánica surgió esta versión, en la cual el jugador disponía también de cuatro regiones en las que pulsar para indicar una dirección. La diferencia radica en que estas regiones ahora eran relativas a la ubicación del protagonista, en vez de absolutas a la pantalla.



De esta forma, al detectar un pulso en cualquier región superior a la posición de la rana se interpretará como movimiento hacia arriba, e igualmente con el resto de lados.

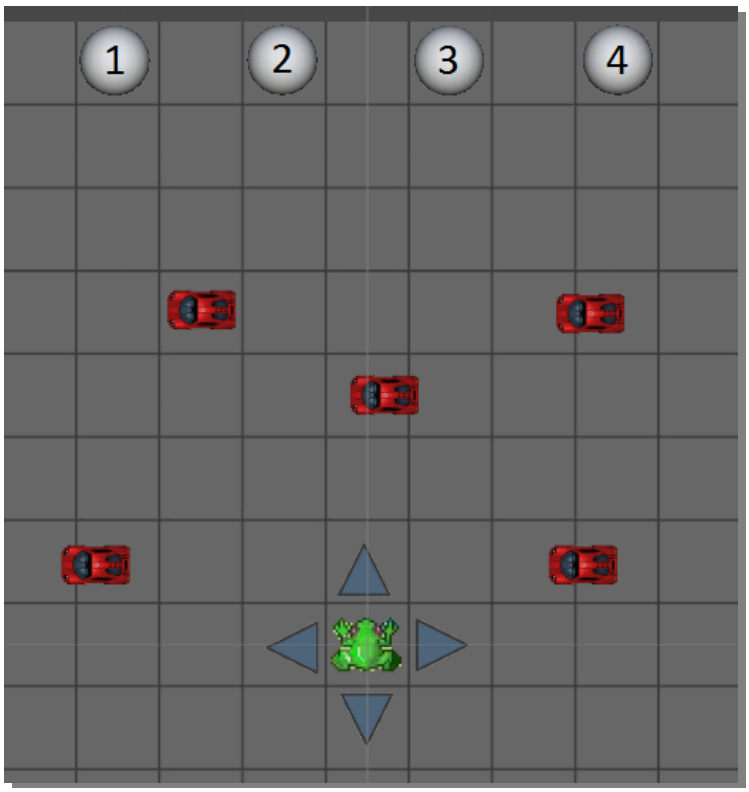
ALPHA 0.8

En esta última versión del prototipo auné las cuatro mecánicas anteriormente mencionadas con el objetivo de poder compararlas en un mismo ejecutable cara a comparativas con diferentes públicos para poder obtener un feedback acerca de cada una de ellas al compararlas de forma más cómoda y sencilla.

Además incorporé objetivos y elementos adversos, intentando realizar una simulación muy primitiva del juego original:

- Los objetivos son unas regiones situadas en la parte superior del nivel, las cuales hay que alcanzar para completarlo. Estas regiones las utilicé para a su vez cambiar de mecánica, siendo cada uno de los "*goals*" un portal a la siguiente mecánica. De este modo es fácil que los usuarios realicen la misma prueba con cuatro tipos de jugabilidades sin salir de la aplicación.
- Los elementos adversos en este caso son vehículos que avanzan lateralmente con una velocidad constante de derecha a izquierda. Estos vehículos se van generando en cada una de las líneas asignadas en intervalos de tiempo aleatorios comprendidos entre 1 y 4 segundos. De esta forma, el jugador puede medir mejor su reacción

con respecto a los controles ante situaciones que requieren una respuesta rápida.



8.3. CHEEKY MOUSE

En la realización de este segundo prototipo ya disponía de ciertos conocimientos básicos acerca del entorno Unity3D que me permitieron avanzar con mayor facilidad, aunque a su vez me planteó nuevos retos que poco a poco fui completando.

En esencia, el videojuego original consiste en eliminar los ratones que descienden para coger el queso. Para ello es posible desplazar al jugador lateralmente e impactar con el martillo en la dirección hacia la que está mirando el protagonista.

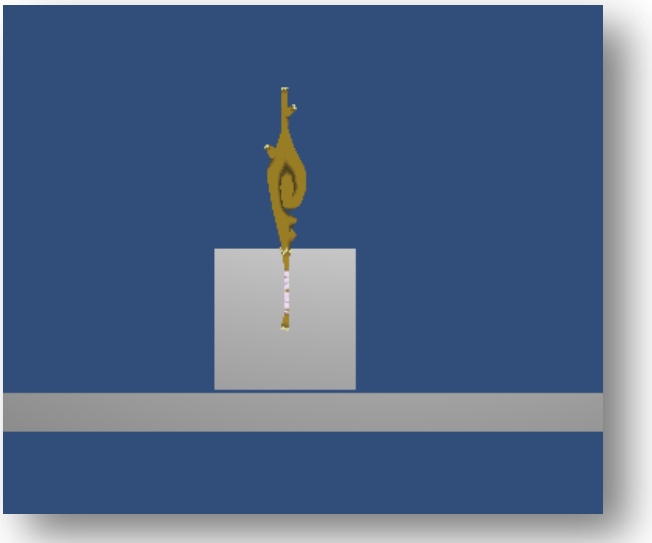
ALPHA 0.1

En esta primera versión implementé un cubo a modo de jugador principal y un entorno sencillo formado por un suelo y dos paredes. Este videojuego se realiza puramente en 2D, siempre en el mismo entorno a excepción de pequeñas variaciones, por lo que los elementos ambientales carecen de importancia cara a la funcionalidad del prototipo.

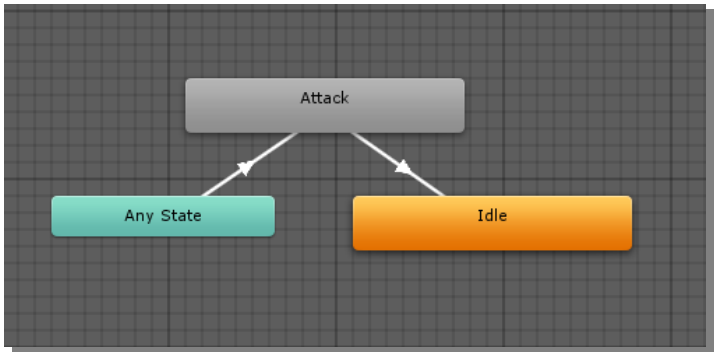
El movimiento de este protagonista viene guiado por pulsos en la pantalla táctil. Cada pulso que se realiza, el controlador de movimiento capta la componente X de la coordenada y aplica una traslación del objeto en el juego hasta dicha posición.

ALPHA 0.2 – MECÁNICA 1

Una vez implementada la mecánica de movimiento procedí a incorporar la segunda mecánica del juego original: impactar con el martillo. Para ello añadí a nuestro protagonista principal una herramienta inicialmente con forma alargada que le permitiera colisionar con los futuros ratones del juego.



Una vez añadido dicho elemento el siguiente paso era generar una animación de este componente, de forma que fuera posible realizar una trayectoria de ida y vuelta, controlando su velocidad y más aspectos técnicos relacionados con colisiones de objetos de juego.

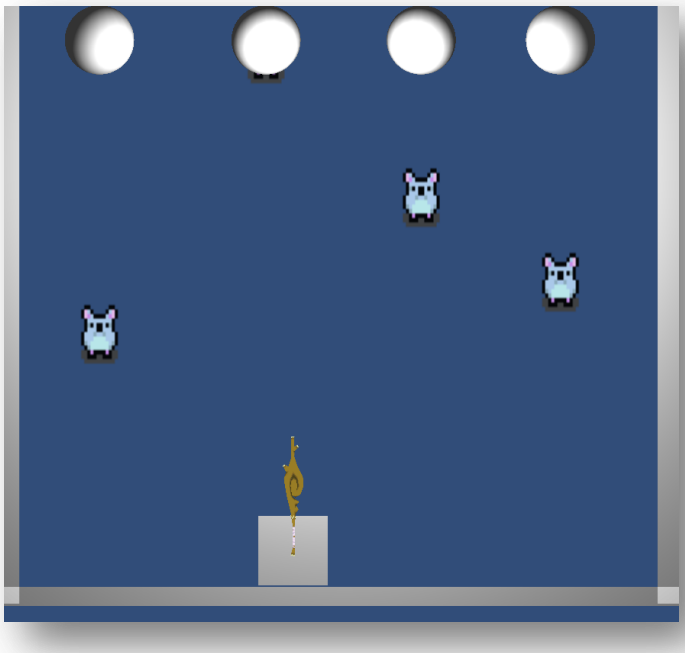


Este controlador de movimiento sencillo permite pasar de cualquier estado (*Any State*) a la animación de ataque (*Attack*) generada, y al finalizar esta volver a un estado de reposo (*Idle*). Este estado de ataque consiste en una transformación de la posición del objeto de ataque hasta impactar con el suelo, así como el recorrido de vuelta. El jugador únicamente podrá eliminar los elementos enemigos durante esta acción, y no mediante colisiones del arma durante el estado de reposo.

El siguiente paso consiste en implementar un método de input que permita ejecutar esta acción de ataque. Y la primera aproximación fue un deslizamiento (*swipe*) sobre la pantalla en la dirección de ataque deseada.

ALPHA 0.3

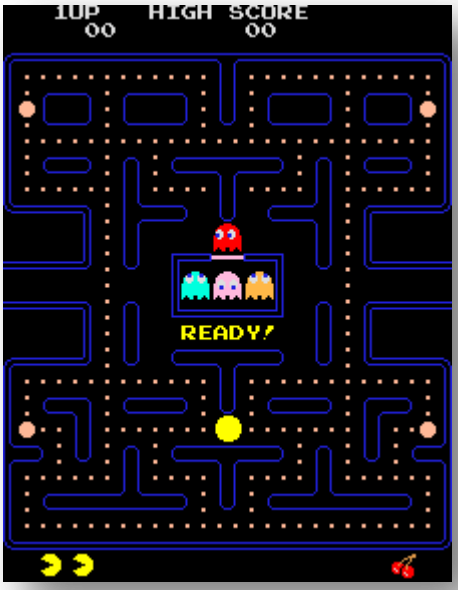
Una vez implementadas las dos mecánicas básicas del juego procedí a incorporar cuatro puntos de generación de ratones enemigos (*spawns*).



En este prototipo los elementos que actúan como ratones descienden a una velocidad constante y atraviesan la plataforma base, al igual que en el videojuego original.

En esta versión ya es posible valorar la respuesta del juego ante ambas mecánicas implementadas con diferentes objetivos móviles. Las sensaciones obtenidas en el resultado de este prototipo no fueron satisfactorias por motivos que explicaré en el apartado de conclusiones de este mismo capítulo. Por este motivo, esta fue la última versión generada de este prototipo.

8.4. PACMAN

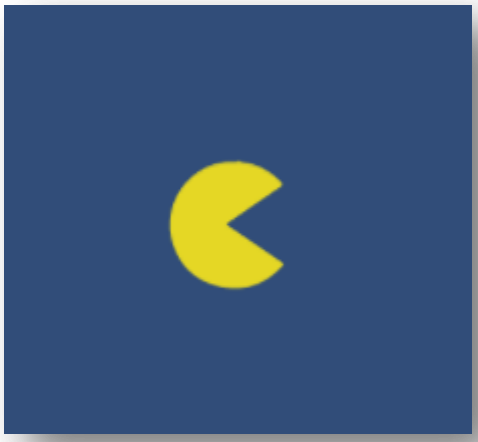


Cara a este tercer prototipo la experiencia acumulada era mayor, y los resultados fueron obtenidos en menor tiempo y con menor esfuerzo. Aunque de nuevo se plantearon nuevos retos, ya que este tercer videojuego planteaba aspectos que ninguno de los anteriores incorporaba.

La mecánica de este popular videojuego es bastante conocida por todos. A modo de síntesis, el protagonista (Pacman) debe limpiar la pantalla de puntos (cocos) mientras huye de fantasmas que intentan eliminarle. Existen cuatro potenciadores que permiten al jugador eliminar a los fantasmas durante un tiempo limitado.

ALPHA 0.1

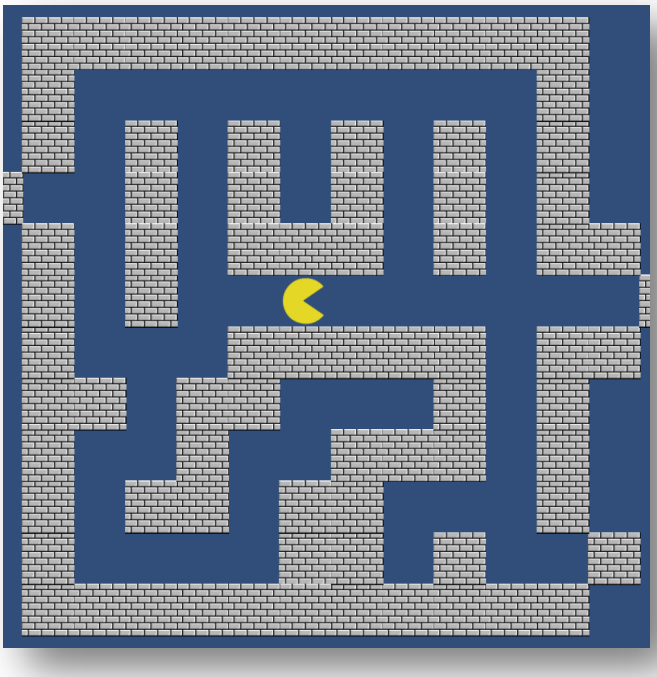
La primera iteración de este prototipo consistió en la generación de un protagonista jugable desde el editor de Unity. Este protagonista puede realizar desplazamientos laterales y verticales visto desde una perspectiva 2D.



Una vez definida una velocidad de movimiento inicial para las pruebas di fin a esta iteración para pasar a incorporar el entorno y la primera versión de la mecánica.

ALPHA 0.2 – MECÁNICA 1

Antes de realizar la implementación de la mecánica generé un entorno básico de pruebas para probar las futuras mecánicas. Para ello ubiqué diferentes paredes a lo largo del espacio de juego, dejando entre ellas siempre un carril con el ancho de nuestro protagonista a modo de caminos.



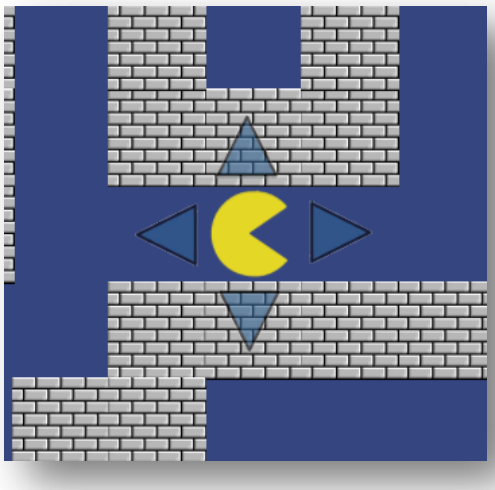
Una vez establecido este escenario de pruebas procedí a implementar el primer tipo de mecánica de movimiento: **deslizamiento en dirección de movimiento**. El jugador debe

desplazar el dedo sobre la pantalla táctil del dispositivo en la dirección deseada para indicar a Pacman que gire.

Este tipo de acción en los terminales móviles es realmente intuitiva y tiene muy buenos resultados cara a los jugadores casuales que utilizan este tipo de dispositivos.

ALPHA 0.3 – MECÁNICA 2

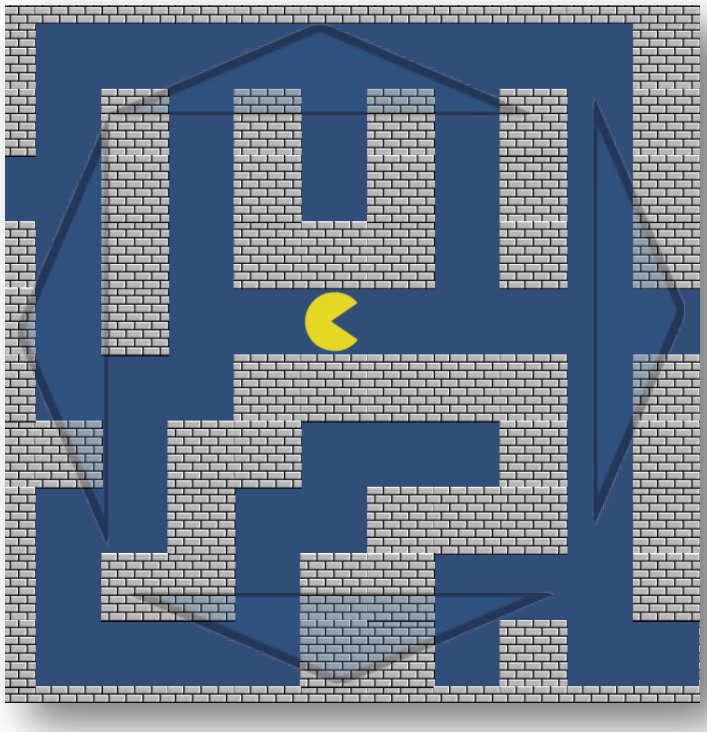
En esta tercera versión implementé la segunda mecánica de movimiento, la cual consiste en flechas de movimiento adyacentes al protagonista en las cuatro direcciones de acción.



Esta mecánica emula de forma más visual el joystick original de este videojuego. Cara a los usuarios menos habituados a videojuegos en general, este tipo de controles les resultan los más directos y autoexplicativos.

ALPHA 0.4 – MECÁNICA 3

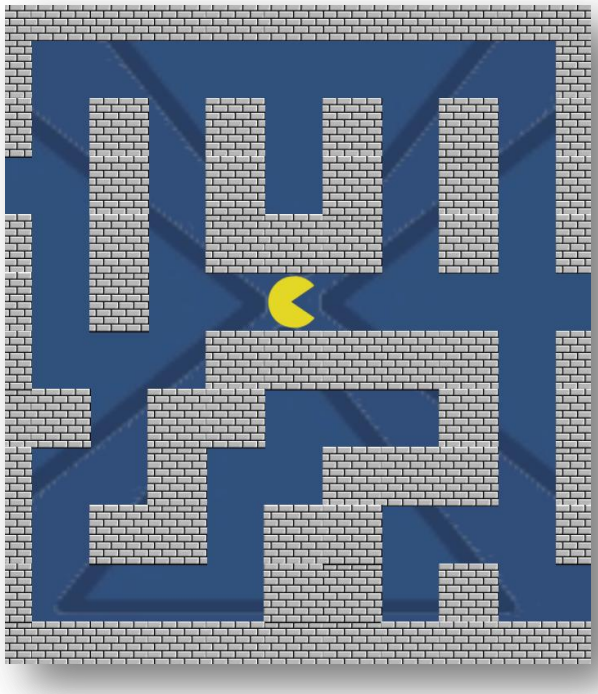
Para la cuarta mecánica utilicé los límites exteriores de la pantalla táctil como áreas de detección de la dirección de movimiento deseada. Es decir, el jugador pulsa cerca del borde que desea, y Pacman se mueve en dicha dirección.



Al igual que en la tercera mecánica implementada para el prototipo de Frogger en la Alpha 0.6, determinadas interacciones para indicar movimiento resultaban antinaturales con respecto a la posición de Pacman. De ese conflicto nace la cuarta y última mecánica.

ALPHA 0.5 – MECÁNICA 4

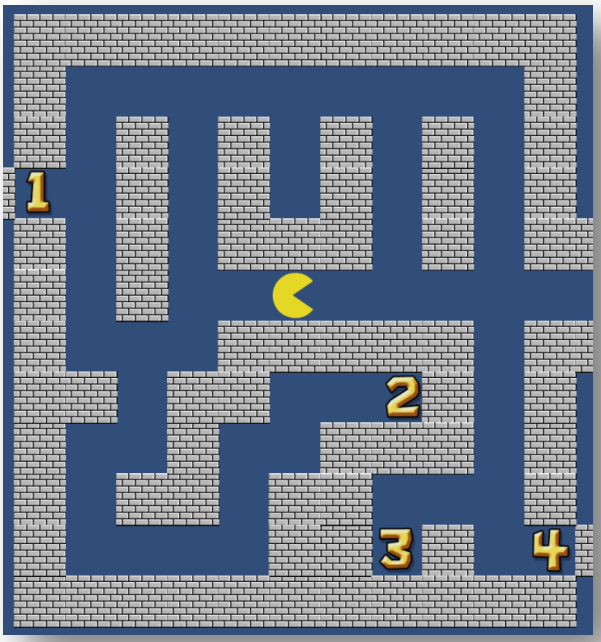
La última versión de la jugabilidad principal del juego consiste en áreas de detección de movimiento relativas a la posición de Pacman, siendo toda la superficie cónica superior al protagonista equivalente al movimiento ascendente, y de igual forma con las otras tres opciones.



