UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de videojuegos clásicos con Scratch

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Miguel Marqués Moros

Tutor: Xavier Molero Prieto

Curso 2015-2016

Resumen

Esta memoria pertenece al trabajo fin de grado en el que se van a desarrollar videojuegos clásicos (Frogger, Duck Hunt y Super Mario Bros) que en su época fueron pioneros dentro de su género. Para el desarrollo se utilizará el lenguaje de programación Scratch proporcionada por el MIT. Una vez terminado el trabajo, pasará a estar a disposición del Museo de Informática de la Escuela Técnica Superior de Ingeniería Informática (UPV), el proyecto se subirá a la página web del museo con el objetivo de demostrar la potencia que supone Scratch para la programación de videojuegos. En cuanto a la estructura de la memoria, primero se realizará una introducción de los videojuegos situándolos en su contexto histórico, seguido de una explicación detallada del lenguaje de programación Scratch empleado para la elaboración del proyecto y a continuación se describirá paso a paso el desarrollo de los videojuegos de forma que el lector pueda observar y aprender cómo utilizar Scratch para programar videojuegos.

Palabras clave: diseño de videojuegos, videojuegos clásicos, pensamiento computacional, videoconsolas, Scratch, programación.

Resum

Aquesta memòria pertany al treball fi de grau en què es van a implementar videojocs clàssics (Frogger, Duck Hunt i Súper Mario Bros) que en la seua època van ser pioners dins del seu gènere. Per al desenrotllament s'utilitzarà el llenguatge de programació Scratch proporcionada pel MIT. Una vegada acabat el treball, passarà a estar a disposició del Museu d'Informàtica de l'Escola Tècnica Superior d'Enginyeria Informàtica (UPV) , el projecte es pujarà a la pàgina web del museu amb l'objectiu de demostrar la potència que suposa Scratch per a la programació de videojocs. Quant a l'estructura de la memòria, primer es realitzarà una introducció dels videojocs situantlos en el seu context històric, seguit d'una explicació detallada del llenguatge Scratch empleat per a l'elaboració del projecte i a continuació es descriurà pas a pas el desenrotllament dels videojocs de manera que el lector puga observar i aprendre com utilitzar Scratch per a programar videojocs.

Paraules clau: disseny de videojocs, videojocs clàssics, vídeo consoles, Scratch, programació, pensament computacional.

Abstract

This memory belongs to the Trabajo Fin de Grado where I will develop classic video games (Frogger, Duck Hunt and Super Mario Bros) which in their time were pioneers in their genre. In this work I have used Scratch, Scratch is a videogame development lenguage for kids provided by the MIT. Once work is completed, it will be available to the Museo de Informática de la Escuela Superior de Ingeniería Informática (UPV), the project will be uploaded to the website of the museum with the aim of demonstrating the power of Scratch for videogame development. As for the structure of memory, first an introduction to the videogames by placing them in their historical context, followed by an explanation of Scratch that I used for the development of the project and then I will describe step by step the development for each videogame so the reader can observe and learn how to use Scratch to develop video games.

Key words: videogame design, retro videogame, console, Scratch, programming, computational thinking.

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

En este capítulo se exponen las razones por las que me he decantado a realizar este proyecto. Se explicarán los objetivos que pretendo cumplir una vez finalizado el proyecto. Además también se dedica un apartado para describir brevemente la estructura que posee esta memoria y otro apartado para explicar cómo se han usado las fuentes bibliográficas en el desarrollo del trabajo. En dicho apartado se referenciarán todas las fuentes bibliográficas y se indicará el por qué se ha utilizado cada fuente.

1.1 Motivación

La principal motivación para realizar este trabajo fin de grado (TFG) ha sido mi pasión por los videojuegos, sobre todo los videojuegos clásicos que fueron los que marcaron una época dorada en mi infancia. Llevo jugando a videojuegos desde hace mucho tiempo, inicié mi andadura en este mundo con una NES, la cual me regalaron cuando yo era pequeño. En ella jugué a muchos juegos, unos me gustaron más, otros menos, algunos de ellos los considero de los mejores videojuegos que he jugado en mi vida y aún a día de hoy sigo jugándolos. En cuanto vi que tenía la oportunidad de reprogramar algunos de estos videojuegos para el TFG supe que este era mi proyecto.

Otra de las motivaciones es demostrar cómo un lenguaje de programación sencillo como Scratch es capaz de emular la programación de videojuegos clásicos, sobre todo en 2D que en su época eran todo un reto tecnológico, pese a ser un lenguaje orientado a todos los públicos para que aprendan a programar de una manera interactiva. Entre las actividades que realiza el Museo de Informática se encuentran los talleres de aprender a programar con Scratch, es por esto que el proyecto se subirá a la página web de forma que todo el mundo pueda pasar un buen rato jugando.

Por último existe un *youtuber* llamado Slobulus el cual también ha supuesto una gran motivación para realizar este trabajo. Este *youtuber* realiza vídeos especiales sobre videoconsolas y videojuegos clásicos explicando sus contextos

históricos y otras curiosidades mientras juega, lo que dota al espectador de unos conocimientos desconocidos.

1.2 Objetivos

Como objetivo general, con este trabajo se pretende aprender a programar en el entorno de programación Scratch, para ello se elaborarán algunos de los videojuegos más influyentes en la historia del videojuego para más tarde incluirlos en la página oficial del Museo de Informática. De esta forma se podrá observar cómo videojuegos que antiguamente suponían un gran esfuerzo tanto tecnológico como físico para su desarrollo, actualmente se pueden imitar en un entorno de programación orientado a todo el público como es Scratch. Para que las recreaciones se asemejen lo máximo posible, todos los videojuegos se han probado previamente emulándolos ya sea en un ordenador o en una videoconsola.

Aparte de la programación de videjuegos, también se parte con objetivo de conocer la historia de estos videojuegos y sus características, así como la repercusión que estos han tenido en la industria de los videojuegos. Los videojuegos escogidos han sido Frogger, Duck Hunt y Super Mario Bros. Todos ellos pertenecen a la época dorada de los videojuegos y fueron todo unos referentes para la industria del videojuego.

1.3 Estructura de la memoria

La presente memoria se divide en ocho capítulos, lo cuales se exponen brevemente a continuación:

- **Capítulo 1:** En él se expone la principal motivación por la que se ha elegido este trabajo, los objetivos planteados previamente, la estructura planteada para la memoria y comentarios sobre la bibliografía utilizada.

- **Capítulo 2:** En este capítulo se viaja al pasado para analizar y conocer la historia de los tres videojuegos implementados en este trabajo. De esta forma se puede observar la evolución y la importancia de estos videojuegos.

- **Capítulo 3:** En él se detalla desde la historia hasta la funcionabilidad del lenguaje Scratch con su plataforma y sus herramientas de programación. También se explica de manera resumida la metodología llevada para la implementación de cada videojuego.

- **Capítulos 4, 5, 6:** Estos capítulos se centran en los desarrollos de los tres videojuegos que se compone este trabajo: Frogger, Duck Hunt y Super Mario Bros. En estos capítulos se explica detalladamente el proceso desde su diseño previo hasta finalizar su implementación.

- **Capítulo 7:** Se explica cómo se han incluido los tres videojuegos en la página web del museo de informática.

- **Capítulo 8:** En el último capítulo se recogen las conclusiones obtenidas una vez realizado el trabajo. También se analizan los objetivos alcanzados y una posible ampliación futura.

1.4 Notas sobre la bibliografía

En este apartado se va a hablar sobre la bibliografía empleada durante el desarrollo de este trabajo, centrándome en explicar cómo se ha aprovechado cada uno de los libros y artículos para aventurarse en el mundo del desarrollo de videojuegos con Scratch.

Como se comenta al principio, una de las motivaciones para realizar este trabajo han sido los vídeos del *youtuber* Slobulus. De ellos he obtenido grandes dosis de conocimiento sobre los videojuegos retro, pues este *youtuber* realiza vídeos en los que explica la historia de videojuegos clásicos mientras lo va jugando.

Para ampliar mis conocimientos sobre los videojuegos históricos he utilizado [1, 5, 6, 11, 16]. En estos libros se habla de los videojuegos más influyentes de la historia y todo el contexto histórico en el que se desarrollaron. Además explican cómo ha evolucionado la industria del videojuego a lo largo de todos estos años.

En cuanto a la estructura de la memoria, se ha consultado un trabajo final de grado. El trabajo corresponde a un antiguo alumno de la escuela, Samuel Villaescusa Vinader, cuyo trabajo se puede consultar en [18]. Este trabajo supone la continuación del de Villaescusa por lo que se ha optado por seguir la misma estructura con los capítulos, dedicando así un capítulo a la historia de los videojuegos implementados y otro al lenguaje de programación Scratch. Además cada videojuego poseerá su propio capítulo para explicar su diseño y su implementación.

Todo lo relacionado con la creación de videojuegos y programación con en lenguaje Scratch se recoge de muchos libros (consultar [3, 9, 10, 12, 13, 17]) Estos libros suponen la mayor parte del contenido de este trabajo, aunque el libro más importante ha sido [17]. En este libro se indican metodologías para programar videojuegos de diferente género con el lenguaje Scratch. Las explicaciones son muy intuitivas pues se adjuntan ejemplos de código que sirven para el desarrollo de otros videojuegos. Además se han consultado artículos sobre Scratch redactados por su creador Mitchel Resnick [15, 14] y Jannette M Wing [19]. También otros artículos relacionados con la historia de los videojuego [7, 2].

Por último, otro de los documentos a tener en cuenta es [4]. Todos los videojuegos se basan en máquinas de estado, esto quiere decir que los elementos del videojuego no mantienen su estado durante todo el tiempo de ejecución. Por ello es importante esta fuente, en el documento se nos explican conceptos relaciona-

dos con esta máquinas de una forma intuitiva. Las máquinas de estado ayudan a diseñar e implementar los videojuegos.

1.5 Derechos de autor

Todos los videojuegos que componen este trabajo son marcas registradas por lo que su implementación en Scratch se hace sin ánimo de lucro. Duck Hunt y Super Mario Bros pertenecen a la compañía Nintendo mientras que Frogger pertenece a Konami. Los videojuegos se han implementado con un fin didáctico para que otros usuarios puedan aprender a programar con Scratch. Así mismo todos los recursos empleados durante el desarrollo de los videojuegos ha sido obtenido gratuitamente desde http://www.spriters-resource.com/ y http://www.sounds-resource.com/. En cada apartado de diseño se cita a los autores que han compartido los recursos. Estos recursos han sido tratados de manera que se adapten lo mejor posible a los videojuegos implementados.

CAPÍTULO 2
Los videojuegos clásicos: una mirada atrás en la historia

En este capítulo se cuenta la historia, las mecánicas jugables y las repercusiones de los videojuegos desarrollados en Scratch. Para ello se hecha un mirada atrás en el tiempo para conocer los videojuegos Frogger, Duck Hunt y Super Mario Bros. Tres de los videojuegos más influyentes en la historia de los videojuegos que todos los aficionados al mundo de los videojuegos deben de jugar alguna vez en la vida.

2.1 Frogger (1981)

Frogger es un videojuego de plataformas lanzado por primera vez en 1981 en salones recreativos. En la figura 2.1 se puede observar la máquina recreativa. El videojuego fue desarrollado por la famosa compañía Konami, muy conocida actualmente en el mundos de los videojuegos, y publicado por SEGA/Gremlin. Más tarde se desarrollarían versiones domésticas para distintas videoconsolas y ordenadores. Actualmente existen innumerables versiones no oficiales circulando por la web.

2.1.1. Historia

A principio de los años 80 se iniciaba una revolución que cambiaría el mundo de los videojuegos. Las videoconsolas comenzaban a tener importancia y los videojuegos cada vez eran de más calidad. Los videojuegos ya no eran mayoritariamente juegos de naves con gráficos simples. Aparecían videojuegos innovadores que aportaban frescura entre tantos juegos de disparos. Frogger junto a Pac Man y otras recreativas fueron los que impulsaron dicha revolución conocida como la edad de oro en los videojuegos.

Konami fue la empresa que se encargó del desarrollo de Frogger. A día de hoy, la empresa nipona es muy conocida dentro del mundo de los videojuegos. Desarrolladores de sagas importantes en la historia de los videojuegos como Me-

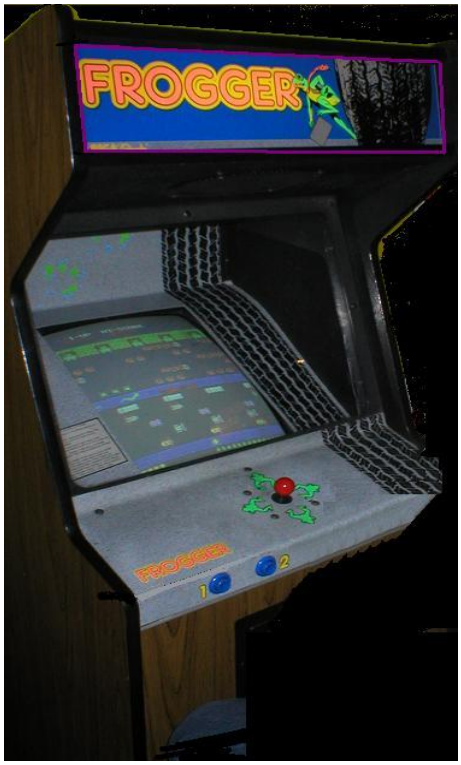

Figura 2.1: Máquina recreativa con el videojuego Frogger lanzada en 1981.

tal Gear Solid, Castlevania o Silent Hill. Pero no siempre la empresa Konami se dedicó a desarrollar videojuegos domésticos. En sus inicios, Konami se dedicaba a desarrollar máquinas de juego lúdico para diferentes casinos en Japón y máquinas arcade para salones recreativos. Una de estas máquinas arcade recibió el nombre de Frogger. Frogger fue lanzado durante 1981 en Japón. Debido a su enorme éxito pronto se expandió al territorio norteamericano gracias a un acuerdo entre Konami y SEGA/Gremlin. SEGA/Gremlin finalmente fue la encargada de llevar el videojuego a tierras americanas como ya había hecho con otros famosos *arcades*[1] japoneses. El videojuego llegó a tierras americanas en octubre de 1981.

2.1.2. El juego

En el videojuego el jugador toma el control de una rana cuyo objetivo es volver a casa. Para ello el jugador se sitúa en la parte inferior de la pantalla y debe llegar a la otra punta. Por eso debe guiar a la rana con un *stick*[2] para cruzar la carretera y llegar al final del río donde se encuentran las charcas. Pero no es tan sencillo, en la carretera aparecen distintos vehículos que deben ser esquivados. Entre ellos se encuentran camiones, turismos, deportivos y excavadoras. Es necesario que el jugador los esquive o de lo contrario atropellan a la rana y tendrá que volver a empezar. Una vez llegada al río, la rana tiene que cruzarlo saltando en los diferentes obstáculos que aparecen en pantalla como troncos, cocodrilos y

[1] Se conoce como *arcades* a los videojuegos que salieron en máquinas recreativas.
[2] Se conoce como *stick* a la palanca que se presiona para el movimiento del personaje.

tortugas, ya que la rana no sabe nadar y se ahogará si se cae al río. En la figura 2.2 se observa el aspecto que tiene el videojuego.

Figura 2.2: Imagen de la rana posicionada al principio de la pantalla en el videojuego Frogger.

Mientras el jugador intenta llegar al final del nivel. Este tiene un tiempo límite para rescatar a todas las ranas de la ronda. También posee un número de vidas limitado para cada ronda. Cuando el tiempo o las vidas llegan a su fin se acaba el juego. Si todas las ranas se llevan con éxito a sus charcas. Se vuelve a cargar el mismo escenario pero con una dificultad más elevada dando lugar a una nueva ronda. Mientras el usuario mantiene una puntuación que aumenta conforme se completan las rondas y se recogen objetos que aparecen esporádicamente en ciertos momentos de la partida. Una vez finalizada la partida la máquina recreativa almacena la puntuación obtenida para que otras personas puedan superarla.

2.1.3. Repercusión

Tras el éxito que supuso la máquina recreativa, sobre todo en Estados Unidos. Konami decidió seguir desarrollando una secuela del videojuego. En 1984 llegaba Frogger 2. Esta vez el videojuego sería lanzado para videoconsolas y ordenadores personales. El videojuego partía de la base de Frogger, sin embargo, debido a la evolución de la tecnología este contaba con tres pantallas que se sucedían de manera consecutiva en vez de una pantalla por nivel.

Después del lanzamiento de la segunda parte, la saga Frogger fue apartada por Konami hasta finales de los 90. Durante este tiempo muchos fans realizaron sus propias versiones del videojuego original. En 1997, en pleno auge de los gráficos tridimensionales, Konami retomó la marca Frogger adaptando la mecánica

del primer videojuego y llevándola a las tres dimensiones (3D). El videojuego salió para la primera Playstation y en ordenadores personales (PC). Para ver el aspecto de esta versión puede verse la figura 2.3. Si bien el juego no causó la revolución que supuso el primero, tuvo una secuela en el año 2000. El videojuego no se adaptó bien al 3D, además la mecánica jugable ya no resultaba innovadora y no despertaba el interés del público. Aun así, se siguieron lanzando versiones de Frogger, aunque la mayoría fueron para dispositivos móviles.

Figura 2.3: Primera versión de Frogger en 3D salida en 1997 para la primera Playstation.

A día de hoy existen muchas versiones del videojuego circulando por la red. Casi todas son desarrolladas por fans que les gustaría volver a ver a la rana en un nuevo videojuego.

En noviembre de 2014 se estrenó en teléfonos inteligentes (*smartphones*) un videojuego llamado Crossy Road. El videojuego copia la jugabilidad de Frogger y la lleva un paso más. Esta vez no hay que llegar a la otra orilla del río o rescatar ranas; es más, ni siquiera el protagonista es una rana. El jugador debe de ir esquivando los diferentes obstáculos de la carretera y ver cuán lejos puede llegar. En la figura 2.4 se puede observar el aspecto jugable de este videojuego. Los protagonistas son una serie de animales que se van desbloqueando conforme se cumplen objetivos durante la partido. El videojuego fue gratuito y estuvo entre los más descargados tanto en App Store (iOs) como en la Play Store (Android).

2.2 Duck Hunt (1984)

Duck Hunt es un videojuego de disparos lanzado en la videoconsola Nintendo Entertainment System o también conocida como NES en Europa. La videoconsola se puede observar en la figura 2.5. El videojuego fue desarrollado y publicado por Nintendo en 1984 en Japón, un año más tarde aparecería en el mercado

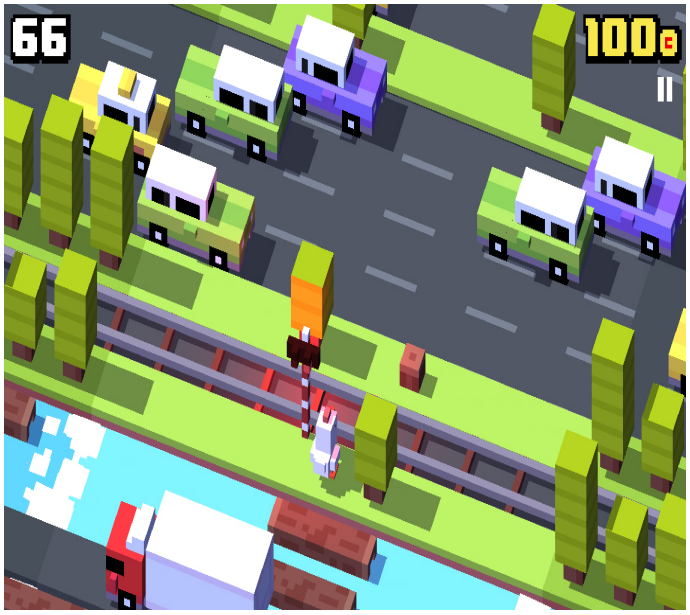

Figura 2.4: Captura del videojuego Crossy Road lanzado para iOs y Android en 2014.

norteamericano y no fue hasta 1987 cuando los europeos pudieron adquirirlo en su territorio. El juego fue producido por el japonés Gumpei Yokoi.

Figura 2.5: Versión de la videoconsola Nintendo Entertainment System (NES) lanzada en Europa, América y Australia.

Junto al juego se lanzó al mercado la Nintendo Zapper la cual se podía adquirir junto al juego o por separado. La Nintendo Zapper es una pistola de luz que permitía disparar a la pantalla de forma que emulara los disparos en el juego. En cuanto a su funcionamiento era bastante sencillo. La pistola se componía de un gatillo y un fotodiodo en la punta del cañón. Al apretar el gatillo, emitía una señal a la videoconsola y la pantalla cambiaba a un fondo negro de duración un *frame* por lo que no era percibido por el jugador. En ese momento, dentro de la pantalla negra el fotodiodo emitía una luz blanca iluminando una parte de la pantalla con un puntero. Este puntero lo emitía la pistola al apuntar el jugador en el televisor. Para más información acerca de la Nintendo Zapper consultar [2].

Como curiosidad, esta pistola sólo funciona en televisiones de refresco de pantalla como son las de tubo de rayos catódicos, no funcionan en televisiones de plasma, LED o LCD. En la figura 2.6 se puede observar el modelo de la Nintendo Zapper en Europa.

Figura 2.6: Caja del periférico Nintendo Zapper para su venta en solitario.

El éxito de este dispositivo animó a otras compañías a desarrollar y lanzar al mercado sus propias pistolas virtuales.

2.2.1. Historia

La idea del juego (cazar patos disparando con un arma de fuego) se remonta 15 años antes de su lanzamiento, cuando la compañía nipona SEGA lanzó una máquina recreativa a la que bautizó como Duck Hunt, el mismo nombre que más tarde utilizaría Nintendo. La idea, cómo no, era cazar patos con una escopeta. La escopeta constaba de 10 cartuchos y por cada pato eliminado la máquina otorgaba una puntuación; la partida finalizaba cuando se agotaban los cartuchos mostrando la puntuación alcanzada. La figura 2.7 muestra el diseño de la máquina recreativa.

A Nintendo le gustó la idea y durante los años 70 desarrolló un sistema llamado Laser Clay System Shooting; detrás de este proyecto ya se encontraba por entonces Gumpei Yokoi (productor de Duck Hunt). El sistema se lanzó en lugares de ocio y consistía en disparar a unos objetivos que aparecían en un proyector enorme. Pero este producto no resultó ser del todo existoso tal y como se esperaba pues Nintendo obtuvo pérdidas de hasta 5 000 millones de yenes.

Años más tarde, convencidos de poder ofrecer un buen sistema de disparos virtual, Nintendo volvió a lanzar al mercado el producto pero esta vez de manera más compacta de forma que se pudiera jugar en los salones recreativos. Con él consiguió remontar las pérdidas obtenidas anteriormente y ya en 1984 con la reciente salida de su propia videoconsola (NES) optaron por adaptar el sistema recreativo a los hogares domésticos para que muchos pudieran disfrutar la experiencia.

Figura 2.7: Máquina recreativa lanzada por SEGA y bautizada como Duck Hunt.

2.2.2. El juego

El objetivo principal del juego es cazar tantos patos como sea posible con nuestra Nintendo Zapper. Para ello el juego se divide en distintas rondas, hasta 99 rondas contiene el juego original, por supuesto en cada ronda que pasa la dificultad se incrementa lentamente. El jugador tiene 3 balas para matar al pato cuando aparece teniendo un tiempo límite para cazarlo o se escapará volando. Cada ronda consta de 10 patos no siendo obligado matar a todos los patos de la ronda para pasar a otra ronda pues se necesita un mínimo de seis patos muertos para pasar de ronda. La puntuación se consigue al matar un pato aunque los hay de distintos tipos, cada tipo de pato proporciona una puntuación debido a su dificultad.

Cada vez que se acierta un disparo aparece un perro agarrando el pato, mientras que si se falla, el perro surge de los matorrales y se ríe burlándose del jugador como se puede observar en la figura 2.8. Esta risa provocó en su época mucha rabia entre los jugadores e intentaban disparle al perro, aunque no era posible.

En cuanto a los modos de jugar, el juego incluye tres modos de juego. Los dos primeros son muy parecidos pues en ambos el objetivo es alcanzar la máxima puntuación cazando patos, la única diferencia es que en el primero los patos salen de uno en uno mientras que el segundo modo los patos salen de dos en dos, incrementando la dificultad. El tercer modo no consiste en disparar a patos, sino a platos; este modo se considera el más difícil porque los platos aparecen mucho más veloces que los patos en la pantalla con lo que se requieren mejores reflejos y rapidez para poder alcanzarlos.

Figura 2.8: Captura del videojuego Duck Hunt cuando el perro se ríe del jugador en el momento que se escapa un pato.

2.2.3. Repercusión

La salida de Duck Hunt al mercado internacional fue todo un éxito para Nintendo, el juego vendió 28,31 millones de copias situando al juego en el puesto número 10 de los videojuegos más vendidos de la historia[3]. La mayoría de las ventas fueron en Estados Unidos donde se vendieron 26,23 millones de copias. Debido a este éxito, otras compañías trataron de llevar al mercado doméstico sus propias pistolas, pero no fue hasta mediados de los 90 con la generación de los 32 bits cuando se logró innovar la mecánica de este tipo de videojuegos sobre todo gracias a la aparición de los polígonos en 3D.

SEGA, que ya había trabajdo en algún proyecto parecido en la generación de los 16 bits, lanza al mercado en 1995 Virtual Cop para su consola de 32 bits (SEGA Saturn). El videojuego consistía en dos policías que trataban de destruir una organización criminal. Para ello el jugador se ponía en la piel de uno de los dos policías disparando a los enemigos que emergían en la pantalla mientras se avanzaba por el escenario; el juego contenía un modo para dos jugadores de forma que pudieran cooperar disparando. Las ventas fueron realmente buenas lo que llevó a SEGA a lanzar otros títulos de este género como es la saga The House of the Dead.

