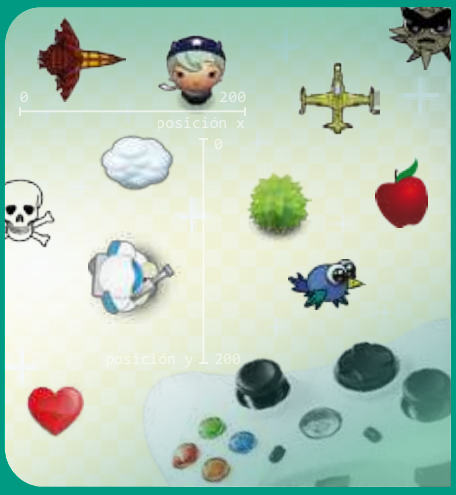


¡QUIERO HACER UN VIDEOJUEGO!

Un e-book para niños y niñas de 8 a 99 años

Alejandro A. Iglesias

unipe: editorial universitaria

HERRAMIENTAS
SERIE TIC

¡Quiero hacer un videojuego!

Un e-book para niños y niñas de 8 a 99 años

¡Quiero hacer un videojuego!

Un e-book para niños y niñas de 8 a 99 años

Alejandro A. Iglesias

Prólogo
Fernando Bordignon

unipe: editorial universitaria | HERRAMIENTAS SERIE TIC

Iglesias, Alejandro Adrián
¡Quiero hacer un videojuego! : un e-book para niños y niñas de 8 a 99 años / Alejandro Adrián Iglesias ; prólogo de Fernando Bordignon. - 1a ed . - Gonnet : UNIPE: Editorial Universitaria, 2016.
Libro digital, PDF - (Herramientas . TIC)

Archivo Digital: descarga y online
ISBN 978-987-3805-20-2

1. Diseño de Videojuegos. 2. Alfabetización en Información. I. Bordignon, Fernando, prolog. II. Título.

CDD 794.8

UNIPE: UNIVERSIDAD PEDAGÓGICA
Adrián Cannellotto
Rector

Carlos G. A. Rodríguez
Vicerrector

UNIPE: EDITORIAL UNIVERSITARIA
Directora editorial
María Teresa D'Meza

Editor
Juan Manuel Bordón

Diagramación
Diana Cricelli

Imagen de tapa
Basada en *Planet Cute*, de Daniel Cook

Camino Centenario nº 2565 - (B1897AVA) Gonnet
Provincia de Buenos Aires, Argentina
www.unipe.edu.ar

Esta edición se publicó en el mes de diciembre de 2016.

ISBN: 978-987-3805-20-2

Índice

Prólogo

Fernando Bordignon
(Director de LabTIC, UNIPE)

El juego es una de las actividades preferidas y más intensamente practicadas por niños y adolescentes. Jugar es un acto que surge de la propia voluntad del participante, no es impuesto por nadie y se configura bajo su libertad. Es muy distinto del trabajo, que en general se basa en una obligación impuesta de manera externa. Esto nos lleva a pensar que jugamos por el placer mismo de hacerlo, para sumergirnos en mundos de aventuras, para explorar escenarios, para lograr vencer numerosos retos parciales en busca de un placentero final, o para compartir con otros esa sensación de felicidad.

Desde que en 1972 se presentó *Pong*, el primer videojuego para computadora comercial, se ha desarrollado una novedosa y enorme industria del entretenimiento. Ciertamente, es una industria con una lógica propia dada por sus modelos y lenguajes de desarrollo; por los modos de publicitar productos, consolas o computadoras de juego; por las diversas interfaces para jugar y hasta por las comunidades de *fans*, entre muchos otros elementos que componen este universo. Por otra parte, el hecho de que los videojuegos digitales se hayan popularizado y masificado ha dejado una marca vivencial fuerte en aquellos que naturalmente nacieron y vivieron de la mano de esos juegos.

Para algunos estudiosos del tema, estos artefactos tecnológicos, que surgieron como parte de una industria del entretenimiento, están modificando nuestras formas de aprender y producir conocimiento (Gros Salvat, 2008: 10). Así mismo, ya existe evidencia de una correlación entre el uso de simulaciones, aplicaciones interactivas o juegos, y la mejora de los logros académicos (Zielezinski y Darling-Hammond, 2014).

El jugar sucede temprano en la vida, dado que los niños entran en contacto con los videojuegos –ya sea a través de tabletas, *notebooks* o *smartphones*– desde muy pequeños, como una forma de ingreso al mundo digital que los rodea. En efecto, los videojuegos se han convertido en el primer elemento de alfabetización digital: es allí donde, más allá de aprender los rudimentos del manejo de la computadora, también aprenden a extender su experiencia de vida en mundos virtuales que exploran solos o acompañados por pares.

Los juegos digitales han evolucionado de modo muy marcado, al margen de la forma de sus interfaces y comandos, como espacios donde se produce la interactividad. También han crecido en su modelo lógico y en la experiencia de inmersión que proponen. En una primera etapa, estaban muy cercanos al conductismo, dado que las tareas del jugador se focalizaban en su conducta, promoviendo la práctica básica de algo, con alguna acción de retroalimentación. La siguiente etapa estuvo marcada por el desarrollo de la interactividad,

con juegos mucho más ricos en el uso de multimedia, incluidas algunas ayudas extras que permitían el avance del jugador así como su inmersión en el juego. La tercera etapa tuvo como eje de desarrollo la participación. Con el avance en las tecnologías de conexión –ya sea en una red local o con acceso a internet– se pudieron poner en práctica nuevos modelos de juegos colectivos donde grupos de jugadores compartían un espacio lúdico y colaboraban o competían entre ellos.

Los videojuegos están empezando a ser valorados de forma positiva como recursos educativos gracias a que tienen una serie de características que propician los procesos de generación de nuevos conocimientos y habilidades: presentan información de manera atractiva, promueven situaciones de análisis de casos donde se requieren decisiones e intervenciones rápidas, y habilitan espacios de concentración para resolverlas.

Hoy, toda un área de investigación en esta temática está tomando impulso y generando las primeras experiencias *ad hoc*. La categoría *serious games* (juegos formativos) habilita nuevas experiencias de inmersión donde el aprendizaje es una experiencia diseñada mediante el uso de mecánicas de juego y del pensamiento lúdico. Aquí, el objetivo es que al estudiante, en su rol de jugador, se lo pueda ayudar a trabajar de manera adicional con contenidos relacionados con un tema específico.

Por eso, jugar está dejando de ser una experiencia pasiva, predeterminada únicamente por la mano de los diseñadores del juego. Ya se han liberado algunos aspectos del diseño a los propios jugadores, brindándoles la posibilidad de ir mucho más allá de la configuración de escenarios, objetos o personajes: se les ofrecen herramientas para terminar de construir o expandir un juego, e incluso para crear sus propios videojuegos desde cero.

Esta posibilidad de expresión a través de la creación de nuevos elementos digitales (nuevas narrativas, escenarios, personajes o metas) implica que el jugador necesariamente desarrolle habilidades superiores del pensamiento en su rol como diseñador y programador. Desde el arte, los jugadores se involucran en seleccionar, adaptar o incluso diseñar nuevos fondos, objetos interactivos pertenecientes a sus propuestas de micromundos, así como los personajes que comandarán. Desde la narrativa, actúan como guionistas y directores de una obra, ya que les dan sentido y pautas de relación (a partir de una interactividad definida por ellos) a todos los elementos de la historia que tratan de construir. Y desde la programación, aprenden a comunicarse con la máquina, mediante un lenguaje de órdenes, para que esta maneje a sus creaciones de la forma en que ellos mismos las pensaron.

Un ejemplo de cómo en la actualidad los juegos intentan desarrollar saberes en las personas, se ve en la campaña estadounidense denominada “La Hora del Código” (https://code.org/), que busca promover el pensamiento computacional en niños y adolescentes de todo el mundo (así como en sus maestros y profesores). Este proyecto ha adoptado el uso intensivo de juegos y personajes populares, así como elementos de programación, como estrategia didáctica en la enseñanza de la resolución de problemas.

Estamos atravesando un estadio de la sociedad en el cual el pensar de forma creativa se ha convertido en un requisito indispensable para el desarrollo. Casi todo cambia tan rápido que los estudiantes tienen que encontrar la forma de dar soluciones innovadoras a problemas desconocidos. En este sentido, el profesor Mitchel Resnick, del MIT Media Lab, indica que los niños y adolescentes manejan la tecnología con fluidez, pero en general no saben cómo crear o expresarse con ella. De alguna manera, es como si supieran leer pero no escribir. Por eso, el desarrollo del pensamiento computacional apunta a empoderarlos, a que puedan expresarse de una forma más completa y compleja. Estos saberes los ayudan a desarrollar habilidades cognitivas superiores que refuerzan su desarrollo intelectual, más allá de que en su futuro laboral esos estudiantes vayan a ser escritores, médicos, psicólogos o informáticos.

Tenemos por delante un futuro prometedor, pero solo si somos capaces de dejar atrás la etapa de admiración por las herramientas y los objetos tecnológicos para pasar a una etapa superior, donde podamos utilizarlos para ampliar nuestra capacidad para expresarnos y para construir conocimientos, inmersos en nuevas comunidades que innoven y compartan sus saberes.

La idea de que haya adultos creando junto a los jóvenes, actuando como mentores y canalizando de la mejor forma posible toda su energía de exploración, es el objetivo de esta obra de Alejandro Iglesias, docente de Tecnologías de la Información y la Comunicación (TIC) de la UNIPE. Con ¡Quiero hacer un videojuego! Un e-*book para niños y niñas de 8 a 99 años* buscamos establecer un puente entre generaciones que propicie el diálogo basado en el hacer, así como la creatividad de los jóvenes. A partir de este diálogo, seguramente podremos entendernos mejor, aprender juntos y llevar adelante nuevos proyectos.

Por eso, quiero invitar a leer, disfrutar y experimentar con esta obra, que es una manera de introducirnos en un nuevo mundo posible donde la creación y la expresión genuina utilizan las pantallas digitales. Su redacción es amena, de una lectura entretenida, con aportes prácticos que acercan a una dimensión real del mundo de los videojuegos. En su segunda parte, el libro invita a tener una experiencia, en primera persona, en la creación de un videojuego. Esperamos que esta sea un disparador de muchos más proyectos.

REFERENCIAS BIBLIOGRÁFICAS

Gros Salvat, Begoña
2008 "Juegos digitales y aprendizaje: fronteras y limitaciones", en íd. (coord.), *Videojuegos y aprendizaje*, Barcelona, Graó.

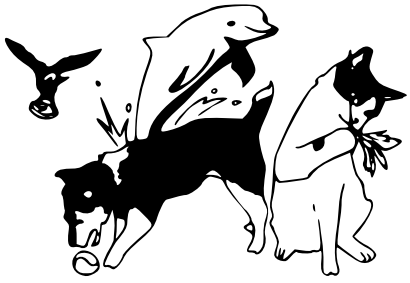
Zielezinski, Molly y Darling-Hammond, Linda
2014 *Technology for Learning: Underserved, Under-resourced & Underprepared Students*, Stanford, Stanford Center for Opportunity Policy in Education.

Introducción al mundo de los videojuegos

En este primer capítulo se van a presentar algunas ideas y conceptos sobre cómo funcionan los videojuegos, de qué están hechos y quiénes los crean. Si quieres empezar directamente con la parte práctica y ponerte manos a la obra, puedes saltearte este capítulo y volver luego, ¡no hay ningún problema! Este libro está escrito de forma tal que puedas acceder a la información en el momento en que sea necesaria utilizando hipervínculos como estos.

1.1. JUGUETES, JUEGOS Y VIDEOJUEGOS: TODOS JUGAMOS

Para empezar a hablar sobre videojuegos es importante entender primero qué es un juego, y para ello vamos a empezar con una simple pregunta ¿Cuál es el juego más viejo que se te viene a la mente? Y cuando hablamos de juego, piensa también en aquellos juegos que no necesitan de botones ni de pantallas.



¿Has pensado en algo como las damas o el ajedrez? ¿Se te ha ocurrido algo más viejo aún, como por ejemplo la escondida? Bien, aquí viene la primera sorpresa: existen juegos mucho más viejos que esos. El juego o el acto de jugar no es un invento nuestro, de hecho es más viejo que los seres humanos. Muchos animales juegan, por ejemplo nuestras mascotas, perros y gatos. Incluso animales más extravagantes como delfines y cuervos también lo hacen. Y ya que he mencionado a los cuervos, ¿alguna vez has visto a uno haciendo snowboard? ¿No? Bueno, no hace falta que me creas, basta con que visites este video en youtube: https://www.youtube.com/watch?v=6uXiAe7Oc-I.

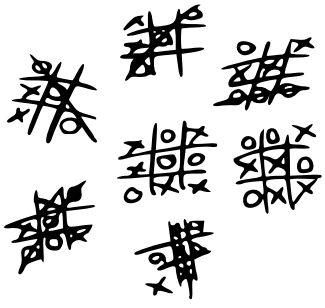
En este otro video puedes ver a un perro jugando a atrapar la pelota él solo: http://www.youtube.com/watch?v=vNvjtyUh2OM.

Pero si los juegos no son un invento humano, ¿entonces qué es un juego? Un juego es una herramienta natural de aprendizaje que hemos estado usando desde que nacimos para entender cómo funciona el mundo que nos rodea. Más adelante vamos a avanzar sobre esto con unos experimentos divertidos con ilusiones ópticas, pero ahora veamos cuáles son las características que definen a una actividad como el juego:

- *Voluntario:* jugamos porque queremos hacerlo.
- *Reglado:* todo lo que hacemos en un juego está controlado por las reglas.
- *Con conflicto:* cuando jugamos estamos tratando de resolver una situación.
- *Con resultado desconocido:* no tenemos que saber si vamos a ganar o a perder.
- *Desafiante:* tiene que costar resolver el juego, sino perdemos el interés.
- *Seguro:* nada nos puede dañar cuando jugamos. Así como los cachorros de león no se lastiman al pelear cuando practican, tampoco nosotros al jugar a la escondida.
- *Con representación:* cada juego cuenta con un grado de representación de la realidad. Por ejemplo en las fichas del ajedrez podemos identificar caballos, torres y reyes.