Esas regiones visualmente marcadas en el prototipo indican las áreas de detección de cada dirección de movimiento, y son relativas a la ubicación del protagonista, por lo que se desplazan a lo largo del nivel con él. De este modo se consigue un resultado más intuitivo a la hora de indicar una dirección a partir de la superficie táctil colindante a Pacman.

ALPHA 0.6

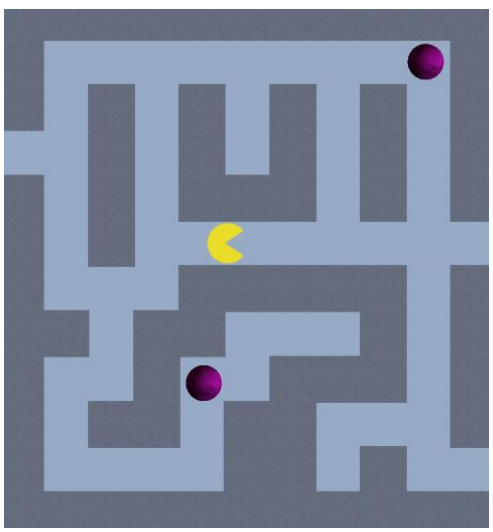
De igual forma que en el primer prototipo realizado en el proyecto, esta versión unifica las cuatro mecánicas previamente en un solo ejecutable. Cada una de las mecánicas es cargada a partir de diferentes objetivos repartidos por el laberinto, indicando con su número la mecánica que cargará.

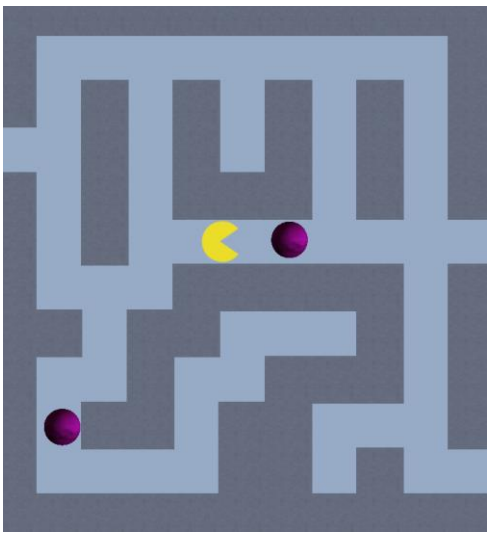


De nuevo, la utilidad de esta herramienta ha sido muy significativa, ya que esta cómoda forma de alternar entre las diferentes formas de interactuar con el videojuego ha hecho que los usuarios que evaluaron las mecánicas no sintieran que se encontraban en un entorno de pruebas.

ALPHA 0.7

El siguiente aspecto clave del juego al que hice frente fueron los enemigos. En el caso del Pacman se trata de fantasmas que, en determinados momentos, persiguen a Pacman. Para ello, para ello incorporé al juego dos esferas en diferentes puntos del laberinto y las doté de cierta inteligencia artificial, de modo que buscaran el camino hasta la posición de Pacman en cada momento.



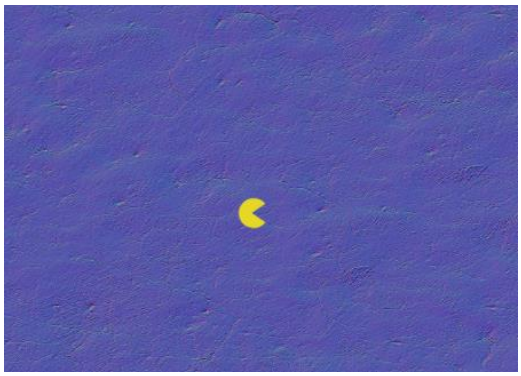


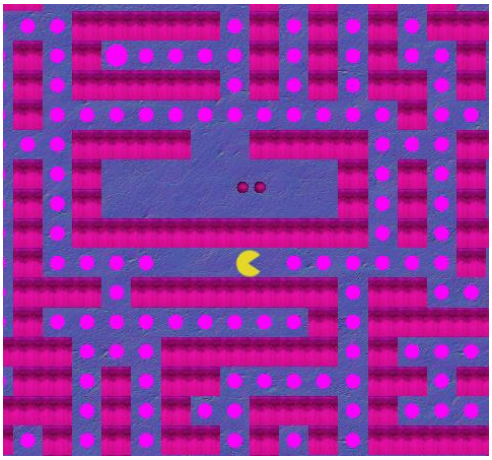
En la primera imagen se observa el estado inicial del juego, con la posición inicial tanto de Pacman como de los fantasmas. En la segunda se muestra el estado del juego transcurridos cinco segundos.

Esta búsqueda por parte de los fantasmas se realiza siempre con respecto a la posición actual de Pacman, por lo que varía a medida que el protagonista va avanzando por el laberinto en tiempo de ejecución.

ALPHA 0.8

En esta nueva versión del prototipo experimenté con la generación del entorno durante la ejecución del juego, de forma que el propio laberinto y otros componentes existentes se generaran al iniciar la aplicación a partir de un mapa predefinido almacenado como string en un fichero.





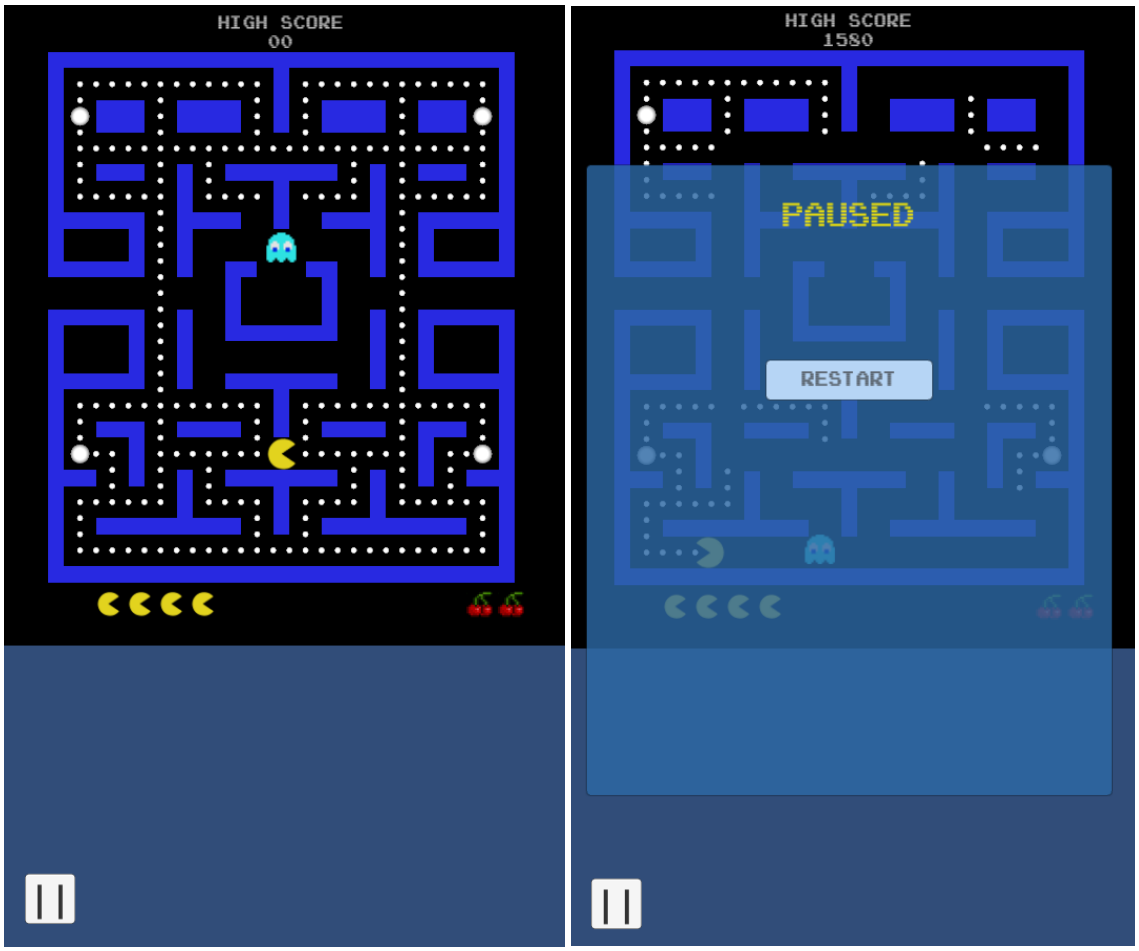
En la primera imagen se observa cómo es el espacio de juego en el editor, y en la segunda se aprecia la generación del entorno en el primer frame de la ejecución del juego.

Este procedimiento aporta numerosas ventajas: la generación de niveles se vuelve relativamente trivial, siendo únicamente necesario editar un fichero de texto con “X” o “.” dependiendo del elemento que se quiera instanciar en cada punto de la cuadrícula. Además, la edición de niveles previamente generados resulta mucho más cómoda.

Este sistema de generación del entorno durante la ejecución fue descartado como método de viable. En el apartado de conclusiones de este mismo capítulo se explican los motivos de esta decisión.

ALPHA 0.9

En esta última versión experimenté con muchos elementos particulares de Unity3D para dar vida a un resultado enfocado a una resolución móvil que pudiera llegar a transmitir una sensación similar a la del videojuego original.



Para obtener este resultado utilicé:

- Controladores de animaciones para Pacman y el fantasma.
- Controlador de colisiones entrantes de Pacman.
- Controlador de nivel (gestión de vidas y puntuación).
- Controlador de vista.
- Interfaz HUD particular de Unity 4.6.
- Control del tiempo de ejecución (pause).

Una vez conseguida esta última versión del prototipo fue más fácil interpretar la sensación que transmitiría el juego final, y este factor es vital para la toma de decisiones inicial cara al desarrollo del proyecto.

8.5. CONCLUSIONES

En este último apartado del capítulo de prototipado explicaré el motivo de cada una de las decisiones de diseño iniciales tomadas en el meridiano del proyecto, previas a la implementación final.

ELECCIÓN MECÁNICA FINAL FROGGER

Las mecánicas creadas para este prototipo fueron las siguientes:

Mecánica	Mecánica 1	Mecánica 2	Mecánica 3	Mecánica 4
Descripción	Desplazamiento mediante flechas de dirección adyacentes al protagonista.	Desplazamiento mediante *swipe* en la dirección deseada.	Desplazamiento mediante áreas de dirección absolutas a la pantalla, fijas en los bordes.	Desplazamiento mediante áreas de dirección relativas al protagonista.

Durante la realización del prototipo del videojuego Frogger realicé varias pruebas con diferentes personas de mi entorno más y menos familiarizadas con videojuegos y con dispositivos móviles. Evalué cada feedback que me proporcionaron y lo utilicé en la propia generación de mecánicas para editar las mismas, y finalmente para seleccionar una u otra definitiva para el prototipo.

En total consulté con 9 personas su opinión sobre las diferentes mecánicas de movimiento, y términos no numéricos el resultado fue el siguiente:

Mecánica	Mecánica 1	Mecánica 2	Mecánica 3	Mecánica 4
Sensación transmitida	Normal	Buena	Mala	Buena

En general, todos los jugadores coincidieron en que la antinaturalidad en muchos casos de la **Mecánica 3** hacía frustrante el movimiento. También hubo un mismo número de opiniones positivas sobre la **Mecánica 2** y la **4**, resultando ambas cómodas para la mayoría de los públicos.

Las generaciones más adultas prefirieron la **Mecánica 4**, mientras que las más jóvenes y habituadas a pantallas táctiles se decantaron por la **Mecánica 2**. Esto es debido a que el *swipe* de la segunda es natural e intuitivo cuando interactúas con este tipo de dispositivos móviles desde joven y se vuelve prácticamente inmediata esa inercia para realizar movimientos a lo largo de la pantalla. Sin embargo, los sujetos menos experimentados preferían algún tipo de señal o marca en pantalla que les indicaran dónde pulsar para definir una dirección.

El problema que planteaba la **Mecánica 1** y por lo cual no tuvo un impacto tan grande en los jugadores de prueba fue el hecho de que a lo largo del juego el usuario tenía que tapar con la mano constantemente una gran parte de la superficie de la pantalla, suponiendo un claro problema, ya que complicaba la detección de los vehículos móviles que se desplazaban y los cuales se debían esquivar.

Debido a todas estas observaciones, las mecánicas candidatas finales de este prototipo fueron la **Mecánica 2** y **Mecánica 4**.

ESTUDIO DE VIABILIDAD DE CHEEKY MOUSE

Este prototipo se detuvo tras el planteamiento de las primeras dos mecánicas en una misma versión jugable. Esto fue debido a las opiniones recolectadas con esta, y observando la

naturaleza y sensaciones que transmitían el juego original con respecto a lo que un juego móvil suele proporcionar.

Todos los usuarios que probaron el conjunto de mecánicas de movimiento + ataque coincidieron en que era confuso realizar dos acciones prácticamente simultáneamente en la misma superficie, simplemente variando el gesto (*touch* + *swipe*). Tanto los jugadores habituales como los más casual afirmaban que no se sentían cómodos y les resultaba frustrante moverse cuando querían atacar y atacar cuando querían moverse. Esta sensación se mantuvo incluso después de experimentar un rato para habituarse.

Este feedback me hizo reflexionar sobre los elementos comunes en los videojuegos móviles. Uno de los más importantes es la sencillez. Un videojuego móvil con una mecánica muy sencilla y divertida conforma la base de un gran videojuego portátil para un público muy amplio.

Por lo tanto, a pesar de que la suma de las mecánicas movimiento + ataque sea cómoda mediante un joystick y un botón de acción en las consolas arcade, esta suma no era tan intuitiva en un dispositivo móvil haciendo uso de características propias de la superficie táctil.

Por este motivo decidí **finalizar** la realización de **este prototipo** en este punto, previendo el poco futuro de las sensaciones que finalmente transmitiría esta adaptación a un dispositivo móvil.

ELECCIÓN MECÁNICA FINAL PACMAN

Las mecánicas generadas para este último prototipo fueron las siguientes:

Mecánica	**Mecánica 1**	**Mecánica 2**	**Mecánica 3**	**Mecánica 4**
Descripción	Desplazamiento con *swipe* en la dirección deseada.	Desplazamiento mediante flechas de dirección adyacentes al protagonista.	Desplazamiento mediante áreas de dirección absolutas a la pantalla, fijas en los bordes.	Desplazamiento mediante áreas de dirección relativas al protagonista.

De nuevo pedí a un total de 9 usuarios probar cada una de estas mecánicas y que me transmitieran sus sensaciones, así como observación por mi parte durante las pruebas. Ante este experimento, los resultados fueron los siguientes:

Mecánica	**Mecánica 1**	**Mecánica 2**	**Mecánica 3**	**Mecánica 4**
Sensación transmitida	Muy buena	Buena	Mala	Buena

Las ventajas y desventajas de cada una de las mecánicas son similares a las del prototipo de Frogger, pero con algunas variaciones.

La constancia de movimiento particular de Pacman hizo que la Mecánica 2, con flechas adyacentes al protagonista, cobrara más fuerza, puesto que las intervenciones sobre su ruta de desplazamiento eran menores. Además, las variaciones del entorno eran menos críticas, ya que no existen amenazas tan constantes como el frenesí de vehículos del primer juego estudiado.

La **Mecánica 3** y **Mecánica 4** comparten la misma situación que en el primer prototipo, resultando una antinatural en determinados casos y la otra bastante más agradable a la mayoría de los jugadores.

Sin embargo, esta vez la **Mecánica 1** causó una sensación mucho más agradable de lo esperado. Cada uno de los jugadores sintió que la respuesta de Pacman ante el *swipe* era divertida, a pesar de que el escenario era de pruebas y sin ningún fin.

En definitiva, la jugabilidad finalista de este prototipo resultó ser la **Mecánica 1** debido a la gran acogida que tuvo entre los jugadores.

ELECCIÓN DE JUEGO

El punto más importante de esta fase de prototipado es la elección definitiva del videojuego que se elegirá cara al desarrollo e implementación en este Proyecto de Fin de Grado.

Inicialmente todos los videojuegos escogidos partían con las mismas posibilidades de elección, pero han sido las pruebas con usuarios el factor que ha hecho elegir un juego frente a los demás.

A continuación se muestra una tabla con las sensaciones transmitidas por cada uno de los juegos:

Mecánica	**Sensación transmitida**
Frogger M1	Normal
Frogger M2	Buena
Frogger M3	Mala
Frogger M4	Buena
Cheeky Mouse M1	Muy mala
Pacman M1	Muy buena
Pacman M2	Buena
Pacman M3	Mala
Pacman M4	Buena

Uniendo estos resultados, la decisión fue relativamente sencilla: el videojuego desarrollado finalmente sería **Pacman** con la **Mecánica 1** (*swipe* para desplazar).

GENERACIÓN DEL ENTORNO EN EDITOR VS. GENERACIÓN DEL ENTORNO EN TIEMPO DE EJECUCIÓN

Una vez determinado el videojuego fue necesario determinar si la generación del entorno se realizaría de forma previa en el editor o realizando una carga desde fichero en tiempo de ejecución.

En el apartado de prototipado del Pacman comenté las ventajas y desventajas de ambas opciones. En conjunto, la opción más eficiente y reutilizable es la generación del entorno en tiempo de ejecución. Pero esto planteó un problema intrínseco a la versión de Unity que estaba utilizando.

Los fantamas que había creado con una aproximación inicial a una inteligencia artificial hacían uso de un elemento de Unity denominado **Malla de Navegación**, el cual permite definir áreas por las que se puede, o no, navegar de la superficie. Quienes están condicionados por esta restricción de navegación son unos elementos denominados **Agentes de Navegación**, los cuales utilizan estas mallas. Los fantasmas generados son objetos que utilizan las propiedades de los Agentes de Navegación, y por lo tanto requieren del uso de estas mallas.

El problema surge debido al siguiente aspecto: las Mallas de Navegación han de ser calculadas a partir del entorno estático del Editor antes de generar la versión ejecutable. Esto implica que, en caso de generar el entorno en tiempo de ejecución, la Malla de Navegación implementada no detectaría ninguno de los objetos existentes en la escena posteriores a la creación del ejecutable.

Esta restricción que acabo de explicar viene dada en la versión **Unity Free**, la cual estoy utilizando. En la versión **Unity Pro** se incluyen instrucciones que permiten recalcular la Malla de Navegación durante la ejecución del programa, permitiendo generar contenido dinámico y hacer que las superficies de movimiento se actualicen en cada momento.

Debido a la ausencia de presupuesto, debo limitarme a las herramientas que me proporciona la versión gratuita de Unity. Por lo tanto, la elección tomada fue **generación del entorno en editor**.

9. DESARROLLO

El objetivo de este capítulo es mostrar el desarrollo de este proyecto como cualquier desarrollo software, tal y como define la Ingeniería del Software.

El desarrollo del videojuego generado en este Proyecto de Fin de Grafo sigue un ciclo de vida especial, ya que combina el modelo de **prototipos** en su etapa inicial, seguido del clásico **modelo en cascada**. El objetivo de esta unión ha sido:

- Obtener en la primera etapa el mayor feedback posible por parte de los usuarios que permitiera definir de forma clara y detallada del diseño del producto final.
- Una vez definido el diseño, seguir un sistema procedural que requiriera la finalización de un estado para poder continuar al siguiente. De este modo me ha sido posible gestionar de forma eficiente las tareas a las que debía dedicarme en cada punto del proceso.