Otra empresa que aprovechó el éxito fue Namco gracias a su máquina recreativa Time Crisis. En 1997 con la Sega Saturn luchando en ventas junto a la Playstation de Sony, Namco lanza al mercado una conversión de uno de sus videojuegos más exitosos em maquinas recreativas, Time Crisis. En la figura 2.9 se contempla una imagen del videojuego. Junto al lanzamiento se creó una nueva y revolu-

[3]Consultado en marzo de 2016 en http://www.vgchartz.com/gamedb/.

cionaria pistola llamada G-CON ambos exclusivos de Playstation. El videojuego tenía un argumento y mecánica parecidos al Virtual Cop lanzado anteriormente por SEGA. Con este videojuego, entre otros, Sony logró ampliar su catálogo de exclusividades y ganar así su batalla contra la consola de SEGA.

Figura 2.9: Imagen del primer nivel del videojuego Time Crisis lanzado por Namco en máquinas recreativas y la primera Playstation.

En 2007 Nintendo lanzó al mercado su consola Wii. La principal innovación de esta videoconsola fue su mando WiiMote, el cual tenía forma alargada y se podía mover por toda la pantalla con el movimiento humano, igual que ocurría con las pistolas. Pero bien, hasta esa fecha la mayoría de pistolas funcionaban como la Nintendo Zapper y tenían el problema de funcionamiento en televisiones que no refrescaran la imagen. Esto ya no era un problema pues el mando de la Wii contaba con un sensor que era el encargado de emular los movimientos en la pantalla. Años más tarde Sony lanzó su propio periférico imitando la funciones del Wiimote, pero fue un fracaso a nivel de ventas. En la figura 2.10 se encuentran ambos periféricos.

2.3 Super Mario Bros (1985)

Super Mario Bros es un videojuego de plataformas lanzado en 1985 para la videoconsola de 8 bits Nintendo Entertainment System (NES) o también conocida en el continente asiático como Famicom. Veáse la figura 2.5 para ver el modelo de la videoconsola. El videojuego fue creado por Shigeru Miyamoto y fue lanzado por la compañía nipona Nintendo. El juego apareció en el mercado nipón el 13 de septiembre de 1985. Un poco mas tarde aparecería en el mercado norteamericano y dos años después a su lanzamiento en Japón aparecería en el mercado europeo. Esto se debió a que la videoconsola de Nintendo no salió en Europa hasta fina-

Figura 2.10: El primer periférico (izquierda) pertence al WiiMote lanzado por Nintendo y el otro (derecha) es el Playstation Move lanzado por Sony.

les de 1986. El videojuego es considerado como el padre de los videojuegos de plataformas ya que sirvió de inspiración para futuros títulos.

Super Mario Bros se llegó a comercializar junto con el videojuego de disparos Duck Hunt (tratado en la sección 2.2) en un mismo cartucho. Véase la figura 2.11.

2.3.1. Historia

La historia de Super Mario Bros se remonta a la salida del primer Donkey Kong. Donkey Kong fue el primer juego diseñado por Shigeru Miyamoto en 1981. El juego tenía a un Mario que por aquel entonces se le conocía como Jumper Man. En el videojuego, Mario tenía que rescatar a una princesa de las manos del malvado Donkey Kong que se encontraba en la cima de la pantalla. Mario tenía que abrirse paso esquivando y destruyendo barriles que el gorila Kong lanzaba al jugador. Los barriles se podían destruir con un martillo que el jugador cogía. Mientras, el jugador iba ascendiendo por las plataformas en distintos niveles hasta llegar a rescatar a la princesa y derrotar al temible Donkey Kong. El videojuego, que fue todo un éxito, fue lanzado en máquinas recreativas de todo el mundo y aún a día de hoy existen jugadores que compiten por obtener un nuevo récord mundial.

Tras el éxito de Donkey Kong en salones recreativos, Shigeru Miyamoto siguió diseñando nuevos videojuegos. Su siguiente videojuego lo llamó Mario Bros y

Figura 2.11: Fotografía del cartucho que contiene Super Mario Bros y Duck Hunt.

fue lanzado en 1983 en máquinas recreativas, en las consolas 2600 y 5200 de Atari y en la NES de Nintendo. En la figura 2.12 se encuentra la videoconsola Atari 2600. El videojuego contaba con el mismo protagonista que se hizo famoso en su videojuego anterior Donkey Kong. Pero esta vez ya no se llamaba Jumper Man sino que fue rebautizado como Mario. Es entonces cuando aparece por primera vez el personaje tal y como se le conoce a día de hoy.

El juego era distinto a Donkey Kong. Esta vez el protagonismo no recaía sobre el gorila Kong. Es más, no llegaba a aparecer en el videojuego. El protagonismo era cedido a Mario, quien junto a su hermano Luigi pertenecían al cuerpo de los fontaneros y tenían que limpiar las cañerías de distintos enemigos mientras avanzaban de nivel (ambos se pueden observar juntos en la figuras 2.13 y 2.15). La incorporación de Luigi permitía a otro jugador unirse a la partida para cooperar o pelear con el jugador principal. Este fue el nacimiento del personaje Luigi, personaje que más tarde acompañaría a Mario en todas sus aventuras.

Mario Bros no obtuvo malas críticas. Pero Nintendo esperaba tener muchas más ventas de las que tuvo. Esto replanteó a la compañía trabajar y cambiar las mecánicas jugables de Mario Bros para su siguiente videojuego que sería Super Mario Bros.

2.3.2. El juego

La princesa Toadstool del reino Champiñón (que más tarde se le conocería como Peach, se encuentra a la derecha de Mario en la figura 2.15) ha sido secuestrada por el rey Koopa, o también conocido como Bowser (situado encima de Mario

Figura 2.12: Fotografía de la videoconsola Atari 2600 para la que salió el videojuego Mario Bros.

y Luigi en la figura 2.15). Mario y su hermano Luigi tendrán que enfrentarse a las fuerzas enemigas lideradas por el rey Bowser hasta encontrar a la princesa y rescatarla de sus manos. Con este simple argumento da lugar la historia principal del videojuego. Como se puede observar, el argumento del videojuego posee cierta similitud con el primer Donkey Kong. Esto se debe a que su creador Shigeru Miyamoto es un fan de los cuentos clásicos en los que el caballero debía de rescatar a la princesa.

Super Mario Bros consta de 32 niveles repartidos en 8 mundos. El diseño de los niveles es variado, desde un alegre bosque hasta un tenebroso castillo, incluyendo algún nivel acuático; en donde el jugador se sumergía en las profundidades del agua. Al final de cada mundo el jugador debía derrotar al rey Koopa en una fortaleza. En ese momento rescatábamos a unos personajes con gorro en forma de seta. Estos personajes formaban parte de la corte del reino Champiñón. Una vez rescatados decían que la princesa no se encontraba en ese castillo. Por lo tanto Mario debía buscar en otro castillo, hasta dar con la princesa.

En cuanto a la jugabilidad, el videojuego destaca por su sencillo control. Con el *pad*[4] direccional se mueve al personaje y manteniendo un botón el personaje puede correr. Además el videojuego responde bien a las acciones del *pad*. Consiguiendo un control fluido sobre Mario, algo muy importante para los videojuegos dentro del género de plataformas.

Una vez en la partida, el jugador tiene que avanzar por la derecha hasta llegar al final del nivel y tocar una bandera (ver figura 2.14). Por el camino, Mario se encuentra con diversos ítems y monedas. Si se consiguen 100 monedas el juego te devuelve una vida (al empezar el juego se dispone de 3 vidas). En cuanto

[4]Se conoce como *pad* a los mandos de las videoconsolas.

Figura 2.13: Mario y Luigi cooperando para limpiar las cañerías de enemigos en Mario Bros.

a los ítems, hay una gran diversidad de ellos, como las setas que hacen crecer al personaje o las estrellas que otorgan la inmunerabilidad. También había flores; éstas transforman a Mario y le permiten disparar bolas de fuego. Al mismo tiempo, Mario va abatiendo distintos enemigos, como los Gombas (pequeñas setas marrones), los Koopa Tropas (tortugas similares al rey Koopa) o las plantas carnívoras que emergían de las tuberías. Uno de los principales problemas del videojuego es no poder volver atrás en el nivel pues el nivel no avanzaba por la izquierda. Esto dificultaba recoger ítems dejados atrás en la pantalla.

El videojuego, al igual que su predecesor Mario Bros, permite jugar a dos jugadores simultáneamente. El jugador uno controla a Mario, mientras que el otro jugador controla a su hermano Luigi. Juntos deberán cooperar para rescatar a la princesa de las manos del temible rey Koopa.

2.3.3. Repercusión

Super Mario Bros es el videojuego que catapultó a la fama a Nintendo. Super Mario Bros se encuentra en el segundo puesto de los videojuegos más vendidos de la historia[5]. El videojuego vendió un total de 40,24 millones de copias en todo el mundo. Tanto el videojuego como su creador Miyamoto, han sido reconocidos por otras empresas de gran prestigio. A su creador se le conoce como uno de los

[5]Consultado en marzo de 2016 en http://www.vgchartz.com/gamedb/.

Figura 2.14: Mario tocando la bandera que da por terminando el primer nivel de Super Mario Bros.

gurús de la industria del videojuego. Tras el éxito de Super Mario Bros muchas compañías adaptaron las innovaciones jugables que ofrecía el videojuego, lo que supuso que pronto aparecieran en el mercado juegos similares a Super Mario Bros.

El videojuego contó con dos secuelas directas. Super Mario Bros 2 y Super Mario Bros 3 aparecieron unos años más tarde en la misma plataforma, la NES. Ambos videojuegos mantenían el esquema y trama argumental del primero. Pero se añadieron nuevas mecánicas jugables. Pese a estas semejanzas con su antecesor, ambos videojuegos fueron exitosos a nivel de ventas. Nintendo había creado un mundo al que explotar y pronto Mario se convertiría en el icono de la compañía.

A lo largo de los años y generaciones han ido apareciendo nuevos videojuegos y con ello nuevos personajes relacionados con la saga. Por ejemplo se añadió a un personaje llamado Yoshi. Yoshi era una especie de dragón feliz que servía de montura para Super Mario (ver esquina inferior derecha de la figura 2.15). Este personaje pronto se convirtió en uno indispensable y de los más queridos por los aficionados. Muchos de estos personajes han protagonizado su propio videojuego. Incluso Nintendo ha lanzado videojuegos de otros géneros en donde reunía a los personajes secundarios de la franquicia (Mario Kart, Mario Tennis y Mario Golf entre otros videojuegos).

En 1991, en plena guerra con Nintendo, SEGA creaba Sonic the Hedgehod para su consola de 16 bits, la Mega Drive. Para conocer más información acerca de

Figura 2.15: Personajes más famosos del universo Mario creado por Shigeru Miyamoto. De arriba hacia abajo empezando por la izquierda se encuentran: Shy Guy, Wario, Toad, Goompa, Peach, Bowser, Mario, Luigi, Koopa Troopa, Daisy, Toadette, Planta Piraña, Waluigi y Yoshi

la guerra entre Nintendo y SEGA se puede consultar [5]. El objetivo de la compañía era convertir a su protagonista en el icono de la empresa al igual que hizo Nintendo con Super Mario. El juego creaba así su propio universo con muchos personajes que más tarde compartirían protagonismo con Sonic igual que los personajes de Super Mario. El videojuego pertenecía al mundo de las plataformas, aunque aumentaba la velocidad del videojuego pues la principal habilidad que poseía Sonic era la de correr a la velocidad del sonido.

Más tarde se sumaría Sony a la batalla con su Playstation. En 1996, Naughty Dog desarrolló un videojuego protagonizado por Crash Bandicoot el cual daba nombre al juego. El objetivo era crear un videojuego de plataformas con un protagonista que sirviera de mascota para Sony [7]. Tal y como Nintendo y SEGA habían hecho. A diferencia de Super Mario y Sonic, Crash Bandicoot hacía uso de las tres dimensiones. Todo el videojuego se desarrollaba en entornos poligonales, esto fue gracias al salto generacional de 16 a 32 bits. Aunque oficialmente nunca llegó a ser la mascota de Sony, si fue uno de los mejores videojuegos de la generación. En la figura 2.16 aparecen ambos personajes.

A día de hoy siguen apareciendo nuevos videojuegos relacionados con el universo Mario. También aparecen videojuegos del universo Sonic. Pero estos tienen una calidad inferior a los desarrollados durante los años 80 y 90. Mientras que los videojuegos de Mario siguen poseyendo la misma calidad o incluso mayor que sus predecesores pues saben aprovechar los avances tecnológicos. En cuanto a Crash Bandicoot, a inicios del siglo XXI, Sony vendió la licencia. Actualmente se encuentra desaparecida, aunque corren rumores sobre una nueva adquisición de la licencia por parte de Sony.

Figura 2.16: A la izquierda de color azul Sonic creado por SEGA; a la derecha Crash Bandicoot, personaje creado por Sony.

CAPÍTULO 3
El entorno de programación Scratch

En este capítulo se va a describir el entorno de programación Scratch, desde el porqué se ha optado por él para desarrollar videojuegos clásicos hasta su historia. También se explica cómo estan organizados los diferentes paneles dentro del programa Scratch y la metodología de desarrollo llevada a cabo en cada uno de los tres videojuegos que componen este trabajo.

3.1 ¿Por qué utilizar Scratch?

Para realizar este proyecto se ha optado por utilizar Scratch como entorno de desarrollo de videojuegos (figura 3.1). Las razones por las que se ha escogido Scratch y no otros entornos de programación ha sido:

- Facilidad de uso: La razón principal por la que se ha escogido este entorno. Scratch permite programar a todo tipo de usuarios. Ya sean expertos programadores o iniciados en el mundo de la programación. Para ello proporciona un sistema basado en el arrastre de bloques que pueden conectarse entre si para añadir funcionalidad a nuestro programa. Esto evita que el usuario necesite conocimientos en otros lenguajes de programación.
- Entorno muy desarrollado: Scratch posee innumerables libros y tutoriales sobre su uso. Además cuenta con su propia página web en donde el usuario puede mantenerse actualizado. Actualmente Scratch se encuentra en su versión 2.0.
- Cuenta con editor online: No es necesario instalar un cliente Scratch en el ordenador para empezar a programar. La página web de Scratch cuenta con su propio editor para programar online desde cualquier lugar.
- Código compartido: Los programas realizados se pueden almacenar en la nube que proporciona Scratch. El usuario puede habilitar la visualización de su código para que otros usuarios puedan aprender nuevas técnicas de

Figura 3.1: El gato que hace de logo en Scratch
y el lema de este que dice «Siempre imagina, programa y comparte».

programar. Incluso otro usuario puede crear una nueva versión del programa. Gracias a esto se favorece tanto a su creador como a los usuarios que desean iniciarse en el mundo de la programación. De ahí viene el lema «Siempre imagina, programa y comparte» (figura 3.1).

- Es gratuita: Scratch es una herramienta totalmente gratuita para todos los usuarios y está disponible en los muchos idiomas entre los que se encuentran español, inglés, francés y alemán.

3.2 El proyecto Scratch

Scratch es una plataforma multimedia de programación desarrollado por el MIT (Massachusetts Institute of Technology). La plataforma cuenta con versión de escritorio y online (si se desea conocer más acerca de la plataforma y como iniciarse con el lenguaje en su versión 2.0, se puede consultar [9]). Esta se centra en un público más novato en programación pues les permite aprender a crear videojuegos de una manera muy sencilla. Mientras desarrollan sus conocimientos sobre la programación, su pensamiento creativo, su razonamiento y el trabajo en equipo. Todo el contenido de la plataforma se puede acceder a través de la página su página oficial que es `https://scratch.mit.edu/`.

El proyecto Scratch nació durante el año 2003 con el grupo de investigación Lifelong Kindergarten del MIT [15], liderado por el norteamericano Mitchel Resnick (figura 3.2). La idea surgió a partir de los pensamientos de los integrantes del grupo. Estos pensaban en la importancia que tenía para los niños crecer aprendiendo a diseñar y expresarse, pues no todos disponían de las mismas oportunidades. El grupo buscaba desarrollar una plataforma para que todos los niños pudieran desarrollar sus capacidades intelectuales. Por ello se basaron en las ideas de Seymour Papert y su concepto de programación en bloques de Lego. Modo en el que se basa la programación con Scratch.

Figura 3.2: Mitchel Resnick, líder del grupo Lifelong Kindergarten y creador de la herramienta Scratch.

El término scratch viene de una técnica utilizada por los *disk jockeys* (DJ) llamada *scratching*. El *scratching* consiste en realizar modificaciones en las canciones a base de reproducir discos de vinilo. El DJ mueve el vinilo con las manos hacia delante y hacia atrás, produciendo diferentes sonidos en la canción de una manera interactiva. Con Scratch pasa algo parecido, pues mezcla lo gráfico con animaciones, imágenes, sonidos, etc. Todo mezclado de una forma interactiva. Se explica de forma detallada en el artículo [14].

Scratch es un proyecto de desarrollo cerrado y de código abierto. ¿Qué significa esto? Significa que el equipo detrás de Scratch elabora enteramente el proyecto estándar para luego liberar su código. De esta forma la comunidad puede experimentar con el desarrollo de nuevas extensiones y modificaciones del programa. En marzo del año 2007 la página web de Scratch, coincidiendo con su nueva versión 2.0, fue completamente rediseñada para introducir el componente social en la plataforma. A partir de entonces los usuarios pueden compartir y visualizar los proyectos de todos los usuarios de la comunidad. Conocidos por el nombre de *scratchers*. En la figura 3.3 se pueden observar comentarios de los usuarios sobre un proyecto en Scratch alojado en la web.

3.3 El pensamiento computacional y Scratch

El pensamiento computacional o también conocido como *computational thinking* es un concepto definido por Jannette M. Wing. Esta autora lo define como «Procesos de pensamiento involucrados en formular problemas y encontrar sus soluciones de manera que las soluciones estén representadas de forma tal que puedan llevarse a cabo por un agente que procesa información (humano o máquina)». Para más información sobre el pensamiento computacional se puede consultar el artículo [19].

Scratch trata de simular dicho pensamiento estableciendo conceptos como secuencia, paralelismo, iteración, datos, operadores, eventos y condicionales. No

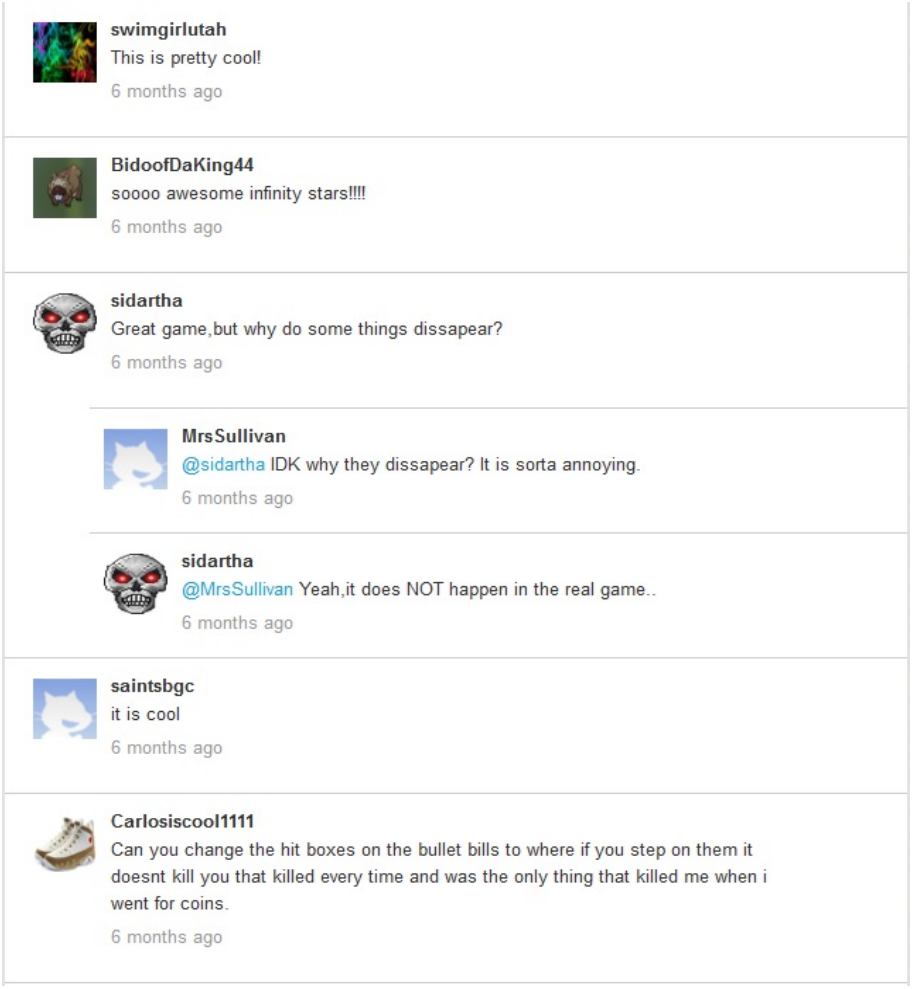

Figura 3.3: Ejemplo de los comentarios de los *scratchers* en un proyecto alojado en la web de Scratch.

sólo establece conceptos, sino también pretende adaptar ciertas prácticas del pensamiento computacional como son:

- En la búsqueda de soluciones ser incremental: divide y vencerás.
- Probar y depurar: no todo sale a la primera, es necesario probar y corregir errores.
- Rehusar y remezclar: no siempre hay que partir desde cero, se pueden reutilizar partes.
- Abstraer, modular y modelizar: se crean modelos para gestionar la complejidad.

Estas prácticas están enfocadas al proceso de pensar y aprender, centrándose mas en cómo lo estás aprendiendo, que qué estás aprendiendo.

Resumiendo, se podría decir que el pensamiento computacional es una habilidad para resolver diferentes problemas de manera algorítmica que, además,

ayuda a mejorar la eficiencia de los procesos. El concepto fue usado por primera vez por Seymour Papert en 1996, y no solo se aplica a la informática, sino a todas las disciplinas existentes.

Scratch ayuda al aprendizaje por medio del pensamiento computacional pues está enfocado a actividades de diseño. Para ayudar a resolver dichas actividades, se ofrecen un amplio abanico de herramientas.

3.4 Programación de videojuegos con Scratch

En esta sección se va a explicar cómo están organizados los diferentes paneles dentro del programa Scratch. Dentro de cada panel se detallarán las principales funciones y su utilidad.

3.4.1. Panel de objetos

El panel de objetos o *sprites* se puede considerar el principal panel dentro de la plataforma Scratch. A diferencia de otros lenguajes de programación, Scratch quiere que el usuario tenga todos los objetos organizados de forma que el código correspondiente a cada objeto se encuentre dentro de él, facilitando así la interacción con ellos y permitiendo que a usuarios con escasos conocimientos de programación les sea más fácil el diseño de videojuegos. Además en cada objeto aparece una imagen de las que se compone dicho objeto.

Los objetos se pueden crear de varias formas. La primera de ellas es a partir de las librerías que posee Scratch. En ellas se encuentran *sprites*[1] sobre distintas temáticas que pueden servir al usuario en caso de no encontrar otros *sprites* por la web. La segunda forma permite al usuario partir de cero con el objeto. Para ello se facilitan herramientas para dibujar su propio objeto. La tercera, es la que se usa para este trabajo. Se trata de cargar los *sprites* a partir de imágenes almacenadas localmente en el ordenador. Las imágenes pueden ser descargadas de distintos sitios web e importarlas dentro del proyecto. Finalmente, la última opción accede a la cámara del ordenador para capturar instantáneas y poder importarlas como *sprites*. En la esquina superior derecha de la figura 3.4 se pueden observar las cuatro opciones.

Además, cada objeto posee cierta información que puede ser visualizada seleccionando la "i" que se encuentra en la esquina superior izquierda de cada objeto una vez seleccionado. En la información se puede cambiar parámetros como el nombre del objeto, la rotación, si puede ser arrastrado por el jugador y si se desea que sea visible. En la figura 3.4 se observa el icono "i" en el objeto *frog* y en la figura 3.5 se encuentra la información detallada.

Por último a la izquierda del panel se encuentra el objeto que define el escenario del videojuego o *stage*. Se encuentra a la izquierda en la imagen 3.4. Para

[1]Son las imágenes que poseen transparencia y son utilizados como objetos en un videojuego.