Estas simples características están presentes en todos los juegos, ya sean videojuegos o no. ¿Has jugado al tatetí alguna vez? Seguramente que sí, y si tienes ganas y alguien con quien jugar, este puede ser un buen momento para hacer un par de partidas. Puedes ir a jugar, aquí te espero.

¿Ya has terminando? ¿Qué sucede cuando juegas muchas veces seguidas contra la misma persona? Las partidas se vuelven repetitivas, el resultado del juego comienza a tornarse predecible y es en ese mismo momento en que los jugadores comienzan a aburrirse. La característica "resultado desconocido" se ha perdido y por lo tanto también la sensación de estar jugando. Así mismo, mientras jugaban, ambos jugadores respetaban sus turnos, dibujaban solo dentro de los marcos y no hacían trampa. Estaban respetando las reglas que definían el juego. Si alguien comenzara a hacer trampa el juego perdería sentido ya que dejaría de ser reglado.



Otra característica que podemos ver en acción con esta experiencia es lo que pasa cuando uno de los participantes no tiene ganas de jugar. Si nosotros insistimos lo suficiente, el otro jugador puede llegar a participar, pero lo hará de mala gana. Esto sucedería porque... ¡dejaría de ser una actividad voluntaria!

Ahora que hemos entendido qué es un juego y cuáles son sus características principales, es momento de hacerse otra pregunta: ¿qué es lo que diferencia a un videojuego del resto de los juegos? La diferencia principal que poseen es la utilización de un dispositivo electrónico para simular las reglas de los juegos. Si el juego está bien hecho, no tenemos que hacer respetar las reglas ni tenemos que leerlas de un manual. En un videojuego, las reglas están escondidas en él y es la tarea de los jugadores descubrirlas y dominarlas.

Realiza el siguiente experimento para terminar de comprender esta diferencia:

- Juega al tatetí usando el programa Paint (en Windows) o el programa Xpaint (si estás usando Linux) para dibujar el tablero y juega con un amigo.
- Si dispones de una conexión de internet juega con esta versión de tatetí en http://www.juegate.com/game/sp/tateti.

Ahora, la pregunta tramposa: ¿cuál es un videojuego?

a) Los dos.
b) Solo la versión que jugaron al dibujarlo con el programa Paint o Xpaint.
c) Solo la versión que jugaron por internet.
d) ¡Ninguno lo es!

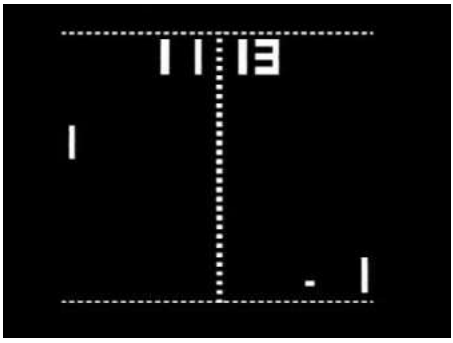
Dejando un poco de suspenso, la respuesta correcta es... ¿Ya la has pensado, no es cierto? ¡La C! El que has jugado utilizando el editor de imágenes (Paint o Xpaint) no es un videojuego porque las reglas las tuvieron que hacer respetar los mismos jugadores, mientras que en el *online* las reglas estaban embebidas en el juego, que obligaba a respetarlas.

Es difícil decir con certeza cuál fue el primer videojuego de la historia. Muchos autores coinciden en que el pionero fue William Higinbotham con *Tennis for Two*, en 1958. *Tennis for Two* nació como un experimento, buscando demostrar que se podían utilizar pantallas para realizar interfaces interactivas.

Una versión comercial del mismo juego conocida como *Pong* y creada por Atari fue uno de los primeros juegos en alcanzar gran éxito y popularidad. Por

si no lo conoces, *Pong* (lanzado en 1972) es un juego muy simple para dos participantes que consta de un par de paletas y una pelota blanca. El jugador que logra atravesar el campo del oponente con la pelota blanca obtiene un punto y el primero que logra acumular 15 puntos es el ganador. El juego en sí es una abstracción del tenis de mesa, simplificado para hacerlo más accesible a los jugadores y a la computadora que lo ejecuta. Como muestra la imagen de abajo, se juega con dos perillas, una para cada jugador.

En esta web puedes encontrar varias versiones para jugar solo o con amigos http://www.ponggame.org/.



Desde entonces, los videojuegos han recorrido un largo camino acompañando a otros avances tecnológicos. Hoy en día los encontramos en casi cualquier dispositivo capaz de mostrar gráficos y reproducir sonidos, desde celulares y *tablets* a computadoras, consolas hogareñas (como la PlayStation de Sony, la Xbox de Microsoft, y la Wii de Nintendo) y consolas portátiles (como la PlayStation Vita y la Nintendo 3DS). Lo que en un inicio fueron unos pequeños píxeles blancos rebotando en una pantalla hoy en día representa una gran industria que utiliza desde joysticks con sensores de velocidad e inclinación, hasta cámaras que permiten saber en qué posición se encuentra nuestro cuerpo.

Más allá de la tecnología que usen, las características que has leído en esta sección se aplica a todos ellos, así que ahora estás más preparado a la hora de pensar qué es un juego. Sin embargo queda mucho más por descubrir y ahora seguramente te estarás preguntando cómo está hecho un videojuego. ¡Basta con que leas la siguiente sección para encontrar la respuesta!

1.2. CÓMO ESTÁN HECHOS LOS VIDEOJUEGOS Y QUIÉNES LOS HACEN

Un videojuego no es solo un mundo ficticio creado por una computadora, también es una experiencia audiovisual interactiva con gráficos y sonidos que deslumbran al jugador. El siguiente esquema te ayudará entender cómo toma forma:

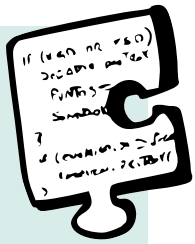
Para hacer un videojuego es necesario un equipo de desarrollo que pueden conformar hasta 50 personas o más –juegos como el *Pro Evolution Soccer* (PES) de Konami–, pero también una sola. Todo depende del tamaño del juego a desarrollar, así que no te preocupes por eso. En lo proyectos grandes, como en el mencionado *PES*, hay personas que se dedican exclusivamente a realizar la simulación del césped del campo de juego, por ejemplo, mientras que otras se enfocan en el diseño de los estadios, en la programación de las simulaciones físicas de los cuerpos de los jugadores y las pelotas, o en crear los sonidos para cada una de las acciones que suceden en el juego.

Eso sí, independientemente del tamaño del juego hay un montón de tareas que deben hacerse y es el trabajo que cubren los siguientes roles:

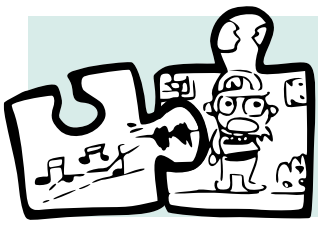
DISEÑADOR DE JUEGOS: es el encargado de crear las reglas que definen al juego, qué cosas puede hacer el jugador, qué clase de enemigos existen, cuánto daño infligen, cuál es la condición para ganar, etc. También es el responsable de hacer que el juego sea divertido y esté balanceado (ni muy fácil ni muy difícil), y de transmitir al resto del equipo cuál es el juego que intentarán hacer. Además, se encarga de desarrollar los niveles, aunque muchas veces hay personas que se dedican exclusivamente a eso y, como podrás adivinar, a ese rol se le llama diseñador de niveles.

Cuando el equipo de desarrollo es grande se debe dejar por escrito cómo es el juego en un documento que se llama “Documento de diseño del juego” o “GDD”, por sus siglas en inglés. Este documento va cambiando conforme el equipo avanza en el desarrollo, ya que no se puede saber de antemano qué cosas pueden resultar más divertidas que otras ni cómo los problemas de tiempos y presupuesto pueden impactar en el diseño del juego.

PROGRAMADOR DE JUEGOS: es el encargado de transformar todas esas locas ideas del diseñador en algo que se pueda ejecutar en una computadora. Cuando el desafío es muy grande los programadores pueden trabajar en equipo para dividirse las tareas y son coordinados por un líder de programación.



Además de embeber las reglas del juego, los programadores tienen que hacer que el juego muestre los gráficos y reproduzca los sonidos, y a la vez lograr que la computadora lo haga lo más rápido posible. En general los juegos están pensados para ejecutarse a 60 cuadros por segundo. Eso significa que el programa debe verificar las reglas del juego y las posiciones de los personajes, calcular las colisiones, mostrar los gráficos y muchas cosas más ¡sesenta veces por segundo!



ARTISTAS VISUALES, MÚSICOS Y SONIDISTAS: los videojuegos en general cuentan con una gran variedad y cantidad de recursos visuales y sonoros.

Los recursos visuales son creados por artistas que se especializan en distintos tipos de gráficos para videojuegos:

- *Artistas con experiencia en animación tradicional* para hacer animaciones cuadro a cuadro en dos dimensiones.
- *Artistas 2d* que hacen fondos para los escenarios del juego.
- *Modeladores 3d* que realizan modelos tridimensionales de los objetos del juego.
- *Artistas 2d que crean texturas* para dichos modelos.
- *Artistas conceptuales* que crean gráficos que sirven para inspirar al equipo y para definir la visión artística del juego.

La música, por su parte, es creada por músicos especializados que en general no forman parte del equipo de desarrollo sino que trabajan de manera paralela. Los especialistas en efectos de sonidos son los responsables de crear o encontrar los sonidos adecuados para los juegos. Por último quedan por describir dos roles que están presentes en todos los equipos de desarrollo.

PRODUCTOR: es el encargado de administrar el proyecto. Tiene que coordinar las tareas para lograr cumplir con el cronograma, y respetar los presupuestos.

TESTERS: son los encargados de encontrar errores y ayudar a mejorar la experiencia de juego. Un tester tiene que probar una y otra vez un mismo nivel de un juego, hasta asegurarse de que no existen errores de programación ni de diseño.

A continuación examinaremos un ejemplo para ver las interacciones de estos roles durante el desarrollo de un videojuego. ¿Conoces el juego *Plants vs. Zombies* (PopCap)? Si no lo conoces, aquí puedes jugar la versión de prueba del juego: http://www.popcap.com/plants-vs-zombies-1.

En *Plants vs. Zombies* el jugador dispone de un jardín que debe defender, con distintos tipos de plantas, del ataque de una horda de zombis. En el juego predominan la estrategia y la planificación, que serán necesarias para poder superar los niveles.

En el siguiente gráfico puedes ver cómo algunos miembros del equipo de desarrollo hacen su aporte (hipotético) a la creación del girasol, una de las plantas del juego:

Programador Creó un reloj interno para el girasol que cada x segundos crea un objeto Sol

Artista Dibujó la planta acorde a un estilo visual y la animó

Diseñador Creó la planta girasol, pensando en ella como una planta capaz de producir recursos que necesita el jugador

Tester Probó los niveles con el objeto de ayudar al diseñador a saber cada cuántos segundos debe la planta producir un Sol para que sea divertido

Si bien todas estas personas trabajan al mismo tiempo, no lo hacen de forma desordenada. Independientemente del tipo de videojuego que se esté desarrollando, todos se hacen respetando las siguientes etapas:

1. *Desarrollo del concepto:* todo comienza con una idea que se irá transformando, con tiempo y mucho esfuerzo, en un videojuego. Lo importante en esta etapa es averiguar cuál es el juego que se va a desarrollar. En el caso de que se trabaje en equipo, también es necesario escribir la idea de manera tal que, cuando otras personas la lean, sepan con qué se van a enfrentar.

2. *Preproducción:* durante esta etapa se planifica y se asignan recursos al equipo de desarrollo, para estimar tiempos y costos. Este también es el momento de probar la idea del juego construyendo lo que se conoce como prototipo. Un prototipo es básicamente una versión mucho más pequeña y fea del juego a desarrollar. Sirve, sobre todo, para saber si es posible hacer el juego y qué tan divertido resultará.

3. *Producción:* es el momento de transformar toda esa planificación e ideas en un juego, un proceso muy largo en el cual el juego va a madurar en la medida en que diseñadores, artistas y programadores le den forma. El juego que se producirá en esta etapa no es necesariamente igual al que se planificó, ya que el diseñador de juego puede ir ajustando el diseño para orientarlo hacia un juego más divertido y con más probabilidades de tener éxito.

4. *Post-producción:* una vez terminado el juego, y una vez que este ha salido al mercado, comienza la etapa de post-producción. Se intentan reparar aquellos errores que puedan haberse escapado al control de calidad, y que los jugadores descubren y reportan. Cuando los programadores son capaces de realizar estas reparaciones se crean actualizaciones que se distribuyen gratuitamente a aquellos que hayan comprado al juego. También durante la etapa de post producción se brinda a los jugadores soporte técnico para los problemas que puedan surgir y resolver las dudas de los usuarios.

En esta sección has aprendido cómo es el proceso de crear un videojuego, quienes trabajan en ello y cuáles son sus tareas. En la sección siguiente podrás descubrir un poco más acerca del mundo del diseño de juegos y, para comprenderlo, sobre cómo funciona nuestro cerebro al momento de jugar. Si te interesa saber qué es lo que hace divertido a los videojuegos, la siguiente sección te gustará. Si tus ganas de hacer un videojuego ya son incontenibles puedes pasar directamente al segundo capítulo y volver después a leer el resto.