Por lo tanto, las fases resultantes de esta combinación han sido las siguientes:

1. Prototipado
2. Diseño
3. Implementación
4. Prueba
5. Mantenimiento

En este capítulo se contemplan las explicaciones de las fases de Diseño, Implementación y Prueba. La fase de prototipado ha sido definida en el capítulo exclusivo para este aspecto tan importante del proyecto. Por otra parte, la fase de mantenimiento depende de la interactuación entre un cliente final real y el desarrollador, por lo que la pospongo hasta disponer de una situación en la que realmente pueda abarcarla.

He decidido omitir la fase de análisis debido a que todos los requisitos del videojuego se han mantenido inalterados con respecto al videojuego original, y el objetivo ha sido introducir aspectos particulares de los terminales móviles actuales.

A continuación comienza la explicación de cada una de estas fases abarcadas (diseño, implementación y prueba).

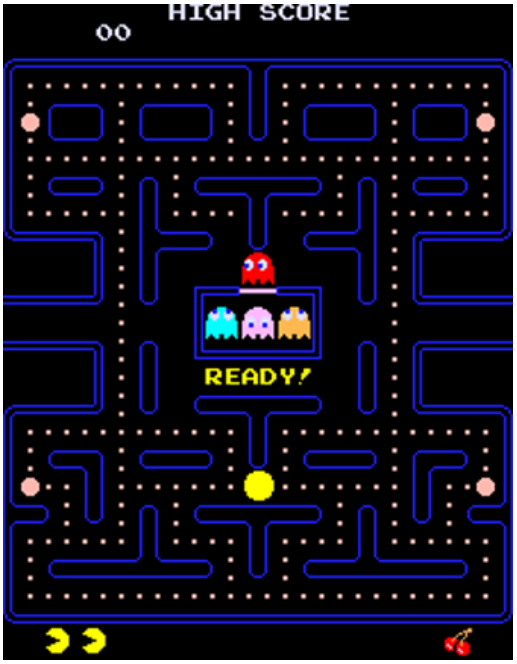
9.1. DISEÑO

La realización de este proyecto en particular supuso una fase de análisis particular. No tuve la necesidad de definir de forma tan explícita los diferentes requisitos funcionales y no funcionales de la aplicación final. Estos requisitos prácticamente son los mismos que los del videojuego que voy a adaptar. Sin embargo, cara a la realización del diseño y la posterior

implementación fue necesario realizar una investigación sobre cada uno de los aspectos en detalle que conforman en Pacman.

A continuación se explican las reglas y detalles de forma breve de este videojuego seleccionado:

ESTUDIO DETALLADO DE PACMAN



La premisa de Pacman es bastante simple: guiar a Pacman en las cuatro posibles direcciones de movimiento (arriba, abajo, derecha e izquierda) a través del laberinto con el objetivo de comerse todos los puntos que hay en él. Cuatro fantasmas se encuentran también en el laberinto y perseguirán a Pacman, intentando matarle. El objetivo final del nivel es limpiar toda la superficie del nivel de estos puntos que lo pueblan y evitar a los fantasmas.

Inicialmente en cada nivel los fantasmas se encuentran en el centro del laberinto, y una vez que Pacman muere, estos regresan al inicio esperando la siguiente ronda. Una vez Pacman completa el nivel, este se resetea para pasar al siguiente. Si Pacman es capturado, pierde una vida y regresa al lugar de inicio. Sin embargo, si no le quedan vidas, el juego ha terminado.

En total existen 244 puntos repartidos por toda la mazmorra, y Pacman debe comerse todos ellos con el objetivo de pasar al siguiente nivel. 240 son puntos pequeños o ***cocos***, los cuales proporcionan 10 puntos al jugador. Los otros 4 puntos grandes son ***energizers*** (*power up*) y proporcionan 50 puntos al ingerirlos. Esta suma proporciona un total de 2.600 puntos por consumir todos los puntos del laberinto en cada nivel. Además, es posible obtener puntos adicionales. Estos se pueden conseguir de dos formas diferentes:

- Darle la vuelta a las tornas consumiendo un energizer, el cual hace que los fantasmas huyan de Pacman y permite a este comérselos. Cada vez que este cambio de tornas ocurra, si se consigue comer un

fantasma, este dará un número de puntos basados en cuántos fantasmas hayan sido comidos previamente esos segundos especiales. Los puntos que se pueden conseguir al comer a cada fantasma son 200 → 400 → 800 → 1600. Una vez los fantasmas han sido comidos, vuelven corriendo a su base e inician de nuevo su persecución dejando de ser vulnerables.

- La segunda forma de obtener puntos adicionales es ingiriendo determinados símbolos bonus (**frutas**) que aparecen directamente debajo de la casa de los fantasmas en el laberinto. Cada una de estas frutas proporciona diferentes puntos adicionales, y su probabilidad y tiempo de aparición depende del nivel en el que nos encontremos.

Los fantasmas generalmente están en uno de estos tres estados principales:

- **Home**: estado en el que se encuentran antes de iniciar la cacería. Cada uno de los fantasmas dispone de un tiempo de reposo en su posición inicial antes de pasar al siguiente estado. Durante este estado no suponen ninguna amenaza para Pacman.
- **Chase**: una vez comienzan a avanzar, estos fantasmas recorren el laberinto con el objetivo de encontrar y capturar a Pacman.
- **Frightened**: cuando Pacman ingiere un energizer, los fantasmas reducen su velocidad y alteran su dirección durante el tiempo que dure este cambio de tornas.

En el videojuego original, Pacman reducía ligeramente su velocidad al ingerir un punto, lo cual le hacía elegir en determinadas situaciones un camino libre en lugar de un camino lleno de puntos cuando estaba siendo perseguido por un fantasma.

Estos fantasmas que comparten protagonismo junto a Pacman disponen de diferentes personalidades claramente definidas, y con ello, comportamientos diferentes. A continuación comentaré brevemente las intenciones de cada uno:

- **Blinky**: El fantasma rojo es el primer fantasma en abandonar la casa de fantasmas al comienzo de cada nivel. Es, de lejos, el fantasma más agresivo de los cuatro, y la mayor parte del tiempo está persiguiendo a Pacman.

- **Pinky**: El fantasma rosa es el segundo en abandonar la casa. Sus movimientos son los más inteligentes de los 4, ya que no persigue directamente a Pacman, sino que se dirige al punto al que Pacman dirige la mirada, intentando prever constantemente los movimientos, en lugar de perseguirle por detrás.

- **Inky**: El fantasma azul es el tercero en abandonar la casa de fantasmas. Es el fantasma más impredecible, ya que a lo largo del desarrollo del juego su comportamiento cambia en numerosas ocasiones, pudiendo ser tan agresivo como Blinky como

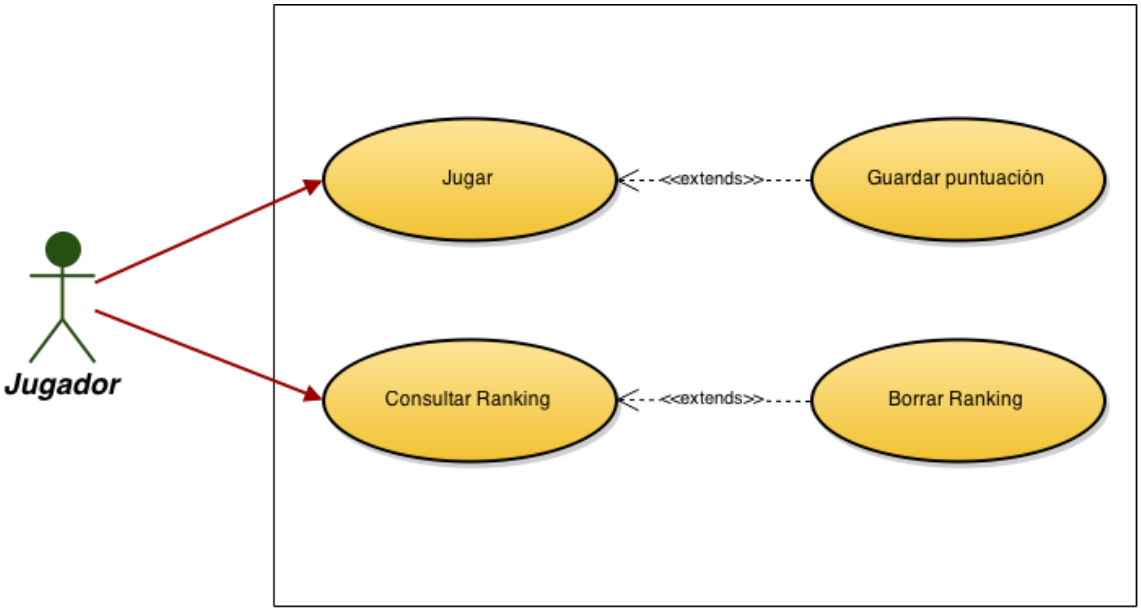
aplicar la estrategia de Pinky. Esta irregularidad en su comportamiento hace que sea peligroso, ya que no se puede prever sus movimientos, al igual que con los dos anteriores.

- **Clyde**: El fantasma naranja es el último en abandonar la casa. El comportamiento de este último enemigo es particular, ya que a lo largo de su recorrido intenta distanciarse de los demás fantasmas, así como de Pacman. Para ello suele recorrer zonas concretas y apartadas de las rutas de los demás, pero en ocasiones se puede considerar que llega a tender emboscadas a Pacman.

Una vez explicados estos aspectos básicos del videojuego en cuestión, indicaré los casos de uso para los que se utilizará esta aplicación.

CASOS DE USO

A continuación se muestra un diagrama en el que se pueden observar los casos de uso principales de la aplicación que se va a desarrollar:



Como se puede observar, el diagrama es realmente simple, ya que el objetivo de un videojuego en general generalmente es permitir jugar al usuario, y es el único motivo por el que accede a él en la mayoría de los casos. En este ejemplo en particular, el usuario además podrá registrar su puntuación al finalizar una partida para posteriormente consultar el ranking, y en la visualización de este ranking podrá borrar todo su contenido en cualquier momento.

ESPECIFICACIÓN DE CASOS DE USO

Nombre del caso de uso	Jugar
Descripción	Jugar partida
Actores	Jugador
Precondiciones	-----

Postcondiciones	Puntuación generada
Flujo Básico	**Flujo Alternativo**
1.El jugador inicia la partida	
2.El jugador juega la partida, pudiendo pausarla, reanudarla, reiniciarla (volviendo al paso 1) o salir.	
3.El jugador finaliza la partida	

Nombre del caso de uso	Guardar puntuación
Descripción	Registrar en la tabla del ranking la puntuación obtenida en la última partida jugada.
Actores	Jugador
Precondiciones	Haber generado una puntuación
Postcondiciones	Puntuación registrada en el ranking
Flujo Básico	**Flujo Alternativo**
1.El jugador introduce su nombre de usuario	
2.El jugador registra su puntuación. Además, el jugador tiene la opción de cancelar este registro en cualquiera de los pasos del proceso.	

Nombre del caso de uso	Consultar Ranking
Descripción	Consultar una tabla ordenada con las mejores puntuaciones registradas en el dispositivo
Actores	Jugador
Precondiciones	-----
Postcondiciones	Tabla de ranking cargada
Flujo Básico	**Flujo Alternativo**
1.El jugador consulta la tabla con las puntuaciones.	

Nombre del caso de uso	Borrar Ranking
Descripción	Eliminar todos los registros de puntuaciones almacenadas previamente en el dispositivo
Actores	Jugador
Precondiciones	-----
Postcondiciones	Eliminadas todas las puntuaciones del ranking
Flujo Básico	**Flujo Alternativo**
1.El jugador elimina todos los registros almacenados en el ranking	

9.2. IMPLEMENTACIÓN

A lo largo del desarrollo de la versión final de este software he utilizado diferentes componentes y características. Estos son generales para la mayoría de desarrollos software, pero otros son específicos y particulares de Unity. A continuación listaré estos elementos clave de la implementación:

- **Controladores**
- **Modelos**
- **Colliders y Triggers**
- **Touch**
- **Mallas de Navegación**
- **Animaciones**
- **GUI**
- **Información persistente**
- **Eventos y delegados**
- **Estados**

En los siguientes subapartados detallaré el uso de cada uno de estos elementos previamente citados, definiéndolos y destacando su utilidad en este desarrollo. Prácticamente la totalidad del código generado está en C#, a excepción de algunos ficheros en JavaScript no incluidos en este informe.

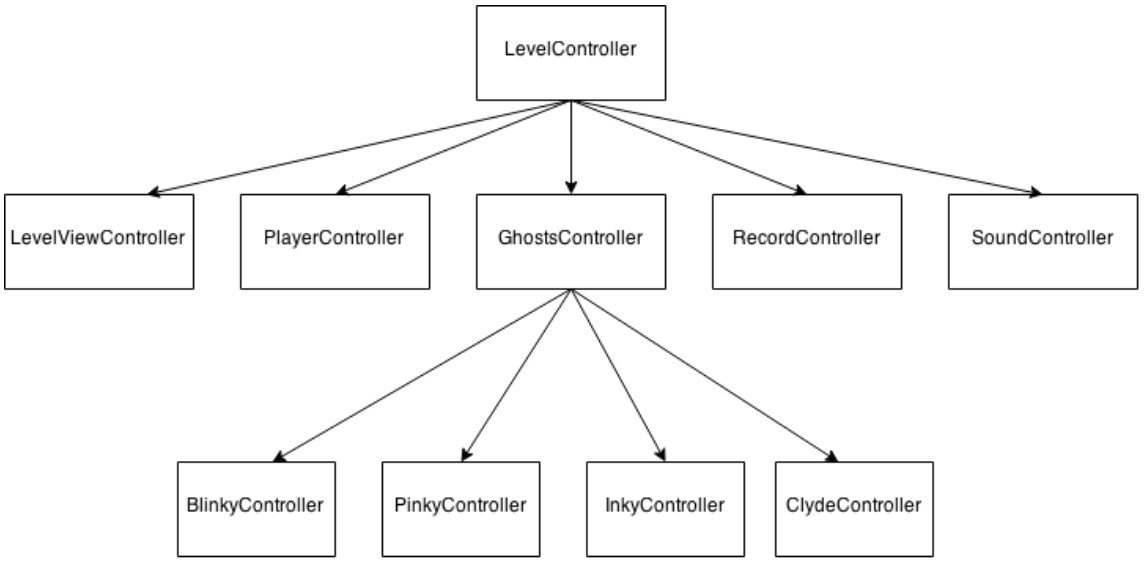
CONTROLADORES

En este desarrollo he aplicado una aproximación al modelo-vista-controlador (MVC), pero sin llegar a adaptarlo completamente. Esto es debido a que Unity es un entorno orientado a componentes, y estos en ocasiones interactúan de forma diferente a lo que el MVC indica. Por lo tanto, he conservado las directrices principales del modelo-vista-controlador, delegando las responsabilidades sobre los controladores, mientras que los modelos se encargan de contener la información más básica, así como ciertos métodos relacionados con estas. Por último, las vistas controlan toda la información que es transmitida al usuario, así como la forma en la que se realiza en cada momento.

Estos tres elementos de este tipo de diseño conviven con determinados flujos de información propios de los componentes propios de Unity3D, los cuales mandan mensajes de forma tanto ascendente como descendente en la jerarquía.

El dominio de este proyecto en un único nivel en el que el jugador puede jugar y repetir tantas veces como quiera, por lo tanto, no existe un controlador que gestione el cambio de niveles. Aun así, dada la estructura planteada, la incorporación de esta capa superior resulta trivial.

A continuación se muestra una relación de los diferentes controladores existentes en la implementación:



El controlador central es el LevelController, el cual invoca en determinadas situaciones a los demás controladores para acciones puntuales.

A continuación comentaré el contenido de los controladores más importante de esta estructura. Es posible encontrar todos y cada uno de ellos en el código fuente al que hace referencia este documento.

LEVELCONTROLLER

Inicialmente, el LevelController instancia cada uno de los controladores que va a requerir durante su ejecución:

```
void Start () {
            ghostsController =
GameObject.Find("GhostsController").GetComponent<GhostsController>();
            playerController =
GameObject.Find("PlayerController").GetComponent<PlayerController>();
            levelViewController =
GameObject.Find("LevelViewController").GetComponent<LevelViewController>();
            recordController =
GameObject.Find("RecordController").GetComponent<RecordController>();
            soundController =
GameObject.Find("SoundController").GetComponent<SoundController>();
            ranking = GameObject.Find("Ranking").GetComponent<Ranking>();

            soundController.siren();
      }
```

Realmente en Unity lo que se realiza es una asignación del componente programático (*script*) de cada objeto de la escena a una variable, mediante la cual se podrá conservar la relación a lo largo de la ejecución.

Una vez obtenidos los demás controladores, la interfaz del controlador sería la siguiente:

```
public void checkGhostPoints (GameObject ghost){
        …
}
private void endOfLevel(){
        …
}
private void gameOver(){
        …
}
public void pause(){
        …
}
public void resume(){
        …
}
public void restart(){
        …
}
public void exit (){
        …
}
public void record(){
        …
}
```

La declaración pública de alguno de estos métodos es debida a que el sistema de eventos de Unity automatiza los eventos *OnClick*, y es necesario poder indicar el método que se ejecutará desde algunos componentes de la vista.

PLAYERCONTROLLER

Este controlador se encarga de capturar todos los inputs relacionados con el movimiento para transmitírselos a Pacman. Además, evalúa el estado del entorno mediante unos detectores que le permiten saber si en una determinada situación Pacman puede girar o no hacia una dirección. Una vez realizada esta comprobación, se ejecuta el movimiento.

En flujo principal de ejecución de este controlador viene definido por el siguiente método:

```
IEnumerator Playing(){
              while (currentState == states.Playing) {
                     checkInput ();
                     updateView();
                     checkEnviroment ();
                     perform ();
                     yield return 0;
              }
       }
```

Este método realmente es una corrutina que proporciona Unity para ejecutar varios hilos simultáneamente. Concretamente el uso de esta corrutina es gestionar los diferentes estados por los que puede pasar el PlayerController, pero se explicará con más detalle en el subapartado Estados de este mismo capítulo.

Como se puede observar, el flujo normal del controlador consiste en llamadas a los métodos principales `checkInput()`, `checkEnviroment()` y `perform()`, los cuales van

acompañados de `updateView()` para modificar parte del HUD mostrando el movimiento que ha realizado el usuario.

GHOSTSCONTROLLER

Este controlador es el encargado de gestionar las acciones de juego que tienen que ver con el conjunto de fantasmas. De esta forma, el LevelController puede abstraerse del número de fantasmas existentes en la partida (aunque originalmente son siempre 4). Ante determinadas acciones que afectan todos los fantasmas de la escena, este controlador se encarga de acceder a cada uno de ellos y cambiar su estado de forma individual. Así mismo, lleva el control temporal de eventos que ocurren en cuando a estados de fantasma se refiere, como puede ser la salida de cada fantasma del punto de inicio o el tiempo restante para la finalización del *power up*.

El inicio de este controlador va ligado a la búsqueda de cada uno de los fantasmas existentes en el juego:

```
void Start () {
                blinky = GameObject.Find("Blinky");
                clyde = GameObject.Find("Clyde");
                inky = GameObject.Find("Inky");
                pinky = GameObject.Find("Pinky");
                time = 0;
                clockRunning = true;
                timeSuperPelletEnds = -1;
                StartCoroutine(Clock());
}
```

Una vez capturados e inicializado ciertas variables privadas, se lanza la corrutina que actuará a modo de reloj. Este reloj controlará diferentes tipos de eventos a lo largo del transcurso del programa.

La interfaz resultante de este controlador es la siguiente:

```
        public void superPelletEaten(){
                …
        }

        private void superPelletFinished(){
                …
        }

        private void setFlashing(){
                …
        }

        public void ghostDead(GameObject ghostKilled){
                …
        }

        public int ghostsEatenThisChase(){
                …
        }

        public void backAllGhosts(){
                …
        }

        public void play(){
                …
        }

        public void pause(){
                …
        }

        public void unPause(){
                …
        }

        private void checkTimeEvents(){
                …
        }

        void HandleGhostIn(GameObject ghost){
                …
        }

        void HandleGhostOut(GameObject ghost){
                …
        }

        void HandleGhostAtHome(){
                …
        }
```

La mayoría de estos métodos reflejan una determinada acción en todos los fantasmas existentes en el nivel, mientras que otros llevan controles generales relacionados con el estado del juego. Este controlador utiliza mucho el sistema de eventos y delegados que explicaré más adelante.