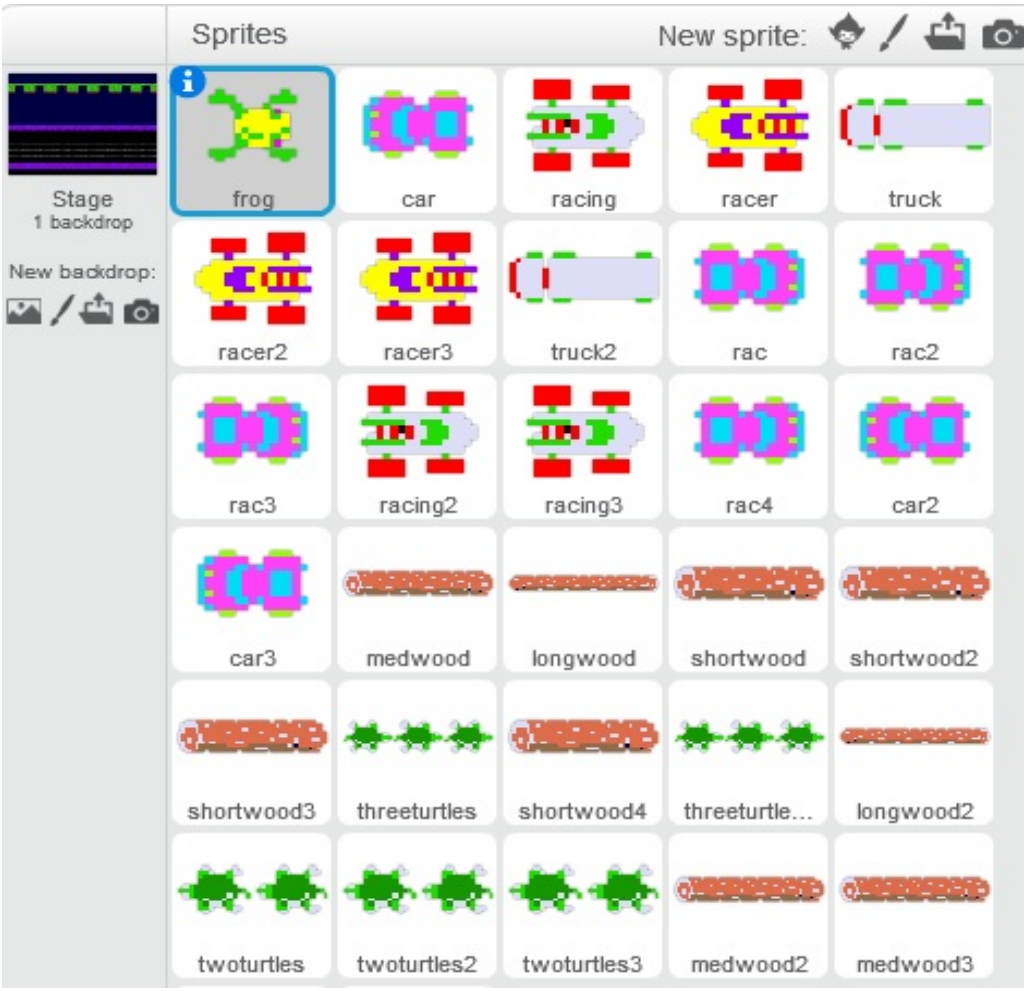

Figura 3.4: Captura de la interfaz del panel de objetos en Scratch.

importar un escenario, se incluyen las cuatro opciones que disponen los objetos comentadas anteriormente. Normalmente este objeto es el encargado de controlar el estado del videojuego indicando cuándo empieza y finaliza el videojuego. También controla las transiciones entre los distintos escenarios del videojuego.

3.4.2. Panel de programas

Cada objeto y escenario dispone de su propio panel de programas o *scripts*. En este panel, como su nombre indica, es donde el programador añade las funciones que cree conveniente a cada objeto con el fin de añadir un comportamiento a cada uno. A diferencia de otros lenguajes de programación, en Scratch no se necesita escribir líneas de código. Tan solo se deben de arrastrar los bloques deseados al panel de programas y juntarlos como si fuesen piezas de LEGO. Una vez construidos ya se habrá añadido una funcionalidad al objeto en cuestión. Así de sencillo es programar en Scratch.

Como se observa en la figura 3.6, hay diferentes tipos de bloques. Cada bloque pertenece a un tipo de bloque. Los tipos de bloques se caracterizan en que cada

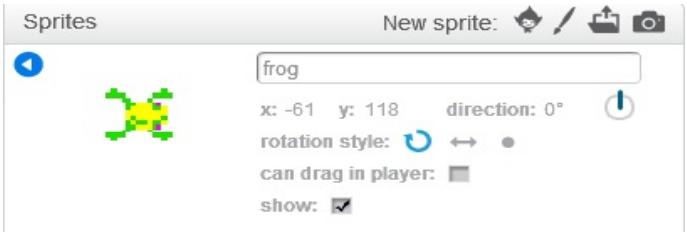

Figura 3.5: Opción del panel de objetos que permite visualizar la información básica de un objeto.

uno de ellos es de un color distinto a los demás. Esto permite al usuario familiarizarse con los colores y facilitar la programación. Entre los tipos de bloques se encuentran:

- **Bloques de movimiento (*motion*)**: Este tipo de bloques se encarga de gestionar los movimientos del objeto. Para ello ofrece bloques que permiten posicionar al objeto en la pantalla, moverlo por la pantalla e incluso contiene bloques para obtener la posición x o y del objeto, entre otros bloques. Estos se caracterizan por un color azul marino.

- **Bloques de apariencia (*looks*)**: Los bloques de apariencia gestionan los diferentes disfraces que posee el objeto. Se pueden utilizar para realizar animaciones a los objetos pues una animación no es más que un conjunto de imágenes. El color característico de estos es un morado oscuro.

- **Bloques de sonido (*sound*)**: Estos bloques controlan los distintos sonidos que posee cada objeto, pudiendo reproducir, pausar o cambiar el sonido entre otras opciones. Los bloques que controlan el sonido son de color morado claro.

- **Bloques de lápiz (*pen*)**: Quizás este tipo de bloques sean los más curiosos de todos. Los bloques de tipo lápiz permiten dibujar en la pantalla durante el tiempo de ejecución partiendo del objeto en donde se han implementado. Para ello existen bloques para subir y bajar el lápiz, cambiar el color de la línea o también el grosor. Son de color verde oscuro.

- **Bloques de datos (*data*)**: Estos bloques son los correspondientes a las variables declaradas para los objetos. Gracias a estos bloques se pueden consultar la variables e incluso cambiar su valor durante la ejecución del videojuego. Se caracterizan por tener un color naranja.

- **Bloques de eventos (*events*)**: Los bloques de eventos son los encargados de gestionar todos los eventos que ocurren durante el videojuego. Es decir, se encargan de enviar mensajes a otros objetos o recibir mensajes de objetos. Se suelen utilizar para inicializar las funciones dentro de cada objeto; pues se indica que hasta que no se reciba tal mensaje no se ejecuten las funciones. Los bloques un poco más oscuros que los de datos.

- **Bloques de control**: Este tipo de bloques es el más utilizado en la programación de videojuegos con Scratch. Dentro se encuentran todos los bloques relacionados con el control del flujo de tiempo. Hay bloques para indicar condicionales, bucles infinitos y con condición o detener la ejecución del programa. Son importantes para la programación de videojuegos pues estos están en constante ejecución y hay que controlar los distintos eventos que suceden a lo largo de la ejecución. Su color característico es el amarillo.

- **Bloques de sensores (*sensing*)**: Los sensores permiten detectar ciertas situaciones durante el tiempo de ejecución del videojuego. Scratch aporta varios bloques que sirven para detectar las colisiones entre los objetos. Lo cual facilita la programación de un videojuego pues uno de los aspectos más importantes de un videojuego son las colisiones. Es decir, para que un personaje se mueva por un plataforma, debe de programarse una colisión entre el personaje y la plataforma. Algo que en un lenguaje de programación estándar requiere ciertos conocimientos. Estos bloques se caracterizan por tener un color azul

- **Bloques de operadores (*operators*)**: Dentro de este tipo de bloques se encuentran todos los operadores algebraicos. Como son suma, resta, mayor que, menor que, igual que, multiplicación, entre otros. Los operadores suelen utilizarse con los bloques de control para indicar las distintas condiciones. Son de color verde claro.

- **Más bloques (*more blocks*)**: Esta opción permite al usuario más avanzado crear su propio tipo de bloque agrupando distintos tipos de bloques. Gracias a esto, se pueden ampliar las posibilidades que ofrece Scratch en su defecto añadiendo nuevos tipos de bloques.

3.4.3. Panel de disfraces

Al igual que con el panel de programas; cada objeto posee su propio panel de disfraces o *costumes*. El panel de disfraces se encarga de gestionar las distintas imagenes que un objeto puede poseer. Para importar los disfraces, el panel ofrece las mismas cuatro opciones que el panel de objetos visto en el apartado 3.4.1. En la figura 3.7 se puede observar la interfaz de dicho panel.

Con los distintos disfraces que posea un objeto se pueden crear animaciones para dicho objeto. Para ello se utilizan bloques de tipo apariencia que permiten el cambio de un disfraz a otro. Por ejemplo, en el videojuego Frogger se desea que al apretar la flecha para arriba la rana se mueva simulando la animación de un salto. Para conseguir esto, se utilizará un condicional que controle la pulsación de la flecha. Una vez pulsada la rana cambiará entre los otros dos disfraces. Para hacerlo de una forma automática se añade un temporizador entre el cambio de cada disfraz volviendo finalmente a la imagen inicial. Todo esto anidado a un bucle. Esta es una forma sencilla de conseguir la animación de salto para la rana, algo que en los lenguajes de programación tradicionales es más complejo.

3.4.4. Panel de sonidos

El último panel disponible en Scratch es el de sonidos o *sounds*. Como el panel de disfraces y programas, todos los objetos poseen su propio panel de sonidos. El panel de sonidos sirve para controlar los distintos sonidos que cada objeto reproducirá durante el tiempo de ejecución del videojuego dadas unas circunstancias. El panel admite diversos formatos de audio como *wav*[2] o *mp3*[3]. Además proporciona herramientas de edición que permiten recortar el audio o añadir efectos. Los sonidos pueden ser importados desde el disco local o de las librerías de Scratch. También permite la opción de grabar el propio audio si se posee un micrófono. En la figura 3.8 se puede observar la interfaz de dicho panel.

Los distintos sonidos de los objetos se pueden controlar con los bloques de sonido. Así se puede seleccionar que audio debe sonar en determinado momento en la ejecución del videojuego.

3.5 Metodología en el desarrollo de videojuegos

En esta sección se van a detallar los pasos realizados durante el desarrollo de uno de los videojuegos pertenecientes a este trabajo. Los pasos se han repetido en el desarrollo de cada uno de los videojuegos que componen el trabajo. Desde el porqué se han escogido estos videojuegos en concreto hasta la corrección de fallos una vez implementados con Scratch.

El primer paso ha sido seleccionar qué videojuegos iba a implementar en este trabajo. La selección no ha sido sencilla, he intentado buscar videojuegos lanzados en los años 80 que fueron revolucionarios para su época. Además pretendía que los videojuegos no pertenecieran al mismo género algo que estrechado más la búsqueda. Al final se han elegido los videojuegos Frogger, Duck Hunt y Super Mario Bros.

Frogger lo he elegido porque de pequeño jugaba a su versión en 3D lanzada en la primera Playstation y me parece una lástima que haya caído en el olvído en estos últimos años. Duck Hunt lo he seleccionado porque me encantan los juegos de disparar con una pistola. Además es el promotor de este tipo de videojuegos en videoconsolas domesticas. Y por último, Super Mario Bros lo considero el videojuego más importante y revolucionario de la historia. Gracias a sus mecánicas jugables supuso toda una revolución en la industria del videojuego.

Ya con los videojuegos a implementar escogidos paso al siguiente paso. El segundo paso ha sido buscar por internet las ROM[4] de todos los videojuegos que iba a diseñar en este trabajo. Una vez obtenidas las ROM, he descargado distintos emuladores que me permitan montar las imágenes obtenidas. Una vez

[2]Se conoce como wav a un formato de audio sin compresión de datos.

[3]El mp3 es un formato de audio cuyos datos se encuentran comprimidos.

[4]Se conocen por ROM a las imágenes de los videojuegos que originalmente salieron en cartuchos. Actualmente se aplica a todos los videojuegos que no son de PC.

montadas las imágenes, he jugado a cada uno de ellos de manera individual. Para obtenier conocimientos sobre cómo eran en su época los videojuegos y cómo podría adaptarlo con Scratch.

Seguido, ya con las ideas claras sobre las mecánicas jugables de los videojuegos, he buscado por internet material relacionado con el videojuego. Todo el material gráfico utilizado lo he sacado de http://www.spriters-resource.com/ y los audios de http://www.sounds-resource.com/. Ambas son páginas en donde los usuarios suben contenido de los videojuegos directamente extraídos de las ROM. Todo el material de ambas webs es gratuito y para un uso sin ánimo de lucro. El material gráfico suele venir todo sobre una misma imagen, por lo que hay que recortar la imagen y en algunos casos añadir transparencia al fondo. La transparencia es fundamental en los *sprites* ya que si se representan como objetos deben tener el contorno transparente.

Para realizar esta tarea, se ha utilizado un programa totalmente gratuito llamado GIMP. El primer paso que se ha realizado ha sido abrir la imagen con el programa. Ya con la imagen cargada, he ido a la opción capa/trasparencia/añadir canal alfa. Veáse la figura 3.9 El canal alfa permite añadir trasparencia en el fondo de la imagen. Después, me ha tocado eliminar el fondo actual para que se muestre el fondo transparente añadido anteriormente. Para eso he utilizado la herramienta de selección por colores. Esta herramienta selecciona el color que se desea por toda la imagen. Una vez seleccionado he eliminado todas las partes seleccionadas quedando así un fondo transparente. En la figura 3.10 se puede observar el resultado. Ahora tan solo he recortado las imágenes una a una para después poder importarlas al proyecto en Scratch.

Documentado y con el material ya preparado, toca pensar en cómo diseñar el videojuego. Cada juego tiene sus propias peculiaridades que hay que tener en cuenta a la hora de programar. Para hacerse una idea global sobre los distintos estados por los que pasan los principales objetos del videojuego he realizado una máquina de estados para entender mejor el flujo de cada uno de ellos. Cada máquina se mostrará y explicará más adelante en su correspondiente capítulo.

En cuanto a las peculiaridades de los juegos, por ejemplo en el videojuego Frogger, la rana debe llegar al otro extremo de la pantalla esquivando diferentes obstáculos. Si uno de estos obstáculos alcanza a la rana, esta pierde una vida y vuelve al principio de la partida. En el caso de Duck Hunt, el jugador debe disparar a unos patos. Estos patos se moverán de forma aleatoria por la pantalla. En el momento que uno caiga, el perro deberá ir a por él. Por lo tanto, hay que controlar la posición en la que se le dispara al pato para pasársela al perro. Además se deben contar el número de patos que se matan para pasar al siguiente nivel. En Super Mario Bros, hay que conseguir que el protagonista Mario se desplace por la pantalla horizontalmente. Logrando así una sensación de velocidad. También hay que controlar las colisiones entre los distintos objetos que aparecen en pantalla.

Ya con los materiales y una idea sobre cómo programar el videojuego, paso a la programación con Scratch, primero creando los objetos que creo necesarios,

a estos objetos se les añaden los disfraces y sonidos que se creen convenientes. Después se intenta plasmar las ideas obtenidas en el paso anterior a código en Scratch. La forma de programar seguida es la de ensayo y error. Gracias a un clic, Scratch nos permite probar nuestro videojuego aunque no haya sido terminado. Así se puede observar cómo va quedando el videojuego y si van apareciendo errores, estos poderlos corregir.

Una vez terminado el videojuego, toca diseñar el menú principal del videojuego para después poder probarlo entero. Durante esta prueba se busca la aparición de ciertos *bugs*[5] que deberán ser corregidos antes de la publicación del videojuego en la web oficial de Scratch. Para la detección de *bugs* he pasado los videojuegos a cuatro amigos, a los que les he dicho que busquen errores e intenten pasarse los videojuegos para tener así una visión de alguien que desconoce cómo se han desarrollado.

[5]Errores o fallos en un programa de computador o sistema de software que desencadena un resultado no deseado.

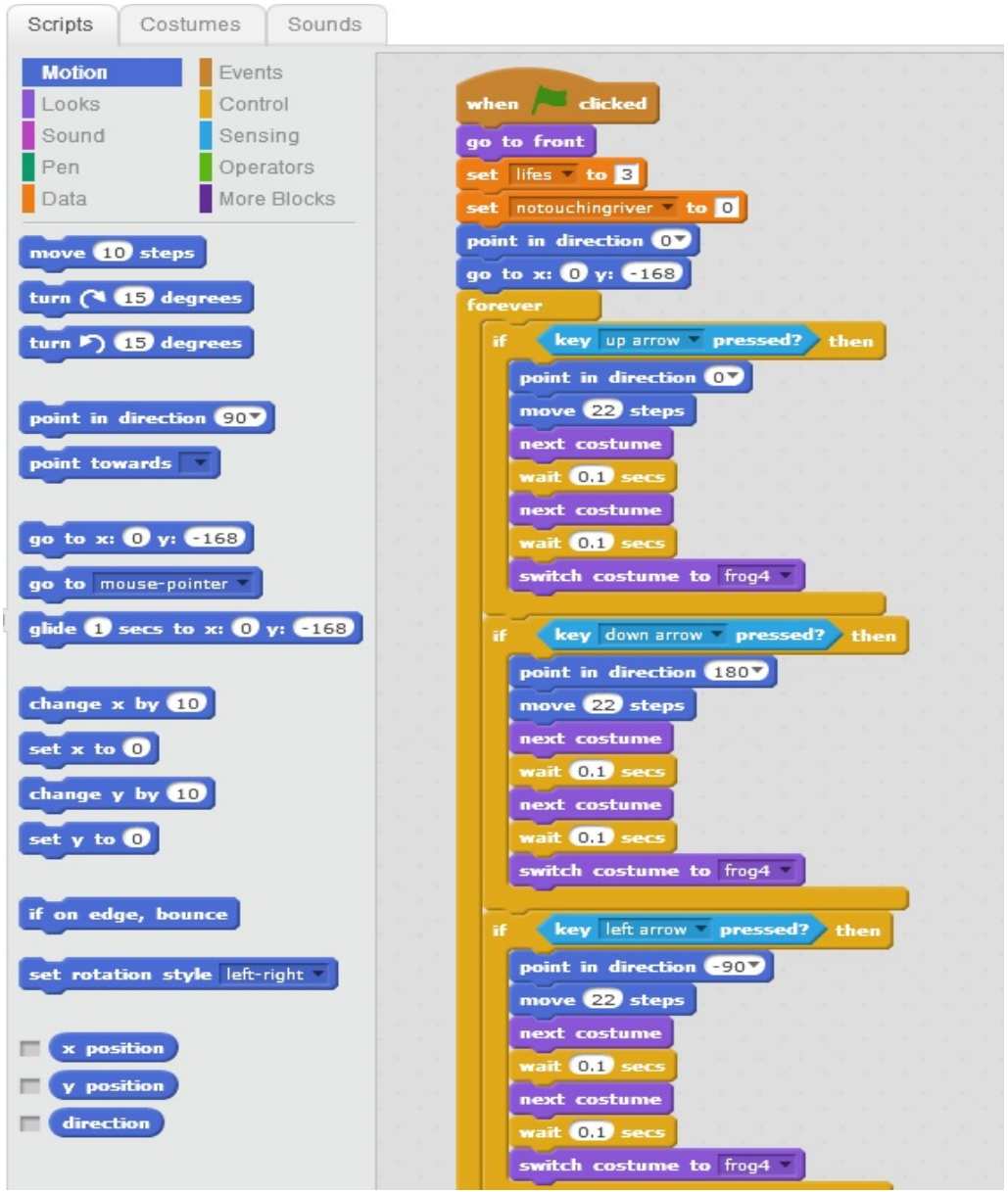

Figura 3.6: Captura de la interfaz del panel de programas en Scratch.

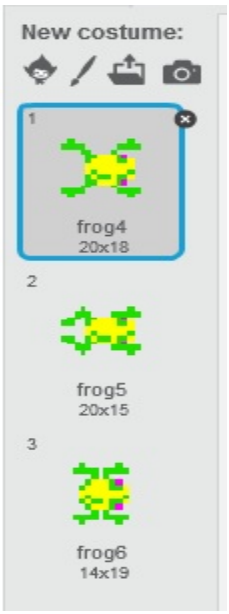

Figura 3.7: Captura de la interfaz del panel de disfraces en Scratch.

Figura 3.8: Captura de la interfaz del panel de sonidos en Scratch.

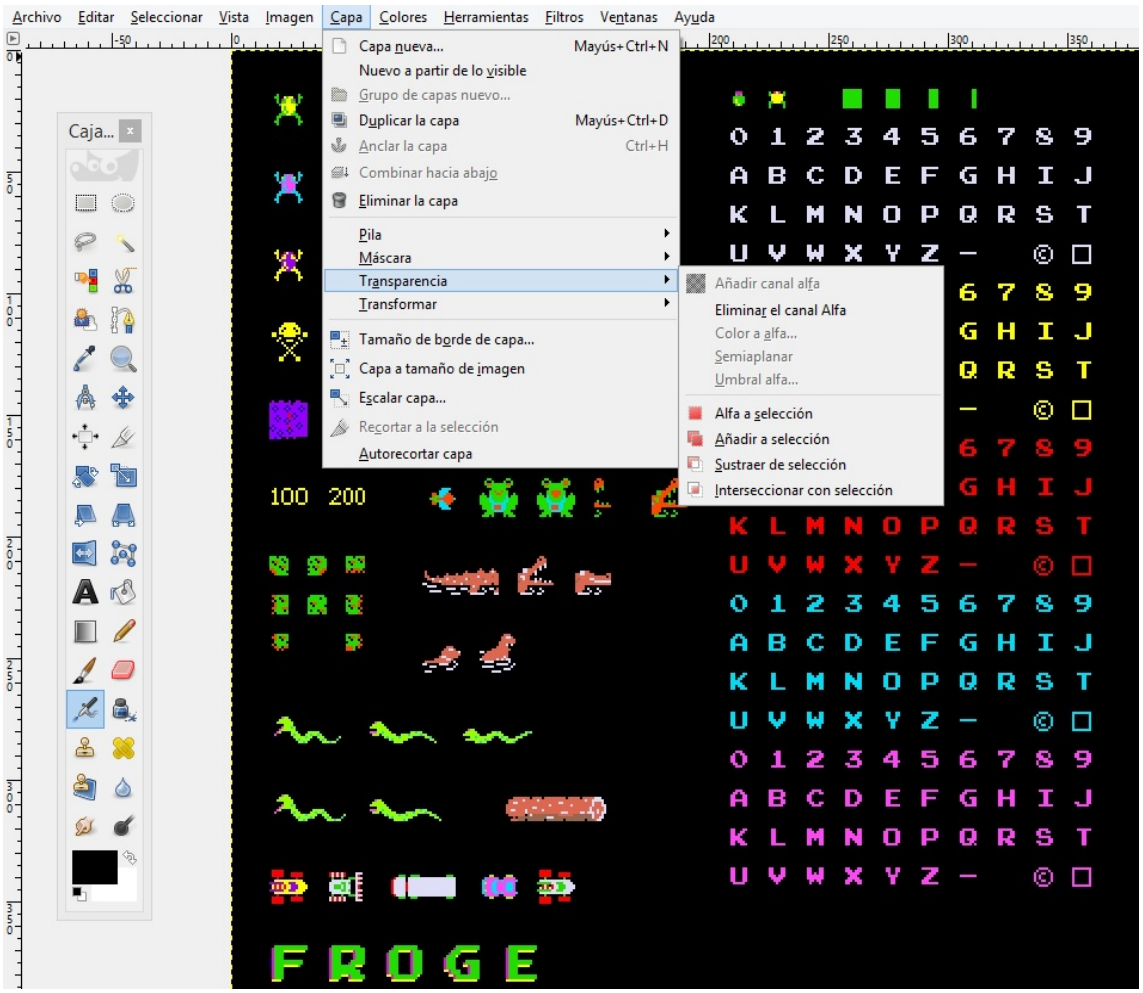

Figura 3.9: Adición de un canal alfa para poder insertar trasparencia a la imagen con GIMP.

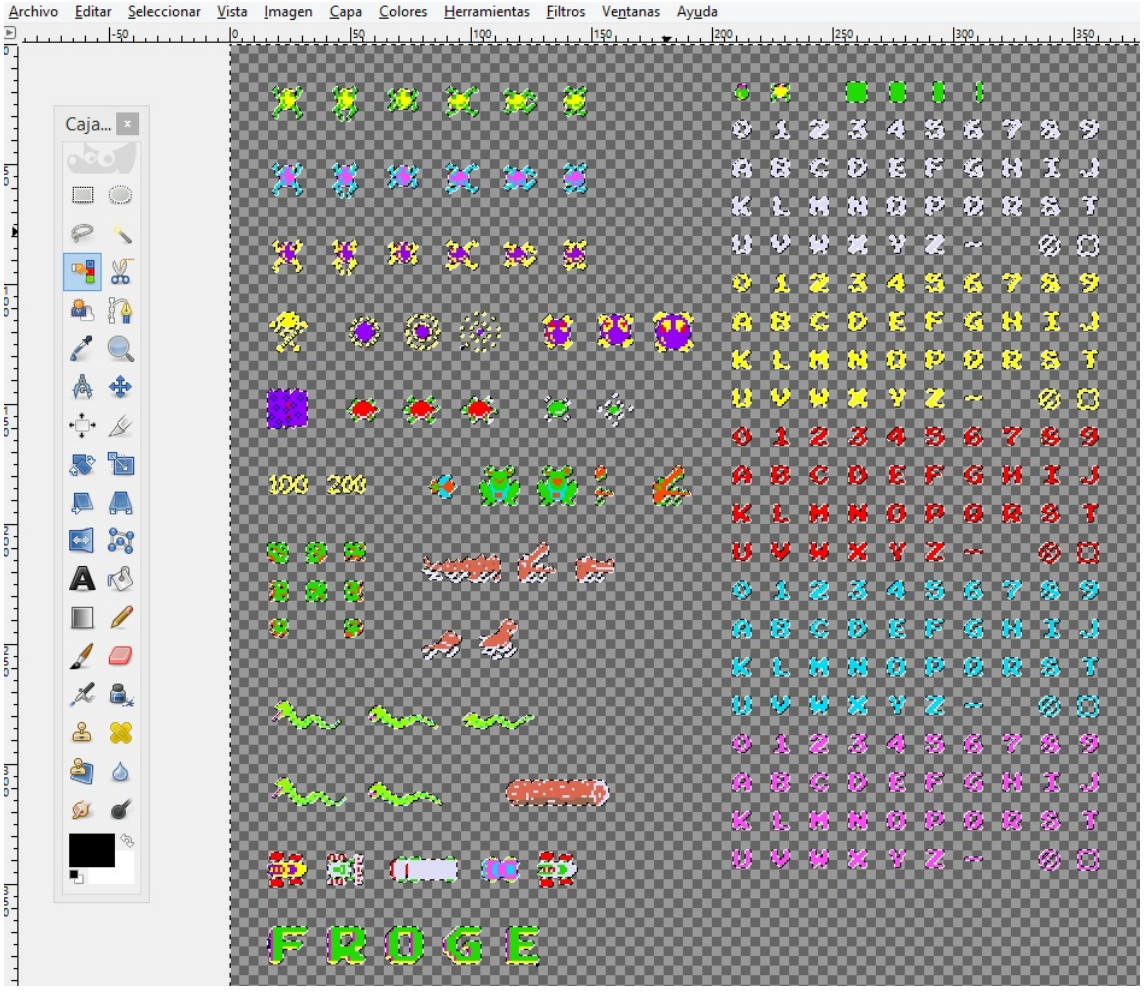

Figura 3.10: Resultado final de la imagen una vez añadida la transparencia.