1.3. CÓMO FUNCIONA UN VIDEOJUEGO DESDE EL PUNTO DE VISTA DE UN DISEÑADOR

Jugar un videojuego es más complejo de lo que parece: existen un montón de procesos mentales que se activan en el mismo momento en que nosotros miramos la pantalla del juego.

El diseñador de juegos Raph Koster experimentó sobre estas ideas y creó una definición más acerca de lo que es un videojuego. A lo largo de su libro *A theory of fun*, él explica que jugar juegos significa, simplemente, "aprendizaje y reconocimiento de patrones".

¿Qué es un patrón?, te preguntarás... Mira las siguientes imágenes. ¿Qué es lo primero que se te viene a la mente?

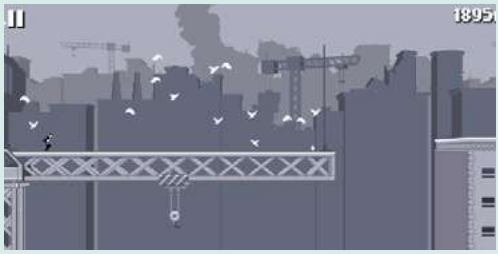
Al observar la primera imagen, no solo puedes ver un par de medidores, sino también dos simpáticas caras que te miran sonrientes, pero... ¿hay realmente dos caras allí?

Mira ahora la imagen con kiwis. Además de las tres rodajas, también parece haber allí, justo en el centro, un triángulo. Sin embargo eres consciente de que allí no hay ningún triángulo, aunque nuestro cerebro insista en decirnos que claramente hay uno.

Lo que está sucediendo es que nuestra mente está haciendo lo que mejor sabe hacer: poner en marcha el proceso de reconocimiento de patrones. Nuestro cerebro conoce las características que definen a un rostro, porque ya ha visto a muchísimos de ellos antes, y cuando encuentra dichas características sobre una nueva imagen (en el caso de la foto de los medidores, un par de círculos colocados sobre una curva) de forma inmediata identifica a esta como una cara. Lo mismo sucede en la imagen de las rodajas de kiwi. Las partes cortadas de cada rodaja nos remiten a las esquinas de una figura que conocemos bien. El cerebro, completando esta información, nos hace ver un triángulo donde, en realidad, solo hay un espacio vacío.

Te estarás preguntando, ¿esto qué tiene que ver con los videojuegos? En principio, se podría decir que TODO. Para el cerebro los videojuegos son solo un montón de patrones mezclados listos para ser descubiertos. No es necesa-

rio que me creas. Para entendero mejor, es momento de jugar a *Canabalt*, de Adam Saltsman, justo aquí: http://adamatomic.com/canabalt/.



Canabalt es un juego muy sencillo, de un solo botón, donde el jugador controla el momento en que su personaje salta y debe intentar sobrevivir en su huida el mayor tiempo posible.

Analizaremos ahora qué es lo que sucede en la mente del jugador mientras juega. Lamentablemente no dispondremos de ningún casco capaz de leer ondas cerebrales así que nos tendrá que alcanzar con la teoría de Raph Koster y con algunos conocimientos sobre neurociencia.

Lo primero que encuentra el jugador es una pantalla de *menu*. Hay dos instrucciones debajo que dicen que se presione X o C para empezar a escapar. Sin embargo es posible que el jugador no lo lea y simplemente comience a presionar al azar las teclas.

Acto seguido empieza el juego comienza y un personaje en blanco y negro comienza a correr. Si el jugador no toca ninguna tecla, el personaje cae de los edificios y pierde. Si bien parece que no ha sucedido mucho, en realidad el jugador hizo en ese simple acto un montón de conexiones con otros patrones que ya tenía en su cabeza y los está relacionando para conseguir una explicación. Veamos el siguiente gráfico:

El juego le está diciendo al jugador un montón de cosas. Al ver el personaje de perfil, el cerebro le avisa que la gravedad va a atraer al personaje hacia abajo; los robots gigantes que caminan por el fondo le cuentan una historia, que el personaje está huyendo de algún tipo de invasión; y por último, el hecho de que el personaje camine sobre los techos le dice al jugador que si se cae, no va a ser muy agradable. Hasta aquí el jugador simplemente reconoció los patrones del mundo real que se pueden encontrar en el juego, ¿pero qué pasa con los nuevos patrones que aprende a partir del juego?

El jugador descubre, a lo largo de las partidas, que existen distintos tipos de edificios y que aquellos que tienen grietas se destruyen luego de ser pisados.

Más sutilmente el jugador aprende que la velocidad del jugador aumenta todo el tiempo y descubre que para controlar la velocidad puede chocarse voluntariamente con los obstáculos como cajas y sillas.

Como el objetivo del juego es sobrevivir la mayor tiempo posible, el jugador intentará aprender todo aquello que esté relacionado con este objetivo:

- Ir muy rápido no le deja tiempo para reaccionar.
- Ir muy lento obliga a hacer saltos más cortos que pueden ocasionar una caída. Tratará de aprender a esquivar obstáculos siempre y cuando no vaya demasiado rápido.
- Saber cuál es el momento de saltar en determinados edificios, como aquellos que tienen ventanas de vidrio, es crucial para superar ciertos puntajes.

El jugador, mediante prueba y error, va a descubrir cuál es el momento justo.

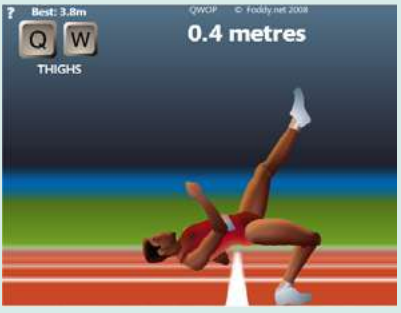
Este proceso de reconocimiento de patrones es idéntico al analizado con las imágenes de los kiwis y los rostros en medidores. El cerebro, a partir de su experiencia, elabora explicaciones mentales que le sirven para interpretar al mundo que lo rodea. ¿Y qué sucede cuando el cerebro completa este ciclo de aprender y reconocer patrones? Sucede algo que todos conocemos como diversión.

El cerebro libera dopaminas como recompensa al proceso de incorporar un nuevo patrón e identificarlo en una situación que resulta útil.

Cuando este proceso de aprender y reconocer patrones sale mal, el cerebro no libera ninguna recompensa y, como resultado, surgen el aburrimiento y la frustración. Pero aquí hay un detalle más. Si continuamos haciendo la misma actividad sin obtener resultados nuevos, a pesar de hacerlo siempre bien, también nos aburrimos. Como ya vimos en la primera sección, el desafío y el resultado desconocido son elementos importantes para un juego.

Sin embargo, mientras que en términos neurobiológicos la diversión consiste en la liberación de dopaminas en el cerebro, para un diseñador de videojuegos la diversión es ese delicado equilibrio que existe entre resolver los desafíos y no abrumarse o frustrarse en el intento.

Sin embargo no somos todos iguales, y el momento en que una persona se frustra o se aburre varía mucho según los gustos y características de cada uno. ¿Has jugado *QWOP*? Bueno, este es tu momento hacerlo, el juego lo puedes encontrar aquí: http://www.foddy.net/Athletics.html.



QWOP es un juego muy particular creado por un desarrollador apodado Bennett Foddy. El jugador debe intentar controlar las piernas de este atleta utilizando para ello cuatro teclas: Q, W, O y P. Cada una de estas teclas controla una parte de la pierna distinta. La paciencia y la coordinación son claves en este juego.

Ahora que has jugado a *QWOP*, ¿cuál fue tu mayor record? El juego es bastante difícil y requiere de mucha práctica, sin embargo a algunas personas este desafío les parece muy interesante y como se puede ver en el siguiente video son capaces de dominarlo: http://www.youtube.com/watch?v=V2GCLgaCcm4.

No hay recetas a la hora de hacer juegos, lo que sí existe son los criterios. El más básico de estos es encontrar el balance entre frustración y éxito, o entre

cosas nuevas y cosas que se repiten, siempre tratando de ofrecer un desafío que sea claro para el jugador y suficientes pistas como para que este sea capaz de aprender a jugarlo y dominarlo.

Por último, en esta sección te presentaré un esquema de análisis de juegos que es muy sencillo y que te ayudará a entender mejor cómo es cada juego. Este esquema lleva el nombre MDA por sus siglas en inglés, que significan mecánica, dinámica y estética.

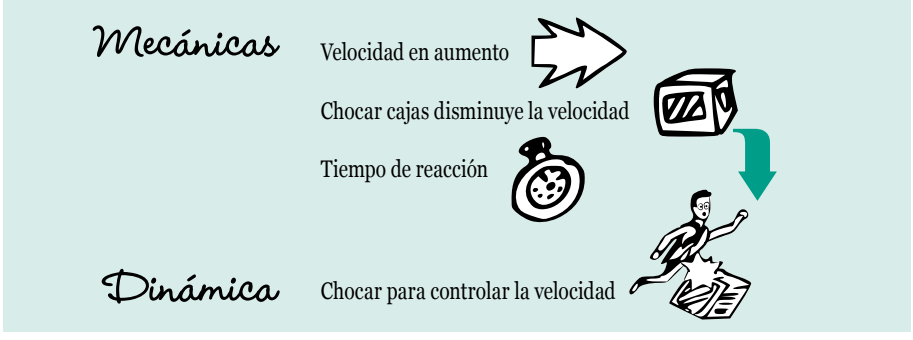
Mecánica

Son las reglas, las cosas que puede o no hacer el jugador, la manera en que se comportan los objetos, el objetivo del juego, etc. Las mecánicas definen la forma en que puede actuar un jugador. En *Canabalt*, por ejemplo, algunas de las reglas son:

- Con la tecla X, C o espacio, el personaje salta.
- Cuanto más tiempo se presiona la tecla para saltar, más alto salta.
- Chocarse con un obstáculo disminuye la velocidad.
- La velocidad aumenta constantemente si el personaje no choca con nada.

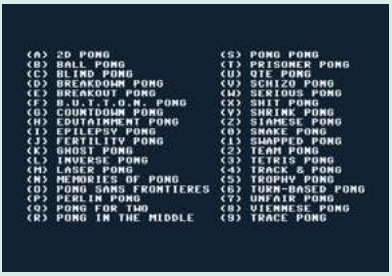
Dinámica

La dinámica es todo aquello que sucede en la cabeza del jugador. Son las decisiones que este toma. Las reglas determinan qué clases de dinámicas pueden surgir de un juego. Un ejemplo sencillo en *Canabalt* es que el jugador debe decidir en qué momento saltar para evitar caer en un precipicio. Otro ejemplo es el decidir si chocar o no con un obstáculo. Esta última decisión es consecuencia de varias mecánicas (reglas) y de las características del jugador, es decir de cuánto tiempo de reacción necesita para esquivar misiles y realizar saltos.



Una forma divertida de ver el impacto de las mecánicas en la forma de jugar y en las decisiones que se puede tomar es probando con un amigo este juego lla-

mado *Pongs*: http://www.pippinbarr.com/games/pongs/Pongs.html. El juego consiste en muchas variantes del *Pong*, presentado en la primera sección de este capítulo.



Pongs es un juego para dos jugadores en la cual los oponentes se enfrentan en una de las 36 variantes del juego original. Se juega con las teclas WASD para el jugador 1 y con las flechas direccionales del teclado para el jugador 2.

Cada una de estas variantes tiene un pequeño cambio en las reglas que transforma completamente la manera de jugar y de esta manera puedes ver el impacto de las mecánicas (reglas) en la dinámica.

Estética

La estética se refiere a los sentimientos que surgen en el jugador al momento de jugar. ¿Cómo es que las mecánicas y la dinámica influyen emocionalmente en el jugador? ¿Qué cosas pueden surgir del juego que el jugador encuentre placenteras? En un artículo publicado en un reconocido sitio de desarrollo de videojuegos,[1] Adam Saltsman contó que al crear *Canabalt* buscaba que el jugador experimentara acción frenética, un juego donde la velocidad lo fuera todo. Efectivamente, al momento de jugar *Canabalt* la presión de estar corriendo todo el tiempo a gran velocidad provoca en el jugador la sensación de encontrarse inmerso en una huida frenética.

Un buen juego

Un buen juego es aquel que tiene una mecánica que permite que el jugador pueda tomar decisiones interesantes y que, al hacerlo, siente aquellas cosas que estaba buscando al momento de ponerse a jugar, ya sea acción frenética en *Canabalt* o partidas estratégicas como en *Plants vs. Zombies*.

Lo importante es que el juego deje que el jugador explore ese mundo fascinante que ha creado el diseñador y que pueda aprender a dominarlo, nave-

1 Saltsman, Adam, "Tuning Canabalt", 29 de septiembre de 2010. Disponible en: <http://www.gamasutra.com/blogs/AdamSaltsman/20100929/88155/Tuning_Canabalt.php>.

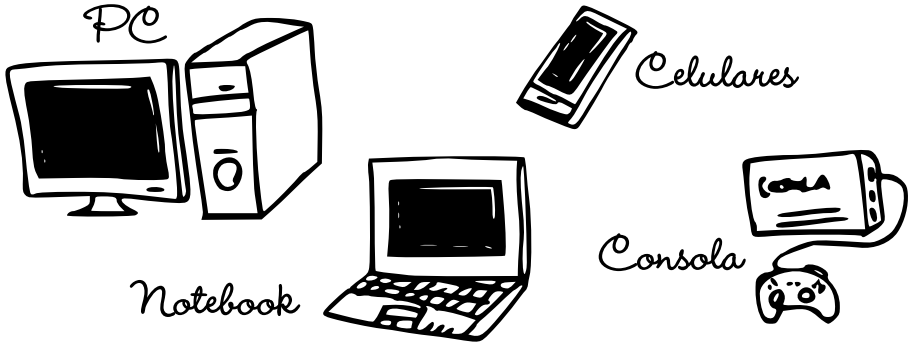
gando en ese delicado equilibrio entre lo desafiante y lo fácil, lo repetitivo y lo nuevo. Un ejemplo de un gran juego es el ajedrez. Sus mecánicas son muy fáciles de aprender (son muy pocas reglas) pero la dinámica que genera es gigantesca, tanto que lleva todo una vida aprender a jugarlo bien.