LEVELVIEWCONTROLLER

Este controlador se encarga de la gestión de todas las vistas existentes en la escena, y es invocado desde los demás controladores para actualizar determinados paneles en función del evento que haya ocurrido en cada momento.

De nuevo, en el bloque Start() se referencian todos los componentes de la escena que se van a utilizar, en este caso, vistas.

```
void Start () {
                pauseMenu = GameObject.Find("PauseMenu");
                pauseButton = GameObject.Find("PauseButton");
                soundButton = GameObject.Find("SoundButton");
                mutedButton = GameObject.Find("MutedButton");

                congratsMenu = GameObject.Find("CongratsMenu");
                gameOverMenu = GameObject.Find("GameOverMenu");
                saveRankMenu = GameObject.Find("SaveRankMenu");

                rightArrow = GameObject.Find("RightArrow");
                leftArrow = GameObject.Find("LeftArrow");
                upArrow = GameObject.Find("UpArrow");
                downArrow = GameObject.Find("DownArrow");

                …
}
```

Cada una de las diferentes vistas que genera este controlador serán mostradas en el apartado GUI de este capítulo.

MODELOS

Los modelos básicos que componen nuestra escena de juego son los siguientes:

- Pacman
- Blinky (Fantasma)
- Pinky (Fantasma)
- Inky (Fantasma)
- Clyde (Fantasma)
- Pellet (punto o coco)
- SuperPellet (energizer)
- Wall

Los elementos que se repiten a lo largo del nivel, como son: Pellet, SuperPellet y Wall, han sido convertidos en Prefabs. Estos Prefabs son generalizaciones del objeto que proporciona Unity, de modo que cada vez que se realiza una instancia de este prefab, este comparte la mayoría de sus atributos y componentes con todas las demás instancias. Es posible realizar ediciones puntuales en cada instancia, pero todos los datos que comparten se almacenarán una sola vez, en lugar de una copia por cada elemento en la escena.

Esta característica es realmente útil en cuanto al rendimiento del programa, ya que ocupará mucha menos memoria y supondrá un menor impacto en tiempo de ejecución.

Los otros elementos cuya aparición es única en la escena, como son: Pacman, Blinky, Pinky, Inky y Clyde, se generan con todos sus atributos desde el editor, y no son duplicados en ningún momento.

El motivo de que los cuatro fantasmas estén separados en modelos independientes es las múltiples diferencias y particularidades de cada uno en el videojuego original, comportándose de forma muy distinta en varios casos y omitiendo diferentes estímulos o siendo más sensibles a ellos.

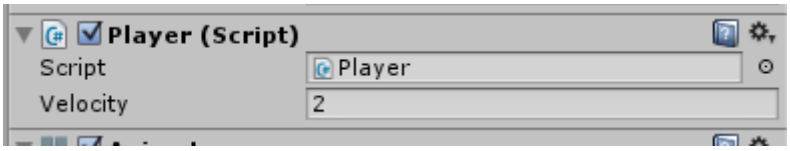
En esta versión realizada para el Proyecto de Fin de Grado, el comportamiento de los fantasmas es relativamente similar, a excepción de las rutas que realizan a lo largo del laberinto. Pero cara a futuras ampliaciones y mejoras, es conveniente mantener separadas las lógicas de comportamiento de cada uno de ellos, ya que pueden diferir de forma considerable.

A continuación reflejaré un ejemplo de clase de modelo en C#:

```
public class Player : MonoBehaviour {

	bool leftFree = true;
	bool rightFree = true;

	bool alive = true;

	public float velocity;

	bool stopped = true;

	public bool isRightFree(){
		return rightFree;
	}

	public bool isLeftFree(){
		return leftFree;
	}

	public void setRightFree(){
		rightFree = true;
	}

	public void setRightBusy(){
		rightFree = false;
	}

	public void setLeftFree(){
		leftFree = true;
	}

	public void setLeftBusy(){
		leftFree = false;
	}

	public void setVelocity(float velocity){
		this.velocity = velocity;
	}

	public float getVelocity(){
		return this.velocity;
	}

	public void stop(){
		stopped = true;
	}

	public void unStop(){
		stopped = false;
	}

	public bool isStopped(){
		return stopped;
	}

	public void setAlive(bool alive){
		this.alive = alive;
	}
	public bool isAlive(){
		return alive;
	}
}
```

Este modelo es el que contiene los atributos propios de Pacman. Los métodos implementados en esta clase son los más básicos y únicamente almacenan y aportan información, ya que es el PlayerController el que se encarga de realizar los movimientos a partir de los inputs.

Unity permite la edición de variables de forma muy sencilla desde el editor siempre y cuando estas sean públicas. Este es el método que se ha seguido para perfilar determinados aspectos relativos a la curva de dificultad durante la fase de pruebas y testeo. A continuación se muestra de qué forma se refleja esto en el editor:



Como se puede observar en las declaraciones del modelo, la única variable pública es la velocidad que tendrá el jugador, y esta no es establecida en ningún momento. A través del editor es posible añadir y modificar este valor, incluso durante la propia ejecución del programa, lo cual facilita mucho las tareas de calibrado.

COLLIDERS Y TRIGGERS

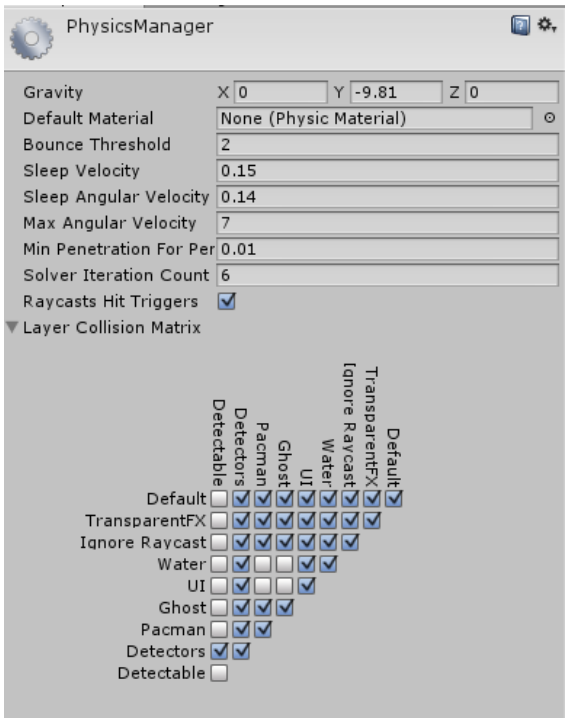
Los Colliders y los Triggers son elementos fundamentales para la creación de un videojuego. Estos componentes son compartidos por la mayoría de entornos de desarrollo. A continuación pasaré a definir cada uno de ellos:

- **Collider**: este componente proporciona a un objeto de juego concreto una superficie de forma variable (esférico, cúbico, plano, cilíndrico, etc.) que colisionará con otros colliders existentes en la escena.
- **Trigger:** este componente es similar al Collider. Concretamente, comparte forma y mismo tipo de vínculo con el objeto al que de adhiere, pero su reacción ante otros objetos es diferente. El Trigger no colisiona con otros Triggers o Colliders, sino que simplemente detecta el paso a través de su superficie.

Estos dos componentes son cruciales para conformar la lógica del juego, ya que los elementos están constantemente teniendo interactuaciones físicas con los demás elementos del juego.

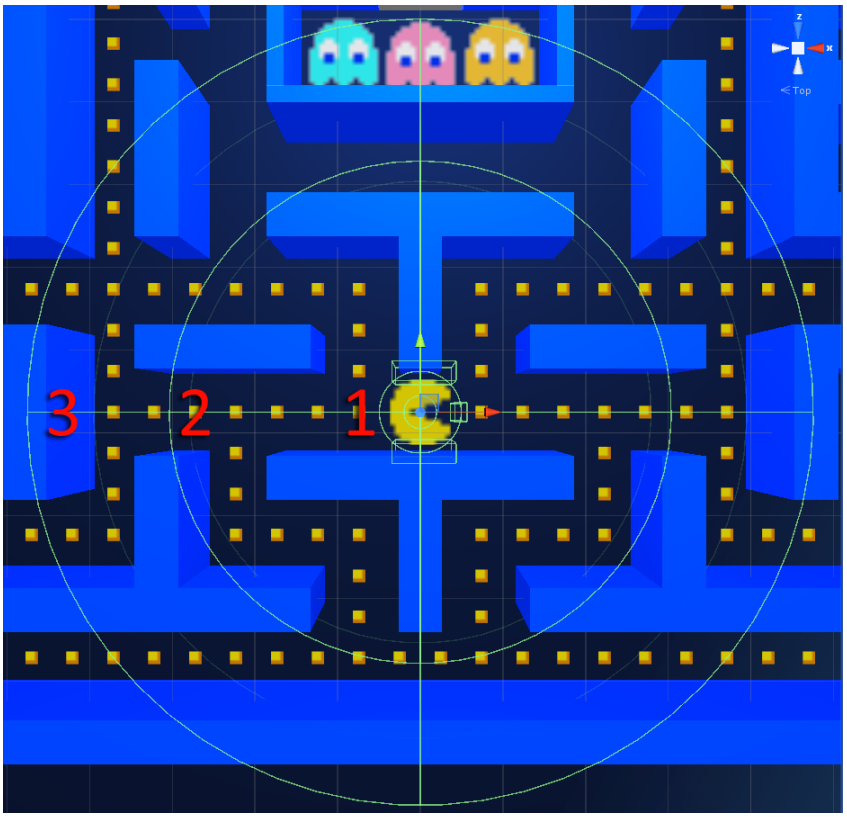
Es posible personalizar las físicas del motor incorporado en Unity para hacer que determinados objetos (y sus respectivos triggers y colliders) ignoren otro tipo de objetos. Esta funcionalidad es realmente importante, ya que no todos los elementos de la escena interactúan físicamente con los demás.

A continuación se muestra el PhysicsManager, el cual permite realizar este tipo de controles de colisiones:

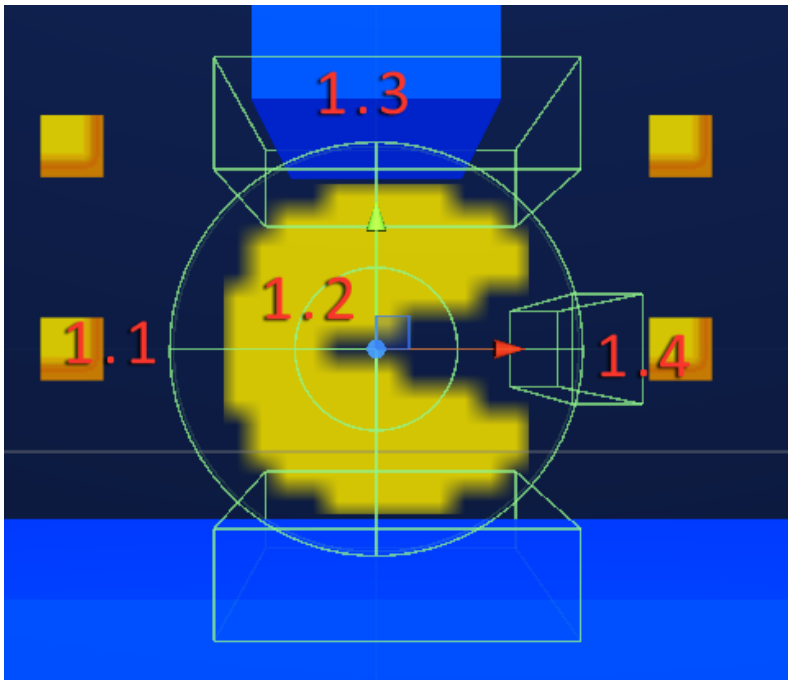


En la imagen superior se observa cómo en la matriz de relaciones físicas he configurado que los elementos que voy a utilizar como ítems detectables únicamente van a reaccionar ante los triggers y colliders de objetos detectores. De esta forma se reducen en gran medida problemas con colisiones del entorno no deseadas.

Una vez ajustadas las físicas paso a explicar la aplicación práctica de estos componentes. Y lo voy a realizar poniendo de ejemplo a nuestro protagonista, Pacman:



He dividido los Colliders y Triggers de este objeto en tres puntos para explicarlos por separado. Comenzaremos por ampliar el primer punto para observar mejor los componentes que rodean la superficie de Pacman.



El círculo verde **1.1** de la imagen superior representa la superficie física de Pacman, la cual es un Collider. Esta detecta las diferentes colisiones que pudiera llegar a sufrir el jugador en su forma más primitiva.

El círculo interior **1.2** es el punto central de Pacman, de nuevo Collider, el cual es el encargado de consumir los puntos que se encuentra por el camino, así como los fantasmas cuando estos son vulnerables. Así mismo, este núcleo es el punto débil de Pacman, por lo que su muerte llegará cuando un fantasma no vulnerable acceda hasta él. De este modo la adaptación se vuelve más fiel al videojuego original, teniendo que estar el fantasma prácticamente encima de Pacman para detectar su muerte.

Los rectángulos **1.3** que se observan a ambos lados de Pacman son Triggers que detectan en cada momento si existen muros a izquierda y derecha de Pacman. Estas superficies van detectando la entrada y salida de paredes durante el avance del juego, y activan un *flag* en el modelo cada vez que alguno de estos laterales queda libre. Este control del entorno es utilizado por el PlayerController para evaluar si un movimiento está permitido en una situación o no.

La superficie **1.4** detecta colisiones de forma frontal, indicando al PlayerController que debe parar el movimiento del jugador. Este componente es de nuevo un Trigger, ya que permite ser invadido por colliders externos.

Una vez terminado este bloque de componentes directamente anclados a Pacman pasamos con otros dos que orbitan a su alrededor y que tienen una gran importancia en la calidad del juego.

La superficie circular **2** que rodea a Pacman a una cierta distancia se trata de un Trigger detector que reacciona ante la aparición de objetos de tipo *Ghost* (Fantasma). Este componente lanza un evento que informa al GhostsController de que un fantasma está cerca de Pacman, y este se encarga de cambiar el estado de dicho fantasma para que comience a perseguir al jugador de forma intensiva hasta que lo pierda de vista.

Es aquí donde entra la función de la superficie circular 3, la cual es de nuevo un Trigger y detecta salidas de objetos tipo *Ghost*. Solamente cuando los fantasmas que persigan activamente a Pacman salgan de este segundo radio volverán a su estado de rutina anterior.

Esta duplicidad de superficies de detección de entrada y salida ha sido generada con el objetivo de eliminar casos límite específicos del juego, como era la situación de que un fantasma detectara proximidad, realizara un pequeño giro en su ruta para dirigirse a Pacman, pero al realizar dicho movimiento, lo perdiera de vista por cuestión de décimas de la Unidad de distancia de Unity. Esto provocaba que el fantasma quedara atrapado en un bucle de detección-pérdida en una esquina del laberinto.

Duplicando las áreas con diferentes radios, en el momento en el que un fantasma detecta la posición de Pacman, tiene un cierto margen de movimiento en sentido opuesto para poder salir de un pasillo y dirigirse hacia a él. Esto convierte los movimientos de los fantasmas en respuestas más naturales y menos erráticas ante la presencia del jugador durante los diferentes niveles.

Programáticamente, Unity proporciona los siguientes métodos de gestión de las interactuaciones físicas de los Colliders y los Triggers:

Colliders

```
void OnCollisionEnter(Collision collision) {
        …
}
void OnCollisionExit(Collision collision) {
        …
}
void OnCollisionStay(Collision collision) {
        …
}
```

Triggers

```
private void checkInput(){
	#if UNITY_ANDROID

	if ((currentState == states.Playing) && (Input.touchCount > 0)){
		Touch touch = Input.touches[0];

		switch (touch.phase){

		case TouchPhase.Began:

			startPos = touch.position;
			break;

		case TouchPhase.Moved:
		case TouchPhase.Ended:

			…

			if (swipeDistVertical > swipeDistHorizontal){
				if (swipeDistVertical > minSwipeDist) {
					float swipeValue = Mathf.Sign(touch.position.y - startPos.y);

					if (swipeValue > 0)//up swipe
					{
						…
					}
					else if (swipeValue < 0)//down swipe
					{
						…
					}
					startPos = touch.position;
				}
			} else {
				if (swipeDistHorizontal > minSwipeDist) {
					float swipeValue = Mathf.Sign(touch.position.x - startPos.x);
					if (swipeValue > 0){ //right swipe
						…
					} else if (swipeValue < 0){ //left swipe
						…
					}
					startPos = touch.position;
				}
			}
			break;
		}
	}
	#endif
}
```

El método simplificado del bloque superior pertenece al PlayerController. Este método **`checkInput()`** se encarga de consultar las entradas táctiles del dispositivo para traducirlas en órdenes para Pacman.

Para lograr eso, hago uso de la clase ***Touch*** proporcionada por Unity, la cual permite detectar de forma unánime los pulsos en pantallas táctiles de todos los dispositivos que soporta este entorno de desarrollo (Android, iOS, Windows Phone y BlackBerry).

Esta clase Touch contiene información de cada pulso realizado en todo momento, y es utilizada durante el flujo de este controlador para captar la ubicación inicial y detectar la

trayectoria para procesar si es un movimiento vertical u horizontal, y posteriormente el sentido de este.

Cada *touch* realizado en la superficie táctil se puede encontrar en los cinco estados siguientes:

- **Began**
- **Moved**
- **Stationary**
- **Ended**
- **Canceled**

En el flujo de este controlador se evalúan los estados que realmente interesan en este caso, que son: **Began**, **Moved** y **Ended**.

Estos estados son controlados por el *switch* que abarca gran parte del método. Durante la **TouchPhase.Began**, el controlador almacena la posición inicial del *swipe* para poder determinar posteriormente su desplazamiento.

El segundo gran bloque capta las fases **TouchPhase.Moved** y **TouchPhase.Ended**. A lo largo de estas líneas de código se captura la distancia actual del *touch* y se compara con la posición original. Si la diferencia absoluta de las distancias es mayor a un mínimo impuesto por un valor de testeo (variable), se interpreta como que se ha realizado correctamente un *swipe* en alguna dirección. Se compara la diferencia de distancias de desplazamiento horizontal y vertical y se determina qué tipo de movimiento se ha realizado. Por último, se evalúa el sentido de este movimiento en función de si la diferencia resulta positiva o negativa.

Inicialmente la mecánica de movimiento era únicamente la finalista de la fase de prototipado, pero durante una de las sesiones de feedback durante el testeo de desarrollo observé que a medida que la complejidad del juego aumentaba, muchos jugadores menos experimentados preferían guiar a Pacman con movimientos concatenados, sin llegar a levantar el dedo de la pantalla en ningún solo momento. Esto me sirvió para darme cuenta de que era posible implementar ambas formas de control de forma natural y que convivieran en la misma versión ejecutable, dejando que el jugador decidiera qué actividad le resultaba más intuitiva.

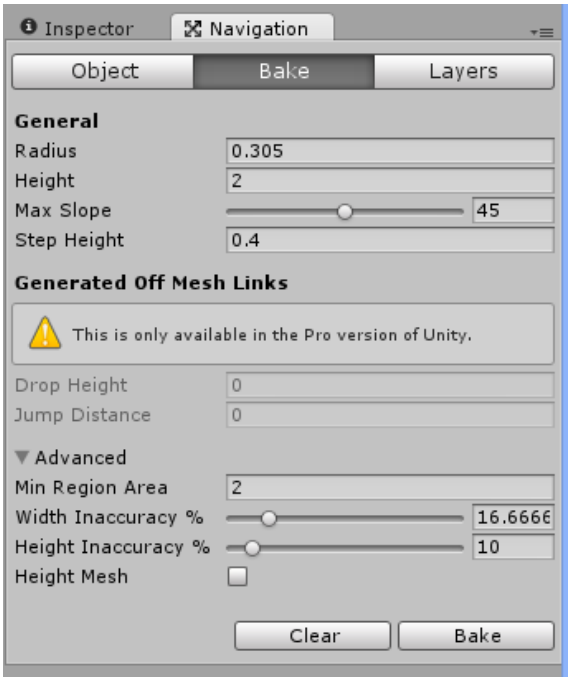
Debido a esta observación existe el control de la **TouchPhase.Moved**, la cual evalúa los movimientos durante su realización, sin esperar a que se llegue a levantar el dedo. Esto aportó solidez y versatilidad al control de las entradas táctiles, dándole un acabado más cómodo para la mayoría de usuarios que probaban el juego por primera vez.