CAPÍTULO 4
Frogger

El primer título que tiene lugar en este trabajo es Frogger. El videojuego es protagonizado por una rana perdida que el jugador debe conducir al otro lado del río para que encuentre su hogar. En el capítulo anterior he explicado los procedimientos llevados para el desarrollo de los videojuegos de una forma general. En este capítulo se explica más detalladamente los procesos llevados a cabo para desarrollar el videojuego con Scratch

4.1 Diseño del videojuego

En esta sección explico los pasos que he utilizado antes de ponerme con el desarrollo del videojuego, pues antes de programar es necesario saber qué y cómo se va a programar.

En primer lugar, he tenido que buscar por la web la ROM original del videojuego que salió en máquinas recreativas. Para montar la ROM y poder jugar he utilizado un conocido emulador llamado MAME 32[1]. MAME 32 es el mejor emulador para montar ROM de videojuegos que salieron en máquinas recreativas. El emulador permite al jugador jugar a los videojuegos tal y como en su época lucían en los salones recreativos. Eso sí, esta vez sin tener que insertar dinero para poder jugar. Para ello, el emulador simula las monedas de forma que con una tecla el usuario puede insertar todas las monedas que desee. Además, gracias a su avanzada configuración, es posible conectar un *arcade stick*[2] para hacer más real la emulación.

Después he probado el videojuego para ver las mecánicas jugables y cómo funcionaba el videojuego. Una vez hecha la idea sobre el videojuego, tocaba encontrar los *sprites* y sonidos que se usarían para el desarrollo de este mismo. En el capítulo anterior indiqué las fuentes para el material gráfico y los sonidos (gracias al usuario "Garycxjk" por haber compartido el material gráfico de Frogger). Para el caso de los *sprites* fue necesario una conversión con el programa de edición de imágenes GIMP. No se entra en detalle pues en la sección 3.5 se detalló

[1] El emulador se puede conseguir de manera gratuita en http://mamedev.org/.

[2] Es como se conocen a los mandos clónicos de las máquinas recreativas.

este proceso. En las figuras 3.9 y 3.10 se puede apreciar la conversión llevada a cabo con los *sprites* para el videojuego Frogger.

Seguidamente, con una idea acerca del videojuego y con los materiales, tocaba ver cómo imitar el videojuego a implementar con el original. Como fuente de inspiración se vieron varios proyectos subidos por *scratchers* en la página web de Scratch. De esta forma me hacía una idea de qué podía hacer y más o menos cómo lo haría. En especial gracias al usuario "cs50" cuyo Frogger me pareció el mejor de la comunidad de Scratch.

Además diseñé una máquina de estados para hacerme un idea sobre los distintos eventos que ocurren sobre objeto rana durante el tiempo de ejecución del videojuego. El programa utilizado ha sido el Bizagi Modeler[3] que aunque es un programa para diseñar otro tipo de diagramas sirve también para realizar posibles representaciones gráficas de diagramas de estados. El resultado del diagrama se observa en la figura 4.1.

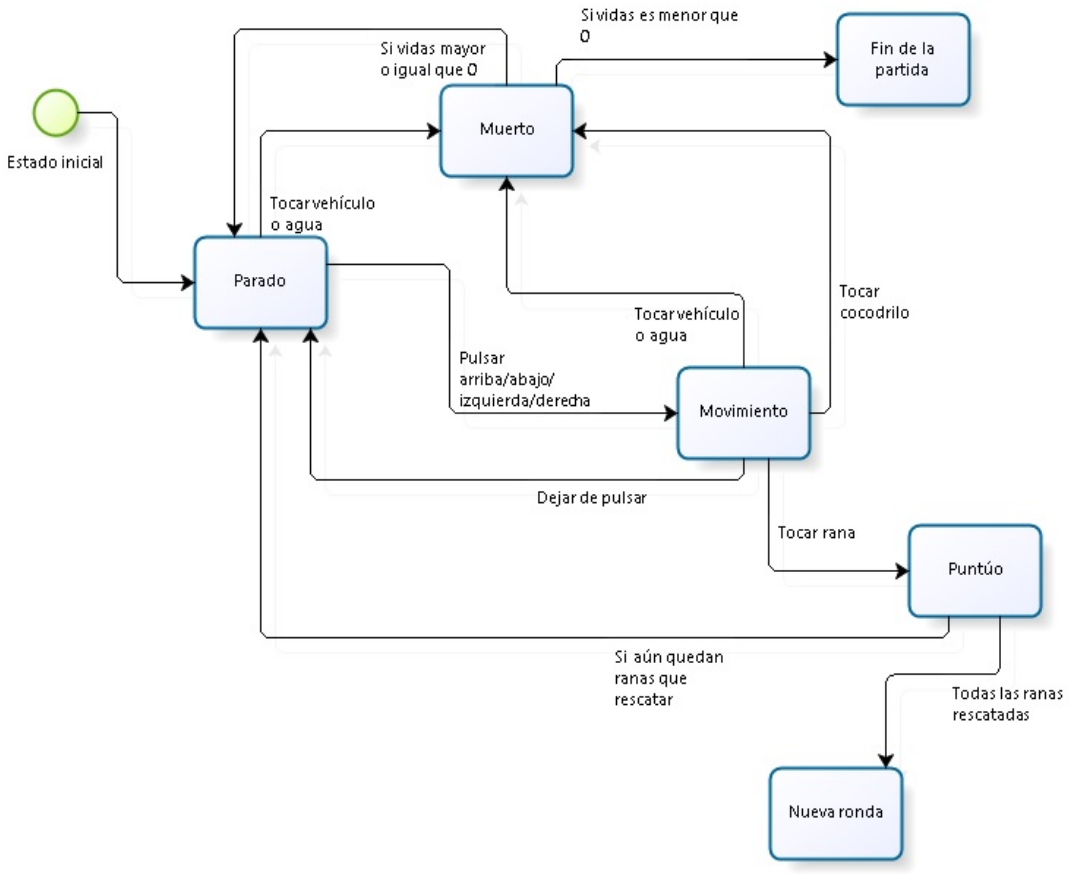

Figura 4.1: Diagrama de la máquina de estados de la rana en el videojuego Frogger.

Este diagrama se centra en los estados del objeto principal, es decir, de la rana que el jugador controla. Al iniciar el videojuego la rana permanecerá parada

[3]La versión gratuita de la herramienta se puede obtener en http://www.bizagi.com/es/productos/bpm-suite/modeler.

al principio de la pantalla. Se mantendrá en este estado mientras el jugador no presione ninguna flecha de movimiento. En el caso de pulsarla, la rana pasará al estado de movimiento hasta que se deje de pulsar la tecla. Momento en el que se volverá al estado de parado. Si mientras la rana se encuentra en uno de estos dos estados toca un vehículo o el agua, la rana morirá. Así pasará al estado muerto, en ese momento si el número de vidas no es negativo se restará una vida y se volverá al estado parado al inicio de la pantalla. Si por el contrario el número de vidas es negativo, se pasará al estado fin de la partida el cuál pondrá fin al videojuego teniendo que reiniciar la partida.

En el caso de llegar al final de la pantalla, si la rana toca al cocodrilo va al estado muerto y se repite lo anteriormente mencionado. Sin embargo, si se toca la rana que está en la orilla, el jugador puntúa. Ahora si aún quedan ranas que rescatar el jugador volverá al principio de la pantalla. Si todas las ranas han sido llevadas a su casa, comenzará una nueva ronda volviendo a empezar desde el estado inicial. Esta ronda será más difícil que la anterior.

En cuanto a otros objetos como son los vehículos, troncos y tortugas se desplazan en horizontal automáticamente a una velocidad determinada. De forma que cuando se inicie el videojuego, estos inician su desplazamiento. En el momento en que lleguen a una determinada posición, desaparecerán y volverán al principio de su desplazamiento. A continuación se explica detalladamente cómo se ha programado todas estas suposiciones en Scratch.

4.2 Implementación del videojuego

Con el material y las ideas claras, paso a desarrollar mi propia versión del videojuego Frogger en Scratch. En esta sección se van a describir los programas de los que se componen algunos de los objetos, pues algunos de ellos son similares entre sí por lo que sólo explicaré a fondo uno de ellos. El resto lo explicaré de una forma breve. En la figura 3.4 se pueden observar algunos de estos objetos.

El primer objeto a explicar es la rana (`frog`). Este objeto es el personaje principal que el jugador debe de controlar por lo que se trata de objeto principal del videojuego. En la figura 4.2 se encuentran parte de los programas que se ejecutan en este objeto. El primero de ellos se podría decir que es el principal, pues se encarga de inicializar las variables importantes del juego, como el número de vidas (`lifes`), número de ronda (`round`), puntuación (`score`), `froggers` y `notouchingriver`. Estas dos últimas se verá más tarde su funcionabilidad.

Una vez iniciadas las variables, se establece la posición inicial de la rana en el escenario. Seguido empieza el bucle que se ejecutará durante todo el tiempo de ejecución del juego. Los cuatro primeros condicionales tratan sobre el movimiento de la rana, dependiendo de cuál de las cuatro flechas se pulse con el teclado, la rana efectuará un movimiento. Por ejemplo, si se presiona la flecha arriba, la rana se moverá 22 pasos hacia arriba. Emitiendo en este tiempo el sonido `hop` y cambiando los disfraces automáticamente para simular una animación. Lo mis-

mo sucede para las otras tres direcciones. El código de este programa continúa con dos condicionales más en la figura 4.3. El primero de ellos controla que la rana no toque el agua. Para ello se utiliza un sensor que es igual a verdadero si la rana toca el color del agua, y la variable `notouchingriver` ha de ser igual a 0. En ese momento la variable `alive` pasa a valer 0, se resta una vida y se envía el mensaje `dead`. El otro se encarga de emitir el mensaje `gameover` cuando se han acabado todas las vidas y supone el final de la partida.

La variable `notouchingriver` controla si la rana está tocando el río o no. El hecho de crear esta variable se debe a que cuando la rana se subía a los obstáculos para cruzar el río, Scratch reconocía que estaba tocando el color del agua y esta moría. Para controlar los valores de dicha variable, creé un programa. Este programa se encarga de controlar dicha variable asignándole 1 (la rana está encima del objeto) o 0 (la rana no está tocando el objeto del río). Aproveché este programa para que la rana acompañase el movimiento horizontal en pantalla de los objetos que pasan por el río. De esta forma la rana no queda quieta al alcanzar los objetos pues se mueve junto a ellos cuando está tocándolos. Para ello he utilizado bloques de movimiento que permiten cambiar las coordenadas x de la rana. A este bloque se le han añadido bloques de operadores para calcular la nueva posición x de la rana. Dicha posición se le suma la posición actual de la rana y la velocidad del objeto, siendo positiva en caso de ir hacia la derecha, o negativa si va para la izquierda. Todo encadenado con varios condicionales `if...else`. En la figura 4.3 se observa la construcción a continuación del programa anterior.

Volviendo a la figura 4.2, se encuentran otros tres programas relacionados con el comportamiento del videojuego. Uno de ellos se encarga de recibir el mensaje `dead` que envía el programa principal cuando la rana muere. Este programa se encarga de restablecer la posición de la rana al principio de la pantalla y cambiar la variable `alive` a 1. Otro de los programas debe de esconde el objeto y las variables cuando recibe el mensaje `everyonehide`. Dicho mensaje se transmite nada mas iniciar el videojuego. Todos los objetos que actúan en el videojuego, a excepción de los que participan en el menú principal, tienen un programa que recibe dicho mensaje. Su función es la misma que en la rana, esconder el objeto para que no aparezca en pantalla hasta recibir el mensaje `startgame` que activa el programa principal. Ya el último programa que posee la rana se encarga de controlar el cambio de ronda. Lo único que hace es esperar a que se emita el mensaje `nextround` para establecer la posición inicial de la rana, reiniciar el número de vidas y el número de ranas a rescatar que se controla con la variable `froggers`.

Con los programas del objeto rana explicados. Ahora se explica la implementación de los programas para los vehículos que pasan por la carretera. Todos los vehículos tienen los mismos programas por lo que sólo se explicará uno de ellos. Lo único de difiere entre ellos es la velocidad de movimiento que en unos será mayor o menor y positiva o negativa. Cada vehículo se ha creado como un objeto único, esto quiere decir que si un mismo coche se repite tres veces en pantalla son tres objetos.

En la figura 4.4 se encuentran los dos programas que componen cada vehículo. El primer programa es el principal pues determina el movimiento del vehícu-

lo. Este comienza cuando recibe el mensaje `startgame` al igual que pasaba con el programa del objeto `frog`. Seguido se indica que se muestre el objeto al que pertenece. Hecho esto, se entra en el bucle que se ejecutará durante toda la partida. Una vez dentro, lo primero que se indica es el movimiento del objeto, la operación que hace para calcular el movimiento lateral es la misma que se calcula para el desplazamiento de la rana cuando se encuentra encima de un obstáculo por el río. Como se observa en la figura, este cálculo es el desplazamiento por la dificultad mas la ronda menos uno. La variable `difficult` tendrá unos valores dependiendo de la dificultad elegido por el jugador.

A continuación aparecen dos condicionales. El primero se encarga de controlar la posición del objeto de forma que si llega al final de la pantalla, este aparezca de nuevo en el otro extremo. El segundo condicional debe de esperar a que el vehículo toque a la rana para quitarle una vida y enviar el mensaje `dead` que como se vio restablece al objeto rana a su posición inicial.

Seguidamente explicaré los objetos que pasan por el río, troncos y tortugas. Todos estos objetos comparten código al igual que los vehículos. Salvo que ahora si tocan a la rana no se restará ninguna vida pues la rana debe subirse al objeto para poder cruzar el río. En el caso de las tortugas el programa es más complejo pues en ciertos momentos estas se deben de hundir costándole una vida a la rana si se encuentra encima de ellas. En la figura 4.5 se pueden observar los tres programas de los que se compone cada objeto tortuga.

Como se observa en la figura 4.5, el programa principal de la tortuga es muy parecido al de los vehículos, salvo que en vez del condicional que enviaba el evento de la muerte a la rana, hay otro condicional. Este se encarga de escoger un número al azar entre un rango de números y compararlo con 102 en este programa. Si se cumple la condición se envía el mensaje `2turtleshink3`. Dicho mensaje activa el otro programa de los que se compone el objeto. Este cambia entre los distintos disfraces de la tortuga para finalmente hundirse. Si en ese momento toca a la rana, se repite el paso del condicional de los vehículos que indicaban la pérdida de una vida. Finalmente se vuelve al disfraz original de la tortuga.

Con los vehículos, obstáculos y la rana explicados, faltan por explicar los objetos de ranas que significan la meta para el jugador. En el juego original estas ranas aparecen una vez la rana ha llegado a su meta en la posición que llega. Para facilitar la implementación, en mi versión ocurre al contrario. Estos objetos indican el lugar al que el jugador debe guiar a la rana para salvarse. Su programa se encarga de controlar el número de ranas que quedan por salvar controlando la variable `froggers`, en el momento que esta valga 0 un programa del fondo enviará el mensaje `nextround`.

Una vez explicados los objetos principales, toca comentar los fondos (*backgrounds*) que componen el videojuego. El juego contiene cuatro posibles fondos, el primero se utiliza para crear el menú principal, el segundo es el escenario en el que se desarrolla el videojuego, el tercero son las instrucciones y el último corresponde a los créditos. Tal y como se aprecia en la figura 4.6, estos programas se encargan de controlar la ejecución del videojuego. El primer programa empie-

za en el momento que el jugador inicia el videojuego. Este nada mas iniciarse envía el mensaje `everyonehide` que como dije lo reciben todos los objetos comentados anteriormente. Se inicializa la canción que suena de fondo junto con una variable llamada `mainmenumusic`. Esta variable se controla con el otro programa cambiando su valor entre 0 (la música deja de sonar o 1 (si debe sonar la música). Inicialmente valdrá 1 hasta que el jugador desee empezar la partida; momento en el que pasará a valer 0. También este otro programa cambia el fondo al del videojuego y se encarga de controlar el paso a la siguiente ronda difundiendo el mensaje ya visto `nextround`. A parte los tres restantes programas se encargan de cambiar entre los fondos del menú principal al recibir su correspondiente mensaje.

4.3 Organización del menú principal

Con el videojuego ya desarrollado, llega el momento de crear un menú principal para facilitar al usuario el acceso al videojuego. El menú principal se compone de tres opciones principales: jugar, instrucciones y créditos. También se crea un submenú para seleccionar la dificultad. Este se encuentra una vez seleccionado jugar. La implementación del menú principal se ha realizado con objetos. Es decir, los títulos de las distintas opciones son objetos creados en Scratch para facilitar la implementación. En la figura 4.7 se puede observar el aspecto que tiene el menú principal del videojuego Frogger.

En el momento que el jugador selecciona la opción jugar aparecerá en pantalla los tres niveles de dificultad. Para conseguir esto, el objeto envía el mensaje `difficult` que será recibido por todos los objetos del menú. Los objetos instrucciones y créditos se esconderán al recibir dicho mensaje. Mientras que los objetos principiante, avanzado y experto se mostrarán en pantalla. Dependiendo de cual de ellos elija el jugador, la variable `difficult` (vista en la sección anterior) tendrá un valor distinto. Si por el contrario, el jugador desea conocer las instrucciones, el proceso es similar. Salvo que ahora no se emitirá el mensaje `difficult` sino el mensaje `tutorial` que al recibirlo el fondo. Este cambiará al fondo con las instrucciones. Exactamente pasa lo mismo al seleccionar el objeto créditos pero mostrando su fondo correspondiente. Si se desea volver al menú principal, se ha implementado una flecha que permite volver al principio del menú.

Para destacar la opción seleccionada en el menú principal, se ha implementado un efecto visual. De esta forma cuando el jugador pase el puntero por una de las distintas opciones, esta resaltará de las demás. El efecto visual escogido para este videojuego se llama ojo de pez. Su efecto es difuminar desde el centro hacia afuera, consiguiendo así resaltar la opción seleccionada. Dicho efecto se aplica a todos los objetos que forman parte del menú principal.

4.4 Comentarios finales

El videojuego ofrece un resultado completo, aunque ha habido algunas cosas del videojuego original que no han sido implementadas. Una de ellas es la aparición de objetos que pueden recogerse y proporcionan puntos extras si se llega al objetivo con ellos. Otra posible ampliación es realizar un temporizador. El videojuego principal disponía de un tiempo límite para terminar una ronda. Para ello se puede crear una variable temporizador, inicializarla a un tiempo inicial y que se le vaya restando uno cada segundo que pase. También es posible cambiar de posición las ubicaciones en donde debe de llegar el jugador. Esto se podría hacer controlando cada ronda de manera individual y enviando un mensaje al final de cada una. Este mensaje lo recibirán los objetos ubicados al final del río. Al recibir los mensajes, estos reposicionan su ubicación en otro lugar de la pantalla.

En cuanto a los problemas, surgieron varios durante la implementación. El problema más grave fue en el momento en que la rana debe subirse a los obstáculos para cruzar el río. Como mencioné en el apartado anterior, creo una variable llamada `notouchingriver` esta variable almacena el valor 1 cuando la rana está encima de un obstáculo y 0 cuando no lo está. La razón por la que creo esta variable es porque facilita la detección de cuando la rana está en agua. Sin esta variable, Scratch detecta que aunque la rana se encuentre encima de un obstáculo esta se encuentra en el agua. Y claro, la rana muere. Otro problema que ha surgido al redimensionar los objetos para adaptarlos al fondo principal. Al ser un videojuego en el que la rana muere si toca algún vehículo, es importante redimensionar estos objetos para tengan todos la misma dimensión. De no ser así, la rana puede morir aunque no esté en el carril del vehículo.

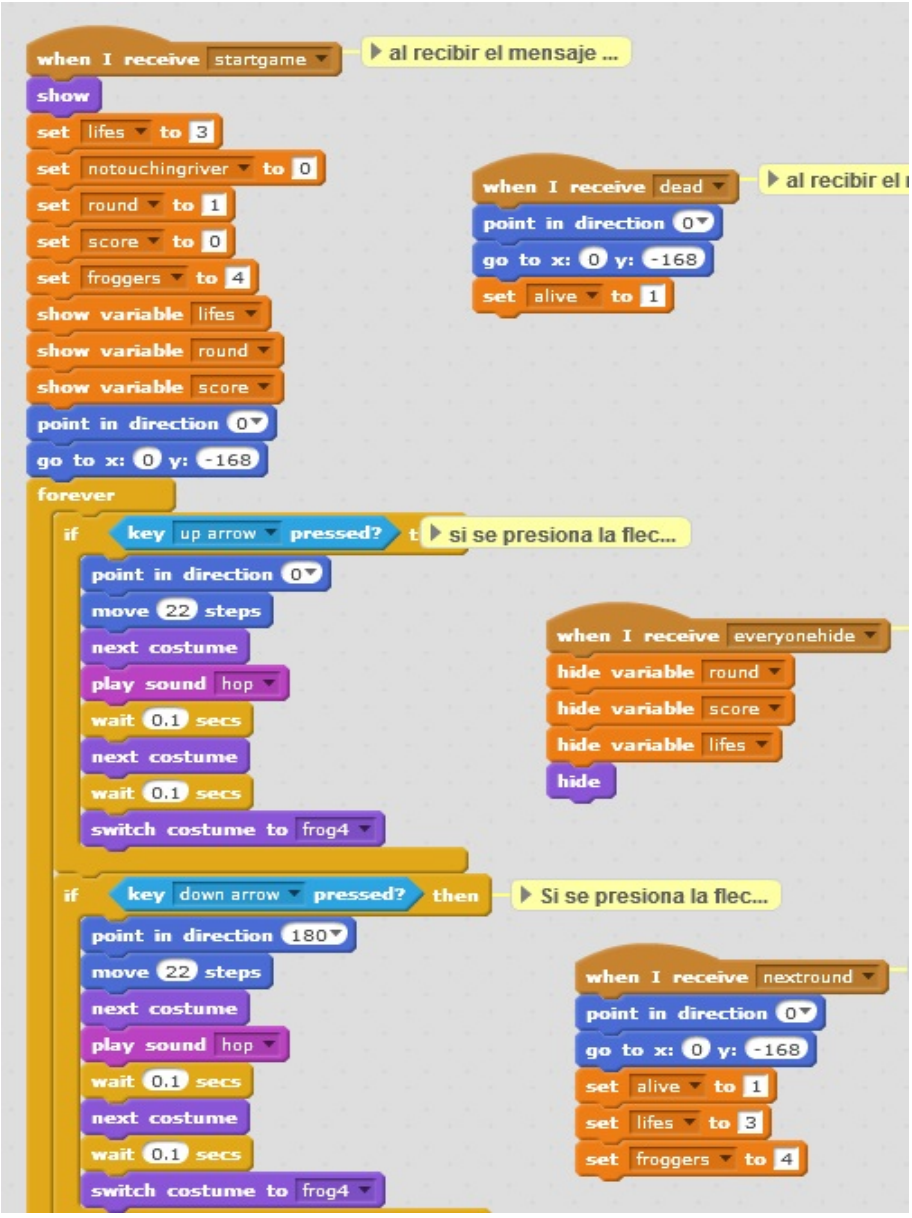

Figura 4.2: Captura de parte de los programas del objeto rana (`frog`) en Scratch.