Durante esta sección has aprendido algunos criterios sobre cómo se diseñan los juegos y cómo funcionan las mentes de los jugadores. Solo queda una última parte de este marco teórico sobre videojuegos, la relacionada con la programación y con cómo funcionan las computadoras. Si quieres saber cómo es que una computadora es capaz de simular todo un mundo audiovisual interactivo, la sección siguiente es ideal para ti. De lo contrario, y como siempre, puedes saltearte esta sección para volver en otro momento y, mientras tanto, ponerte manos a la obra con la parte práctica.

1.4. CÓMO FUNCIONA UN VIDEOJUEGO DESDE EL PUNTO DE VISTA DE UN PROGRAMADOR

Los videojuegos, lamentablemente, no se programan solos. Como si fuera poco, además de programarlos, los desarrolladores deben asegurarse de que los juegos puedan "correr" en diferentes computadoras, smartphones, consolas, o cualquier otra plataforma en la cual el juego estará disponible.

Los videojuegos consumen muchos recursos de una máquina e intentan sacarle el máximo provecho para ofrecer experiencias interesantes al jugador. Para poder entender cómo es posible que un videojuego, y en general un programa, funcione sobre una computadora es necesario saber mínimamente cómo está compuesta una computadora y qué es lo que hace cada parte de la misma. No te preocupes que será un vistazo breve y, además, aprenderás cosas nuevas que no te esperabas.



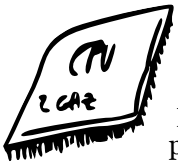
Tanto las PC, *notebooks*, *netbooks*, celulares, consolas, *tablets* y hasta los modernos *smart tv*, son computadoras. Todos tienen las mismas partes y funcionan de la misma manera. Por lo tanto las siguientes descripciones servirán para analizarlas a todas ellas.

Placa madre: la placa madre tiene, entre otras funciones, la tarea de conectar cada una de las partes de la computadora —así como los periféricos— mediante el bus de datos, que es el canal de comunicación interno.

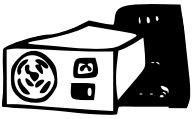
Almacenamiento: es el lugar donde la computadora guarda información de forma permanente. Las imágenes, los videos, los sonidos, los archivos de datos y los programas se guardan aquí. En general tienen mucha capacidad, lo que significa que pueden almacenar mucha información.

Memoria RAM: es el lugar en donde se encuentran los programas que están ejecutándose, junto con sus datos. Esta memoria es muchísimo más rápida que los medios de almacenamiento permanente, lo que significa que se pueden extraer y guardar gran cantidad de datos en muy poco tiempo.

Procesador: es el componente central de la computadora, responsable de ejecutar las instrucciones que tienen los programas. Cuenta con registros para guardar pequeñas cantidades de información y realizar operaciones matemáticas y lógicas. Utiliza la memoria RAM para cargar y guardar los datos que necesita para ejecutar los programas. Cuanto más rápido puede trabajar un procesador, más instrucciones por segundo puede ejecutar. Tal vez también hayas oído hablar de procesadores de más de un núcleo. Esto significa que el procesador está formado por más de una unidad de procesamiento y que por lo tanto es capaz de ejecutar instrucciones en paralelo (es decir, hacer dos cosas al mismo tiempo).



Fuente de alimentación: como te podrás imaginar, es el lugar de donde obtiene energía la computadora y desde donde se distribuye a todos sus componentes.



Estos son, a grandes rasgos, los elementos más importantes de una computadora y también los más importantes para entender cómo funcionan los programas.

Todos los datos y programas se guardan en los medios de almacenamiento permanente, que albergan la información tengan o no energía. Es decir que si estás escribiendo un documento de texto y lo has guardado en el disco duro, no lo perderás aunque se corte el suministro de energía eléctrica.

El microprocesador carga los programas que están en los medios de almacenamiento y los guarda en la memoria RAM, que es muchísimo más rápida. Sin embargo, si se cortara el suministro de energía toda la información que guardaba la memoria RAM se perdería. Es por esta razón que no se usa para guardar datos de manera permanente (además de que su costo es más elevado que el de los medios de almacenamiento permanentes).

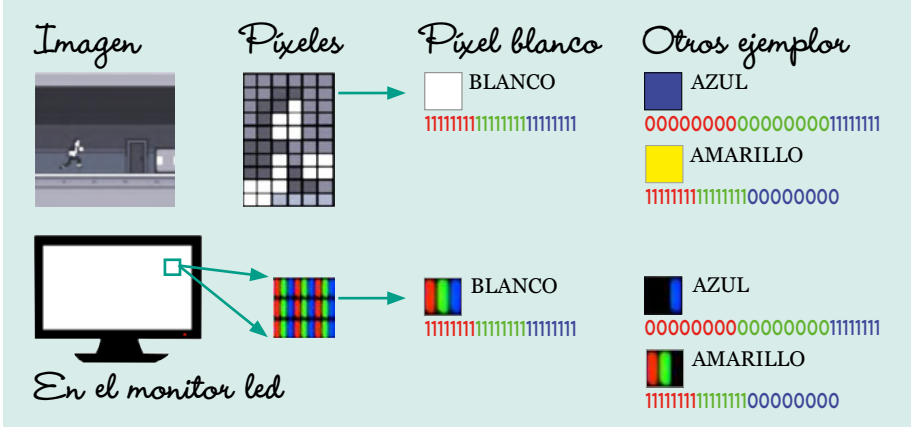
El uso de memoria y de poder de procesamiento tiene que ser eficiente y no se puede malgastar, ya que es limitado. Si estos recursos se desperdician, los programas se ejecutan más lentamente e incluso pueden dejar de funcionar en caso de exceder la capacidad de la computadora. En los videojuegos es crucial que el programa funcione rápido, con el ideal –tal como se contaba en la sección 2 de este capítulo– de los sesenta cuadros por segundo.

Es probable que hayas escuchado hablar del código binario. Si no es así, no te preocupes, que es algo muy simple. En una computadora, la información (y las instrucciones) está codificada en este famoso código, que no es más que unos y ceros.

El ASCII, por ejemplo, es un código que asigna a cada letra y carácter un código binario de 8 bits. Un bit es la unidad mínima de información, es decir un uno o un cero. A la letra A mayúscula, en el código ASCII, le corresponde el valor 01000001.

Es posible que te preguntes cómo es posible que los videojuegos, con su esplendor audiovisual, puedan ser traducidos a simples unos y ceros. Lo cierto es que todo puede ser almacenado utilizando con cierta precisión el código binario.

Las imágenes, por ejemplo, se codifican en píxeles. ¿Sabes lo que es un píxel? Una imagen de computadora está compuesta por muchos puntos y cada uno de esos puntos tiene asignado un color. Bueno, esos puntos de color son los mencionados píxeles. Los colores son los que definen el valor de cada píxel, y cada píxel está compuesto por un componente rojo, uno verde y otro azul. Para entender mejor este enredo de palabras mira la imagen a continuación:



Los sonidos también son codificados a uno y cero, pero de manera un poco distinta a los píxeles. ¿Sabes lo que es el sonido en realidad? El sonido consiste en vibraciones en el aire que el oído capta gracias a su fisionomía. Cuanto más grandes son las variaciones en esas vibraciones, más fuerte es el sonido resultante; cuanto menos espacio las separa, más agudas se escuchan. Para grabarlas en un dispositivo, los micrófonos toman muestras de estas vibraciones en el aire y las almacenan. Los valores de estas vibraciones se miden lógicamente de 1 a –1 y, a su vez, se traducen a código binario. La siguiente imagen te ayudara a entenderlo:

Las instrucciones que le dicen qué hacer a la computadora también están codificadas en código binario. ¡Pero no te asustes! Cuando programamos un videojuego en la actualidad no es necesario escribir con ceros y unos. Existen programas que traducen los lenguajes de programación a uno que puede comprender la computadora. Aún mejor, durante el segundo capítulo utilizaremos un programa que nos permitirá editar visualmente las opciones e instrucciones sin que tengas que escribir casi nada.

En la computadora funcionan muchos programas que trabajan a la vez: desde los que se encargan de interpretar las teclas presionadas en un teclado hasta otros más complejos como el sistema operativo, responsable de manejar los recursos de la computadora para que los usuarios y otros programas puedan utilizarla.

Lo importante de esta sección es que tengas una visión superficial de cómo funciona una computadora para entender que no hay nada de magia atrás de ello: partes interconectadas, abstracciones en código binario y programas trabajando codo a codo con otros programas forman la base del funcionamiento de la computadora, nos permiten navegar por internet, escuchar música, escribir documentos y, por supuesto, ¡jugar videojuegos!

Ahora sí, ya no quedan más excusas, estás preparado para avanzar con la parte práctica del capítulo 2. ¡Mucha suerte con tu primer videojuego!

Desarrollo de un videojuego paso a paso

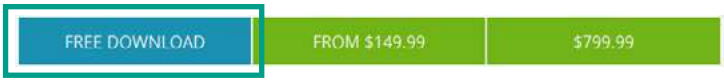
En este capítulo aprenderás a utilizar la herramienta Game Maker para desarrollar tu propio videojuego. Guiado paso a paso, realizarás una serie de experimentos y ejercicios que te darán los conocimientos necesarios para desenvolverte sin problemas con el programa.

Al final de este capítulo hay una guía con detalles de las principales acciones y condiciones que te serán necesarias para encarar tus propios proyectos utilizando Game Maker.

2.1. PRESENTACIÓN DEL CAPÍTULO Y ALGUNAS NOCIONES BÁSICAS

La herramienta que utilizaremos en este libro se llama Game Maker (GM). Si bien es una herramienta paga, Yoyo Games provee una versión gratuita que se puede utilizar sin problemas para crear juegos y ejecutarlos en PC o Mac. Al final del capítulo se presentarán alternativas a este programa, para que tengas otras opciones para elegir, pero no te preocupes porque todos se manejan de manera muy similar a GM.

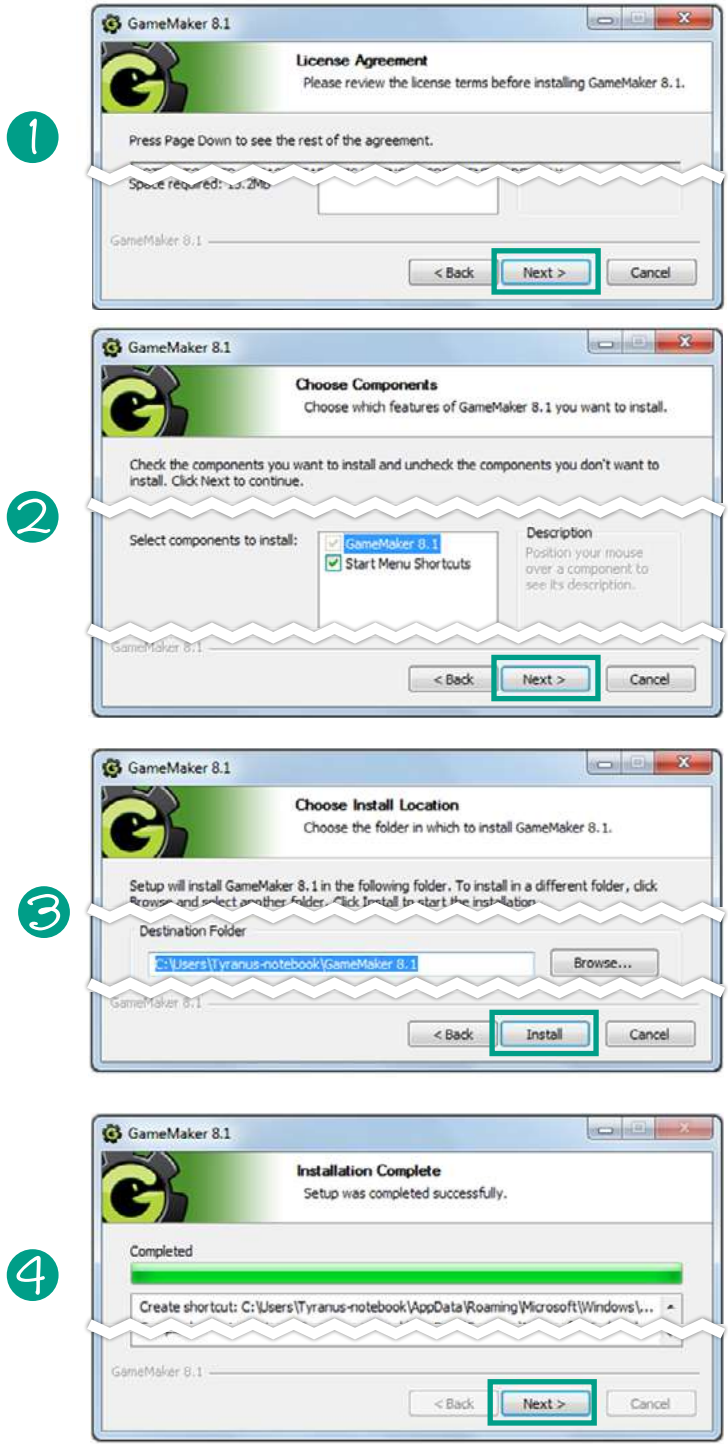
Antes que nada, hay que instalar el programa. Para hacerlo, ingresa en la siguiente web: https://www.yoyogames.com/studio/download. Una vez allí, haz clic en el link que dice "GET GAMEMAKER" y elige la siguiente opción:



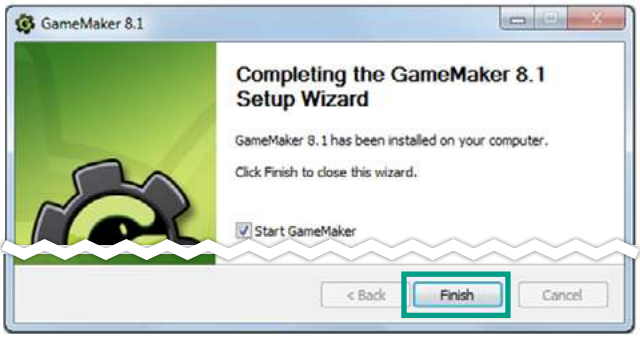
Cuando termine la descarga, haz doble clic en el archivo ejecutable. Si ya has instalado programas antes, puedes saltearte esto y seguir a partir de aquí. Si es la primera vez que instalas algo, vas a descubrir que no es nada raro ni difícil. Primero, ejecuta el archivo. Luego:

1) Haz clic en *I agree* para aceptar la licencia de uso del programa.
2) Haz clic en *Next* para realizar la instalación por defecto.
3) Haz clic en *Install* para instalar el programa en el disco C.
4) Cuando la barra de progreso se complete, haz clic en *Next* para terminar la instalación.