MALLAS DE NAVEGACIÓN

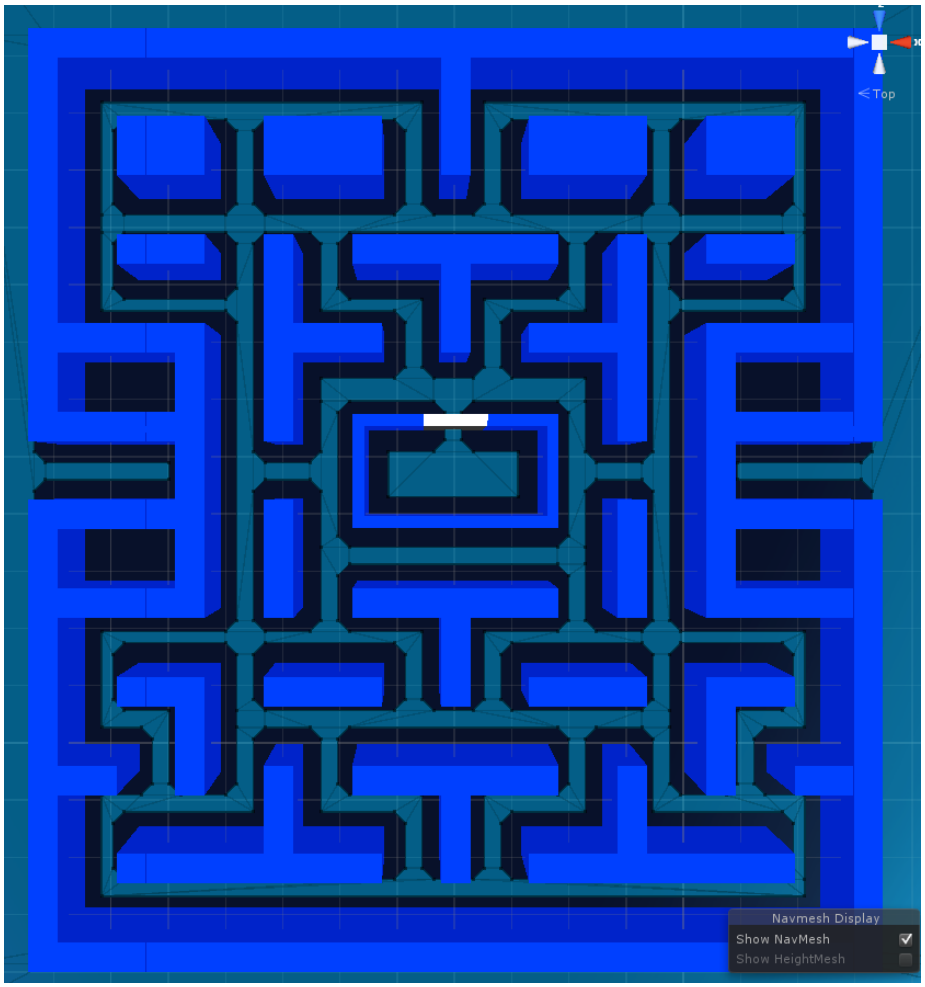
A continuación explicaré la utilidad de otro elemento clave propio de Unity3D. Se trata de las Mallas de Navegación.

Estas Mallas de Navegación son superficies definidas como *walkable* (o “caminables”), y se vinculan a determinados objetos que queremos definir como terrenos u obstáculos. Para definir estas Mallas de Navegación es necesario seleccionar los objetos del entorno que

queremos que actúen como terrenos y obstáculos, y a continuación acceder al panel Navigation del editor de Unity.



Es necesario definir los objetos que queremos que actúen como parte del entorno transitable, así como los obstáculos, como estáticos. Una vez realizado esto es necesario acceder al panel Navigation mostrado en la imagen superior y seleccionar *Bake*. Esta opción calcula la malla de navegación transitable a partir de los objetos estáticos previamente seleccionados, y nos proporciona un resultado similar al siguiente:

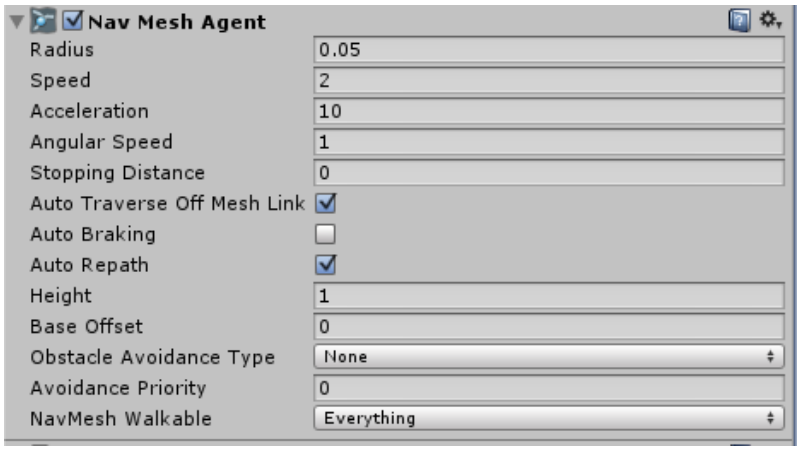


En esta imagen se observan las superficies *walkable* generadas en esta malla, las cuales se destacan mediante el color azul claro en el interior de los pasillos. El grosor y más características de esta superficie resultante es editable desde el panel Navigation mostrado anteriormente, pero algunas de las opciones están reservadas a la versión Unity Pro.

Una vez definida esta superficie entra en acción los Agentes de Navegación. Estos componentes permiten a un objeto de la escena automatizar determinados movimientos a través de esta malla generada. El objetivo es simplificar la gestión del movimiento de determinados elementos del videojuego.

En nuestro caso, este componente Agente de Navegación ha sido vinculado a cada uno de los fantasmas de la escena. De esta forma es posible indicar los movimientos de estos enemigos indicándoles únicamente una posición de destino, y es el Agente de Navegación el encargado de detectar la ruta más corta hasta dicho destino.

Estos agentes son altamente personalizables, pudiendo especificar los siguientes atributos:



Una vez definidas cada una de las variables para generar el agente que más se adapta a nuestro tipo de juego, únicamente resta indicar programáticamente la posición de destino de los fantasmas en cada una de las situaciones. Esto se lleva a cabo mediante una **máquina de estados** explicada más adelante, en el subapartado Estados de este mismo capítulo.

Como comenté en el capítulo de Prototipado, la generación de esta malla se debe realizar desde el editor en la versión Unity Free, siendo imposible obtenerla mediante la generación en tiempo de ejecución del entorno. Este es el motivo principal por el cual se optó por una generación del entorno de juego en el propio editor.

ANIMACIONES

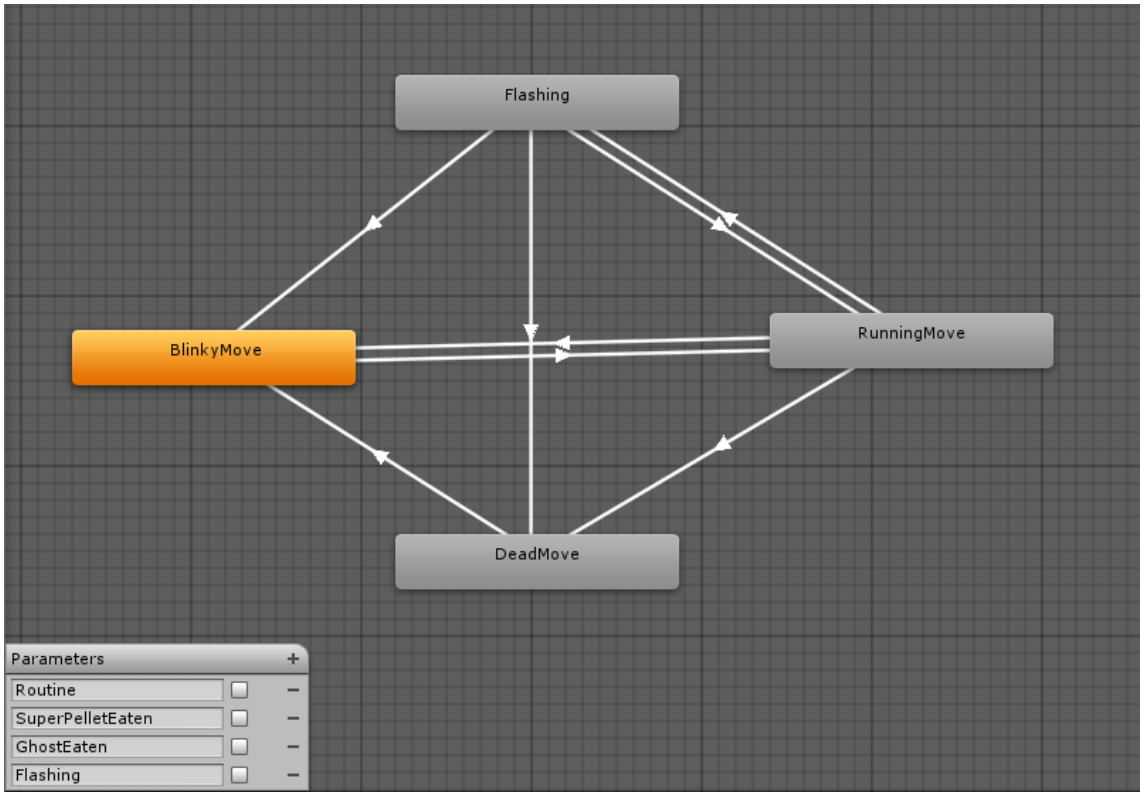
Una funcionalidad realmente cómoda y práctica de Unity son las Animaciones y los Controladores de Animaciones. Estos dos se complementan para lograr diferentes animaciones dependiendo del estado en el que se encuentre un objeto. Estas animaciones suponen un gran impacto visual para el jugador, y la ausencia de ellas reduce considerablemente la sensación transmitida por el videojuego.

El componente más básico de este apartado son las Animaciones, las cuales no son más que un conjunto de *sprites* (imágenes planas) en el caso de un videojuego 2D como es el que estamos realizando. Estos sprites que conforman la animación disponen de un tiempo asociado a cada imagen, que en nuestro caso es el mismo para todas. Al reproducir esta animación simplemente se suceden las imágenes que previamente hayamos incluido en esta animación.

Desde el editor es posible generar animaciones simplemente seleccionando el conjunto de sprites que se desean utilizar y arrastrándolos al entorno de desarrollo, de esta forma se añade a nuestra jerarquía dicha animación, pudiendo ser editada en cualquier momento.

Una vez generadas las animaciones es necesario crear un Controlador de Animaciones. Este controlador gestiona una máquina de estados por los que va a pasar el objeto asociado. Estos estados son puramente gráficos, y no están necesariamente vinculados a los estados lógicos del objeto en cuestión desde el controlador o modelo.

A continuación se muestra como ejemplo el controlador de animaciones de uno de los fantasmas:



En este caso se trata del controlador del fantasma Blinky. En él se observan los diferentes estados animados en los que se puede encontrar:

- **BlinkyMove**: en este estado el fantasma dispone de una animación de movimiento estándar, en la cual está definido su color y se altera entre dos sprites para lograr este efecto.
- **RunningMove**: en este estado el fantasma pasa a ser de color azul, el tamaño de sus ojos disminuye y la velocidad de la animación de movimiento es menor. Se pasa a este estado cuando Pacman consume un *power up*.
- **FlashingMove**: en este estado el fantasma parpadea de color blanco mientras aún está huyendo de Pacman, pero durante menos tiempo. Este estado indica que en breve el fantasma dejará de ser vulnerable para volver a la cacería.
- **DeadMove**: este estado tiene como animación únicamente *sprites* de ojos, sin cuerpo de fantasma. Se accede a este estado cuando Pacman se ha comido al fantasma y está volviendo al lugar de inicio para regenerarse.

Uniendo todos estos estados se consigue el flujo de animaciones de estos enemigos que pueblan el nivel.

En este recorte de la imagen anterior se observan los parámetros generados para el control de los estados. Los cambios entre estados se realizan en función a los valores de cada una de estas variables de la máquina de estados. En determinados puntos de los controladores que gestionan cada objeto animado se accede al componente *Animator* y se cambia el valor de estas variables, siendo posible gestionar estos cambios simultáneamente con el cambio de estados lógicos de cada controlador.

A continuación se indica la forma de acceder a estas variables para cambiar su estado de forma programática:

```
ghost.GetComponent<Animator>().SetBool("Routine", true);
```

Esta gestión de estados de animación se realiza en los siguientes objetos de juego:

- Pacman
- Blinky (Fantasma)
- Pinky (Fantasma)
- Inky (Fantasma)
- Clyde (Fantasma)
- SuperPellet (*energizer*)

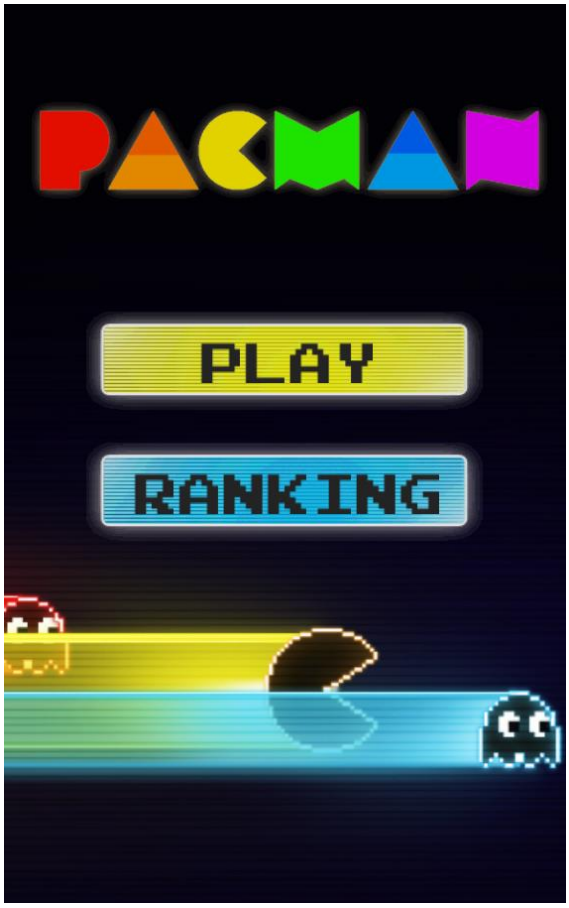
En cada uno de ellos se realiza un control diferente de estos estados, y disponen de las variables necesarias para los cambios existentes.

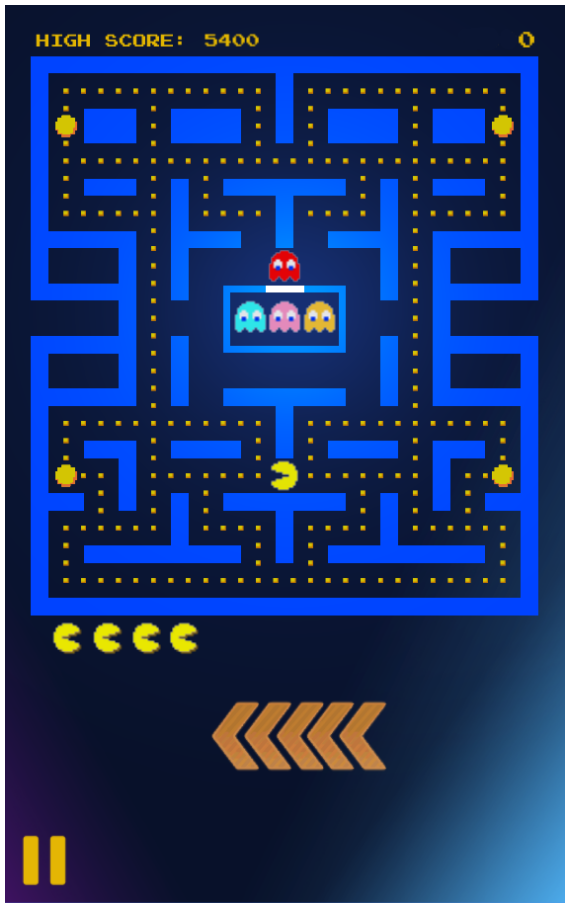
GUI

Los menús y el HUD incorporados en este proyecto han sido realizados con las nuevas herramientas de diseño incluidas en Unity 4.6, las cuales aun están poco documentadas pero que prometen ser realmente potentes cara a futuros desarrollos a medida que se adquiere cierta experiencia.

Los elementos utilizados para la generación de fondos, animaciones e interfaz han sido en su totalidad recursos gratuitos, un gran número de ellos creados directamente por mi.

A continuación se muestran las diferentes vistas generadas durante el desarrollo del videojuego:

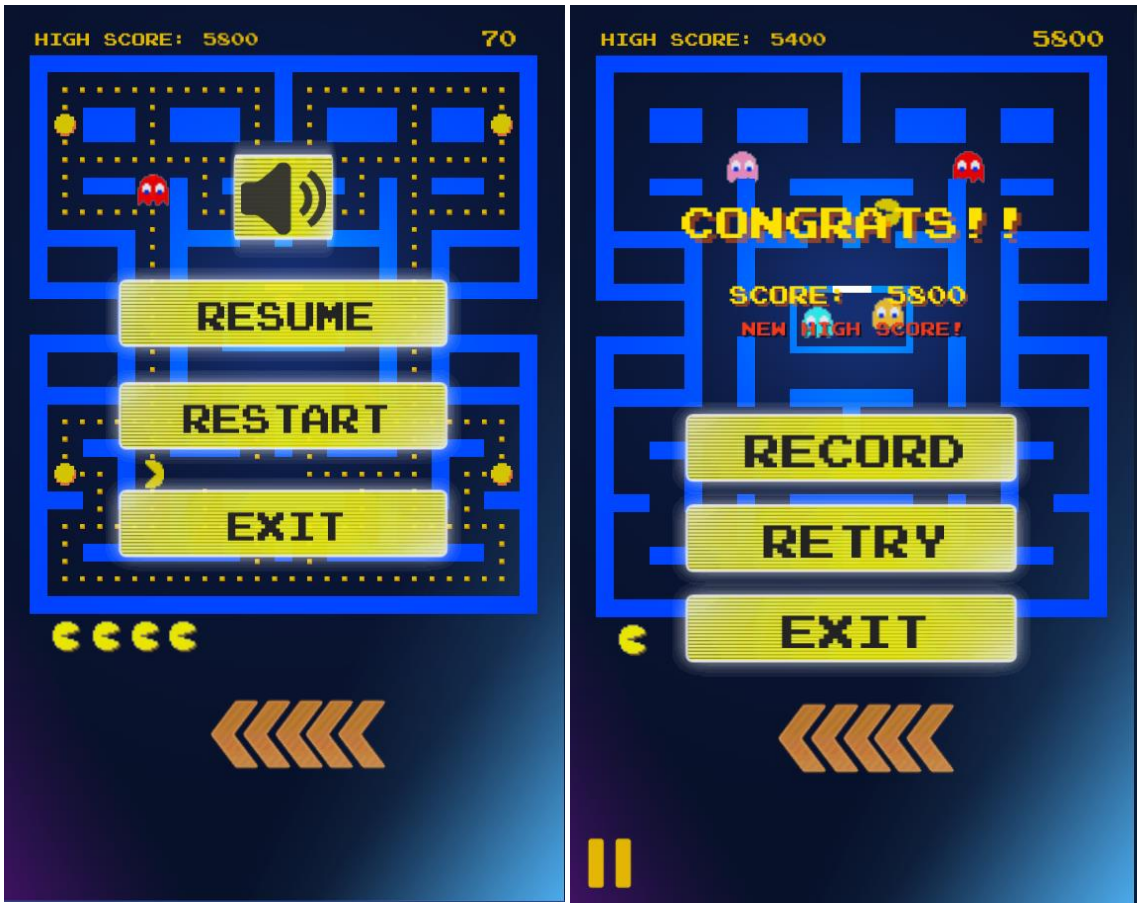




En la primera captura se observa el menú inicial del videojuego. He elegido un fondo, título y botones que permitieran una futura redistribución en caso de realizar mejoras añadiendo nuevas funcionalidades a la aplicación. Es importante tener este factor en cuenta, ya que es posible que se llegue a escoger un diseño que no permita variaciones en la interfaz sin arruinar la composición visual que se ha intentado conseguir.