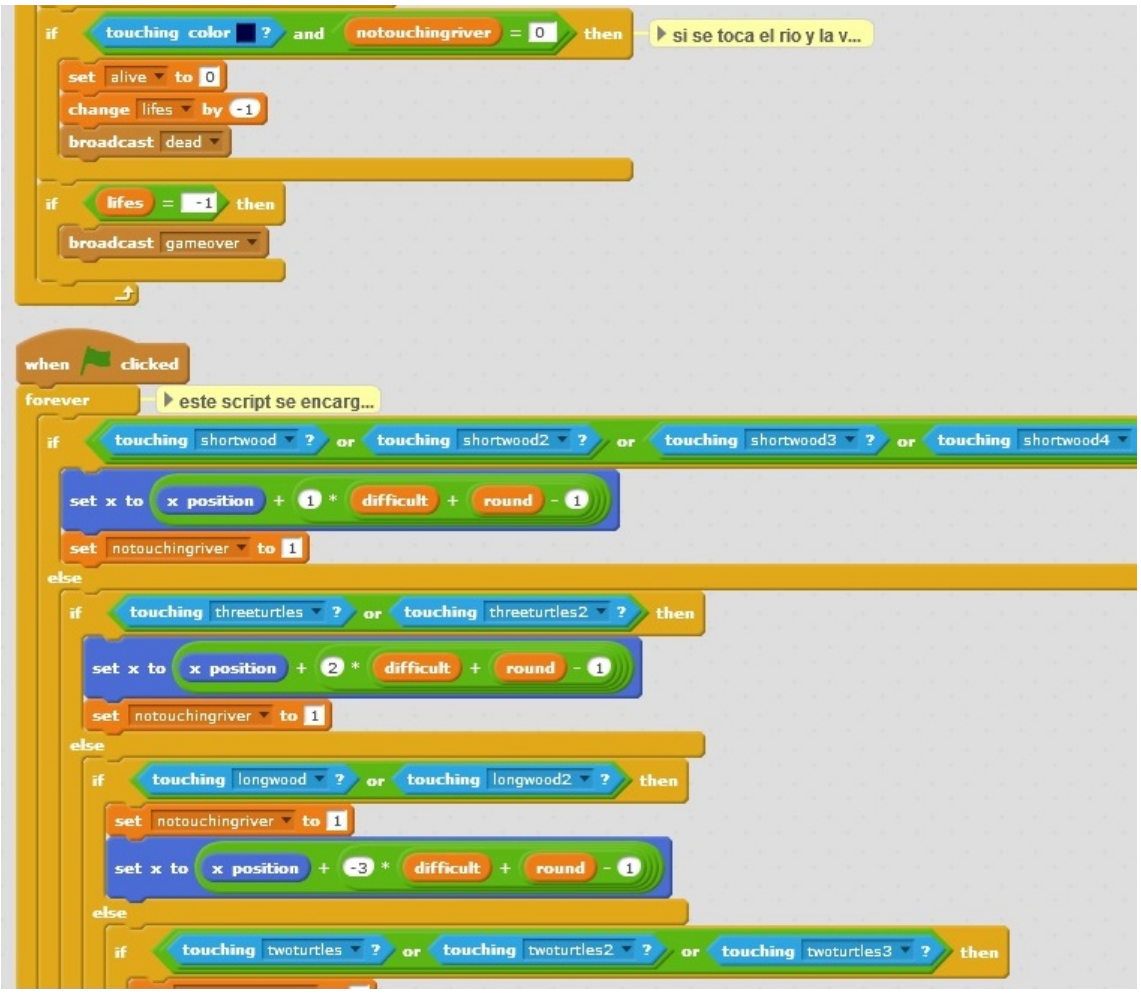

Figura 4.3: Captura de la segunda parte de los programas del objeto rana (`frog`) en Scratch.

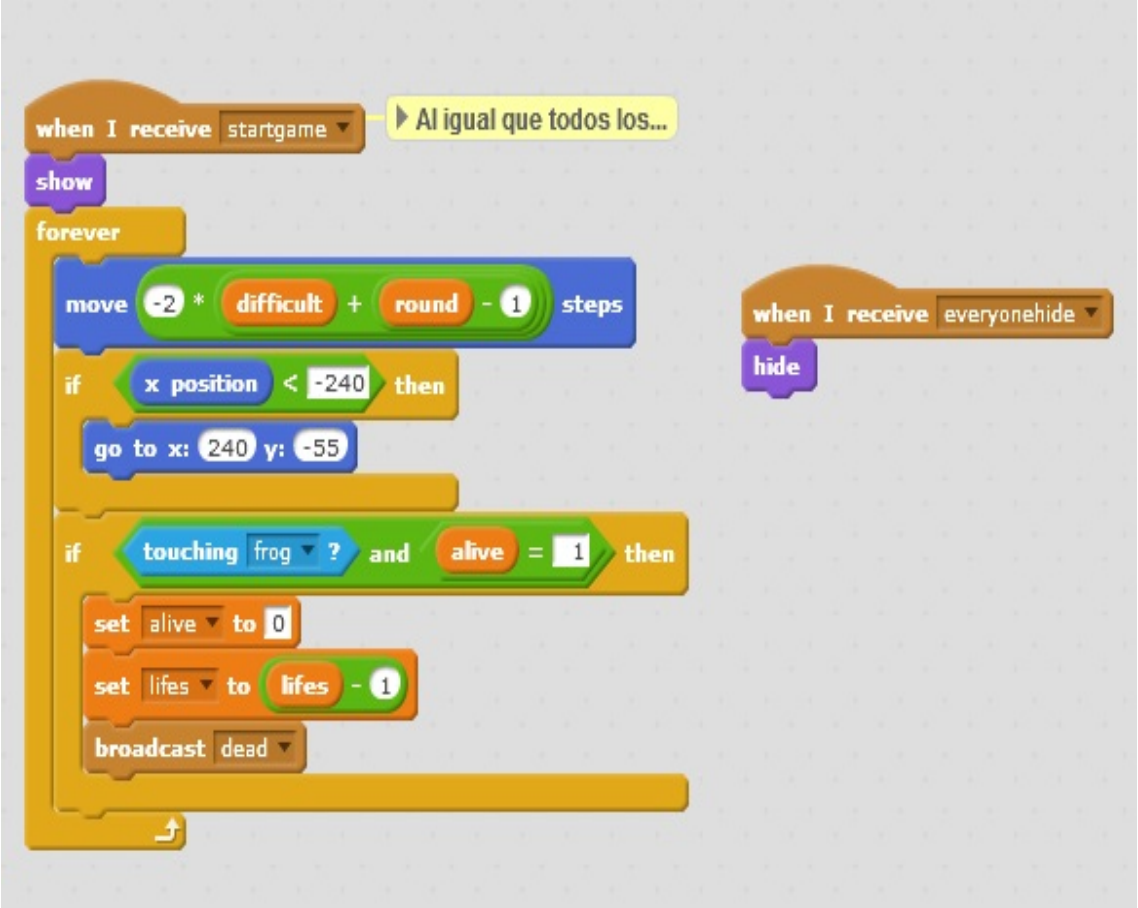

Figura 4.4: Captura de los dos programas que se componen los vehículos en el videojuego Frogger.

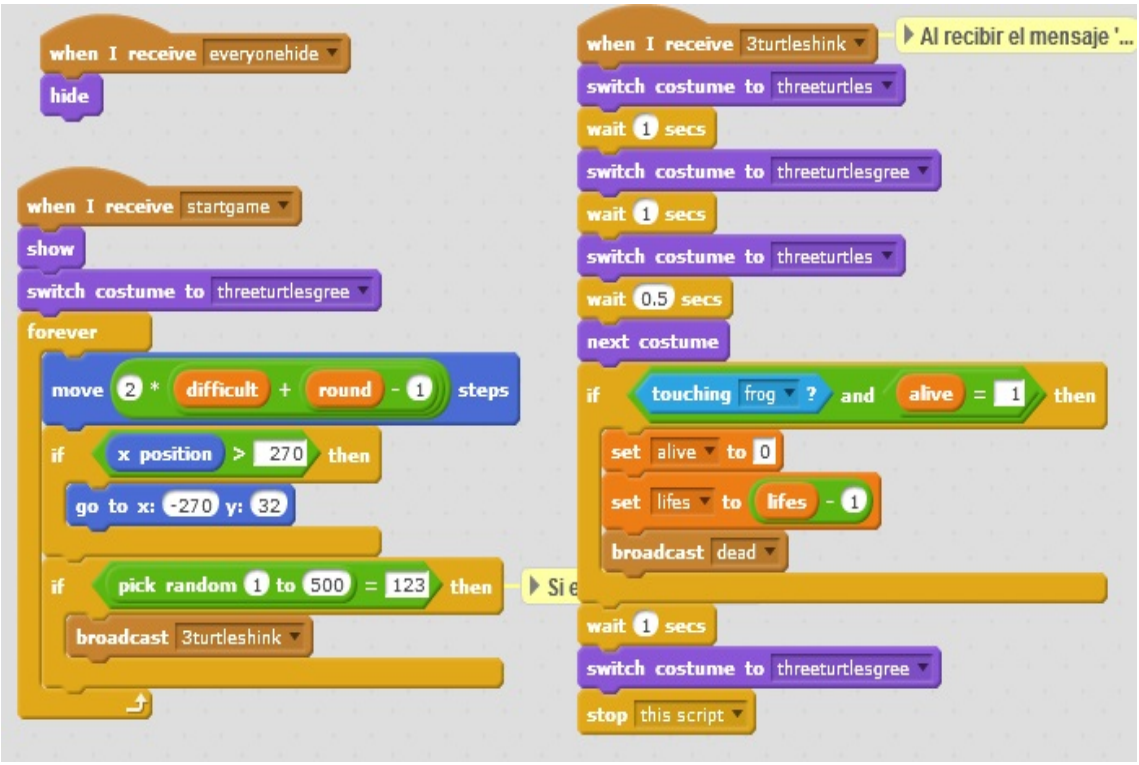

Figura 4.5: Captura de los dos programas que se componen las tortugas en el videojuego Frogger

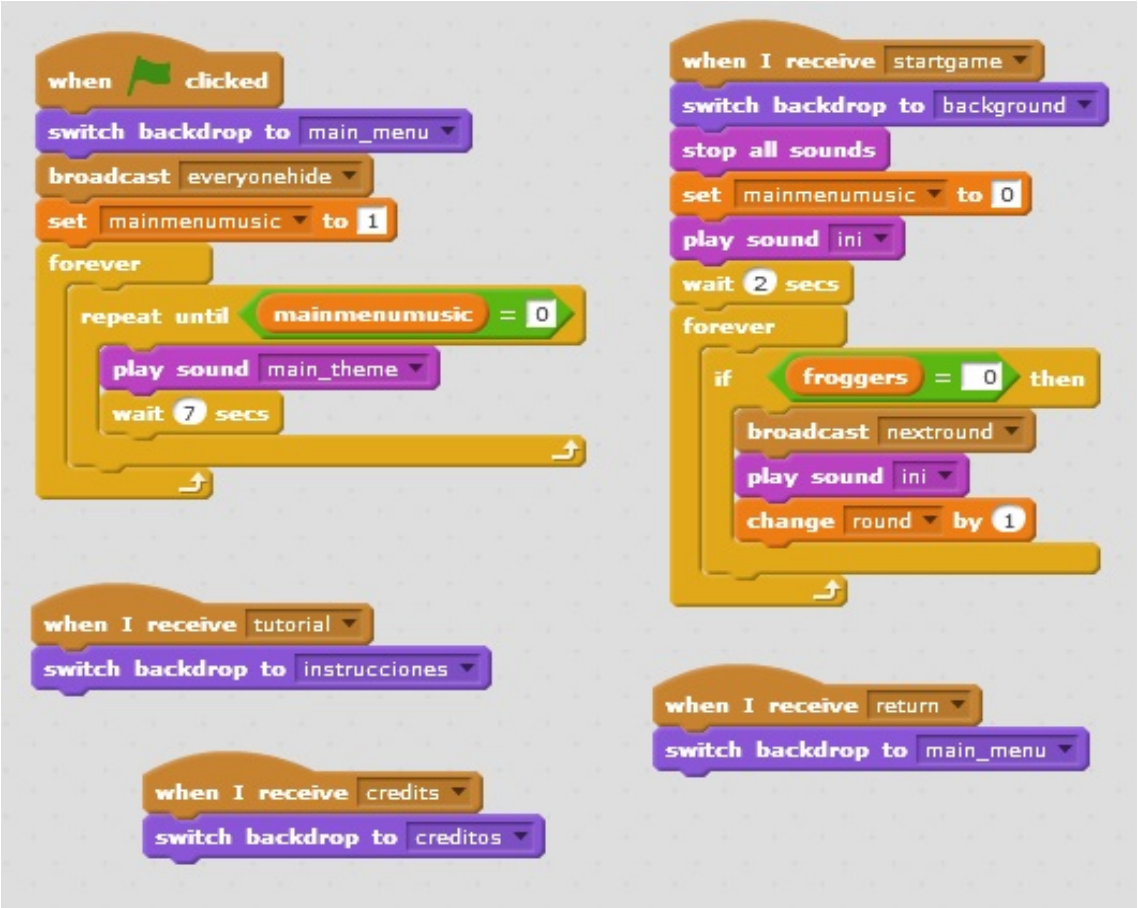

Figura 4.6: Captura de los dos programas que se componen los fondos en el videojuego Frogger.

Figura 4.7: Menú principal de la versión implementada del videojuego Frogger.

CAPÍTULO 5
Duck Hunt

Con el videojuego Frogger explicado, ahora toca explicar otro de los títulos que tienen lugar en este trabajo. El videojuego se llama Duck Hunt. Este videojuego de disparos en primera persona tiene como protagonista al jugador, que tendrá que abatir tantos patos como sea posible para alcanzar la mejor puntuación. Para su desarrollo se han seguido las mismas pautas que en el anterior videojuego, vistas en el apartado 3.5. En este capítulo se explica de una forma más precisa todo el desarrollo del videojuego.

5.1 Diseño del videojuego

El primer paso realizado para conocer todo acerca del videojuego Duck Hunt ha sido probar el videojuego. Para ello al igual que pasó con Frogger, he tenido que encontrar la ROM por internet. Por la red me he encontrado con mucho emuladores para la videoconsola NES. De entre todos el emulador escogido se llama VirtualNes[1]. Este emulador se ha utilizado para probar el videojuego Duck Hunt y el Super Mario Bros, de este último hablaré en el siguiente capítulo.

Una vez probado el videojuego, he buscado por internet todo el material gráfico que podría ser útil en el desarrollo de Duck Hunt. Al igual que en el videojuego anterior, el material gráfico lo he sacado de la página web `http://www.spriters-resource.com/` y `http://www.sounds-resource.com/`. Especialmente gracias al usuario "Silver" por compartir los `sprites`. Como ya ocurría en el videojuego Frogger, los `sprites` vienen todos sobre la misma imagen. Estos deberán de extraerse, para ello se utiliza el programa gratuito GIMP cuyo proceso de extracción se explica detalladamente en el apartado 3.5.

Con el material multimedia, procedo a estructurar cómo se debe de implementar el videojuego Duck Hunt con Scratch. A diferencia de Frogger, este videojuego tiene dos objetos principales, el pato y el perro. Para esquematizar sus comportamientos, he representado dos máquinas de estados con el programa gratuito Bizagi Modeler. Una enfocada en el pato y la otra en el perro.

[1] El emulador se puede descargar gratuitamente desde `http://www.emulator-zone.com/doc.php/nes/virtuanes.html`.

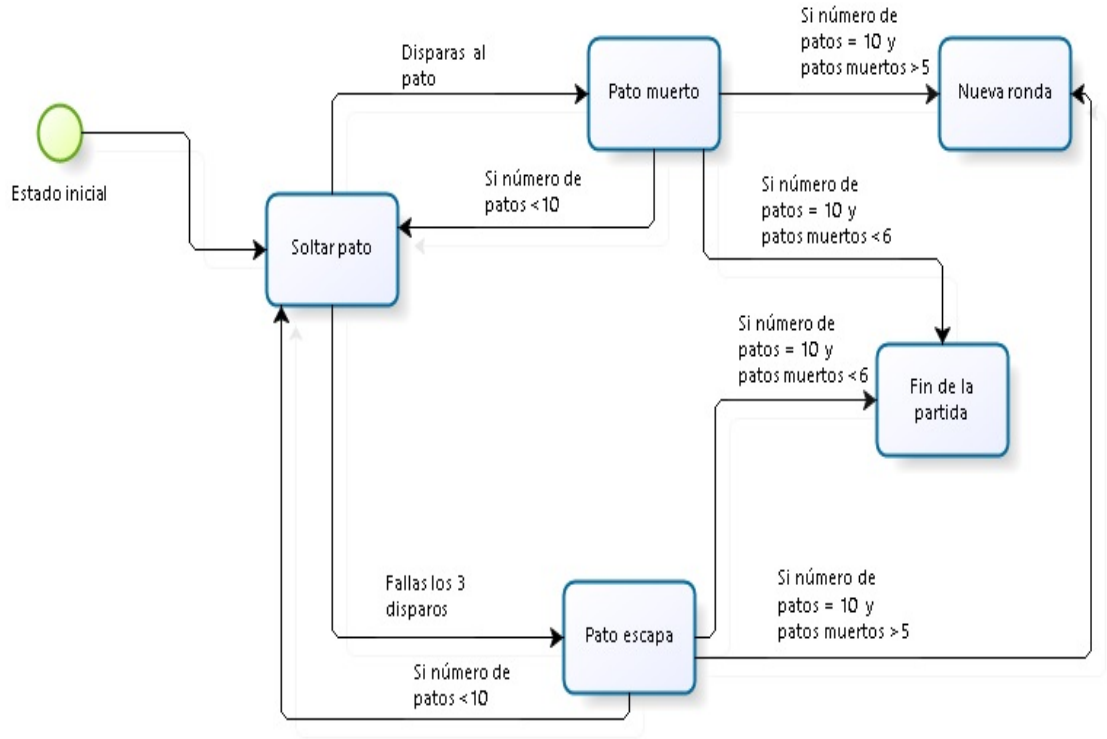

Figura 5.1: Representación de la máquina de estados del objeto pato.

El primer objeto principal a tratar es el pato. Como se observa en la máquina de estados de la figura 5.1, al empezar el videojuego se suelta el pato. En ese momento el jugador tiene tres disparos para cazar al pato. Si consigue acertar, el pato pasa a estado muerto y cae al suelo. Si no es así, el pato pasa al estado escapar, con lo que saldrá de la pantalla. En ambos estados, el pato volverá a aparecer en el caso de que el número de patos sea menor que diez. En cada ronda aparece un total de diez patos. Una vez aparecidos todos los patos (pueden haber muerto o escapado) llega el recuento de patos cazados durante la ronda. Si el número de patos acertados es mayor que cinco, da comienzo una nueva ronda. Si por el contrario no han sido cazados esté número mínimo de patos, la partida finaliza.

El otro objeto principal es el perro, cuyos estados también han sido representados con una máquina de estados. En la figura 5.2 se contemplan los distintos estados de este objeto. Cuando el jugador inicia la partida, el objeto perro permanece escondido entre los matorrales. En el momento que el objeto pato pase a su estado muerto, el perro aparecerá en pantalla cogiendo el cuerpo del pato. Una vez hecho esto, el pato volverá al estado escondido automáticamente. Algo parecido pasa con el siguiente estado del perro. Si por el contrario, el pato se escapa, el perro pasa al estado aparecer pero sin pato en mano. Ahora sale y empieza a reírse del jugador para luego volver a esconderse. Si se empieza una nueva ronda el perro permanecerá escondido hasta que muera o escape un pato. Pero en caso de final de la partida, el perro permanece en su estado jocoso hasta que el jugador decide empezar de nuevo.

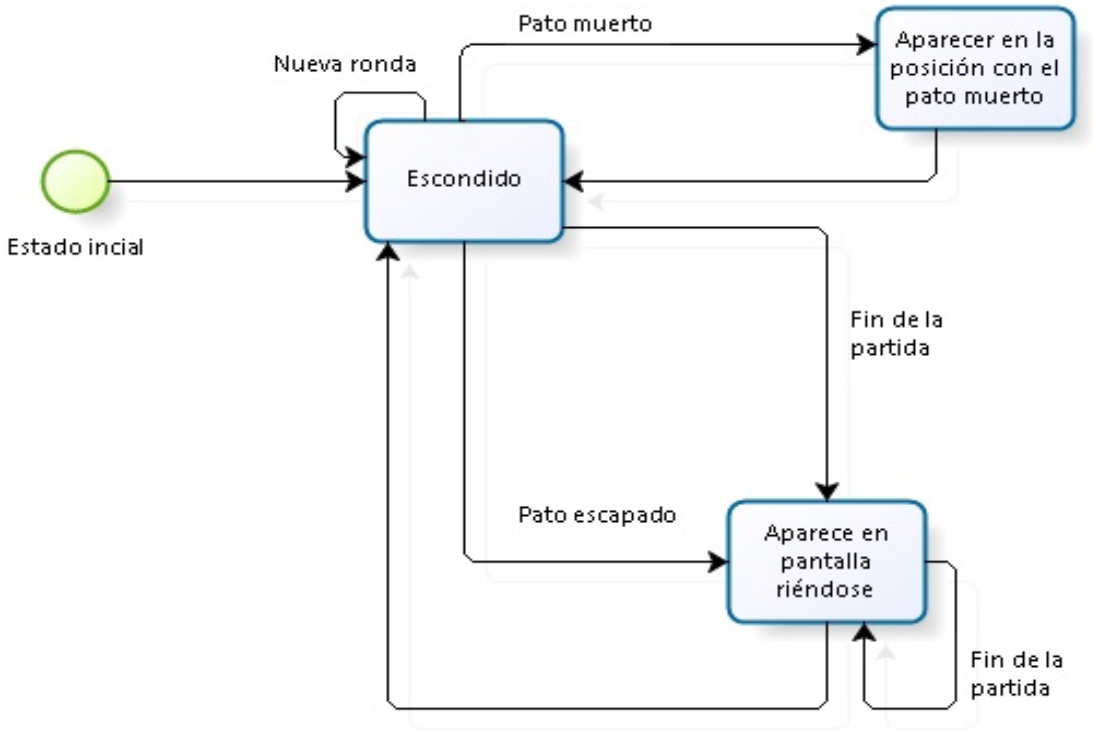

Figura 5.2: Representación de la máquina de estados del objeto perro.

5.2 Implementación del videojuego

Una vez representados los diagramas de estados para los dos objetos principales, obtenido el material multimedia necesario y con una idea clara sobre cómo adaptar el videojuego con Scratch. Empieza la implementación de Duck Hunt con Scratch. En esta sección se explicarán detalladamente los programas de los objetos más importantes del videojuego.

Los primeros programas a explicar son los del objeto perro (`dog`), este objeto es junto al pato uno de los principales a la hora de programar el videojuego. En la figura 5.3 se pueden observar todos los programas de este objeto que se explicarán a continuación.

El objeto se compone de seis programas, dos de ellos son los más importantes. Estos tratan los estados del perro vistos en la sección 5.1. Cuando el objeto reciba el mensaje `doglaugh`, empieza el programa que se encarga de hacer que aparezca el perro de los matorrales y empiece a reírse del jugador por haber fallado. Para ello el perro cambia de disfraz y saldrá en la posición x del último movimiento del pato. Para esconder al perro entre las hierbas, se ha creado un objeto copiando las hierbas del fondo de pantalla. De esta forma, el perro está siempre debajo de la hierba. Cuando tiene que salir tan solo debe de cambiar su posición y sumándole 2 hasta llegar a -30. De esta forma saldrá por encima del objeto hierba creado. Una vez haya salido de la hierba, se emite el mensaje `music`. Este mensaje lo recibe otro de los programas del perro y reproduce el sonido con las risas del

perro. La razón por la que se ha creado dicho mensaje es para que se ejecute el sonido en paralelo mientras el perro realiza la animación de reírse. Esta animación comienza justo después de emitir el mensaje y dura lo mismo que el sonido. Una vez terminado, el perro vuelve a cambiar su posición y, esta vez sumando -2 hasta esconderse por completo. Seguidamente controla la reaparición de los patos en pantalla. Si la variables `gameover` es igual a 0 y el número de patos soltados es menor que 10, envía el mesaje `duckrespawn`. Si por el contrario esto no se cumple, envía los mensajes `ducknumbercount` y `duckorder` y esperará. Estos mensajes iniciarán la cuenta de los patos cazados en sus respectivos programas. Estos programas decidirán si hay una nueva ronda o es el fin del juego.

Se acaba de ver el programa que ejecuta el perro cuando un pato escapa. En caso de acierto el programa creado es muy similar al anterior. Este programa se ejecutará cuando reciba el mensaje `dogsmiley` que emitirá el objeto pato al morir. En ese momento el perro saldrá de los matorrales con el pato en la mano. Su implementación es igual que el programa anterior, salvo por el sonido que ahora se ejecutará en el mismo programa pues ahora el perro no tiene ninguna animación. Por lo demás el código es igual. Otro de los programas creado se encarga de inicializar todas las variables referentes a los patos muertos y es que se han creado diez variables; una para cada pato soltado. Esto se ha hecho para facilitar la representación de la interfaz gráfica que indica los patos cazados y escapados. Estas variables se inicializan a 0 y pasarán a valer 1 si ese pato es cazado.

Los dos programas restantes son los menos importantes. El primero de ellos se encarga de posicionar al perro cuando recibe el mensaje que da comienzo al juego, y el otro es el programa que tienen todos los objetos e indica que si se recibe el mensaje `everyonehide` el objeto deberá de esconderse.

Con los programas del perro explicados, paso a explicar los programas del objeto pato (`duck`). Los programas del pato se pueden ver en las figuras 5.4 y 5.5. El primer programa de la figura 5.4, se encarga de inicializar las variables principales del videojuego que son la puntuación (`score`), ronda (`round`), pato muerto (`duckshot`) y el número del pato (`ducknumbers`). Después comienza el bucle que se encargará de cambiar el disfraz al pato creando así la animación de volar. Los disfraces se irán intercambiando mientras la variable `duckshot` sea igual a 0; es decir, mientras el pato siga vivo. Paralelamente se ejecutan otros cuatro programas al recibir el mensaje `startgame`. Centrándome en los programas de la figura 5.4, el que se sitúa a la derecha del programa anteriormente descrito tiene que controlar la dirección hacia la que el pato se mueve. Para ello se escoge un número aleatorio entre 1 y 4. Dependiendo del número el pato apuntará hacia una dirección. Finalmente esperará un determinado tiempo antes de cambiar la dirección. Este tiempo será menor con cada pato soltado en una ronda, incrementando así la dificultad de esta.

Siguiendo con los programas de la figura 5.4, los dos siguientes programas también se ejecutan paralelamente junto a los anteriores. El de la izquierda tan sólo se encarga de dar una velocidad para que el pato se desplace por la pantalla. Esta velocidad se calcula con una expresión algebraica que suma las variables `duckspeed` y `round`. La variable `duckspeed` tomará un valor distinto dependiendo

de la dificultad escogida por el jugador. La variable `round` incrementa constantemente en cada nueva ronda. De esta forma la velocidad del pato se aumenta en cada ronda que pasa. El último de estos programas tiene que cambiar la posición del pato cuando este toca los bordes de la pantalla o el matorral. Una vez tocados, el pato cambia su dirección por la contraria. Es decir, si el pato apunta hacia la derecha (90) al tocar uno de estos objetos cambia su dirección por la izquierda (-90).