5) Por último, haz clic en *Finish* y acto seguido se ejecutará el programa.



5



Ahora el programa ya está listo para que lo uses. Cuando lo ejecutes aparecerá algo así:

En este menú podrás elegir diferentes opciones. Las más importantes son:

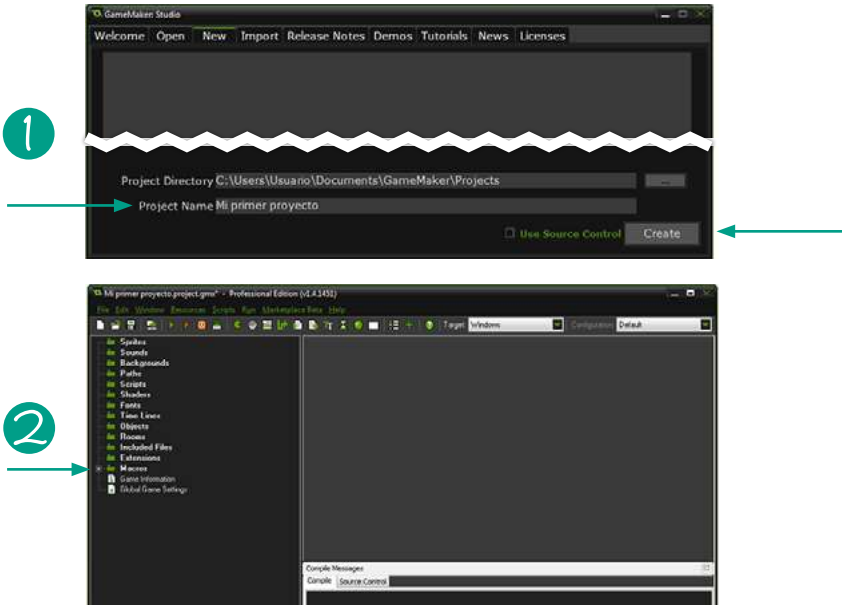
- *Open* para abrir un proyecto.
- *New* para crear uno nuevo
- *Demos* para acceder a los ejemplos que pueden ser descargados desde la página de Yoyo.

2.2. DESARROLLO DE UN VIDEOJUEGO CON GAME MAKER

2.2.1. Tu primer experimento: “El recoge frutas”

Este es tu primer proyecto y hay un montón de cosas por aprender, pero no te preocupes porque empezaremos con algo muy sencillo. Vamos a realizar un juego de recoger frutas: las frutas caerán desde arriba y el jugador tendrá que agarrarlas para ganar puntos. Es muy fácil de programar y te ayudará a entender cómo funciona este programa a la vez que creas tu primer videojuego.

Lo primero que aprenderás es cómo crear un nuevo proyecto. Una vez abierto el programa, haz clic en la solapa *New* y a continuación se abrirá una ventana (1). Lo que tienes que hacer es poner donde dice *Project Name* el nombre del proyecto que vas a crear, en tu caso será "Mi primer experimento". Luego tienes que hacer clic en *Create* y *a* continuación se abrirá el nuevo proyecto, totalmente vacío (2).



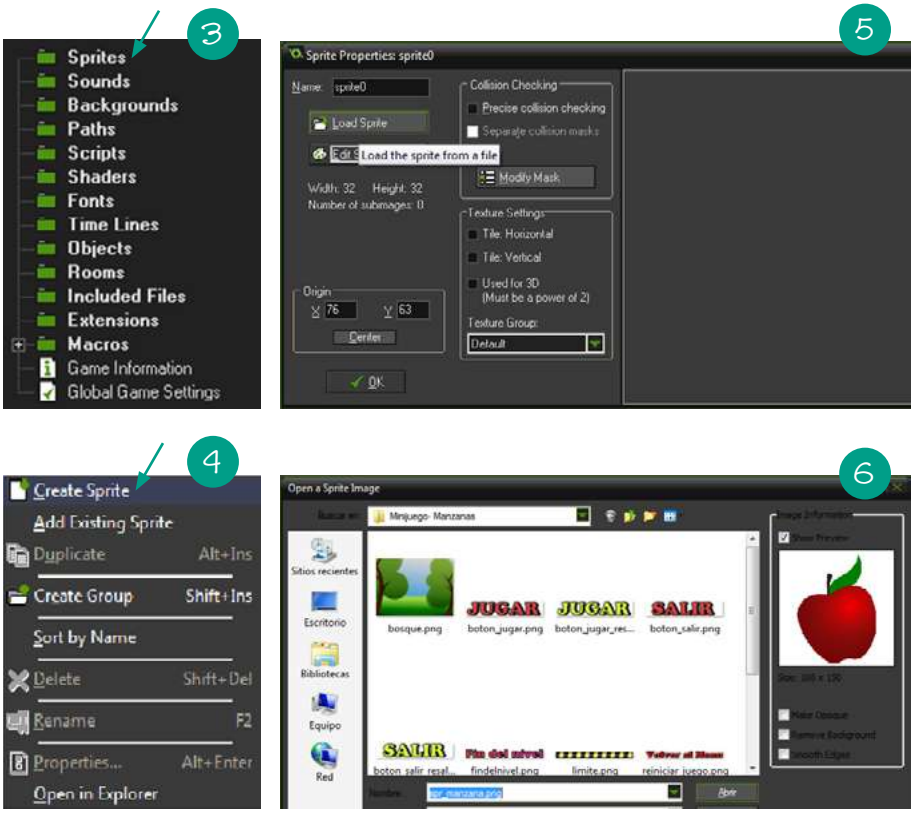
Échale un vistazo rápido a la interfaz. A la izquierda, agrupados por carpetas, se encuentran los elementos que compondrán el juego. Los más importantes son:

- *Sprites:* son las imágenes.
- *Sounds:* los sonidos y música.
- *Backgrounds:* son las imágenes de fondo que utiliza el juego.
- *Paths:* son caminos que pueden seguir los objetos.
- *Scripts:* son instrucciones que se le pueden asignar a los objetos para que las ejecuten.
- *Objects:* son objetos como el personaje principal, los enemigos, etc.
- *Rooms:* son los niveles y menús.

Ahora sí, para comenzar con el juego de recoger frutas lo primero que necesitarás serán frutas. Un lugar donde conseguir unos lindos gráficos y gratuitos es este: http://www.lostgarden.com/2007/05/dancs-miraculously-flexible-game.html. Antes de comenzar, en http://editorial.unipe.edu.ar/videojuego/ puedes encontrar todas las imágenes y recursos utilizados en los proyectos de ejemplos. Se trata, en todos los casos, de diseñados realizados por Daniel Cook para Lostgarden.com y bautizados como "Planet Cute".

Lo primero que vas a crear en Game Maker (además del proyecto) es un sprite. Un sprite es una imagen –puede estar animada o no– que se utilizará para visualizar los objetos en el juego. Para hacer eso, haz clic derecho sobre la carpeta *Sprite* (3), en el panel de la izquierda, y elige la opción *Create Sprite* (4). Al hacer clic allí se abrirá una ventana (5).

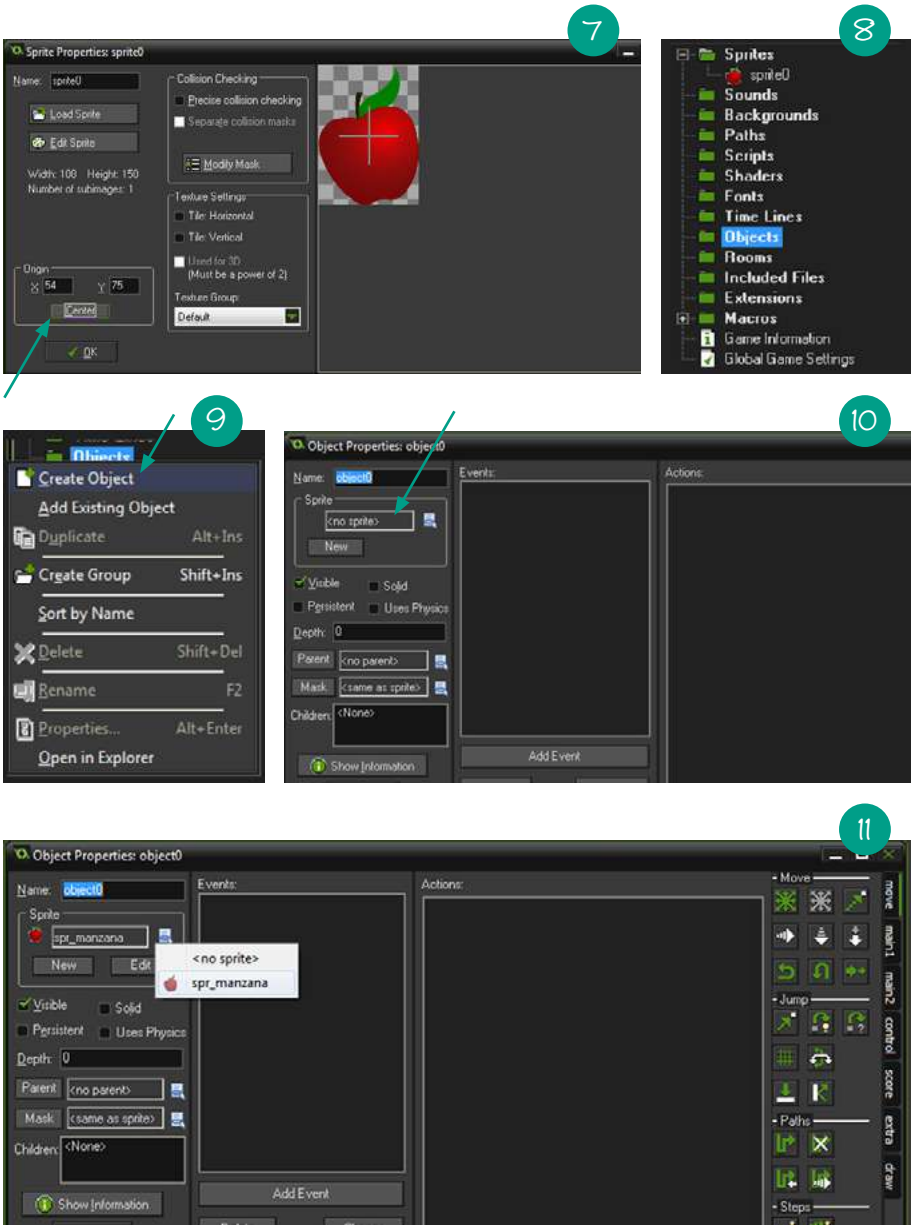
En esta ventana puedes asignarle un nombre al sprite. Para hacer las cosas más fáciles después, a todos los sprites les pondrás "spr_" al principio del nombre y así evitarás muchas confusiones. Este sprite se llamará entonces "spr_manzana". Para cargar un sprite haz clic en la opción *Load Sprite*, y busca el archivo que tenga la imagen que quieres utilizar. En este caso será "manzana.png" en la carpeta de recursos que descargaste y extrajiste (6).



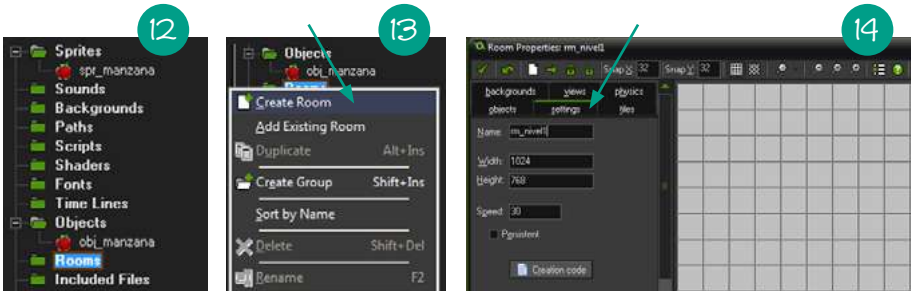
Haz doble clic en la imagen y ya está cargada. A continuación tendrás que hacer clic en *Center* para que el punto de origen de la imagen esté en el centro y, luego, en *OK* (7). ¡Perfecto! Ya creaste el primer sprite. Lo siguiente será crear un objeto manzana al que luego le agregarás comportamiento. Para crearlo, ve al panel de la izquierda, haz clic en la carpeta *Objects* (8), y luego en *Create Object* (9).

Al hacer clic allí, se abrirá una ventana (10). De nombre le pondrás "obj_manzana" (todos los objetos empezarán con "obj_" para evitar confusiones).

A continuación haz clic en donde dice <no sprite> (11) y busca allí el sprite de la manzana. Para finalizar, haz clic en *Ok*. El resto de las opciones las examinaremos más adelante.