En la imagen de la derecha se observa una captura inicial del nivel. En ella se puede ver los elementos del HUD orbitantes a la escena. Entre ellos se encuentra:

- La puntuación más alta obtenida hasta el momento (High Score).
- La puntuación obtenida hasta el momento (Score).
- Vidas restantes (simbolizado mediante figuras de Pacman).
- Flecha de dirección de movimiento (indica al jugador el último swipe realizado).
- Botón de pausa.



En la captura de la izquierda nos encontramos con el menú de pausa, el cual permite al jugador las siguientes opciones:

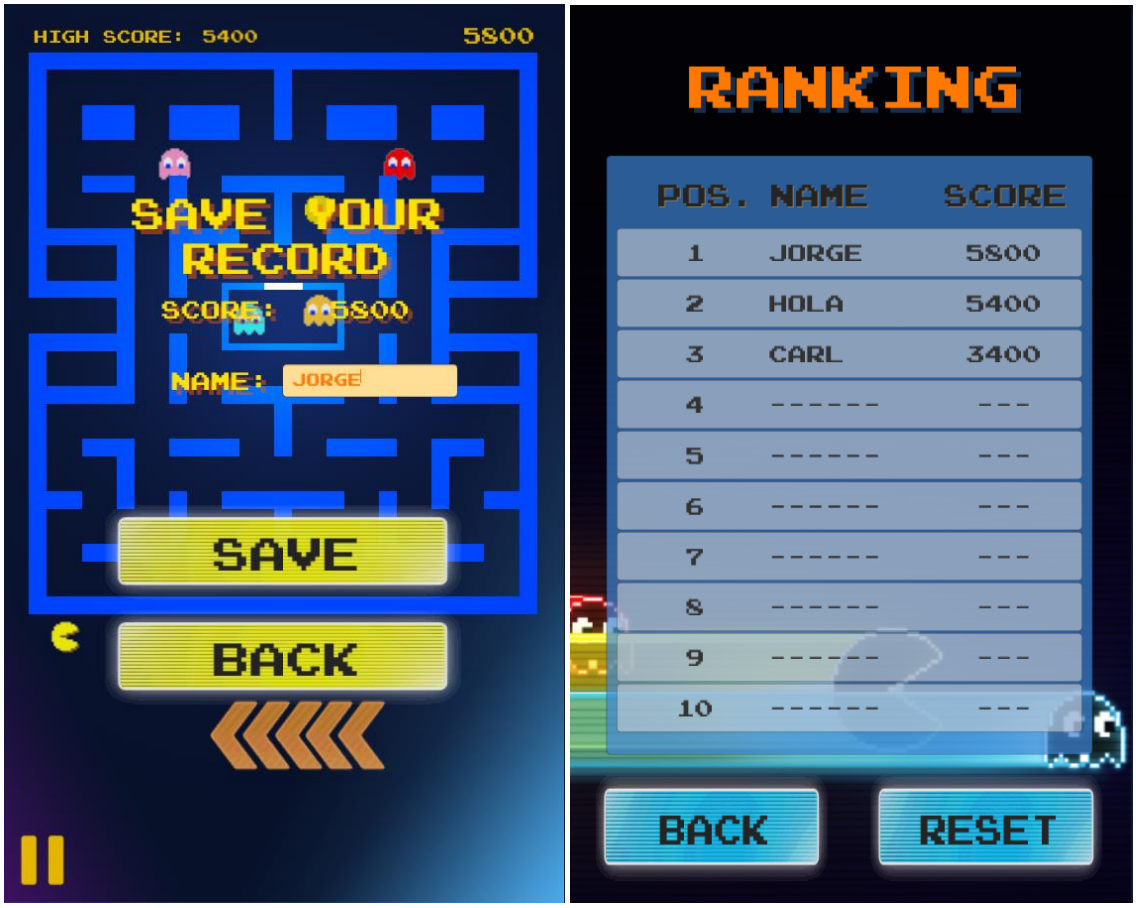
- Activar/Desactivar sonido del juego.
- Continuar la partida.
- Reiniciar el nivel.
- Salir al menú principal.

En la imagen de la derecha tenemos la pantalla de victoria, la cual aparece al despejar todo el laberinto de los puntos que lo pueblan. Una vez completado el nivel se informa al usuario de su puntuación obtenida, así como si es o no un nuevo record.

Asimismo se le permiten las siguientes acciones:

- Registrar la puntuación obtenida en el ranking.
- Reintentar el nivel.
- Salir al menú principal.

Existe una pantalla de derrota similar a la de victoria. Se informa del fin de la partida, muestra la puntuación obtenida hasta el momento y ofrece la opción de reintentar y de salir al menú principal.



En la captura de la izquierda nos encontramos con la pantalla de registro de puntuación, a la cual se puede acceder tras la pantalla de victoria. En esta es posible introducir un nombre al cual se asociará nuestra puntuación. A continuación, es posible guardar esta puntuación o regresar a la vista anterior.

La captura de la derecha muestra la tabla de las puntuaciones más altas registradas en el dispositivo. Estas puntuaciones pueden ser consultadas en cualquier momento desde el menú principal, y es posible eliminar las marcas previamente almacenadas mediante la opción Reset.

INFORMACIÓN PERSISTENTE

Una de las necesidades que surgieron en el diseño de esta aplicación fue la persistencia de algunos datos en el sistema, de forma que si el usuario cerraba la aplicación completamente y accedía en otro momento, estos estuvieran consolidados y fueran accesibles independientemente del tiempo y las circunstancias que hubieran transcurrido.

Para lograr esto he hecho uso de determinados métodos que proporciona Unity para guardar variables sencillas (strings, floats…) en la memoria del dispositivo en el que se está ejecutando la aplicación móvil sin llegar a crear de forma específica sistemas de almacenamiento y carga de datos específicos para cada terminal en el que se pueda ejecutar la aplicación.

Esta abstracción del dispositivo final ahorra mucho trabajo en caso de realizar una publicación multiplataforma, y siendo una solución altamente eficiente.

A lo largo de este desarrollo ha sido necesario almacenar dos tipos de datos:

- **Ranking**: listado con puntuaciones y el usuario que la ha conseguido.
- **Opciones de sonido**: en este caso particular, únicamente la opción de "Muted" (silenciado).

Para ello creé un script encargado de la carga y guardado de ambos tipos de datos. A continuación mostraré los métodos claves del controlador de persistencia del sonido:

```
public bool loadSound(){
		if (PlayerPrefs.GetInt("Sound") == 0){
			return false;
		} else {
			return true;
		}
	}

	public void saveSound(bool sound){
		if (sound){
			PlayerPrefs.SetInt("Sound", 1);
		} else {
			PlayerPrefs.SetInt("Sound", 0);
		}
		onSoundChanged(sound);
	}
```

Como se puede observar, mediante las sentencias PlayerPrefs.GetInt("Sound") se obtiene el valor de la variable "Sound" almacenada previamente en la memoria del dispositivo. Si no existe ningún registro previo y al tratarse de un entero, el valor por defecto es 0.

Así mismo, la sentencia para registrar un nuevo valor de esa variable es la siguiente: PlayerPref.SetInt("Sound", 1);.

En nuestro otro caso de persistencia, el **Ranking** con el listado de puntuaciones y nombres, durante la ejecución los controladores trabajan con un SortedList, pero este objeto es demasiado complejo para el almacenamiento que proporciona por defecto Unity, por lo que he creado una interfaz en el controlador de carga y guardado que traduce esta SortedList en una string. Una vez generada esta string, el procedimiento para guardar la información en memoria es idéntico al caso anterior.

```
public SortedList loadList(){
		SortedList list = new SortedList();
		string srank = PlayerPrefs.GetString("Ranking");
		if (srank == ""){
			return list;
		}
		string score = "";
		string name = "";
		bool field1 = true;
		for (int i = 0; i<srank.Length; i++){
			…
		}
		return list;
	}

	public void saveList(SortedList list){

		string srank = "";
		if (list.Count != 0){
			for (int i = list.Count-1; i>=0; i--){
				…
			}
		}
		PlayerPrefs.SetString("Ranking", srank);
	}

	public int getHighScore(){
		SortedList list = loadList();
		if (list.Count == 0){
			return 0;
		} else {
			return (int) list.GetKey(list.Count-1);
		}
	}

	public void saveRecord(int score, string name){
		SortedList list = loadList();
		list.Add(score, name);
		saveList(list);
	}
}
```

EVENTOS Y DELEGADOS

A lo largo de la ejecución de la aplicación se generan un alto número de situaciones que deben ser gestionadas por algún controlador o reflejar cambios en la información en determinados modelos.

Este tipo de situaciones son capturadas mediante eventos y referencias a delegados. A continuación se muestra un caso básico de captura y tratamiento de eventos:

Por un lado, el objeto que genera el evento.

```
public class CollisionScript : MonoBehaviour {
	public delegate void eaten(GameObject food);
	public static event eaten onEat;

	void OnTriggerEnter(Collider col){
		if (col.gameObject.name == "PacmanCenter"){
			if (onEat != null){
				onEat(this.gameObject);
			}
		}
	}
}
```

Y por otro, el receptor del evento:

```
void OnEnable(){
	CollisionScript.onEat += HandleOnEat;
	…
}

void HandleOnEat(GameObject food){
	…
}

void OnDisable(){
	CollisionScript.onEat -= HandleOnEat;
	…
}
```

De esta forma, en la clase emisora del evento se invoca a un método vinculado previamente a un delegado. Una, ninguna o múltiples clases pueden estar suscritas a los eventos de ese delegado en concreto, y ante la aparición de un evento se ejecuta el método oportuno de la clase receptora.

Esta clase receptora debe disponer de métodos `OnEnable()` y `OnDisable()`, los cuales se ejecutan automáticamente con la creación y destrucción del objeto. De esta forma se evitan referencias a delegados desde clases que están inactivas o han sido eliminadas.

Esta gestión de eventos se realiza para informar de las siguientes situaciones en diferentes puntos de la ejecución:

- Un punto ha sido comido.
- Un power up ha sido comido.
- Un fantasma ha sido comido.
- Pacman ha muerto.
- Un fantasma ha vuelto a su posición inicial
- Todos los fantasmas han vuelto a su posición inicial

El objetivo de dividir los dos últimos elementos tiene el objetivo de respetar las responsabilidades de cada controlador. El evento de llegadas individuales de fantasmas a la base es gestionado por el GhostsController, mientras que la llegada final de todos y cada uno de ellos supone que puede comenzar la siguiente ronda, y por lo tanto es gestionado por el LevelController.

ESTADOS

A lo largo de la creación de este videojuego ha sido necesario definir un total de cinco Máquinas de Estados Finitos. El uso de estas MEF ha sido estructurar y definir de forma más modulada y escalable los comportamientos de los diferentes elementos que hacen uso de ellas.

Concretamente, estas máquinas de estados han sido implementadas en las siguientes clases:

- **PlayerController**
- **BlinkyController**
- **PinkyController**
- **InkyController**
- **ClydeController**

El objetivo de dividir el tratamiento de los controladores de los cuatro fantasmas es dotar a cada uno de estados completamente diferentes, así como las transiciones entre estos. Esto no ha sido posible en su totalidad en esta versión desarrollada para el Proyecto de Fin de Grado, pero la separación permite poder realizar estas modificaciones como mejoras futuras.

Como objeto de análisis para esta explicación del uso de estados durante la implementación he decidido seleccionar un controlador genérico de los fantasmas:

```
public enum ghostState{
		Initial,
		Routine,
		Chasing,
		Running,
		Dead,
		Backing,
		Paused
}
void Start (){
	…
	StartCoroutine(FSM());
}
protected IEnumerator FSM(){
	while(true){
		yield return StartCoroutine (currentState.ToString());
	}
}
IEnumerator Initial(){
		while (currentState == ghostState.Initial){
			…
		}
		yield return 0;
	}
}
IEnumerator Routine(){
		while (currentState == ghostState.Routine){
			…
		}
		yield return 0;
	}
}
IEnumerator Chasing(){
		while (currentState == ghostState.Chasing){
			…
		}
		yield return 0;
	}
}
…
```

Inicialmente es necesario crear una variable enum que contenga todos los estados por los que podrá pasar el objeto. En el caso del fantasma genérico:

- **Initial**: estado inicial, en el cual se mantiene realizando pequeños movimientos sobre su posición hasta que se le ordene pasar al estado de rutina.
- **Routine**: estado de exploración del fantasma. Durante este estado cada fantasma recorre el laberinto de una forma diferente mediante puntos de ruta fijados de forma particular en cada caso.
- **Chasing**: estado de persecución sobre Pacman. Cuando se encuentran en este estado, el objetivo de los fantasmas de forma general es siempre la posición del jugador.
- **Running**: estado en el que entran los fantasmas cuando Pacman ingiere un power up. Durante este corto periodo de tiempo los fantasmas reducen su velocidad y huyen del jugador.

- **Dead**: estado en el que entran los fantasmas una vez han sido comidos por Pacman. Durante este estado el fantasma vuelve rápidamente a la casa de fantasmas.
- **Backing**: estado al que pasan los fantasmas una vez han capturado a Pacman. Durante este estado los fantasmas vuelven rápidamente a sus posiciones iniciales.
- **Paused**: estado de detención de movimiento generado por un menú o la opción de pausa.

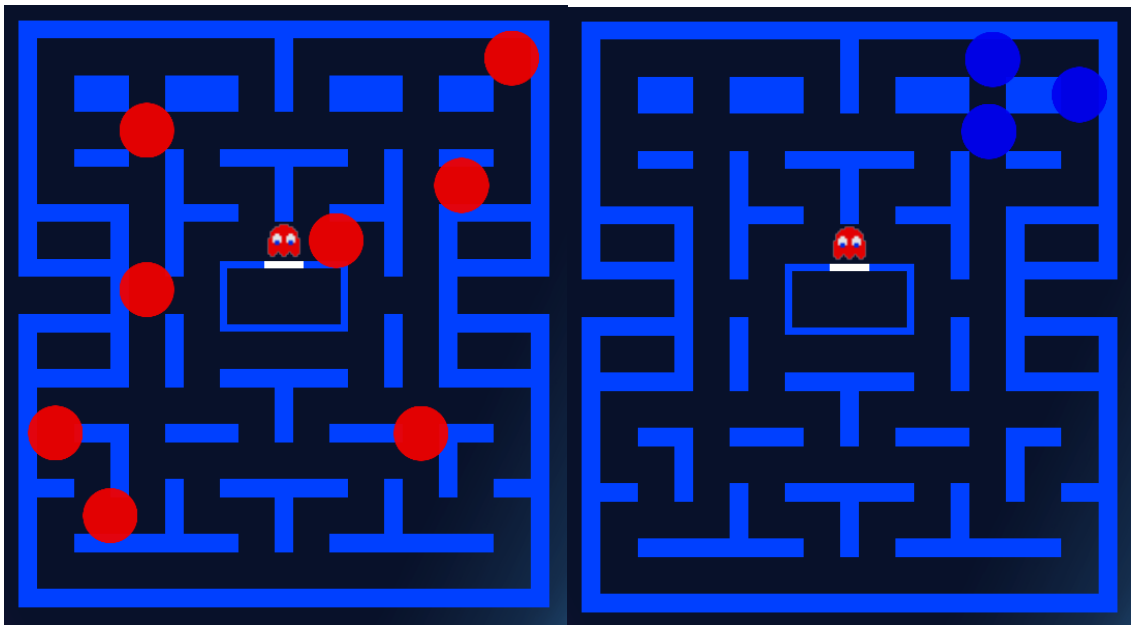
A continuación es necesario iniciar una corrutina que se encargará de invocar en cada situación al método asociado a cada estado. En este caso, los estados también son corrutinas. Estas corrutinas contienen para cada estado las iteraciones que se deben realizar.

Los cambios de estado se realizan mediante el método `changeState(ghostState)`.

Uniendo todos estos elementos se obtiene una estructura de control de los diferentes comportamientos que va tomando cada objeto dinámico de forma sencilla y fácil de ampliar y modificar.

Debo hacer especial hincapié en la particularidad del estado Routine y Running de cada uno de los fantasmas, ya que en función de la personalidad original creé una estructura de puntos de ruta diferente para cada uno. A continuación mostraré el comportamiento por defecto que tiene cada uno de estos fantasmas sin la presencia de Pacman:

1. **Blinky**

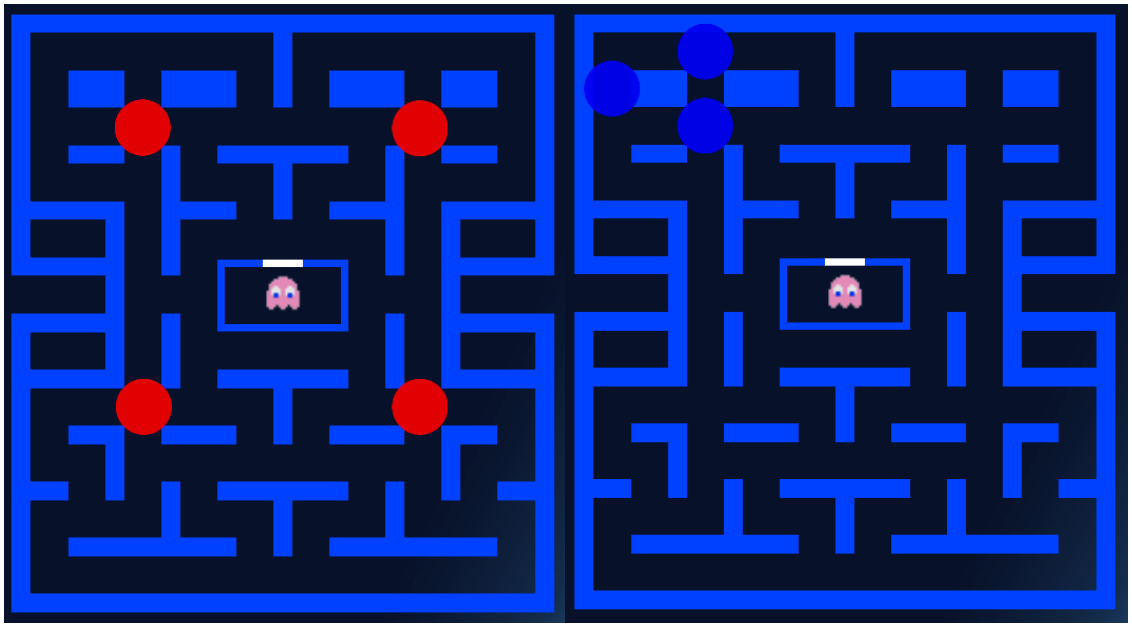


En la imagen de la izquierda se pueden observar los puntos de ruta que sigue Blinky de forma cíclica hasta que detecta la posición de Pacman. En la imagen de la derecha se representan los puntos de ruta que tendrá cuando este sea vulnerable, es decir, mientras huya de Pacman. Estos puntos de ruta son fieles al videojuego original, ya que cada fantasma se

dirigía a dar vueltas a un determinado punto del mapa una vez Pacman había consumido un *power up*, pero generalmente no disponían de tiempo para llegar a este objetivo.

Con esta rutina establecida para Blinky pretendía simular su posición agresiva del juego original, siendo el que más número de pasillos recorre con el objetivo de encontrar a Pacman y a partir de ahí perseguirle.