Con la primera parte explicada, ahora explico la segunda figura. En la figura 5.5 se encuentra el último programa del objeto pato que se ejecuta al recibir el mensaje `startgame`. Al igual que se hizo para ejecutar el sonido de la burla del perro sin que fastidiara la ejecución de los programas. Para el sonido del pato se ha hecho algo similar; este sonido se ejecuta en paralelo de forma que no interfiera en la ejecución de los otros programas. Además se aprovecha el programa para asignar la posición x del pato a la variable `duckx` que es la que recibe el perro para aparecer en la última posición del pato.

Continuando en la misma figura, en esta parte se tratan los estados muerto y escapado, ambos vistos en la sección anterior. Cada uno de estos dos estados posee dos programas, uno que se encarga de gestionar los disfraces del pato y enviar los mensajes `dogsmiley` en caso de cazar el pato o `doglaugh` en caso de que escape. Como se vio estos mensajes activan parte de los programas del objeto perro. El otro programa debe de gestionar cada pato que se suelta si se mata o no. Es importante tener en cuenta que patos escapan y cuáles no para gestionar la interfaz gráfica que indica el progreso al jugador. Para ello se crea una variable por pato haciendo un total de diez variables.

Aún en la misma figura pueden observarse más programas. Uno se ejecutará al recibir el mensaje `duckrespawn` enviado por el perro para soltar un nuevo pato mientras no se hallan soltado los diez patos. Otro se ejecuta al recibir el mensaje `duckcheck` que debe de conocer si el pato se le ha tocado con el ratón (disparado) o por el contrario perder una bala. Si el pato es alcanzado se enviará el mensaje que cambiará el estado del pato a muerto. O por el contrario si no quedan balas el mensaje `duckflyaway` deberá enviarse y cambiará el estado a escapado.

Una vez explicados los programas de los dos objetos principales, queda por explicar otros objetos secundarios. Uno de ellos corresponde con las balas que el jugador dispone, su código es bastante sencillo. El objeto tan sólo debe de cambiar de disfraz en función de la variable *shots* que como vimos son los disparos que al jugador le quedan. Otro de los objetos es la hierba que permite esconder al objeto perro hasta que salga. El puntero que el jugador controla es otro objeto. Este debe de seguir los movimientos del ratón y emitir el mensaje *duckcheck* al pato en el momento del disparo. Además reproduce el sonido que simula un disparo. Además para la animación del perro al principio del programa se ha creado un objeto. Este cambia de disfraz cada cierto tiempo lo que permite simular la animación del perro paseando para después ocultarse entre los hierbajos.

Finalmente quedan los objetos que componen la interfaz gráfica que indica al jugador el progreso de la caza. Para su implementación se han utilizado once

objetos. Uno es el fondo de la interfaz y los otros representan a cada uno de los diez patos que componen un ronda. En la figura 5.6 se encuentran los programas que compone el icono del primer pato de la interfaz. Gracias al usuario "cs50" de la comunidad Scratch por compartir los programas que permiten manipular la interfaz en tiempo de ejecución así como realizar una animación al final de cada ronda. A continuación tan solo se explicarán los programas del primer icono. El resto de iconos sus programas son iguales.

Al recibir el mensaje `startgame` el icono se sitúa en primera posición dentro de la interfaz y espera hasta que el primer pato sea cazado. En ese momento el objeto cambia el disfraz por el mismo icono pero de color rojo, lo que indica que el pato está muerto. Paralelamente, hay un programa que esconde y muestra el icono de manera intermitente mientras el pato está en pantalla indicando al jugador en todo momento el número del pato en la ronda. El resto de programas se activan de manera consecutiva al finalizar la ronda. Como se vio anteriormente, los objetos dos objetos principales enviaban dos mensajes al finalizar la ronda. Estos eran `ducknumbercount` y `orderducks`. Pues bien, estos mensajes activan dos programas del presente objeto. Al recibir `orderducks` uno de los programas controla el número total de patos disparados con la variable `duckshots`. Después envía un mensaje a cada uno de los objetos que componen la interfaz si se han cazado todos los patos. Es decir, si tan solo se cazan tres patos, el mensaje se enviará a los tres primeros patos independiente del orden de caza. Mientras al recibir el mensaje `ducknumbercount` el objeto reconoce si su pato asignado ha sido cazado o no modificando la variable `duckshots` que controlará el programa anterior. Al final si el objeto se encuentra dentro del número de patos cazados, este realizará la animación de aparecer y desaparecer mientras suena la música de fondo. Dependiendo de si se pasa de ronda o es el final del juego, sonará una música distinta.

5.3 Organización del menú principal

Ahora que el videojuego ya está implementado, llega la hora de crear un menú principal para estructurar de una mejor manera el videojuego. Al igual que con el videojuego Frogger, este menú dispondrá de las mismas opciones: jugar, instrucciones y créditos. Además permitirá seleccionar el nivel de dificultad para la caza de patos. Para su implementación, se han creado seis objetos. Cada objeto representa una opción dentro del menú principal. En la figura 5.7 se observa el aspecto final del menú principal.

Estos objetos interactúan con los fondos del videojuego. Por ejemplo si el jugador desea conocer las instrucciones del videojuego tendrá que hacer clic en su correspondiente opción. En ese momento el objeto instrucciones envía un mensaje que recibe un programa del objeto fondo. Este deberá de cambiar el fondo actual y mostrar el nuevo fondo. Además todos los objetos que no deben de estar visibles se tienen que esconder, para ello estos objetos recibirán también el mensaje que iniciará un programa para esconderse. Si el jugador selecciona la opción

jugar, aparece en pantalla tres niveles de dificultad. Dependiendo de la dificultad escogida el pato se moverá a una velocidad distinta. La variable `duckspeed` obtendrá un valor mayor si se selecciona la dificultad más difícil o un valor menor si se desea en fácil. Una vez seleccionada la dificultad, dará comienzo el videojuego.

Cabe destacar que mientras el jugador se encuentra en el menú principal suena el tema original del videojuego. Para su reproducción se ha seguido el mismo método que con el videojuego Frogger. El sonido se controla con un programa que reproduce el sonido mientras una variable llamada `mainmenumusic` vale 1. Al salir del menú, esta variable pasará a valer 0 y la música se parará.

5.4 Comentarios finales

Al igual que ya se vio en el capítulo 2, mas concreto en la sección 2.2. El videojuego original traía varios modos de juego. En este trabajo me he centrado en implementar el modo principal, pero se podría hacer una ampliación adaptando los otros dos modos. Uno de ellos era similar al modo principal, lo único es que en ese modo se sueltan dos patos en vez de uno. Para implementarlo podría ser tan sencillo como añadir otro pato y que el jugador deba de cazar los dos patos para pasar al siguiente. El otro modo se trata de disparar platos en vez de patos. Los platos se lanzan mueven en una perspectiva hacia el fondo simulando lejanía. Este modo quizás sea un poco más difícil de implementar, ya que hay que ir haciendo más pequeño el plato conforme está en pantalla para conseguir el efecto de lejanía hasta desaparecer.

En cuanto a los problemas surgidos, el principal fue hacer la interfaz gráfica que indica los patos muertos y escapados. Para solucionar dicho problema tuve que recurrir a la comunidad de Scratch. Allí vi un proyecto que implementaba perfectamente dicha interfaz. El dueño del proyecto es "cs50" por lo que gracias a él pude continuar mi desarrollo. Otro problema surgió una vez implementado y probado por mis amigos. Resulta que el juego podía disparar cuando no estaba el pato en la pantalla, esto causaba un *bug* que dificultaba continuar con la experiencia. Para solucionarlo tuve que añadir una variable que controlara si hay pato o no. Esta variable se le pasaba al puntero permitiéndole disparar o no.

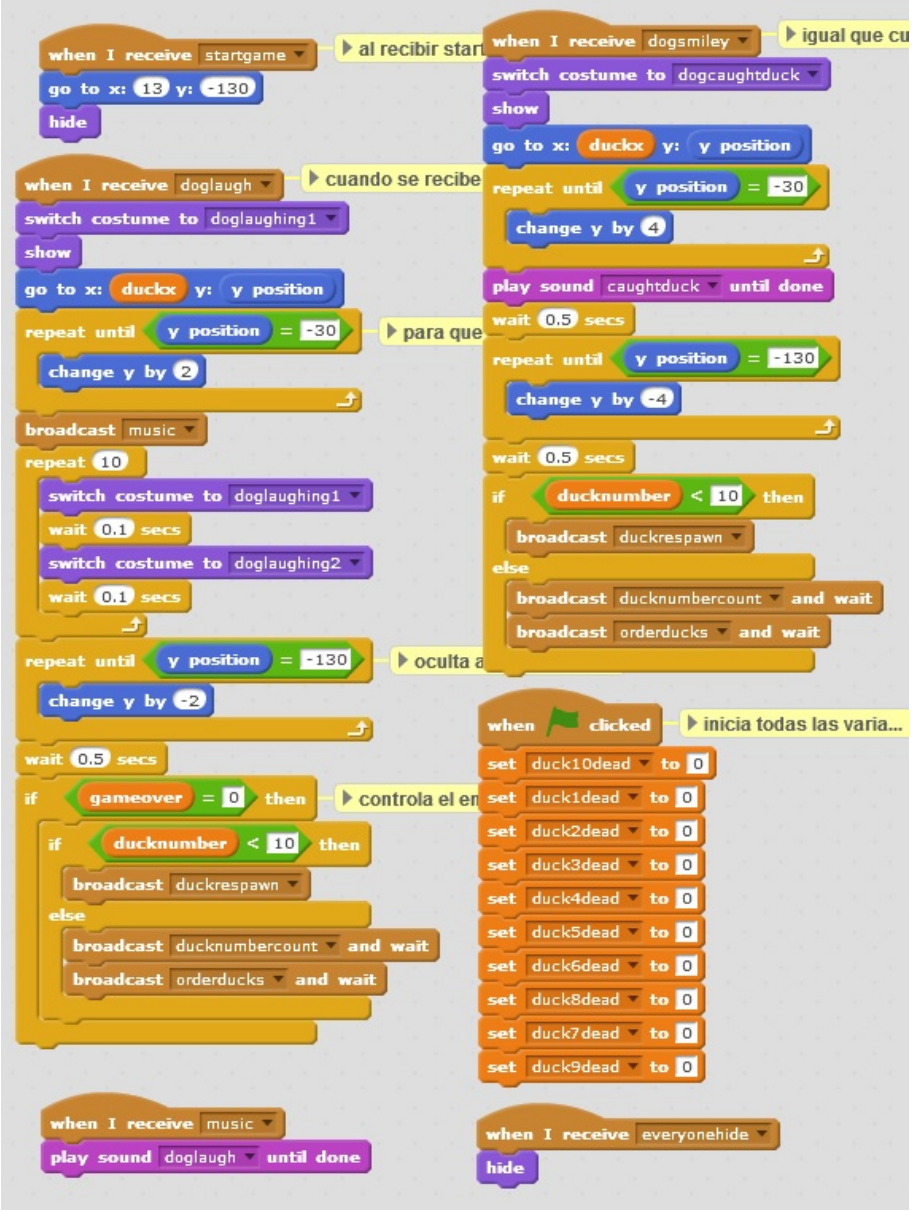

Figura 5.3: Captura de todos los programas del objeto perro (dog) en Scratch.

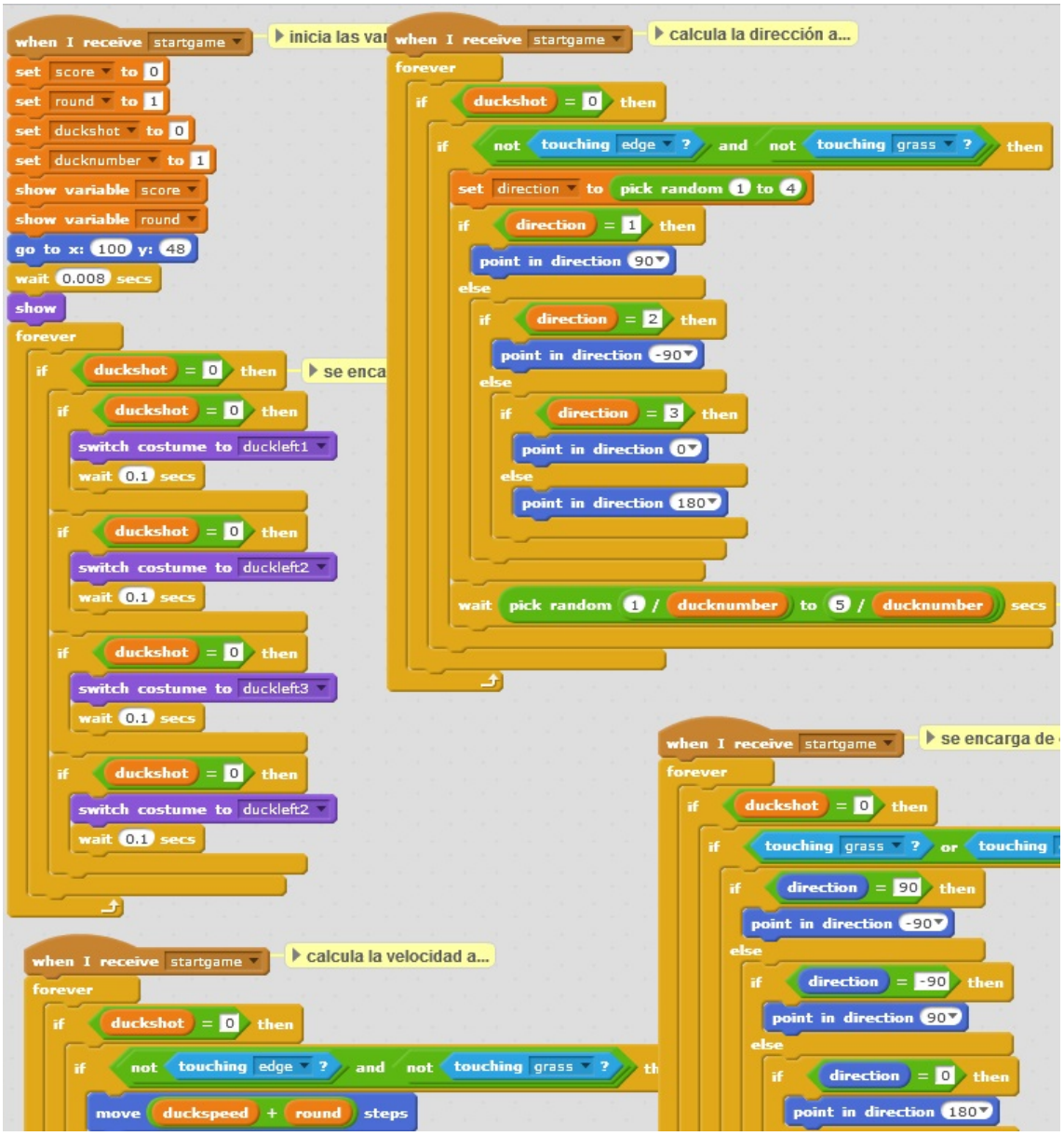

Figura 5.4: Captura de la primera parte con los programas del objeto pato (`duck`) en Scratch.

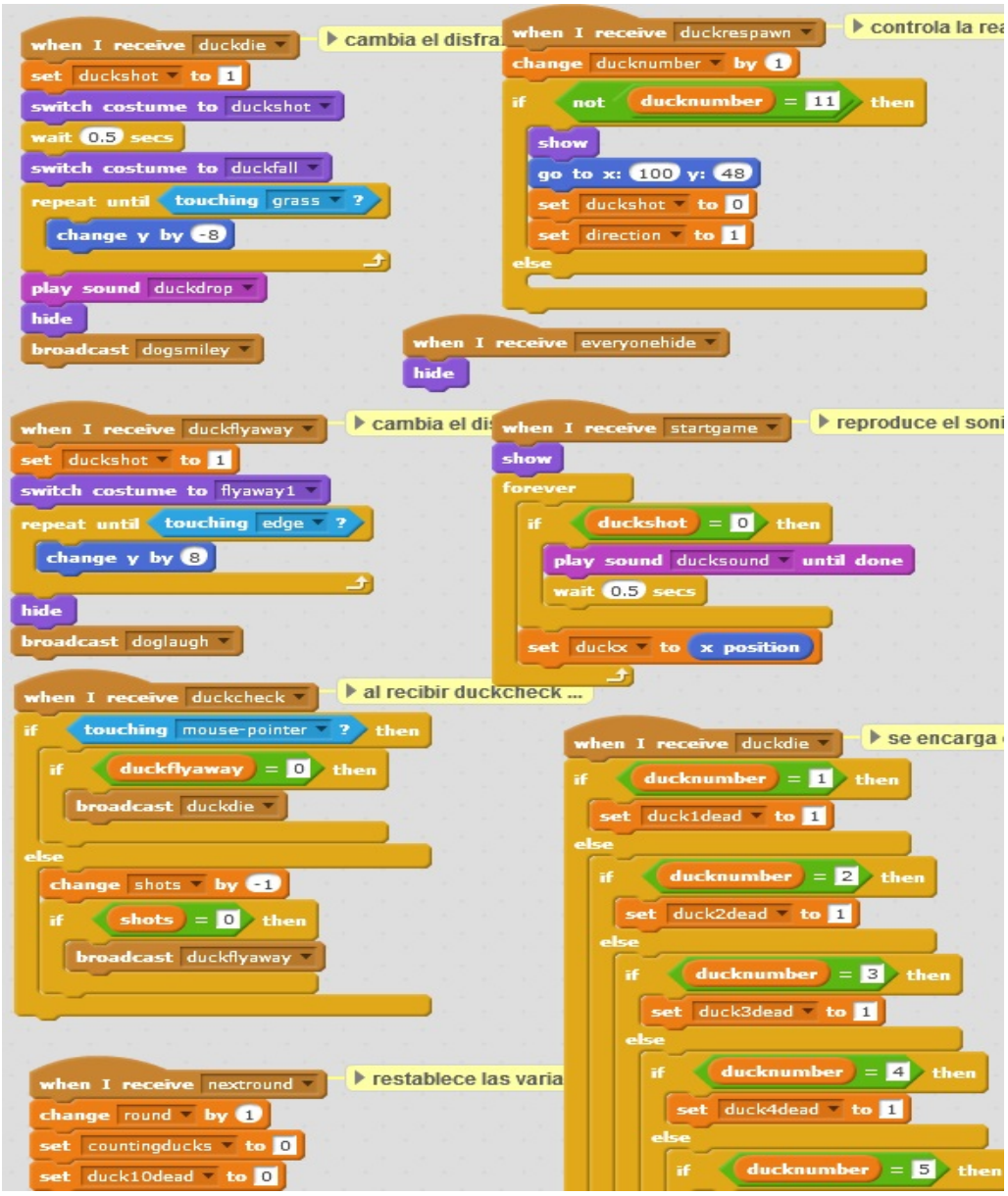

Figura 5.5: Captura de la segunda parte con los programas del objeto pato (`duck`) en Scratch.

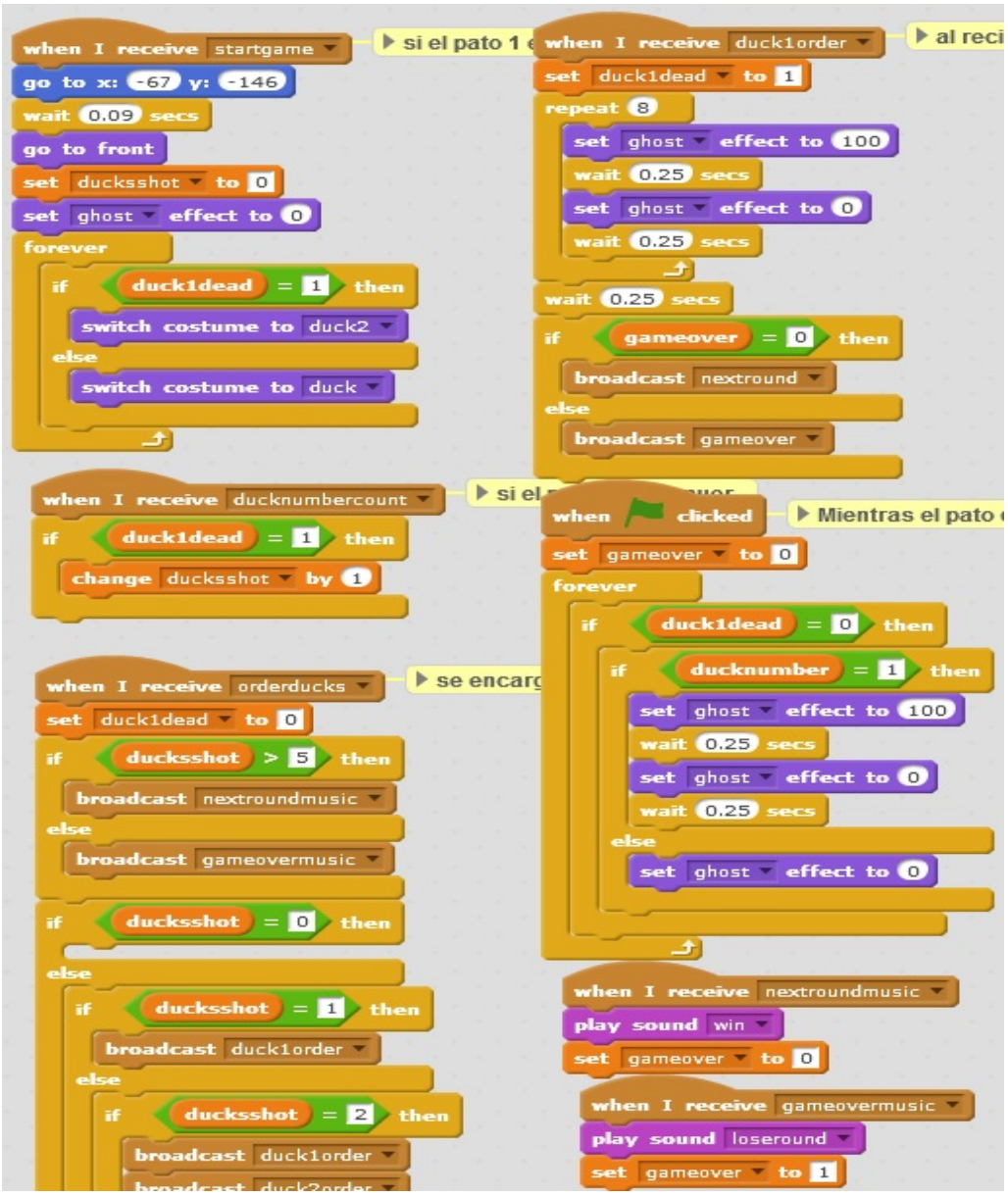

Figura 5.6: Captura de los programas del primer objeto patito de los diez que hay en la interfaz que indica al jugador el progreso de la partida.

Figura 5.7: Menú principal de la versión implementada del videojuego Duck Hunt.

CAPÍTULO 6
Super Mario Bros

El último videojuego tratado en este trabajo es Super Mario Bros. Este videojuego pone al jugador en el papel de Mario que tendrá que enfrentarse a grandes peligros para rescatar a su princesa. En este capítulo se describe de manera específica las fases llevadas a cabo para el desarrollo de este videojuego empezando por cómo ha sido el diseño previo a la implementación, seguido de su implementación con Scratch, la creación de un menú principal y por último un apartado con comentarios adicionales.

6.1 Diseño del videojuego

Para desarrollar Super Mario Bros se han seguido las mismas pautas ya mencionadas en capítulos anteriores. Antes de implementar el videojuego con el lenguaje de programación Scratch es necesario conocer las bases del videojuego y tener una idea sobre cómo trasladar las mecánicas a Scratch. Para este videojuego no ha sido necesario buscar una ROM por internet. El videojuego lo adquirí de la tienda oficial de Nintendo (Nintendo eShop) para jugarlo en la videoconsola Nintendo 2DS. Esta tienda ofrece a cambio de unos precios asequibles poder adquirir algunos de los videojuegos más influyentes de la compañía. El videojuego viene acompañado de su propio emulador para poder reproducirse en las consolas de nueva generación.

Una vez probado el videojuego es momento de buscar todo el contenido multimedia. Al igual que con Frogger y Duck Hunt, todo el material se ha descargado gratuitamente de las páginas web http://www.spriters-resource.com/ y http://www.sounds-resource.com. Gracias a los usuarios ”khgamer6” y ”dryd3418” por compartir los recursos públicamente.

Con el material necesario ya en el ordenador, toca intentar adaptar la versión original del videojuego. El principal problema para su adaptación son las limitaciones que posee Scratch. En Super Mario Bros, el jugador mueve al personaje Mario por la pantalla al mismo tiempo que esta se desplaza horizontalmente dando lugar a nuevos elementos que aparecerán en pantalla conforme Mario se desplace. Scratch no posee ninguna herramienta para implementar videojuegos

de este tipo, por lo que es tarea del programador conseguir que la pantalla se desplace conforme se mueve el personaje. El truco para conseguirlo está en que Mario siempre permanece en el mismo sitio y es el escenario el que se desplaza de un lado a otro. Para ello es necesario declarar una variable que controle el desplazamiento de los objetos que componen el escenario.

Con la idea principal clara, llega el momento de realizar una representación gráfica de una máquina de estados para los movimientos del objeto principal (Mario). Para su representación he utilizado el programa Bizagi Modeler pues permite diseñar todo tipo de representaciones gráficas. La máquina de estados se centra en los movimientos de Mario. Aunque en realidad Mario no se mueva, si que realiza las animaciones de caminar y saltar mientras el escenario se desplaza, transmitiendo al jugador una sensación de movimiento. Cada estado representa una animación. Por ejemplo si el objeto se encuentra en el estado caminando, Mario realizará la animación de caminar y el escenario se desplazará hacia la derecha o izquierda, dependiendo de la dirección que el jugador desee. En la figura 6.1 se observa dicha máquina de estados.