El siguiente paso es crear un nivel o room para poner tu linda manzana. Para ello haz clic derecho en la carpeta *Rooms* del panel de la izquierda (12) y elige la opción *Create Room* (13). Se abrirá una ventana y, a continuación, harás clic en la solapa *settings* (14).



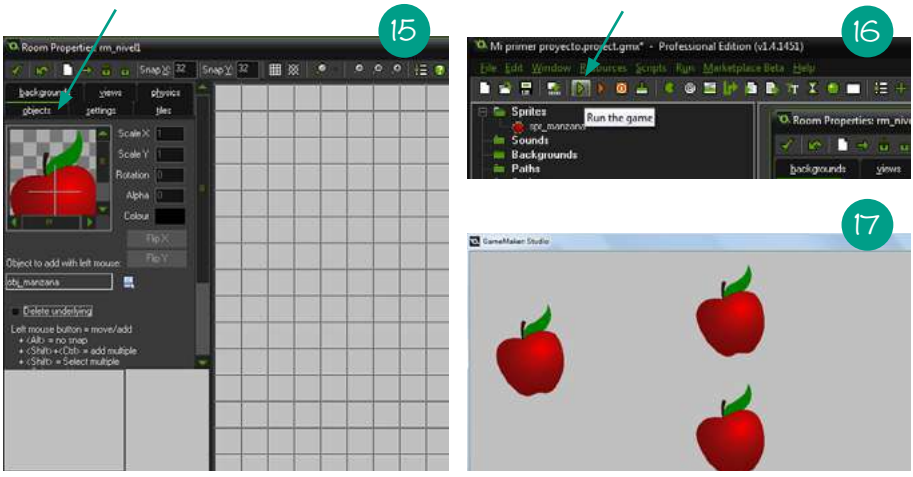
Aquí se le puede cambiar el tamaño al área del nivel, pero por el momento puedes dejarlo en 640 de *Width* (ancho) y 480 de *Height* (alto). Si quieres puedes cambiar el tamaño siempre y cuando no sea ni exageradamente pequeño (1x1) ni exageradamente grande (50000x50000). Luego le pondrás de nombre "rm_nivel1" (a todos los niveles le pondrás "rm_" al inicio para evitar confusiones).

A continuación haz clic, en esa misma ventana, en la solapa *objects* (15).

Una vez hecho eso, en el área gris cuadriculada puedes, simplemente haciendo clic, agregar tu objeto "obj_manzana" cuantas veces quieras.

Bien, hasta ahora has aprendido a crear sprites y cargar imágenes, crear un objeto, crear un room y poner objetos en un room. Para poder ejecutar el ejemplo haz clic en el botón *play* como muestra la imagen 16.

Si has colocado manzanas en el room, en este momento tendrías que ver algo muy parecido a la imagen 17.



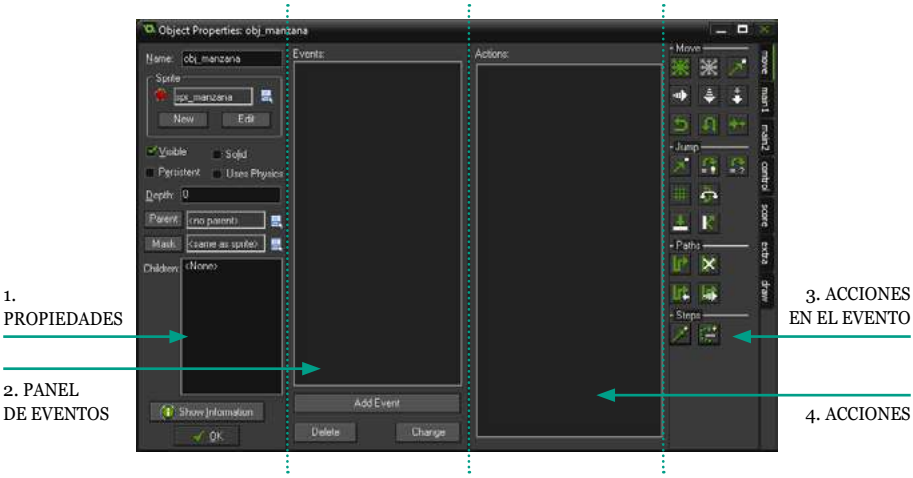
No podemos decir que esto sea tu primer juego, pero estamos cada vez más cerca. Ahora lo que nos falta es la interacción, ya que a simple vista lo que acabamos de crear parece solo una imagen fija. Vamos a introducirnos en el mundo de la programación..

Tal como se dijo en la sección Cómo funciona un videojuego desde el punto de vista de un programador, una computadora se encarga de ejecutar instrucciones y eso es lo único que sabe hacer. ¿Pero

cómo se hace para decirle qué hacer a la computadora en Game Maker? Lo cierto es que para poder hacer eso hay que tener en cuenta tres elementos importantes de Game Maker:

- *Los objetos:* contienen información, comportamientos y el *sprite* que los representa en pantalla.
- *Los eventos:* son momentos en que una determinada condición se cumple. Por ejemplo, cuando se presiona la tecla espacio se produce el evento "se presionó la tecla espacio". Cuando dos objetos chocan entre sí, se produce el evento "colisión entre dos objetos".
- *Las acciones:* son las instrucciones que tiene que ejecutar un objeto cuando se produce un determinado evento.

Haz doble clic en el objeto "obj_manzana", se abrirá una ventana como esta:



En la columna "1 Propiedades" se encuentran las propiedades del "obj_manzana". Por ejemplo, en *Sprite* se muestra cuál es el sprite que representa al objeto en pantalla ("spr_manzana").

La columna "2 Panel de eventos" es el lugar donde se pueden agregar los eventos a los cuales debe responder el objeto.

La columna "3 Acciones del evento" es el lugar donde se agregan las acciones para cada evento.

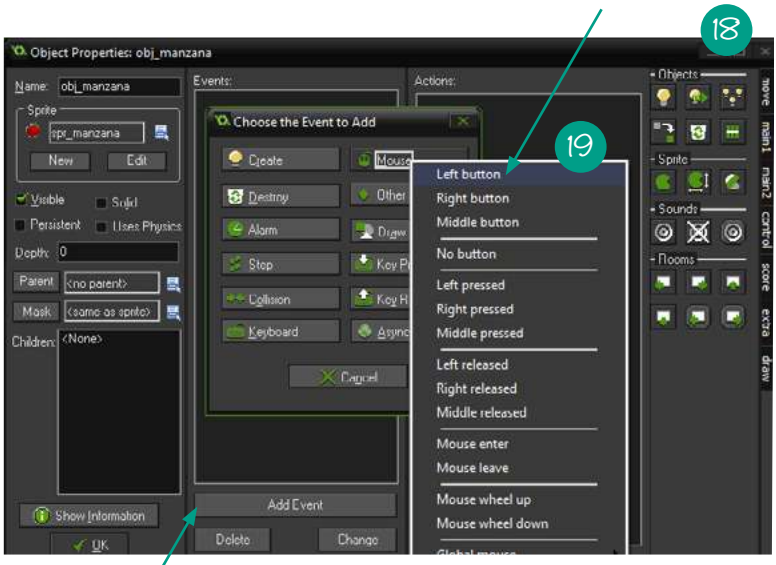
La columna "4 Acciones" es el lugar donde se encuentran todas las acciones disponibles para utilizar en eventos. De allí nosotros elegiremos cuáles son las que queremos usar, arrastrándolas hasta la columna 3.

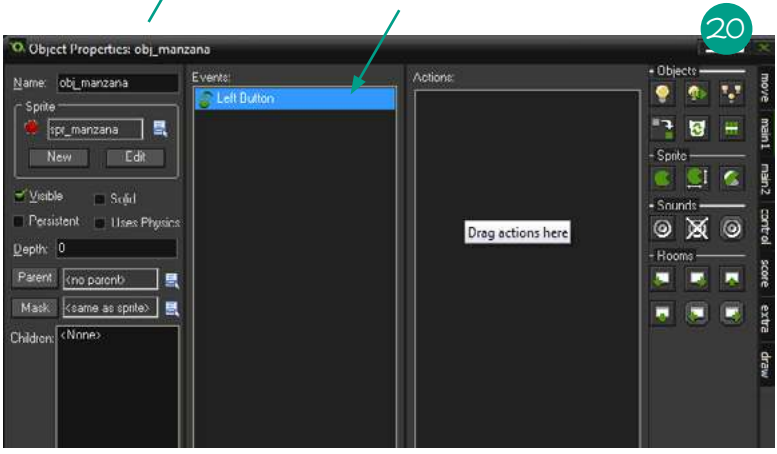
Lo que nosotros queremos hacer es que cuando el jugador haga clic en una manzana esta desaparezca y sume puntos. Traducido en términos de Game Maker, lo que queremos es agregar las acciones "eliminar el objeto manzana" y "sumar puntos" cuando se produzca el evento "hacer clic en objeto manzana". Para ello, en la parte "2 Panel de eventos" del objeto "obj_manzana", haz clic en *Add Event* (agregar evento) (18).

Al hacerlo, se abrirá la ventana *Choose the Event to Add* (elige el evento a agregar). En esta nueva ventana hay un listado de los eventos que podemos agregarle a nuestro objeto "obj_manzana". El evento *Create*, por ejemplo, se produce cuando se crea el "obj_manzana" (apenas aparece la manzana en la pantalla). El evento *Step*, en cambio, se produce todo el tiempo. Al final de este capítulo hay una sección con la descripción de cada uno de los eventos más importantes.

El evento que estamos buscando es el evento que tiene que ver con el mouse, por eso harás clic en *Mouse* y a continuación se abrirá un menú (19).

En este caso, queremos que cuando se presione el botón izquierdo desaparezca la manzana, y para eso harás clic en *Left Button* (botón izquierdo). A continuación verás que en la lista de eventos se agregó el evento *Left Button* (20), lo que quiere decir que las acciones que estén en la sección que dice *Actions*, a su derecha, son las que se ejecutarán al hacer clic en el objeto "obj_manzana".





Miremos ahora, por un momento, la sección que está más hacia la derecha, la parte "Acciones". Allí se encuentran todas las acciones disponibles ordenadas en paneles. En este panel en particular están las asociadas al movimiento que veremos después. Lo que debes hacer ahora es clic en la solapa que dice *main1* (21).

En el panel 2 están las acciones que aplican a los objetos. La primera de todas tiene este ícono y sirve para crear un nuevo objeto. La que tiene este ícono sirve para cambiarle el sprite al objeto. Al final de este capítulo también hay una sección con las acciones más importantes y una descripción de cómo funcionan.

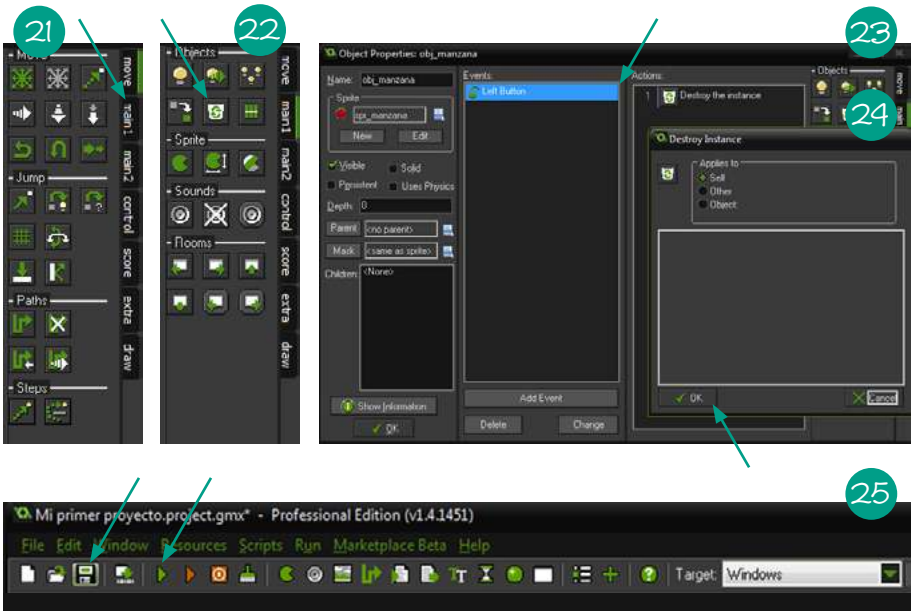
Lo que queremos hacer ahora es que la manzana desaparezca, se elimine. Para ello usaremos la acción con el ícono , que es la de eliminar un objeto (22). Tienes que arrastrar el ícono hasta la sección de acciones, como muestra la figura de abajo.

Una vez "soltado" allí, creará la acción *Destroy* en la sección de acciones del evento *Left Button*, como muestra la pantalla 23.

La misma ventana nos pregunta sobre qué objeto debe recaer la acción. En este caso es en el mismo objeto, es decir, el "obj_manzana", por lo que dejamos seleccionada la opción *Self* (que significa "sobre sí misma" en ingles) y cliqueamos en *OK* (24).

Ahora ya estás listo para probar tu juego, haz clic en *guardar* y luego en *play* (25).

¡Ahora sí! Las manzanas desaparecen al cliquearlas. Esto significa que cuando nosotros hacemos clic sobre una manzana, se activa el evento *Left button* en el objeto "obj_manzana" y se ejecuta la acción *Destroy* que agregamos previamente.