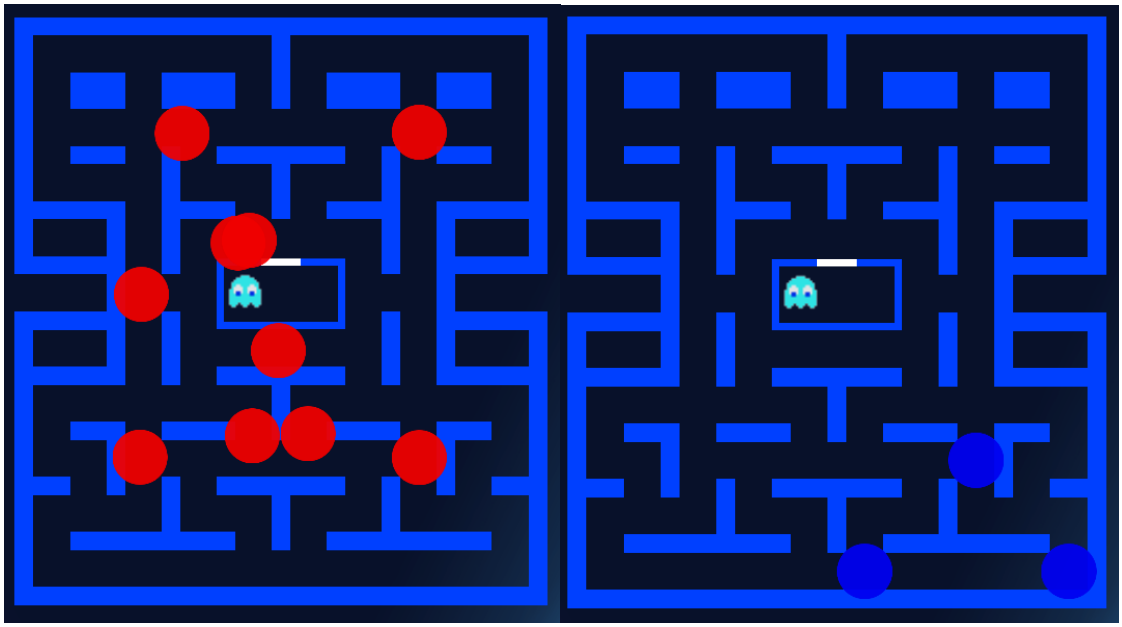
2. Pinky



De nuevo, a la izquierda se muestran los puntos de ruta cíclicos realizados durante la rutina de Pinky, y a la derecha los puntos de huida.

Pinky es el fantasma más ingeligente y previsor en el videojuego original, y pretendía transmitir esa sensación mediante una ruta específica que abarcara los puntos clave de unión de los cuatro cuadrantes del laberinto, ya que el jugador deberá pasar por ellos en repetidas ocasiones para avanzar. De esta forma es más probable que Pinky coincida con Pacman rápidamente.

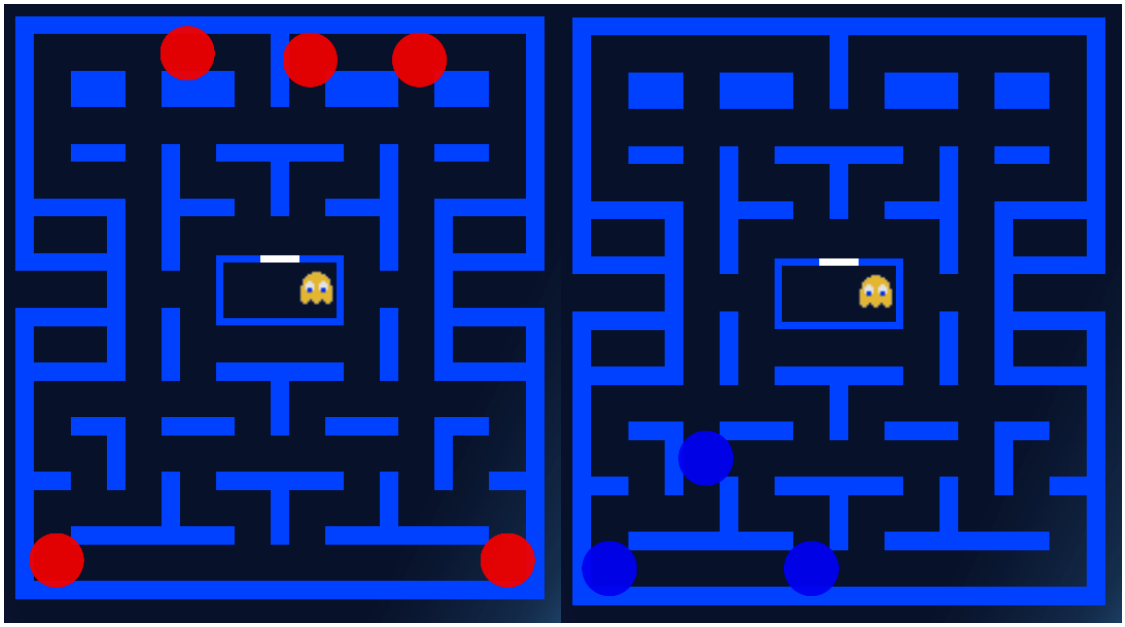
3. Inky



A la izquierda tenemos la rutina cíclica de Inky, y a la derecha la ruta de huida.

Este fantasma actúa como el más impredecible de los cuatro, puesto que su comportamiento alterna entre el agresivo de Blinky y el previsor de Pinky. He intentado reflejar esto con una mezcla de patrones de ruta, controlando los cuatro nexos claves del laberinto a la par que algunos movimientos de retroceso en determinadas ocasiones para explorar más superficie.

4. Clyde



A la izquierda la rutina cíclica de Clyde, y a la derecha su ruta de huida.

Clyde tenía la personalidad más tímida y distante de todos los fantasmas, e intentaba constantemente alejarse de sus compañeros y del propio Pacman. Para ello le he generado

una ruta bastante aislada y distante en diferentes extremos del mapa, omitiendo la zona central, lugar donde se produce un mayor número de intersecciones.

9.3. PRUEBAS

A lo largo de la implementación realicé pruebas unitarias en casos puntuales para verificar el correcto funcionamiento de determinados controladores, modelos o scripts concretos. Este tipo de pruebas resultaron en su mayoría exitosas y no fueron una gran carga de trabajo.

Sin embargo, dado que mi desarrollo consiste en la implementación de un videojuego, una fase vital del proceso es el testeo. Durante esta fase se da a probar el videojuego a diferentes usuarios una vez ha sido completamente implementado, y se evalúa:

- La dificultad
- La atención que capta
- La diversión generada
- La existencia o ausencia de reto
- La existencia o ausencia de deseo de seguir jugando

Por lo tanto, en este último punto no llegué a realizar una fase de pruebas tradicional del ciclo de vida en cascada, sino que me adentré en diversas pruebas con jugadores para evaluar sus sensaciones con el videojuego.

Estos ajustes que se realizan durante el testeo generalmente son numéricos, asociados a variables públicas, de modo que es inmediato realizar cambios hasta llegar al estado óptimo durante la realización de las pruebas.

Las variables más críticas que afectaban a la complejidad del nivel principalmente eran las siguientes:

Modelo	**Variable**	**Valor Inicial**
Ghost	BasicGhostSpeed	2
Ghost	BackingGhostSpeed	BasicGhostSpeed * 2
Ghost	RunningGhostSpeed	BasicGhostSpeed * 0.9
Pacman	BasicPacmanSpeed	2
Pacman	DetectionAreaRadio	3.8
Pacman	LoseSightAreaRadio	5.43

A medida que se dio a probar el videojuego a diferentes jugadores y se observó que la dificultad de esta primera versión era excesiva realicé una serie de ajustes seguidos de más pruebas, hasta que finalmente obtuve una versión más divertida y agradable para los usuarios. Esta versión suponía un reto inicial pero a su vez resultaba relativamente fácil de superar a la mayoría de los usuarios.

A continuación se muestra una comparativa de los valores finales con respecto a los originales:

Modelo	Variable	Valor Inicial	Valor final
Ghost	BasicGhostSpeed	2	1.7
Ghost	BackingGhostSpeed	BasicGhostSpeed * 2	BasicGhostSpeed * 2.5
Ghost	RunningGhostSpeed	BasicGhostSpeed * 0.9	BasicGhostSpeed * 0.6
Pacman	BasicPacmanSpeed	2	2
Pacman	DetectionAreaRadio	3.8	2.82
Pacman	LoseSightAreaRadio	5.43	3.99

En esta versión depurada, los fantasmas corren más lento que Pacman, vuelven más rápido al lugar de inicio cuando Pacman muere y reducen aún más su velocidad cuando el jugador consume un *power up*, de forma que le sea más fácil alcanzar a los fantasmas que huyen de él.

Por otra parte, el radio de detección de Pacman ha sido reducido considerablemente, al igual que el de pérdida de vista del jugador. De esta forma es más fácil evitar a los fantasmas al pasar a una distancia media de ellos y poder seguir con el juego sin tanta tensión.

10. CONCLUSIÓN

10.1. CONCLUSIÓN DEL TRABAJO REALIZADO

La realización de este proyecto me ha permitido entrar en contacto con del desarrollo de videojuegos de forma práctica y directa, así como con diferentes entornos de desarrollo, tanto programáticos como gráficos o de diseño. Específicamente me ha proporcionado gran soltura en el manejo de Unity. Estos conocimientos tanto de programación como de diseño me permitirán entrar en el mundo de desarrollo de videojuegos con más perspectiva y un background más relacionado con las herramientas que probablemente tenga que asimilar en el futuro.

Una de las etapas que más esfuerzo y dedicación han supuesto de este desarrollo ha sido la fase de prototipado. Inicialmente consideraba que la mayor carga de trabajo vendría dada durante la fase de implementación, pero estaba equivocado.

Durante la etapa de **prototipado** plasmé en resultados jugables cada una de las ideas que iba planteando como posible videojuego de este proyecto. Cada uno de estos intentos supuso un reto, ya que eran mi primera aproximación al entorno que utilicé, así como a los conceptos relacionados con el desarrollo de videojuegos. Estos prototipos tuvieron que ser probados por diferentes jugadores y evaluadas sus reacciones, con el objetivo de detectar cuál de todos los esbozos planteados podía servir como semilla para la realización de un videojuego realmente exitoso y agradable de jugar.

Por otra parte, una vez seleccionado el prototipo finalista, la fase de **implementación** resultó menos compleja, ya que todos los aspectos críticos del videojuego ya habían sido decididos. En esta etapa únicamente restaba reconstruir la idea desde cero, ya con una cierta experiencia, utilizando una estructura mejor planteada, ampliable y reutilizable. Pero, en esencia, conservando la idea plasmada en el prototipo.

Con la realización de este proyecto me he dado cuenta de que es muy importante saber dejar a un lado las expectativas en determinadas situaciones y afrontar los resultados tal y como son de forma objetiva, buscando caminos para mejorar estos en caso de que fuese necesario. Esto lo comento porque la idea inicial del videojuego planteado no siempre termina convirtiéndose en el producto terminado. Esta evolución durante el desarrollo puede desanimar al desarrollador, y es importante seguir con el proceso hasta el final. Una vez obtenidos los resultados, es posible que el concepto generado haya variado con respecto al original, pero esto no es siempre algo negativo. Muchas ideas buenas surgen de esta evolución a lo largo del desarrollo, y es importante tener presente el esfuerzo realizado y valorar de la forma más objetiva posible el resultado obtenido. Esta abstracción del trabajo personal permite enfocar mejor el público al que realmente se podrá dirigir este producto generado, así como prever su aceptación.

10.2. LÍNEAS FUTURAS

En este apartado se exponen todos aquellos aspectos del proyecto que puedan ser mejorables cara a futuras revisiones, pero que debido a la restricción temporal existente y a la falta de experiencia inicial no ha sido posible incluir en esta versión.

Refactorizar el código del software: a lo largo del diseño y la implementación he intentado seguir una estructura lógica y ordenada que contemple todos sus elementos desde el comienzo, de forma que el resultado sea coherente. Pero a medida que avanzaba el proyecto fueron necesarios determinados ajustes puntuales desligados de la arquitectura debido a restricciones o flujos propios del entorno de desarrollo utilizado. Estos factores hicieron que el resultado no fuera tan claro y elegante como había planteado en un comienzo. Como mejora futura reorganizaría la estructura teniendo en cuenta estas particularidades del entorno descubiertas durante el desarrollo. Esta experiencia me permitirá también ser más coherente y realista con diseños creados para futuros videojuegos.

Plantear un **enfoque de negocio**: durante la realización de este proyecto he sido consciente del fin educativo de este, y lo he utilizado como un camino para orientarme a una salida profesional específica. Sin embargo, es interesante enfocar el desarrollo de ideas a un fin económico concreto. Esta planificación de modelo de negocio, vías de ingreso, público objetico, etc. permite orientar el propio desarrollo del proyecto de una forma u otra. Me gustaría llegar a trabajar con este tipo de enfoques para adquirir esa perspectiva de negocio de forma práctica. Buscando una aplicación para este videojuego es posible generar compras de vidas por parte del usuario, de forma que sea posible seguir avanzando una vez perdidas todas. Otro método clásico es la inclusión de publicidad.

Mejorar el **aspecto visual**: mis conocimientos como diseñador gráfico dejan mucho que desear, y la interfaz visual generada ha sido una pequeña aproximación al resultado visual final que había planteado darle al videojuego. Como línea de mejora clara incluiría un mayor número de animaciones a diferentes elementos del entorno, así como un lavado de cara a las paredes que conforman el laberinto, diseñando para cada pieza una imagen semitransparente similar al videojuego original. Otro elemento que mejoraría en gran medida la sensación transmitida sería dotar de efectos de iluminación o partículas a los diferentes mensajes con texto repartidos por la interfaz cuando estos cambien. Este feedback visual provoca en el jugador sensaciones positivas que le invitan a seguir utilizando la aplicación.

Incorporación de varios **niveles**: en esta implementación he generado un único nivel que contiene la mayoría de elementos que conforman el videojuego. Pero en la versión original de Pacman se disponía de un total de 20 niveles con variaciones de dificultad específicas. El proyecto ha sido diseñado de forma que sea muy sencillo cargar niveles de dificultad al juego al comienzo de cada pantalla, así como la generación de un menú de selección de nivel con persistencia de datos.

Ralentización de Pacman: en el videojuego original el jugador sufría una ligera desaceleración en su trayectoria al ingerir cada uno de los puntos repartidos por el nivel. Este aspecto no ha sido reproducido en esta adaptación, aunque sí contemplado. El motivo fue el uso de las 300 horas del proyecto en tareas más prioritarias del desarrollo.

Adaptación de la interfaz a todo tipo de pantallas: durante las siguientes etapas del proyecto tuve acceso únicamente a un dispositivo móvil Android, y se trataba de un Tablet Samsung Galaxy Note 10.1, por lo que las pruebas de interfaz que realicé fueron generadas enfocadas al ratio de aspecto de esta pantalla. El diseño general de los módulos de la interfaz ha sido configurado de forma que respete los valores relativos de cada pantalla. De cualquier forma, cara a mejoras futuras, es necesario realizar pruebas con diferentes tipos de pantallas para finalmente adaptar el diseño de esta interfaz a una completamente dinámica.

Aplicar **tiempos de espera** programados: un elemento que no ha sido reproducido en esta adaptación ha sido una serie de aproximaciones a cinemáticas muy básicas. Con esto me refiero a tiempos de espera en los que se produce alguna animación o se espera un número de segundos para esperar a que el jugador se prepare. Me refiero al interludio presentado al cargar un nivel en el que se esperan varios segundos esperando a que el jugador observe el entorno, y a continuación comienza la fase. De igual forma, la victoria y derrota van precedidas de algunos segundos de observación por parte del usuario antes de mostrar los respectivos mensajes. Este tipo de tiempos de espera intencionados son elementos que ayuda al jugador a asimilar la situación en la que se encuentra, y son aspectos interesantes a incluir en un futuro.

11. BIBLIOGRAFÍA

A continuación procederé a indicar los recursos sin los cuales el desarrollo de este proyecto habría resultado mucho más complejo. En esencia son pocas las webs que he utilizado a lo largo del proceso, ya que un alto número de explicaciones y resolución de dudas han provenido de mis tutores y entorno de estudio.

DOCUMENTACIÓN DE UNITY3D

El portal más consultado a lo largo de este desarrollo ha sido sin lugar a dudas la documentación suministrada por el equipo de Unity3D. En ella se explica el funcionamiento tanto de forma gráfica como programática de cada una de las herramientas que incorpora, acompañado de ejemplos y vídeos en determinados casos.

Enlace: http://docs.unity3d.com/Manual/index.html

PACMAN DOSSIER

Toda la información técnica sobre el funcionamiento del videojuego Pacman original ha sido obtenida a partir de este documento que recopila y analiza cada uno de los aspectos técnicos y de diseño presentes en el videojuego. Este análisis es realmente interesante y completo.

Enlace: http://home.comcast.net/~jpittman2/pacman/pacmandossier.html

INTERNET ARCADE

Durante la fase de prototipado fue necesario acceder a un gran número de videojuegos clásicos cara al análisis de la viabilidad como videojuego candidato a ser adaptado a dispositivos móviles. Estos accesos a diferentes videojuegos fue realizado a través del apartado Internet Arcade de la web archive.org. En él se listan un total de 649 videojuegos (en el momento de la realización del proyecto) arcade preparados para ser ejecutados desde el navegador como si se tratara de la versión original.

Enlace: https://archive.org/details/internetarcade

WIKIPEDIA

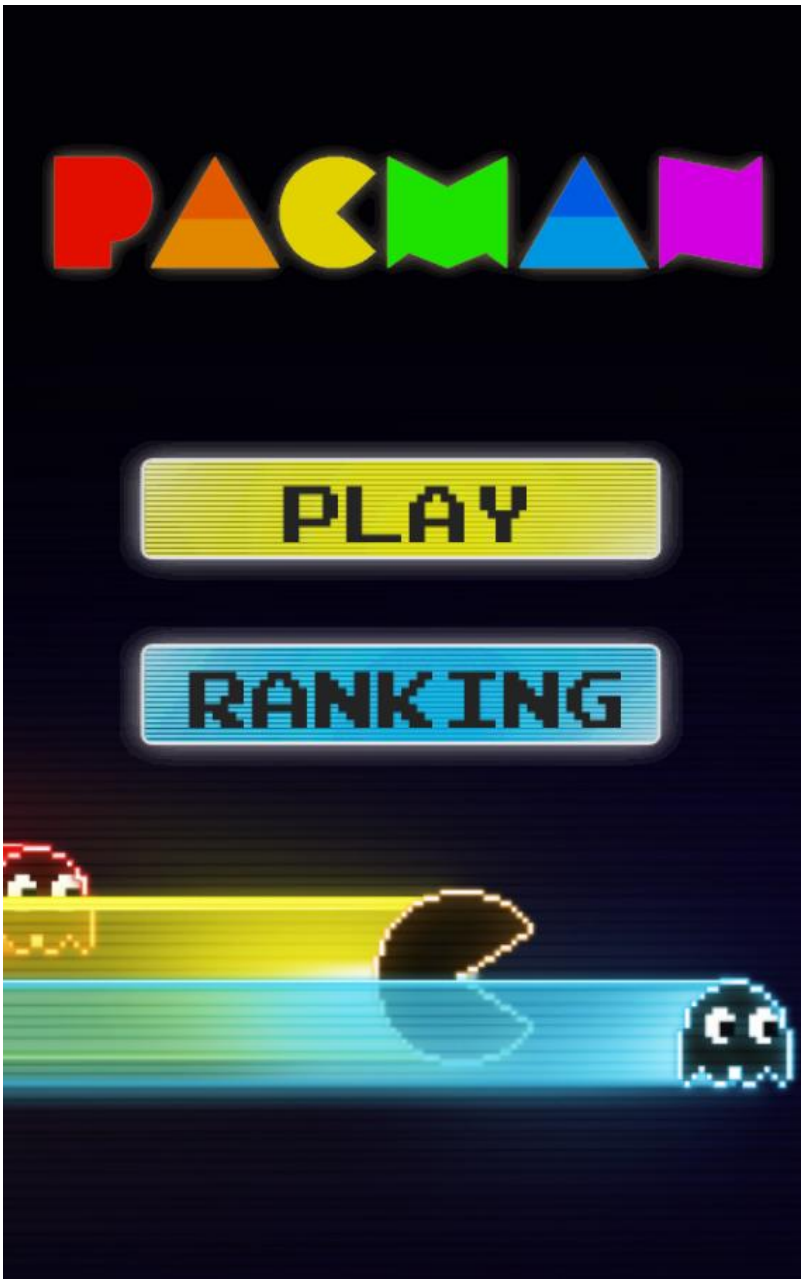
Las tareas de recopilación de información histórica realizadas durante las primeras etapas del proyecto han recibido prácticamente la integridad de los datos de diferentes artículos del portal Wikipedia. Estos accesos han sido en su mayoría a la versión inglesa, puesto que en términos generales es la más completa y profunda en temáticas relativas a la informática.