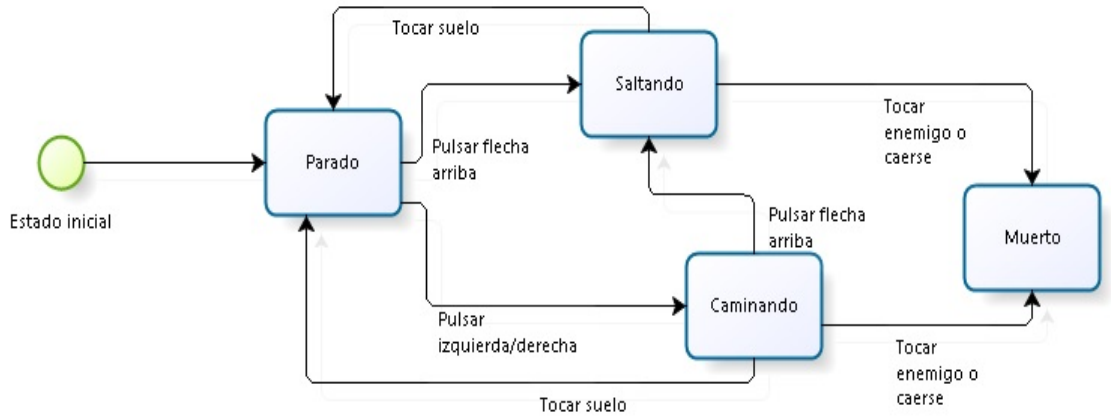

Figura 6.1: Representación de la máquina de estados del objeto Mario.

Cuando el jugador inicia la partida permanece parado hasta pulsar una de las direcciones del teclado. Si pulsa la flecha hacia arriba, el objeto Mario cambiará su animación por la de saltar y se elevará en la pantalla. Este estado puede alcanzarse también si Mario está caminando y se pulsa la flecha para arriba. Para volver al estado parado será necesario que Mario toque el suelo de los objetos que hay en pantalla. Para pasar al estado caminado es necesario tocar las flechas izquierda o derecha. Dependiendo de cuál se presione Mario avanzará hacia un lado u otro. En caso de tocar algún enemigo o caerse por el precipicio Mario pasará automáticamente al estado muerto.

Además durante la aventura Mario puede coger diferentes ítems que aparecen al golpear las cajas. Estos ítems dotan a Mario de poderes. Por ejemplo, si se coge la seta, Mario se volverá más grande hasta que un enemigo le golpee, en ese momento Mario volverá al tamaño original. Sirve como vida adicional pues Mario no muere si es grande. Otro ítem es la estrella que vuelve a Mario invencible durante un tiempo limitado.

Otro aspecto importante a tener en cuenta son las colisiones. Como vimos en la sección 3.4.2, Scratch posee sus propios bloques para detectar si un objeto está en contacto con otro. En un principio se plantea utilizar estos bloques para crear todas las colisiones del videojuego. Aunque al final debido a imprecisiones en su detección no todas las colisiones se han recreado con estos bloques. En la sección 6.2 explicaré el sistema que he implementado para las colisiones frontales y traseras entre Mario y otros objetos.

6.2 Implementación del videojuego

Ahora llega el turno de utilizar los conceptos adquiridos para implementar el videojuego Super Mario Bros en Scratch. El objeto principal del videojuego es Mario a quien el jugador debe de controlar. El truco para la implementación está en que el objeto Mario no se desplaza por la pantalla sino que permanece en todo momento en el mismo lugar de la pantalla. Lo que si se desplaza es el escenario, de esta forma transmite al jugador la sensación de desplazarse cuando mueve a Mario. Así es como se implementan todos los videojuegos que hacen uso del *scroll*[1] pues el lenguaje Scratch no ofrece herramientas para desarrollar este tipo de videojuegos. Se debe de declarar una variable llamada en nuestro caso `scrollX` que aumente su valor o disminuya conforme el jugador mueva al personaje. Por ejemplo en nuestro Super Mario Bros, esta variable se inicializa a 0 y se le resta 6 cada vez que el jugador pulsa la flecha derecha o suma 6 si pulsa la flecha izquierda. Que se resta 6 significa que el escenario se mueve horizontalmente hacia la izquierda y si se suma 6 el escenario se moverá hacia el lado contrario.

El objeto que se encarga de gestionar el valor de la variable `scrollX` es Mario. Este se considera el objeto principal del videojuego. En la figura 6.2 se pueden contemplar parte de los programas de dicho objeto. El primero de ellos y más extenso es el programa principal y se encarga entre otras de sumar o restar un valor a la variable `scrollX`. El programa empieza una vez recibido el mensaje `startgame` que lo enviará el menú principal en el momento que el usuario quiera jugar. Una vez iniciado se declaran cuatro variables que serán de utilidad para la ejecución del videojuego. Estas variables son `gravity` inicializada a $-0{,}5$, `velocity` inicializada a 0, `onGround` cuyo valor vale 1 y la ya mencionada `scrollX` que empieza valiendo 0. Seguido se posiciona el objeto en pantalla apuntando hacia la derecha. Acto seguido comienza un bucle infinito dentro del mismo programa. Una vez dentro en cada iteración del bucle se actualiza la variable `marioy` obteniendo la última posición y de Mario. Esta variable es útil para el objeto sensor que lleva Mario en la cabeza, más tarde se comentará a fondo su utilidad.

El primer `if...else` se encarga de la gravedad del personaje cuando está en el aire, es decir, la velocidad de caída del personaje hasta tocar un objeto. Si la variable `onGround` es igual a 1 significa que Mario se encuentra sobre algún objeto,

[1] Se conocen como juegos de *scroll* a los videojuegos cuyo escenario se desplaza horizontalmente o verticalmente dando a conocer nuevos elementos en el escenario.

en ese momento la variable `velocity` pasa a valer 0 y la posición y del objeto no varía. Si por el contrario la variable *onGround* pasa a valer 0 significa que Mario está en el aire, momento en el que la variable `velocity` toma el valor $-0{,}5$ de `gravity`. Este valor se resta a la posición y de Mario hasta que `onGround` igual a 0. Esta variable la controla un programa paralelo en tiempo de ejecución y que controla si Mario está tocando alguno de los colores de los objetos para establecer la variable a 1, de lo contrario tomará el valor 0. El programa se encuentra a la derecha del principal en la figura 6.2.

Continuando con el programa principal, después de controlar la velocidad de caída de Mario comienzan los condicionales que se encargan de modificar la variable `scrollX`. Para los desplazamientos hacia la derecha e izquierda el código es similar, salvo las variables de entrada a los condicionales. Si se presiona la flecha derecha del teclado y el sensor derecho no percibe colisión, Mario apuntará hacia la derecha y a la variable `scrollX` se le restará 6. Si se pulsa la flecha izquierda y no se detectan colisiones en esta dirección sucede los mismo pero se suma 6 a `scrollX`. Para implementar el salto su código es algo diferente pues si que cambia su posición y al saltar. Para ello se anidan dos condicionales al principal. En uno se comprueba si Mario está tocando por la parte inferior algún objeto. Si esto se cumple entonces aparece el otro condicional indicando que su sensor superior no debe estar en contacto con otros objetos. Si ambas condiciones se cumplen, entonces la variable `onGround` pasa a vale 0 y se le asigna el valor 9 a la variable `velocity`. Dicha variable se suma a la posición y de Mario hasta que actúa la gravedad y hace que caiga el personaje. El último condicional del programa se encarga de controlar cuando Mario se cae por el precipicio, momento que se enviará el mensaje `dead`. En la figura 6.3 se encuentra el programa que se inicia la recibir dicho mensaje.

Otro programa que se ejecuta paralelamente se encarga de cambiar el disfraz del objeto. En este disfraz Mario se encuentra en la posición de salto. Mientras la variable `onGround` permanezca a 1 se mantiene el disfraz pues significa que Mario está en el aire. También controla que Mario pueda regresar al suelo al tocar superiormente un objeto simulando así un efecto de rebote. Los cuatros programas restantes de la figura 6.2 son menos importantes para el funcionamiento del videojuego. Uno de ellos se activa al recibir el mensaje `mariobig`, este se envía cuando Mario recoge una de las setas, en ese momento el personaje se hará más grande. Otro de los programas se activa al recibir el mensaje `mariosmall` y se encarga de invertir el proceso anterior. Es decir, pasar del Mario grande al Mario normal en el momento que se toque a un enemigo. De los dos restantes programas uno se encarga de esconder el objeto al recibir el mensaje `everyonehide` y el otro de parar los programas del objeto al recibir el mensaje `victory` que indica que se ha terminado el nivel.

En la figura 6.3 se encuentran los tres programas restantes del objeto principal Mario. Dos de ellos se ejecutan al recibir el mensaje `startgame` el cuál ya recibían otros programas del mismo objeto. Uno de estos se encarga de inicializar todas las variables necesarias que no tienen que ver con el movimiento en *scroll* del personaje. Además paralelamente reproduce un sonido para cuando Mario salta. Este

sonido se reproduce únicamente al inicio de cada salto. El otro programa tiene como misión cambiar entre los disfraces del objeto para simular el movimiento de Mario. Lo único que hace es cambiar de disfraz si se pulsan las flechas derecha e izquierda. Por último existe un programa que se activa al recibir el mensaje `dead`. Este debe de parar todos los programas del objeto y los sonidos para indicar al jugador que Mario ha muerto, momento en el que Mario cambia de posición y de disfraz automáticamente para finalmente aparecer el mensaje *game over* (fin del juego).

Una vez explicado el objeto Mario, llega el momento de crear el escenario. Como comenté anteriormente, Scratch no posee herramientas para desarrollar videojuegos con uso de *scroll*, por eso los escenarios deben de implementarse objeto a objeto. Es decir, cada elemento que aparece en el escenario se ha creado como un objeto en Scratch. El tamaño de la pantalla en Scratch es de 360x480. Para la implementación del escenario se va a dividir en distintas partes, cada parte tendrá el tamaño de la pantalla en Scratch. Lo realmente importante para su implementación es el ancho de pantalla. El primer trozo partirá desde 0 hasta 480, el segundo trozo de 480 hasta 960. Cada parte de la división medirá 480 de ancho que se sumará al tamaño de las partes ya recorridas del escenario; mientras cada objeto se posiciona en una de las partes en las que se divide el escenario. El escenario del videojuego lo he dividido en ocho partes. En la figura 6.4 se contempla el programa de una de las tuberías que pertenece a la segunda parte del escenario.

Esta tubería permanece escondida mientras la variable `scrollX` es mayor que -294 o menor que -768. Esto es para que no se muestre siempre el objeto en la pantalla si no permanecería siempre en la pantalla. Si el `scrollX` se encuentra dentro de ese rango entonces deberá de aparecer en la pantalla. Para simular su movimiento por la pantalla, recordamos que Mario no se movía sino que se mueven los objetos del escenario, se aplica unas operaciones algebraicas para establecer la posición x del objeto. Dicha operación consiste en sumarle a la variable `scrollX` el resultado de la anchura de la pantalla por el número de la parte que pertenece, siendo 0 para la primera parte, 1 para la segunda, y así sucesivamente con todas las partes de las que se compone el escenario. Conforme el jugador pulse la flecha derecha `scrollX` disminuye su valor en 6 haciendo que cada vez el valor de esta variable sea menor, es por eso que se le suma el tamaño máximo de la parte que pertenece el objeto. Así el objeto consigue desplazarse hacia la izquierda cuando el jugador quiere desplazarse hacia la derecha o viceversa. De esta forma parece que Mario se mueve por la pantalla, pero lo que se mueven son los objetos del escenario en función del valor que tenga la variable `scrollX`. Este proceso se repite con todos los objetos del escenario, tanto suelos como enemigos.

Explicado el desplazamiento de todos los objetos que componen el escenario, a continuación se explica el programa que poseen los enemigos (ver la figura 6.5). Esta figura pertenece a un enemigo ubicado en la segunda parte del escenario al igual que la tubería anterior. El primer `if...else` trata el movimiento del objeto al igual que el programa de la figura 6.4. Además en el programa de los enemigos se introducen dos condicionales adicionales. El primero de ellos se encarga

de cambiar el disfraz del enemigo cuando el objeto Mario lo toca. Como existen dos sensores (uno frontal y otro trasero) la única forma de tocar al enemigo es saltando sobre él. En ese momento se cambia el disfraz del enemigo para que se vea aplastado y después se para el programa. El otro condicional es para cuando Mario ha recogido la seta y se ha hecho grande. Para ello se crea la variable `mariobig` que valdrá 1 o 0 dependiendo de si se ha cogido la seta. Si Mario es alcanzado por algún enemigo mientras está con los efectos de la seta, este perderá los efectos volviendo al tamaño original pero no morirá. Poseerá una segunda oportunidad para terminar el nivel. Se envían los mensajes `dead` o `mariosmall` que activarán dos de los programas anteriormente comentados del objeto Mario.

A parte existen otros enemigos que aparecen de las tuberías, estos son conocidos como plantas carnívoras que devorarán a Mario si se encuentra con una de ellas. Para su implementación, se parte del programa de los otros enemigos salvo que estas plantas no pueden ser eliminadas por Mario. Estas plantas aparecen de repente por las tubería y permanecen estáticas hasta volver a esconderse. Para ello se utilizan dos condicionales y generador de números aleatorios, en el momento que coincida el número aleatorio coincida con el elegido por el programador se entrará al condicional. Este mostrará a la planta salir de la tubería mientras el otro condicional se encarga de esconder la planta cuando coincida con otro número.

Los siguientes objetos a explicar son los tres sensores creados para detectar las colisiones (frontal, superior y trasero). En la figura 6.6 se puede observar el programa del sensor derecho. Los tres programas poseen una estructura similar al primer `if...else` la única diferencia es la variable que actualizan. La variable `collisionRight` controla si el sensor derecho está tocando algún objeto, `collisionLeft` para las colisiones del sensor izquierdo y `collisionUp` para las superiores. Todas obtendrán el valor 1 si se está en contacto con otros objetos o por el contrario 0. Estas variables se pueden observar en los condicionales para el incremento o decremento de la variable `scrollX` (véase en la figura 6.2). La idea de crear tres objetos para detectar las colisiones proviene del *scratcher* "khgamer6" en cuyo proyecto utiliza dos objetos para su detección de colisiones. Si bien Scratch ofrece bloques para facilitar la detección de choques, carece de precisión al utilizarse imágenes pequeñas, por esa razón he optado por crear tres objetos nuevos para realizar la función.

¿Y qué pasa cuando Mario salta y se eleva por la pantalla? Para eso se utiliza la variable `marioy`. Como ya se vio esta variable contiene la posición y del objeto Mario en todo momento. Dicha posición la reciben los tres sensores y hará que acompañen a Mario en todo momento independientemente de la posición y de él mismo. Por último para ocultar estos tres objetos, se utiliza el efecto desvanecer (`ghost`), el cual dota al objeto con transparencia haciéndolo imperceptible al ojo humano.

El último que queda por comentar es el objeto seta. Se trata de un objeto que aparece al golpear la parte inferior las cajas con el interrogante. En el momento que Mario golpea una de estas cajas el mismo objeto cambia de disfraz y emite un mensaje que activa la aparición de la seta encima de la caja. Una vez Mario

recoja el ítem aumentará su volumen y tendrá una vida más. Para ello el objeto seta envía el mensaje `mariobig` que activa el programa ya visto en la figura 6.2. En el momento que se recoge el ítem se para la ejecución del programa pues ya no debe de volver a aparecer la seta.

6.3 Organización del menú principal

Como ya ocurrió con otros dos videojuegos de este trabajo, se debe crear un menú principal para facilitar el acceso al jugador. El menú principal se crea una vez finalizado el videojuego y permite al jugador escoger entre tres opciones que son jugar, instrucciones y créditos. La metodología empleada para su creación ha sido la misma que en los videojuegos anteriores. Todas las opciones se han creado como un objeto único. Estos objetos interactúan con los programas del escenario. Estos programas se encargan de cambiar el fondo del menú. Por ejemplo si el jugador desea conocer las instrucciones para jugar al videojuego debe seleccionar instrucciones en el menú principal. Gracias al efecto visual `pixelate` aplicado en cada uno de los tres objetos el jugador puede conocer que opción está seleccionando. Al hacer clic en instrucciones, este objeto emite un mensaje llamado `tutorial`. Una vez recibido por el fondo, este cambiará la imagen que corresponde a las instrucciones y los otros objetos que estaban en el menú principal se esconderán hasta volver atrás. En el momento que el jugador seleccione la opción jugar se enviará el mensaje `startgame` que iniciará los programas principales de los objetos.

Con el menú principal se pretende familiarizar al jugador con el videojuego, es por ello que se incluye un apartado explicando los controles del videojuego y su objetivo principal. También agradecer a todos los colaboradores externos al proyecto que han servido de ayuda durante el desarrollo del videojuego. A diferencia de los anteriores proyectos, para Super Mario Bros solo se ha implementado una única dificultad para jugarlo y no tres como en los anteriores videojuegos. En la figura 6.7 se puede observar el resultado final del menú principal para el videojuego Super Mario Bros.

6.4 Comentarios finales

El videojuego no ha sido recreado fielmente al original, debido a las dificultades de Scratch para implementar videojuegos que hacen uso de *scroll*, por ello se han obviado algunas cosas del Super Mario Bros original. El objetivo con este videojuego era demostrar que con un lenguaje que no da soporte a este tipo de videojuegos se podía conseguir un producto parecido pero no igual que el original. Super Mario Bros tiene contenido que no ha sido implementado pero podría añadirse en un futuro. En esta versión tan solo otorga poder la seta, pero en el videojuego original existen otros dos ítems que otorgan a Mario poderes, estos son una estrella y una flor. La estrella proporciona inmunerabilidad durante un

determinado tiempo y la flor permite a Mario disparar bolas de fuego. A parte el jugador posee un número de vidas y puede recoger monedas para aumentar dicho número. Esto se podría haber implementado, pero al crear sólo un nivel he preferido que el jugador posea una única vida para terminar el nivel. Esto añade una dificultad extra al videojuego. Si se deseara que el jugador tuviera un número de vidas bastaría con crear una variable con el número de vidas y restarle uno cada vez que Mario muera. Una vez no queden vidas sería el final del juego.

Como comenté en el capítulo dos, el Super Mario Bros original permitía que otro jugador se uniese a la partida con Luigi. Esto sería imposible implementarlo en Scratch pues los dos personajes no deben moverse, lo que se mueve es el escenario. Al haber dos personajes no es lo mismo que con uno ya que ambos permanecen estáticos en la pantalla y el escenario debe moverse cuando uno de ellos avance por lo estropea la experiencia jugable del personaje que no se mueve.

Durante el desarrollo del videojuego han ido surgiendo distintos problemas, el principal ha sido el tratamiento de las colisiones. En un principio traté de implementarlas con los bloques que ofrece Scratch, pero debido al tamaño de los objetos daba problemas de precisión. Por ello gracias a la comunidad de Scratch se me ocurrió crear tres objetos para detectar las colisiones, uno a la derecha de Mario, otro a la izquierda y uno superior para detectar el contacto con otros objetos al saltar. Otro problema ha surgido a la hora de implementar el movimiento horizontal de los enemigos. Este problema no he conseguido solucionarlo pues si bien conseguía que el enemigo se desplazara horizontalmente, este no acompañaba al desplazamiento del escenario consiguiendo que el objeto no tuviera un comportamiento adecuado. Finalmente no he conseguido solucionar este problema y he optado por dejar estáticos a los enemigos.

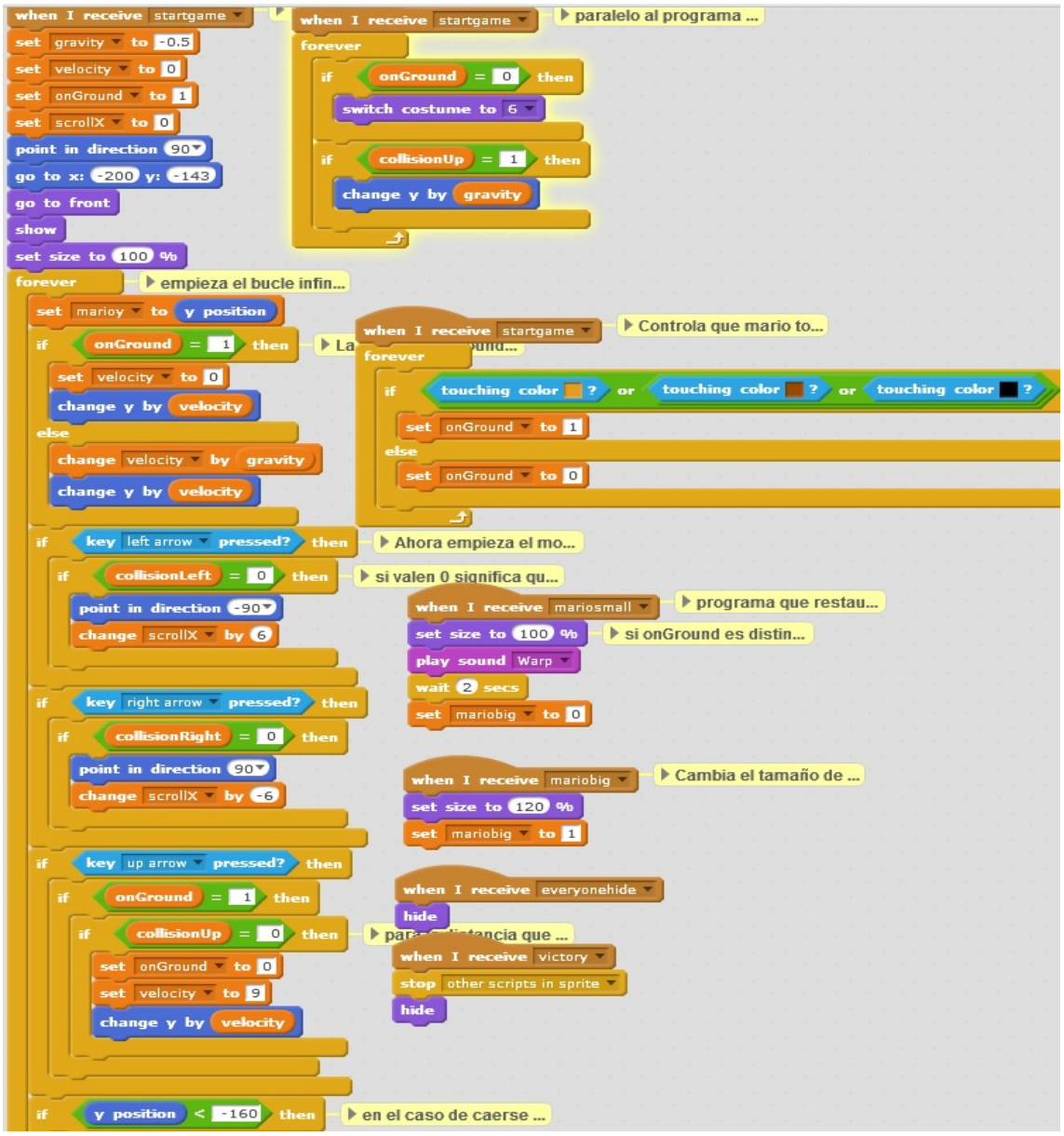

Figura 6.2: Captura de la primera parte de los programas que tiene el objeto principal Mario en el videojuego.

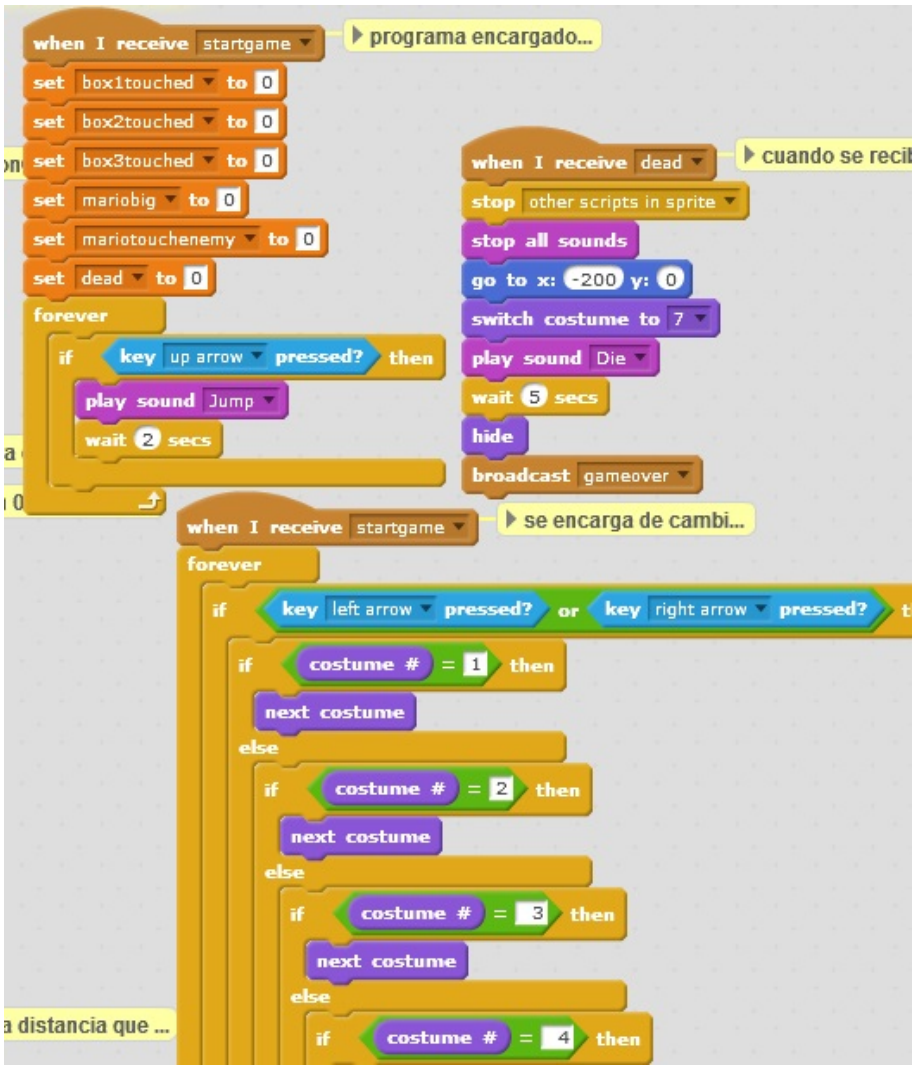

Figura 6.3: Captura de la segunda parte de los programas que tiene el objeto principal Mario en el videojuego.