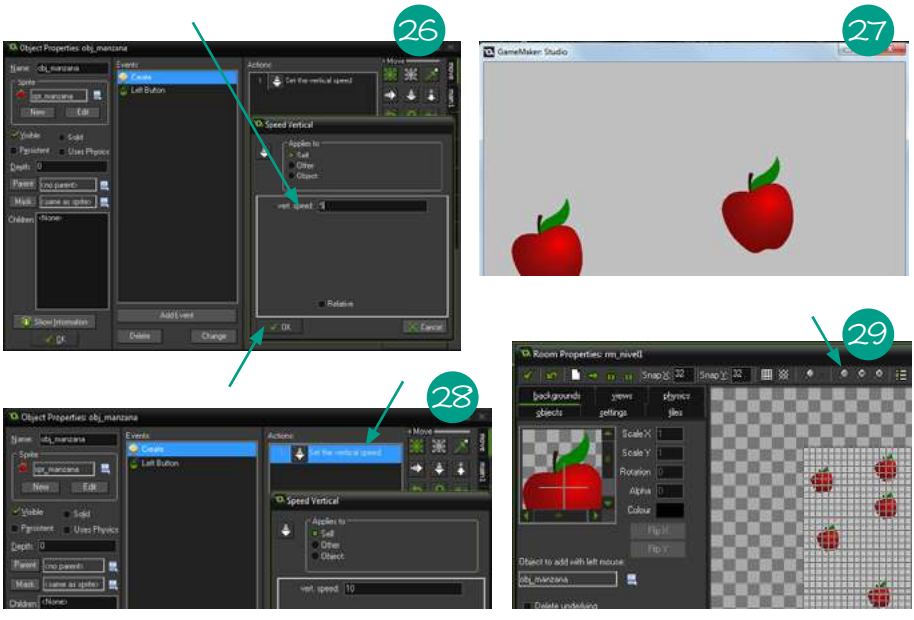
Pero eso es demasiado fácil. ¡Las manzanas deberían moverse! Para hacer eso tenemos que agregar un nuevo evento. Queremos que apenas aparezcan las manzanas, es decir al momento de crearse, comiencen a caer. Para hacerlo, agrega el evento *Create* en el "obj_manzana". Y ahora elige la acción *Speed Vertical* (velocidad vertical). Al agregarla al evento *Create* se abrirá una ventana (26).

Donde dice "vert. speed" tienes que elegir con qué velocidad quieres que caiga. Empieza poniendo un uno allí y luego haz clic en *OK*. Guárdalo y prueba (27). Las manzanas caen, pero muuuy lentamente. Bien, ajustaremos ese número. Para ello vuelve a abrir el "obj_manzana" (si lo cerraste), haz clic en el evento *Create* en la lista de eventos y luego doble clic en la acción *Speed Vertical*, como muestra la imagen 28.

Ahora sí, introduce un número mayor, haz clic en *OK*, guarda y prueba. Repítelo hasta que estés satisfecho con la velocidad a la que caen. Como ves, el juego ya está quedando un poco más complicado.

Bien, con unos pocos clics ya tenemos la estructura de este pequeño juego: caen manzanas que podemos hacer desaparecer al cliquear.

El problema ahora es que las manzanas se acaban muy rápidamente. Para hacer que el nivel dure más tiempo agregarás más manzanas fuera del área visible del jugador. Haz doble clic en la room "roomo" y usa la imagen (29) como guía: el área que está marcada en rojo en la imagen es la que corresponde al área visible por el jugador. Usando la herramienta de lupa con el símbolo menos puedes achicar el tamaño de los objetos en el editor de room, y presionando la tecla espacio y el clic izquierdo del mouse, puedes navegar para poder ver la parte superior del nivel. De esta manera nos aseguramos de que unas cuantas manzanas sean parte del desafío.



Nos falta el puntaje, para que el jugador pueda medir su rendimiento y competir contra alguien (o contra sí mismo). Pero antes de hacer que se acumule ¡necesitamos verlo! Para ello crearemos un nuevo objeto llamado "obj_controlador". Ahora haremos uso de un nuevo tipo de evento, el evento *Draw* (dibujar). Este evento se produce cuando el juego dibuja las imágenes en pantalla, y es justo lo que necesitamos ya que el "obj_controlador" será el encargado de mostrar el puntaje en pantalla.

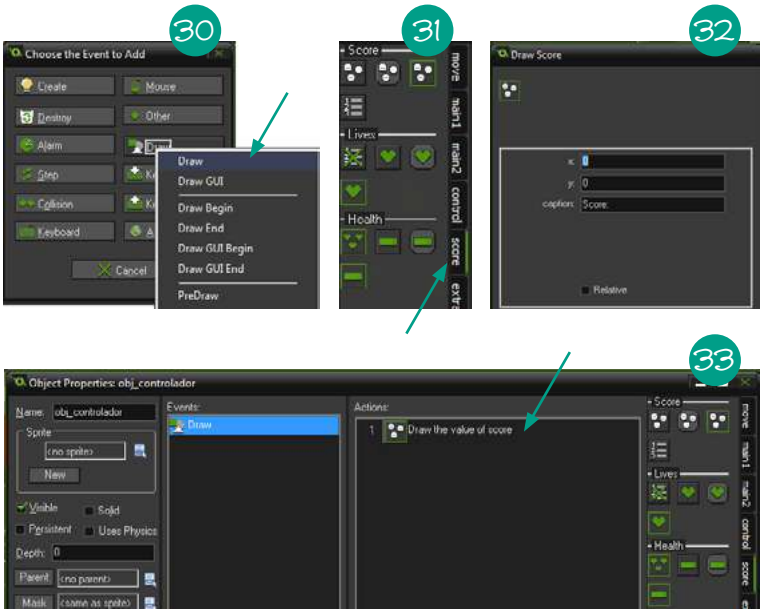
Luego de agregar el evento *Draw* ve al panel de acciones (30), y elige nuevamente *Draw*.

Una vez creado el evento, necesitamos una acción del panel de acciones que nos permita dibujar en pantalla el puntaje. Verás la solapa *score* (que significa puntaje en inglés) y dentro de ella varias acciones que nos pueden ser útiles. Mira la imagen 31 como referencia.

Este ícono  , por ejemplo, sirve para darle un valor al puntaje; este otro  para verificar si el jugador ha alcanzado cierto puntaje; y este último  sirve para dibujar el puntaje en la pantalla, justo lo que necesitamos.

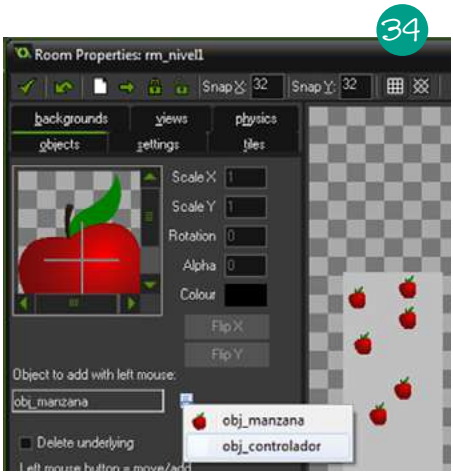
Arrastraremos el ícono de la acción *Draw Score*  hacia el área de acciones del evento *Draw* de nuestro objeto "obj_controlador". Al hacerlo se abrirá una ventana (32).

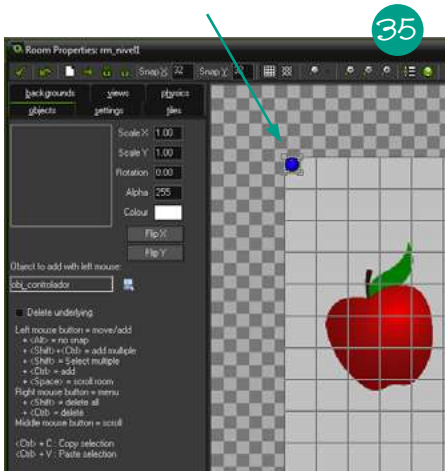
Las coordenadas "x:" e "y:" marcan la posición en que se dibujará el puntaje. "x" es a cuánta distancia horizontal debe estar con respecto al borde izquierdo e "y" es a cuánta distancia vertical debe estar el puntaje desde la parte superior del área visible. La opción "caption" se refiere a qué texto debe acompañar el puntaje. En nuestro caso, como hablamos castellano, colocaremos "Puntaje:" en lugar de "Score:". Luego haremos clic en *OK*. El objeto "obj_controlador" quedará como muestra la imagen 33.



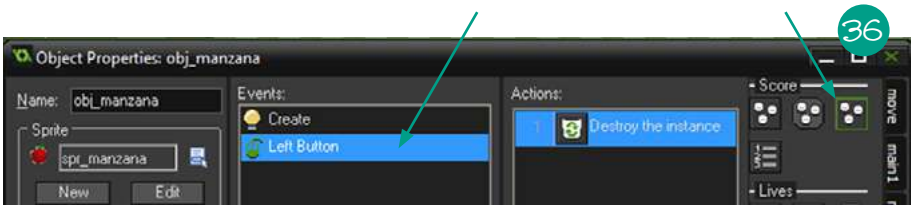
Bien, aquí hay un detalle importante, el objeto controlador no tiene ningún sprite asignado, por lo que no tendrá un gráfico asociado. Sin embargo, para que se pueda dibujar en pantalla el puntaje, necesitamos agregar al room el objeto "obj_controlador" de todas formas. Para ello, en el "roomo" del editor de room agregarás el "obj_controlador" donde tú quieras. Una vez abierto el editor de room haz clic en el panel *Objects* y en donde dice "Object to add with left mouse:" (objeto para agregar con el clic izquierdo) elige el "obj_controlador" y agrégalo una sola vez. Usa la imagen 34 como guía.

Al agregarlo, como el "obj_controlador" no tiene sprite, el programa nos muestra en su lugar un signo de pregunta (35). Ahora que ya agregamos el objeto encargado de mostrar el puntaje, haz clic en *guardar* y en *probar*.





Vemos que al hacer clic en las manzanas el puntaje no cambia. Para hacerlo tendremos que agregar al "obj_manzana" una acción para cambiar el valor del puntaje. ¿En qué evento sería? En el preciso momento en que el jugador hace clic en la manzana. Abre el "obj_manzana" y agrega la acción *Set Score* , como muestra la imagen 36.



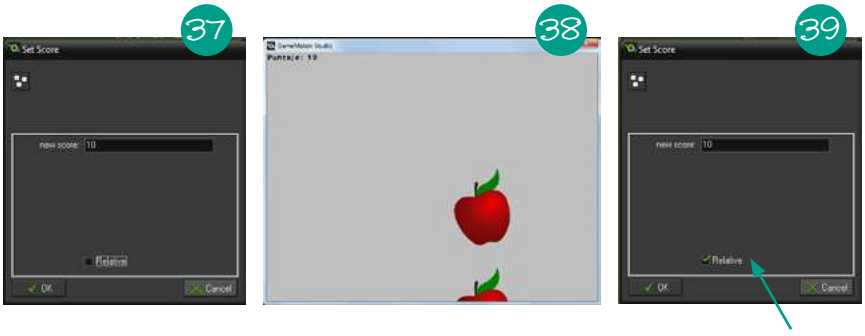
Al agregarlo se abrirá una ventana (37). Aquí podremos poner cuántos puntos queremos otorgar cada vez que se recoge una manzana. En donde dice "new score:" coloca al valor en puntos que quieras que sume. En mi caso, pondré 10. Luego haz clic en *OK*, guarda y prueba. ¿Qué es lo que sucedió? ¡El puntaje es siempre el mismo! (38).

Esto ocurre porque cada manzana, en lugar de sumar diez puntos pone diez como el puntaje total.

Para hacer que sume, ve al objeto "obj_manzana"-> evento *left mouse* -> acción *Set Score* y coloca un tilde en *Relative* para que el puntaje se sume, como muestra la imagen 39.

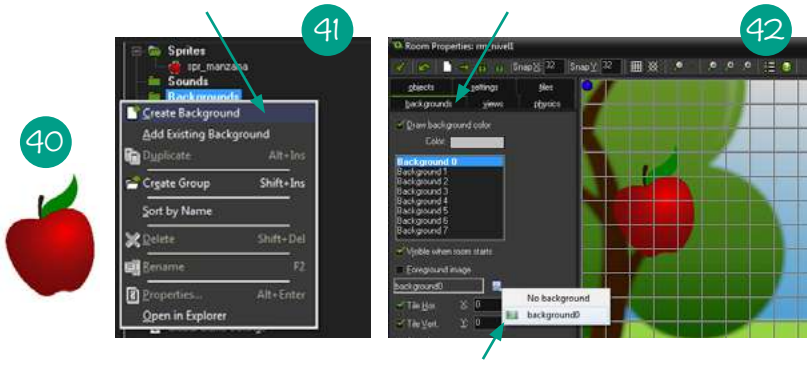
Ahora sí, ¡los puntajes se acumulan! Un dato curioso, que te va a ser útil para la siguiente actividad, es que si en lugar de poner 10 puntos en la manzana pusiéramos –10 (con el símbolo de menos adelante), en lugar de sumar diez puntos los restaría, haciendo que el jugador perdiera puntos.

Con esta idea en mente, intenta crear una manzana podrida, similar a la sana, pero que en lugar de sumar puntos los reste. Para ello, crea el sprite "spr_manzanaMala" y cárgale la imagen "Manzana_Mala.png" (40). Luego crea el objeto "obj_manzanaMala", haz que caiga, y que reste puntaje y desaparezca cuando se le haga clic encima. ¿Fue fácil, no?



A nuestro juego le falta un poco de decoración, le vendría muy bien un fondo. Para ello crearemos un background (fondo, en inglés). Haz clic derecho sobre la carpeta *Backgrounds*, en el panel de la izquierda, cliquea en *Create Background* (41). Luego –dentro de la ventana del nuevo background– haz clic en *Load Background* y carga la imagen de la carpeta de recursos llamada "bosque.png" (si quieres puedes cargar cualquier otra). Por último, nómbralo "bkg_bosque" (los nombres de los objetos no pueden ir con espacios) y haz clic en *OK*.

Ahora, en el editor de room elige el panel *Backgrounds* y busca allí el fondo "bkg_bosque", como muestra la imagen 42.