Enlace: http://www.wikipedia.org/

12. APÉNDICES

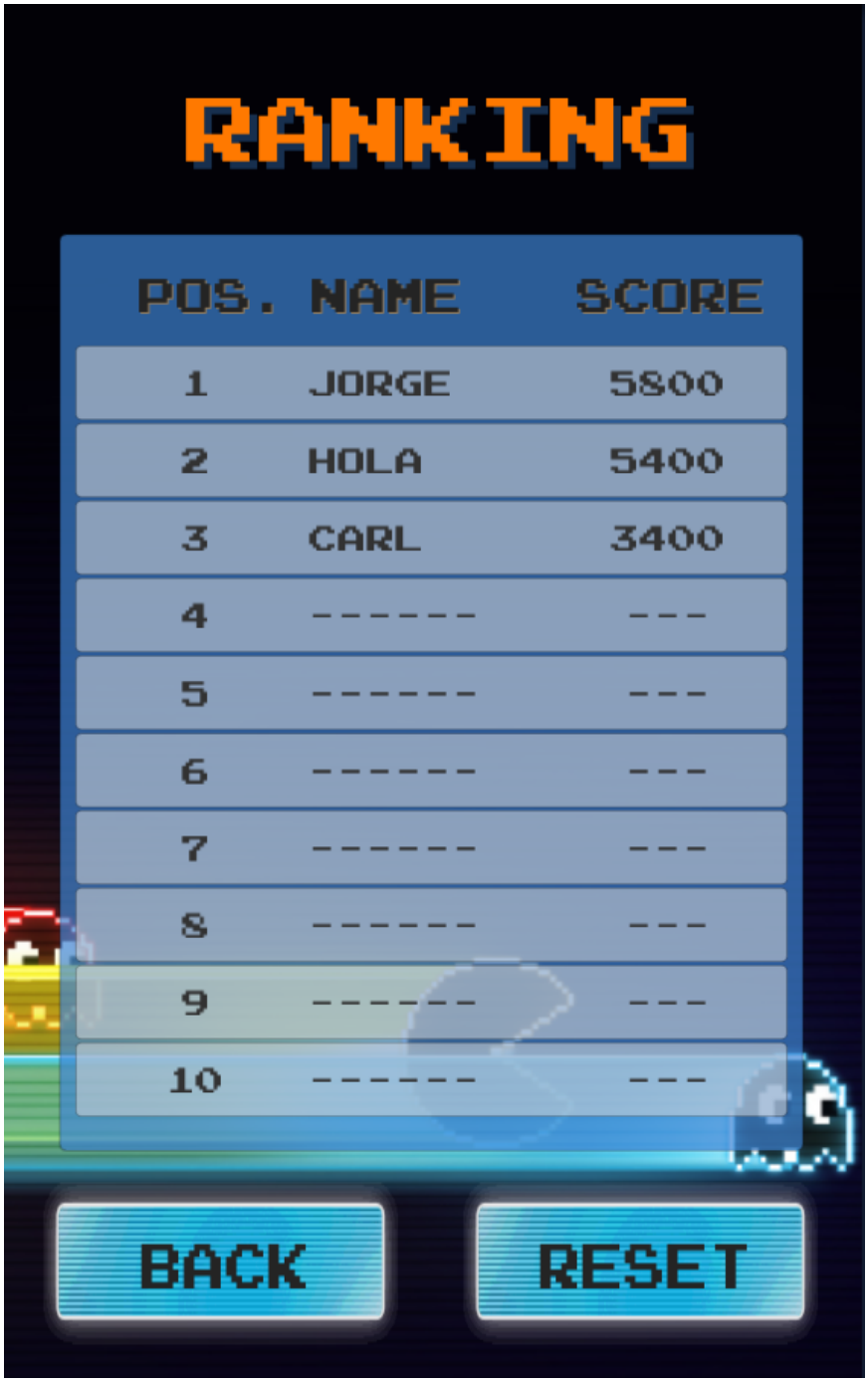
12.1. MANUAL DE USUARIO

En este apéndice se incluye un manual de usuario que permita a los futuros jugadores comprender cómo utilizar la aplicación de forma realmente simple. A continuación se comenzará con una vista inicial de la aplicación una vez iniciada:



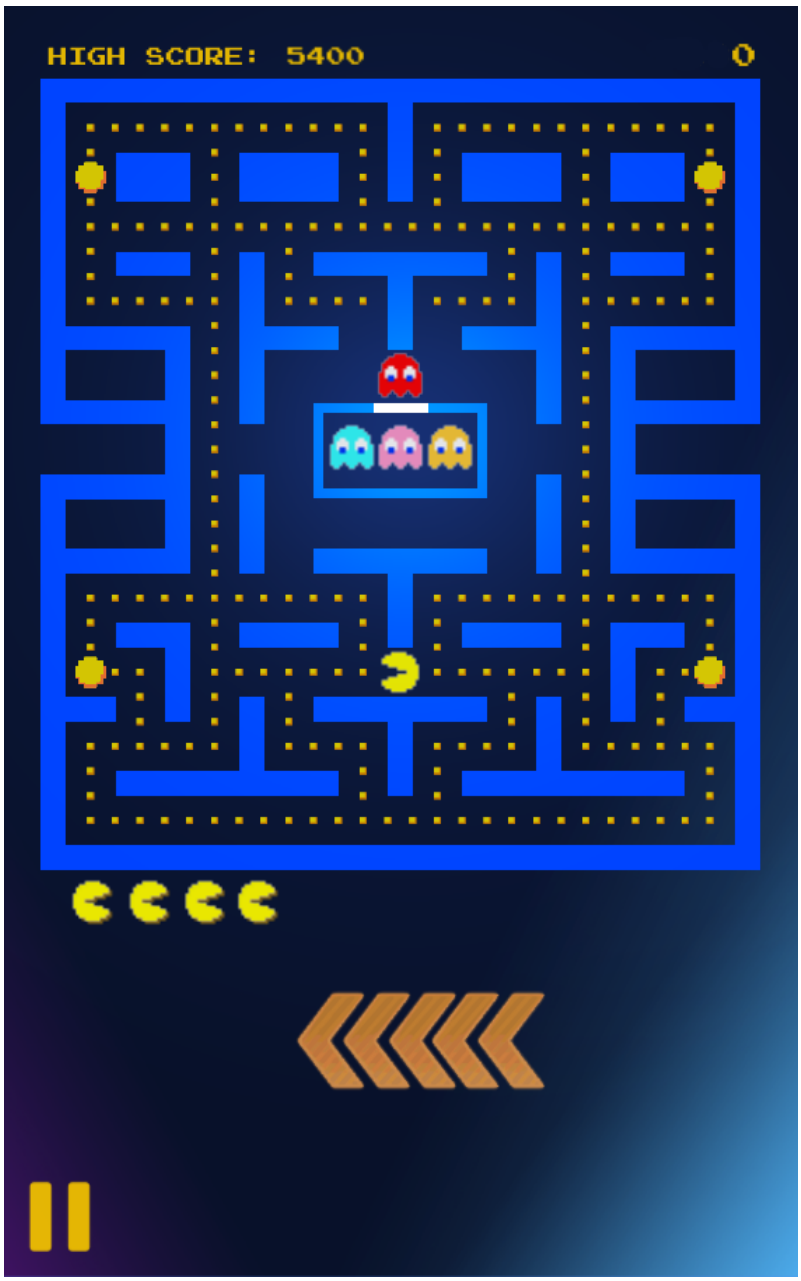
Opción	Descripción
PLAY	Inicia el primer nivel del juego.
RANKING	Muestra la tabla de ranking.

En la siguiente captura se muestran el **listado de puntuaciones** conseguidas hasta el momento:



Opción	Descripción
BACK	Volver al menú principal.
RANKING	Eliminar todas las puntuaciones almacenadas.

La siguiente captura refleja el **inicio de una partida**:



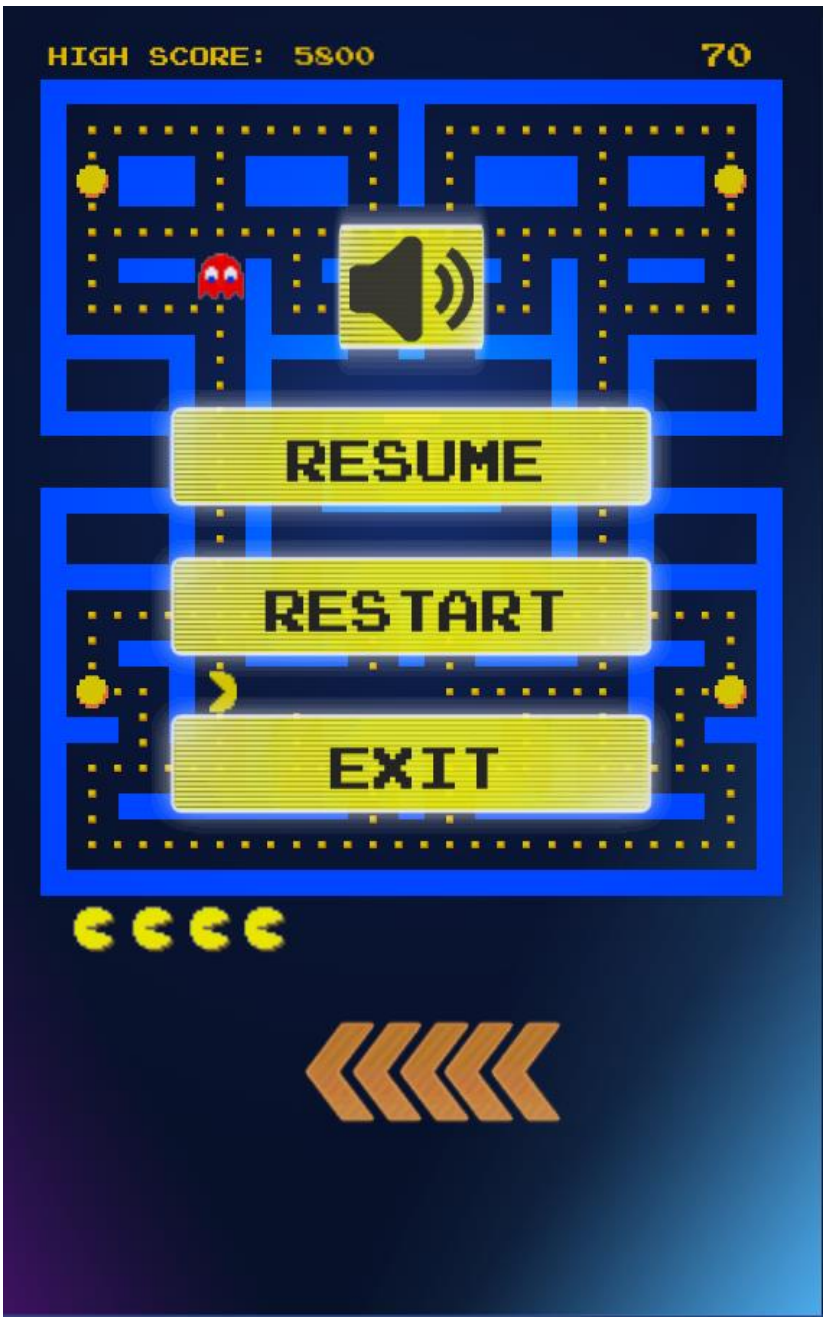
Para desplazar a Pacman basta con deslizar el dedo por la pantalla en la dirección en la que deseas moverlo.

La **flecha** de la zona inferior de la pantalla indica la dirección a la que se moverá Pacman siempre y cuando sea posible.

En la esquina inferior izquierda existe un botón que permite **pausar** la partida en cualquier momento.

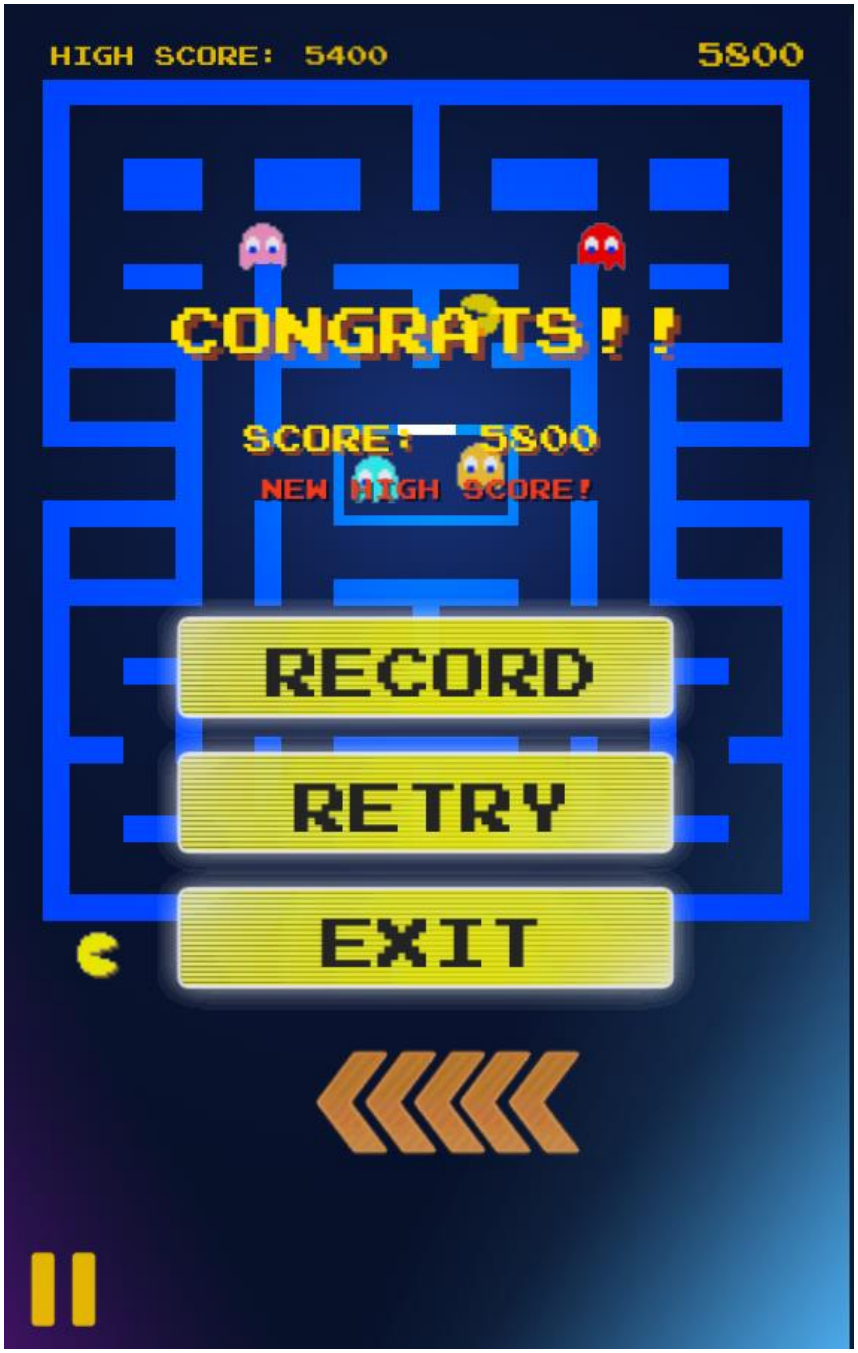
En el área superior al laberinto se muestra a la izquierda la **puntuación máxima** obtenida hasta el momento, y a la derecha la **puntuación actual**.

La siguiente captura muestra el **menú de pausa**:



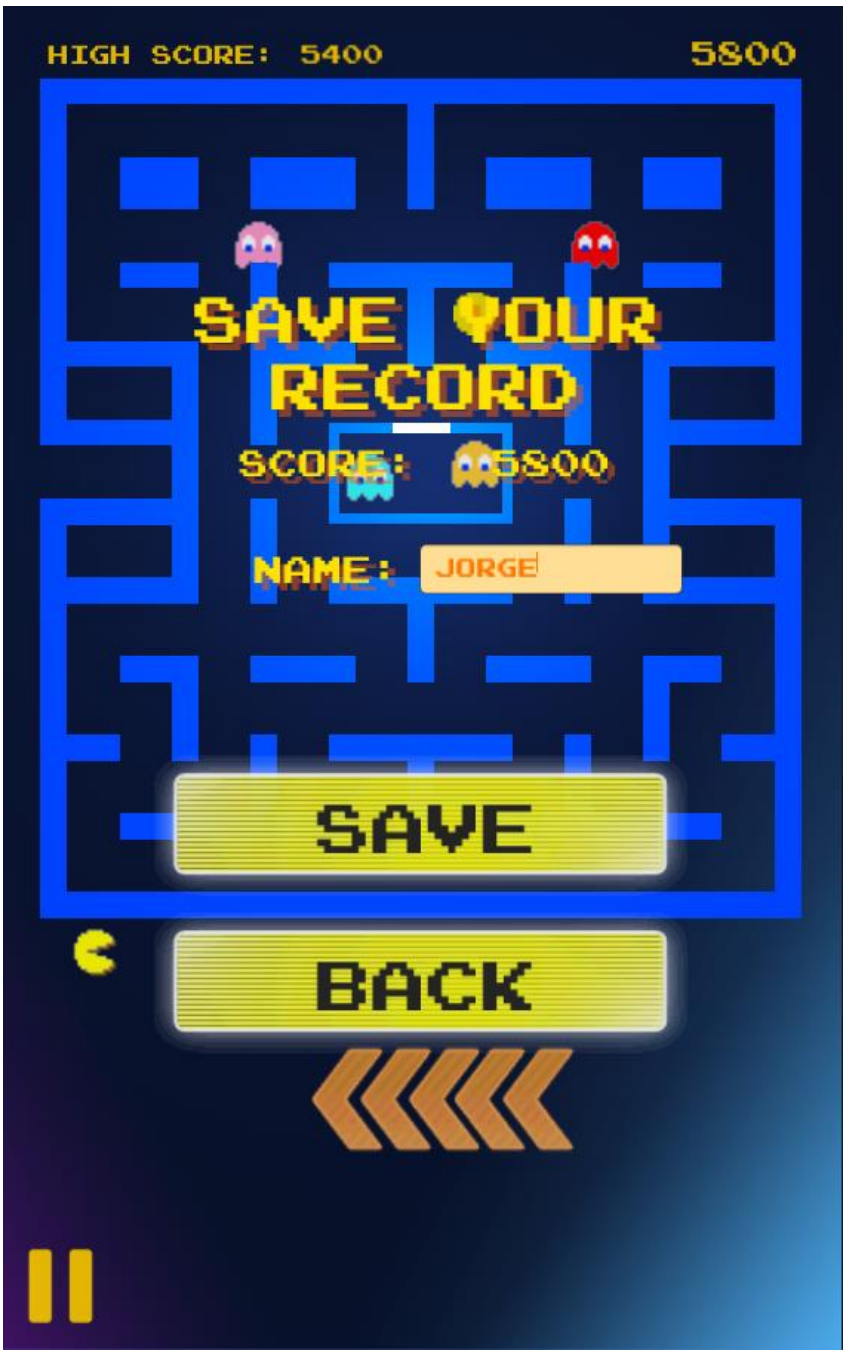
Opción	Descripción
MUTE	Permite activar y desactivar el sonido del juego.
RESUME	Continúa la partida.
RESTART	Reinicia el nivel.
EXIT	Vuelve al menú principal.

La siguiente captura muestra el **menú de victoria** una vez se ha completado con éxito el nivel:



Opción	Descripción
RECORD	Permite registrar la puntuación en el ranking.
RETRY	Reinicia el nivel.
EXIT	Vuelve al menú principal.

Esta última pantalla permite **guardar la puntuación** obtenida en la última partida:



Opción	Descripción
SAVE	Guarda la puntuación con el nombre introducido.
BACK	Vuelve a la pantalla de victoria sin guardar la puntuación.