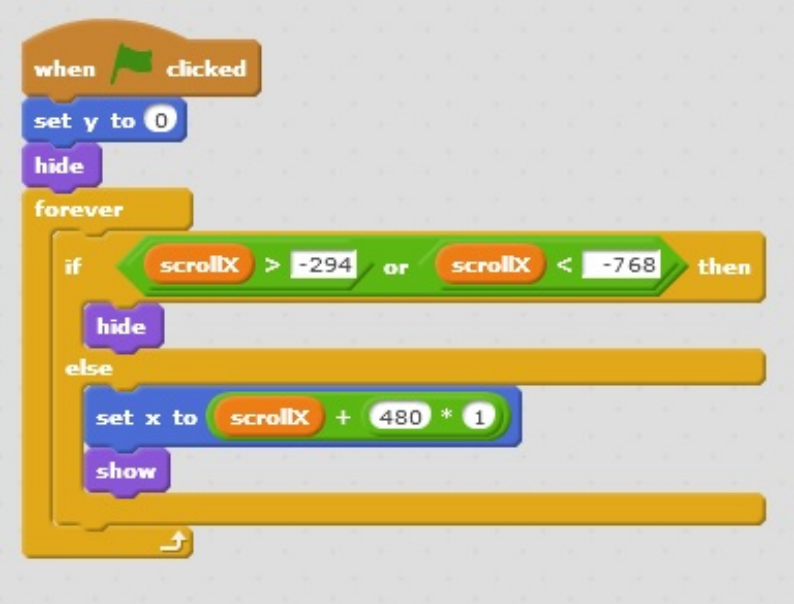

Figura 6.4: Captura del único programa perteneciente a uno de los objetos tuberías.

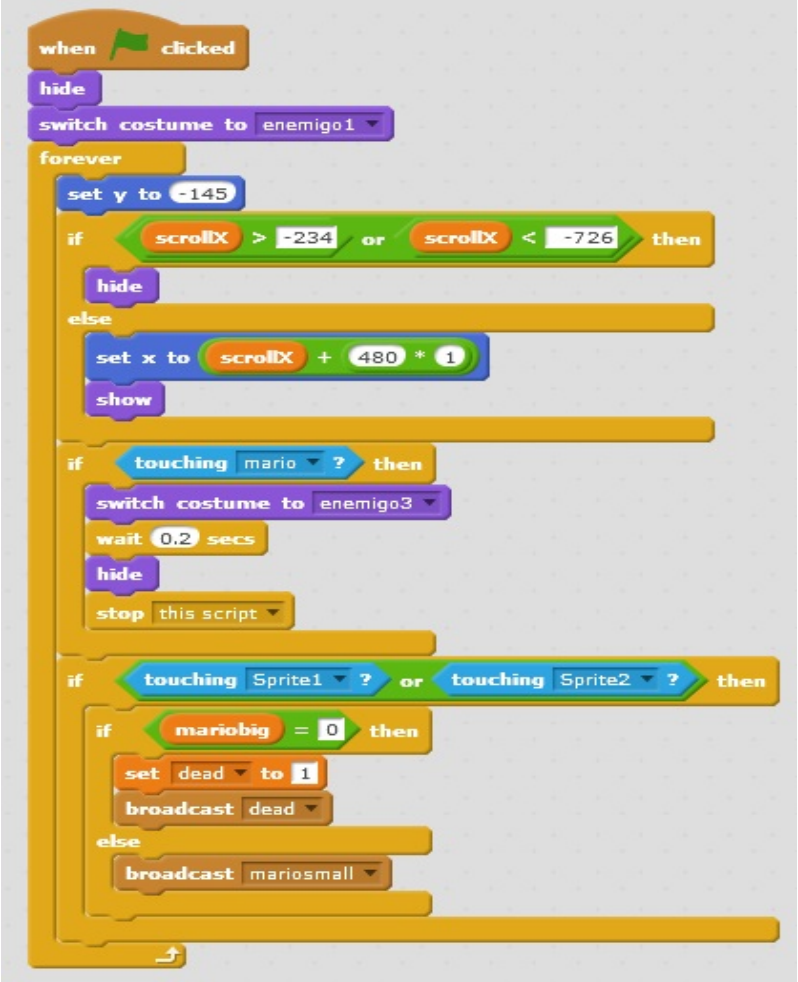

Figura 6.5: Captura del programa principal perteneciente a uno de los enemigos.

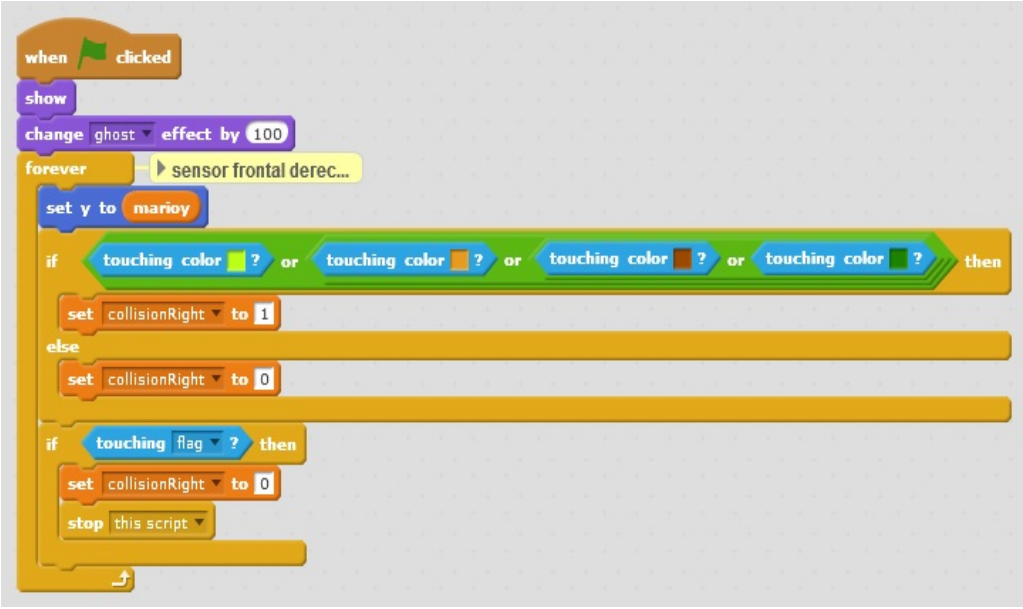

Figura 6.6: Captura del programa perteneciente al objeto que hace de sensor frontal.

Figura 6.7: Menú principal de la versión implementada del videojuego Super Mario Bros.

CAPÍTULO 7
Diseño de la página web

El presente capítulo se centra en la elaboración de una página web para el museo de la escuela. En ella se incluirán los tres videojuegos implementados en este trabajo una vez compartidos con la comunidad de Scratch. Todos los videojuegos se embeberán en el código HTML y se adjuntará un enlace a la página web de Scratch. Los usuarios podrán acceder a los videojuegos y reinventarlos si lo desean.

7.1 Implementación

Como ya se vio en el primer capítulo el objetivo de este trabajo era la creación de tres grandes videojuegos de la historia mediante el lenguaje de programación Scratch. Una vez terminados, tendrían que compartirse en la página web del Museo de Informática de la Escuela Superior de Ingeniería Informática para que los visitantes pudieran disfrutar de ellos jugando y motivarlos para empezar a programar con Scratch.

Para poder crear la página web se ha pedido acceso para editar la página web del museo. La página web del museo está administrada mediante la platadorma WordPress. Una vez obtenido el acceso se ha creado una nueva página dentro de la web del museo. Para la creación de la página se he partido de la web creada por Samuel Villaescusa Vinader para el museo el año pasado. Por ello para la implementación se ha copiado el código HTML de la página creada por Samuel y se ha cambiado el contenido por el de los videojuegos que componen este trabajo. En la figura 7.1 se observa la interfaz gráfica que proporciona Wordpress para la edición de la web.

7.2 Organización de la web

Al comienzo de la página, se incluye un título que da nombre a la página web. Al ser la continuación de la web creada por Samuel se ha optado por el mismo título que la suya pero indicando que es la segunda parte. El título queda

de la siguiente forma"videojuegos clásicos con Scratch (II)". Seguido se incluye el nombre del encargado en adapatar los tres videojuegos con Scratch. A continuación se incluye una breve introducción a la que sigue el primer videojuego (Frogger). De cada videojuego se explica un poco de su historia para poner en contexto a todos los usuarios que visiten la página. Acto seguido se muestra el objetivo del videojuego para que los jugadores tengan claro cuál es su misión dentro del videojuego. Por último antes del videojuego se indican las instrucciones básicas para que el usuario no tenga problemas dentro del menú principal. Finalmente se incluye el videojuego alojado en la página web de Scratch. El videojuego se incluye en dos formas. En la primera de ella se encuentra embebido en el código HTML lo que permite a los usuarios jugar desde la web del museo. Para embeberlo en el código Scratch ofrece una opción dentro de cada proyecto que incluye un código. Este código tendrá que copiarse y pegarse en el código HTML. La otra forma es jugarlo desde la web de Scratch, para ello se muestra un enlace al videojuego dentro de la comunidad.

En el caso del videojuego Super Mario Bros, se incluyen una notas adicionales pues debido a las limitaciones del lenguaje Scratch no ha sido posible adaptar fielmente el videojuego. Por último al final de la pantalla se vuelve a incluir el nombre del creador de los tres videojuegos junto con el nombre del profesor que ha tutelado el proyecto. Si se observa la figura 7.2 vemos el resultado final de la página web. Para acceder a la página web visitar el siguiente enlace `http://museo.inf.upv.es/es/juegosscratch2/`.

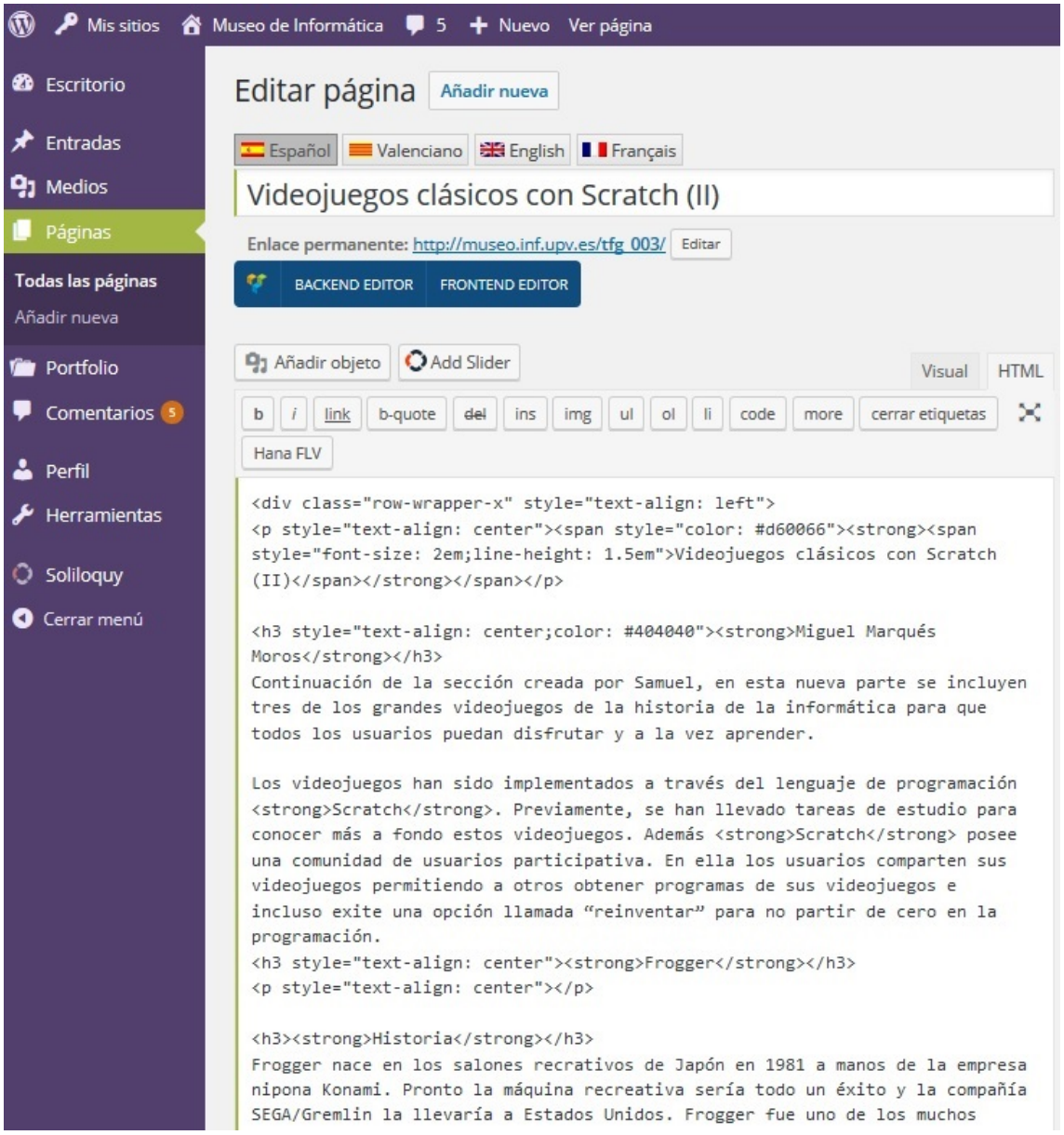

Figura 7.1: Interfaz gráfica que proporciona la herramienta Wordpress para la edición de la página web.

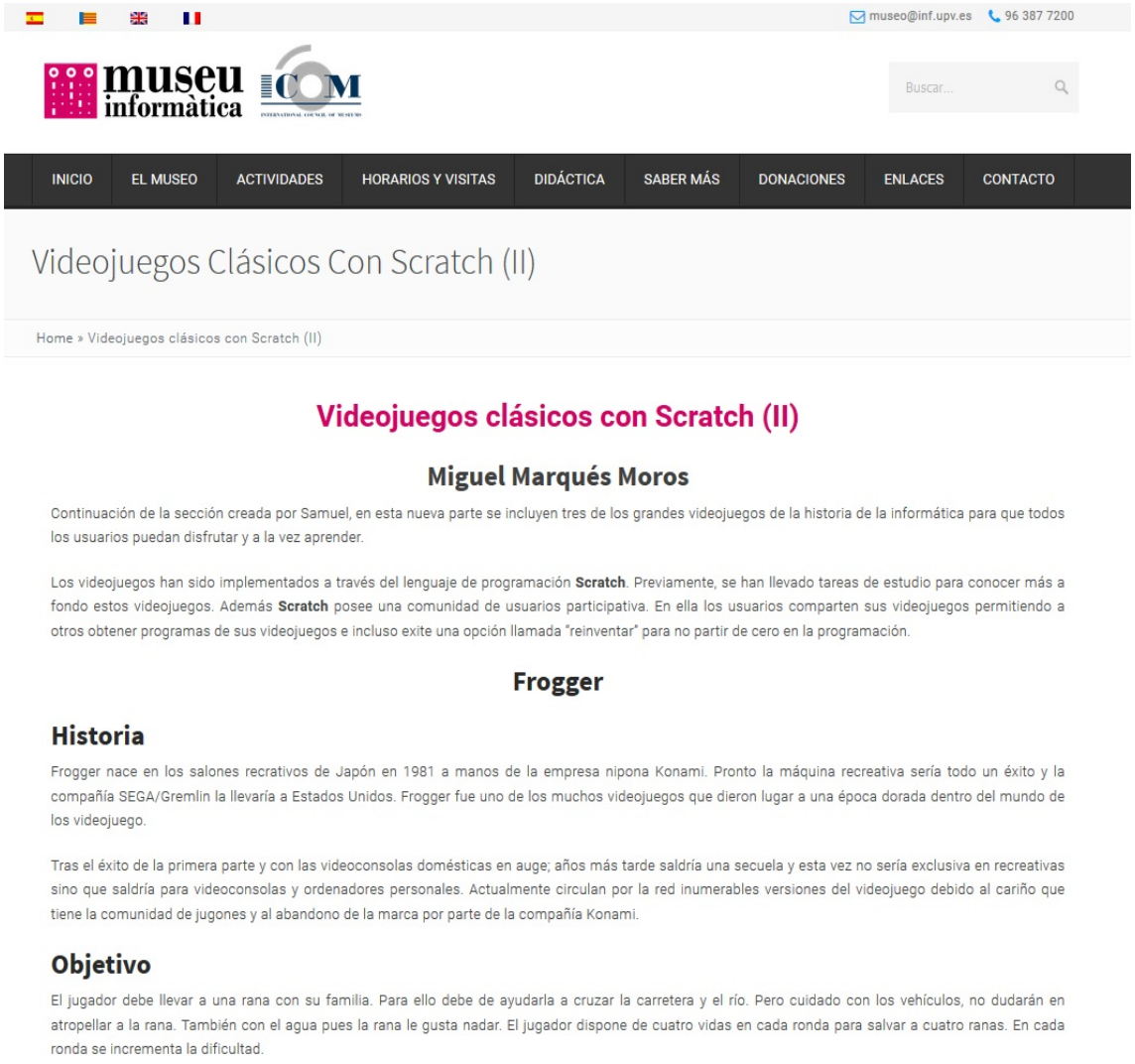

Figura 7.2: Ejemplo de la página web del museo en la que se incluye una breve introducción y el objetivo e historia de uno de los videojuegos implementados con Scratch.

CAPÍTULO 8
Conclusiones

Último capítulo de la memoria, en él se plantean unas consideraciones finales obtenidas a partir del trabajo realizado durante el presente curso. Se realiza un resumen final y se analizan los objetivos alcanzados. Asimismo se incluyen líneas de trabajo futuro.

8.1 Consideraciones finales

En el presente trabajo se ha demostrado que con un lenguaje de programación orientado al público infantil como es Scratch, pueden recrearse sencillamente videojuegos que en su época supusieron todo un reto tecnológico para las empresas desarrolladoras. Para ello se han utilizado una serie de artículos y libros, los cuales pueden consultarse en la bibliografía al final del trabajo.

Con este trabajo se ha puesto especial énfasis en el lenguaje Scratch pues a día de hoy supone todo un avance tecnológico. Como ya se explicó en el capítulo tres, éste introduce a usuarios noveles en el mundo de la programación. Para ello ofrece un lenguaje intuitivo basado en la construcción de bloques (como si de bloques de LEGO se tratase) que no requiere escribir líneas de código como en otros lenguajes de programación convencionales. Esto elimina la fase de aprendizaje de la escritura de código, dando lugar a un simple rompecabezas en el que el usuario tan solo debe escoger unos bloques y montarlos, de esta forma se genera un fragmento de código totalmente funcional. Estos fragmentos de código pertenecen a los objetos y se encargan de manejarlos durante el tiempo de ejecución, añadiendo así funcionalidad a los objetos. Si bien facilita la programación con respecto a otros lenguajes, puede resultar un poco limitada para recrear cierto tipo de videojuegos en comparación con otros. Como se vio en el capítulo seis, los videojuegos que hacen uso de *scroll* como es el caso de Super Mario Bros, se ven afectados por estas limitaciones. Cosa que con un lenguaje de programación como Java estas limitaciones desaparecerían, aunque el desarrollo sería más complicado.

Lo primero que se ha realizado en el trabajo ha sido una aproximación histórica en cada uno de los videojuegos desarrollados (Frogger, Duck Hunt y Super Mario Bros) para que el lector comprenda lo importante que fueron en su época

y cómo han influido en la industria del videojuego que actualmente conocemos. Seguido se explica el lenguaje Scratch, desde sus orígenes hasta las herramientas que ofrece para la programación. Después se describe la metodología de desarrollo llevada a cabo para todos los videojuegos que componen este trabajo. A continuación se pasa a explicar detalladamente los procesos de desarrollo de los tres videojuegos ya finalizados, haciendo especial hincapié en el diseño previo y su implementación con Scratch.

Tras terminar el desarrollo de los videojuegos, se dedica un capítulo a la elaboración de una página web que permanecerá en la web oficial del Museo de Informática de la Escuela Técnica Superior de Ingeniería Informática. De esta forma todos los visitantes de la página pueden probar los videojuegos además de conocer un poco acerca de su historia. Los videojuegos están alojados en la comunidad de Scratch y se adjunta un enlace para poder reinventar los videojuegos si se desea. Con esto se pretende despertar el interés a todo tipo de usuarios y se animen a realizar modificaciones mientras disfrutan de los videojuegos.

8.2 Trabajo futuro

Para finalizar, al igual que este trabajo se considera una continuación del proyecto de Samuel Villaescusa Vinader (se encuentra en [18]) planteo que se sigan desarrollando nuevas adaptaciones en Scratch de videojuegos clásicos que han influido en la historia. Por ejemplo juegos como Tetris, Bouble Bouble, Gauntlet, The Legend of Zelda son solo algunos clásicos que podrían adaptarse en futuros trabajos. Aunque existen muchos otros videojuegos que pertenecieron a la edad de oro de los videojuegos durante la década de los años 80. Si se prefiere, se pueden ampliar los videojuegos desarrollados en este trabajo. Al final de sus respectivos capítulos, en la sección llamada comentarios finales, se explican posibles ampliaciones que se podrían realizar partiendo por supuesto del videojuego ya implementado. Estas son algunas de las ideas planteadas por si alguien se anima a seguir desarrollando videojuegos clásicos en Scratch.

Si el alumno optara por continuar desarrollando videojuegos clásicos, estos no tendrían porqué ser algunos de los cuatro videojuegos mencionados. Estos cuatro ejemplos son sólo algunas de las joyas lanzadas durante la década de los 80. Existen grandes joyas en la historia del videojuego por lo que hay un amplio abanico de videojuegos que escoger.

Bibliografía

[1] DEMARIA, RUSEL. WILSON, JOHNNY L. WILSON (2004). *High score.* McGraw-Hill, Osborne.

[2] GALLEGO, ANDRÉS (2011). *Así funcionaba la pistola Nintendo Zapper de NES.* España, Vida extra. http://www.vidaextra.com/juegos-retro/asi-funcionaba-la-pistola-nes-zapper [Consulta: febrero 2016.]

[3] GARRIDO, SERGIO (2015). *Scratch par niños... y no tan niños.* Francia: Amazon Media EI S.à.r.l.

[4] GUTIÉRREZ OROZCO, JORGE A. (2007). *Máquinas de estados finitos.* Ciudad de México: Escuela Superior de Cómputo.

[5] HARRIS, BLAKE J. (2014). *Console wars.* United Kingdom: HarpperCollins.

[6] KENT, STEVE L. (2001). *The ultimate history of video games.* USA: Three Rivers Press, New York.

[7] LEIVA, CARLOS (2012). *Naughty Dog creó a Crash Bandicoot para que fuera la mascota de PlayStation.* España, Vandal. http://www.vandal.net/noticia/69800/naughty-dog-creo-a-crash-bandicoot-para-que-fuera-la-mascota-de-\playstation/ [Consulta: marzo 2016.]

[8] LÓPEZ BARRINAGE, BORJA (2010). *JUEGO Historia, teoría y práctica del diseño conceptual de videojuegos.* España: Alesia (Games and Studies).

[9] LÓPEZ GARCÍA, J.C. (2013). *Guía de referencia de Scratch 2.0.* Colombia, Eduteka. http://www.eduteka.org/pdfdir/ScratchGuiaReferencia.pdf [Consulta: marzo 2016.]

[10] MCMANUS, SEAN (2013). *Scratch programming in easy steps.* United Kingdom: East Steps Limited.

[11] MOTT, TONY (2010). *1001 video games you must play before you die.* Cassell Illustrated.

[12] POLLOCK, WILLIAM (2010). *Super Scratch programming adventure!* Canadá: The Lead Project.

[13] RABIN, STEVE (2010). *Introduction to game development*. Boston: Course Technology, a part of Cengage Learning.

[14] RESNICK, MITCHEL (2009). *Scratch: Programming for all*. Communications of the ACM, 52(11):60-67.

[15] RESNICK, MITCHEL (2013). *Lifelong Kindergarten.* Cultures of Creativity, LEGO Foundation. http://web.media.mit.edu/~mres/papers/CulturesCreativityEssay.pdf [Consulta: marzo 2016.]

[16] STEINBERG, SCOTT (2007). *Videogame marketing and PR.* Power Play Publishing.

[17] VAN PUL, SERGIO. CHIANG, JESSICA (2014). *Scratch 2.0 game development*. United Kingdom: Pack Publishing Ltd.

[18] VILLAESCUSA VINADER, SAMUEL (2015). *Diseño de videojuegos clásicos en Scratch.* Trabajo final de grado, Escuela Técnica Superior de Informática, Universidad Politécnica de Valencia.

[19] WING, JANNETTE MARIE (2009). *Computational thinking. Viewpoint.* Communications of the ACM, 49(3):33-35.