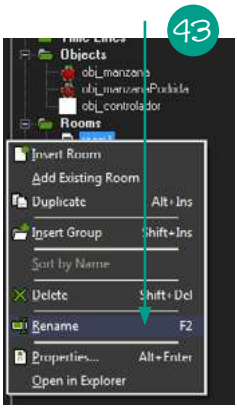


Lo último que necesitamos es una pantalla de menú y una pantalla de "Fin del juego".

Primero vamos a cambiarle el nombre a la room en que colocaste las manzanas, que por el momento se llama "room0". Le pondrás "rm_juego". Para hacer eso, selecciónala del panel de la izquierda, haz clic derecho sobre la room, elige la opción *Rename* (renombrar) y luego colóca el nuevo nombre (43).

Ahora sí, crea otra room y ponle el nombre "rm_menu". Luego agrégale el fondo que gustes. Yo utilizaré el mismo que para el juego, el background "bkg_bosque". A continuación necesitamos los botones y, para eso, los sprites de los botones y los objetos.

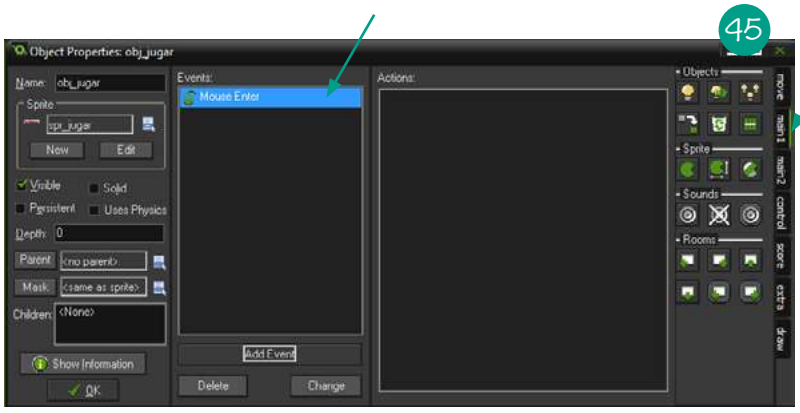
Crea un sprite, carga la imagen "boton_jugar.png" y nómbralo "spr_jugar". Luego agrega otro sprite, carga la imagen "boton_jugar2.png" y nómbralo "spr_jugar2". ¿Te preguntarás por qué dos imágenes del mismo botón? El motivo es que queremos que cuando el mouse pase por arriba del botón, este se ponga amarillo, y cuando salga, se vuelva a poner rojo. Le dará un efecto muy bonito, ya verás. Ahora es el momento de crear el "obj_boton_jugar", asignarle el sprite "spr_jugar" y colocarlo en la room "rm_menu" con el editor. Quedará como muestra la figura 44.

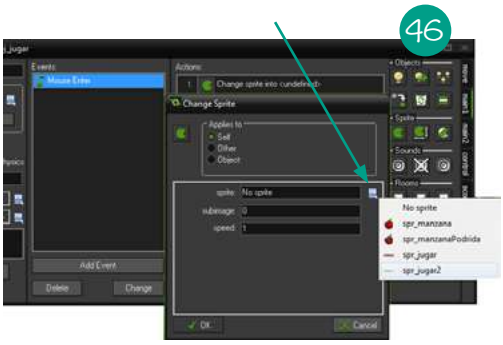


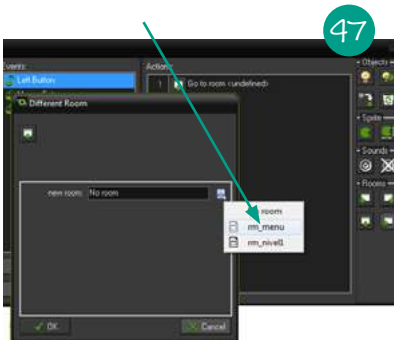
Ahora volvamos al objeto "obj_boton_jugar". Lo que queremos hacer es que cuando el puntero del mouse pase sobre el botón se ponga el sprite que está amarrillo. Por suerte, para eso hay un evento del mouse que se llama *Mouse Enter* (cuando el mouse está por arriba) que es justo lo que necesitamos. Agrégalo y a continuación busca en la solapa de acciones *main1* la acción  *change sprite* (cambiar el sprite) (45).

Al hacerlo se abrirá una ventana (46). Aquí, selecciona el sprite "spr_jugar2". Guarda y prueba qué es lo que sucede.

Si el juego no empieza en el menú es porque la room "rm_juego" está antes que "rm_menu" y Game Maker siempre ejecuta primero la room que se encuentre más arriba. Para arreglar esto simplemente selecciona la room "rm_menu" y arrástrala hasta arriba de la room "rm_juego", como muestra la figura 47.



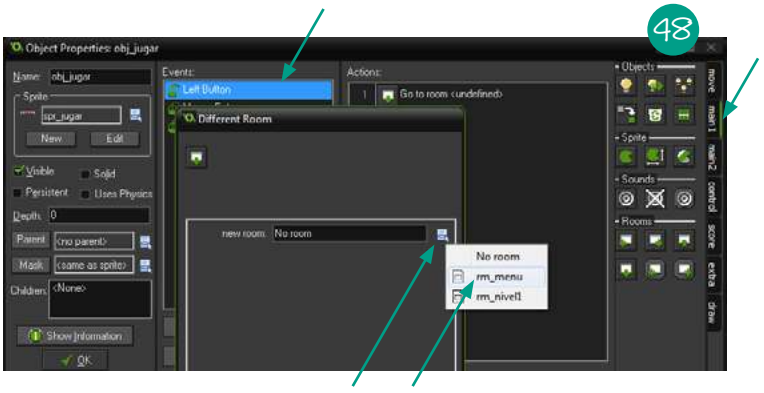




Ahora sí, guarda y prueba nuevamente. Como puedes notar, cuando el mouse pasa por encima del botón este se pone amarillo. Pero no vuelve a estar en rojo de nuevo, queda para siempre en amarillo.

La solución es agregar el evento *Mouse Leave* (cuando se va el puntero del mouse) y hacer que vuelva a cambiar el sprite con esta acción pero, esta vez, con el sprite normal "spr_jugar".

Ya está listo tu botón de jugar. Ahora lo que queremos es que al presionarlo (un evento que ya conocemos, *Left Button*) se pase a la room "rm_juego". Usa la imagen 48 como guía.



Ahora es tu turno de agregar el botón salir, usando lo que aprendiste hasta ahora. Cuando lo hagas y crees los eventos correspondiente busca entre los paneles de acción la acción *Quit game* para salir del juego al presionar el objeto "obj_salir".

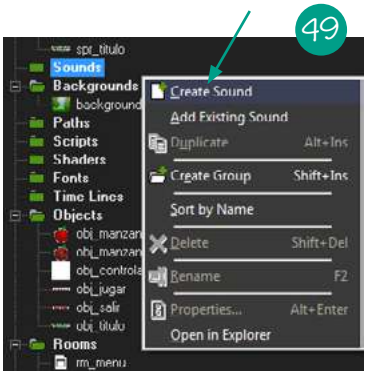
Por último, con lo que ya sabes, agrega un objeto con un sprite cargado con la imagen "titulo_atrapa_manzanas.png" y colócalo a modo de titulo. Quedará algo similar a esto:

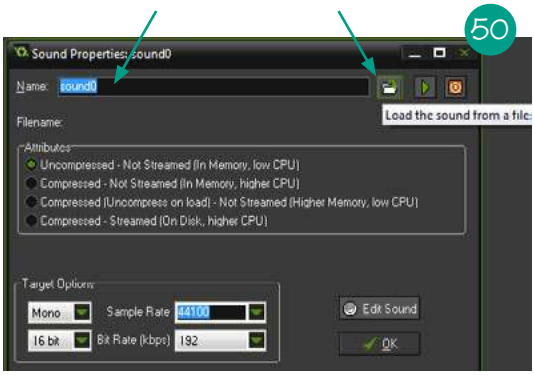
Es momento de agregar, por primera vez, sonidos para nuestro juego. Para hacerlo seguimos un proceso muy similar al de crear cualquier otro elemento. En la carpeta *Sounds* (sonidos) haz clic derecho y elige la opción *Create Sound* (49).

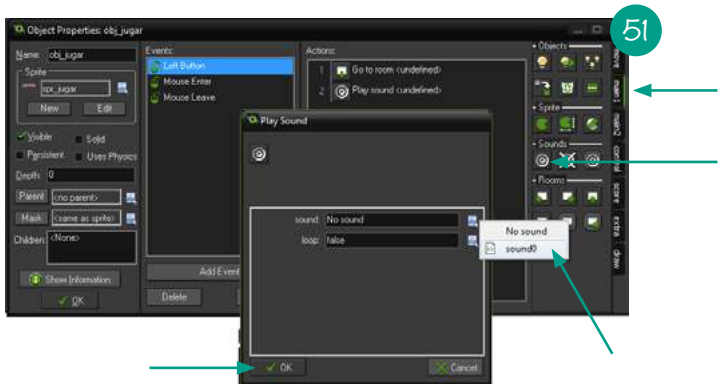
A continuación colócale el nombre "snd_boton" y haz clic en *Load sound*, como muestra la imagen 50.

Allí carga el sonido de la carpeta de recursos llamado "boton.wav" (si quieres puedes cargar otro) y haz clic en *OK*. Ya tienes el sonido agregado a Game Maker, ahora solo falta reproducirlo.

En el evento *Left button* del objeto "obj_jugar" agrega esta acción y selecciona el sonido "snd_boton" en el cuadro que dice "sound:", usando la imagen 51 como guía.







¡Perfecto! ¡Ahora el botón jugar hace ruido al ser presionado!

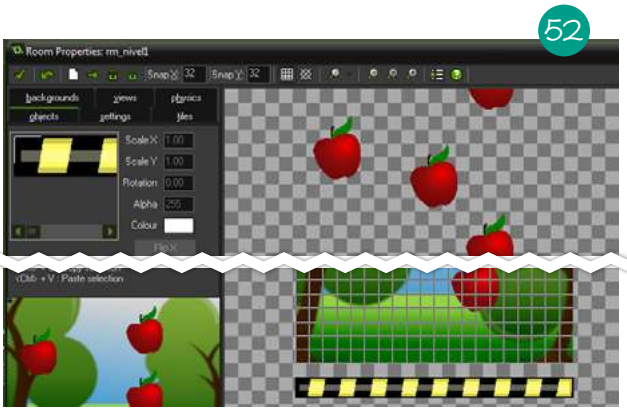
Con lo que has aprendido agrégale sonido también al botón salir, a cuando se agarran las manzanas comunes y a cuando se agarran las podridas (si quieres puedes utilizar los sonidos “manzana_agarra.wav” y “manzana_podrida.wav” de la carpeta de recursos).

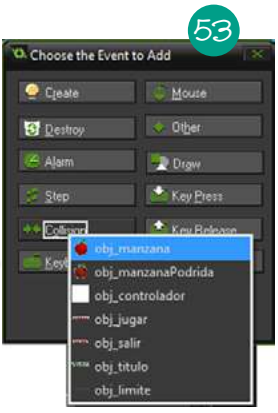
Para que las manzanas no queden tiradas por ahí cuando el jugador no las atrape tenemos que hacer que las manzanas desaparezcan al terminar de caer. Una forma sencilla de hacer esto es crear un objeto grande que elimine las manzanas cuando choquen contra él.

Crea un nuevo sprite, llámalo “spr_limite” y carga la imagen “limite.png”. A continuación, crea un objeto llamado “obj_limite” y colócalo al fondo de la pantalla, donde no se vea, tal como muestra la imagen 52.

Lo único que falta es encontrar un evento en el objeto “obj_manzana” que nos sirva para saber cuándo es que este objeto manzana choca contra el “obj_limite”. Este tipo de evento se llama colisión y es muy útil cuando programamos videojuegos.

Como puedes ver en la imagen 53, se puede detectar cuándo es que un objeto choca contra cualquier otro. En nuestro caso nos interesa el momento en que choca con el “obj_limite”. Asi que debes hacer clic *en Add event* -> “obj_limite”.





¡Perfecto! Una vez hecho esto, lo que tienes que agregar es la acción *Instance destroy* a este evento. Haz exactamente lo mismo pero con el "obj_manzanaPodrida". Ahora sí, todas las manzanas desaparecerán al caer.

¡Tu primer juego está casi terminado! Solo falta hacer que, cuando caigan todas las manzanas, el juego muestre el puntaje total y nos permita regresar al menú para decidir si queremos seguir jugando o no.

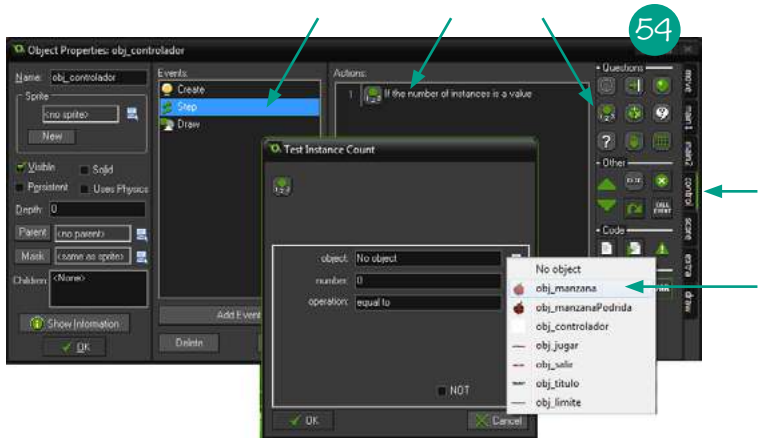
Lamentablemente no existe un evento en Game Maker que sea "si no hay manzanas terminó el juego". Pero no te asustes, hay una forma muy sencilla de hacer esto y solo tienes que aprender un par de conceptos más para lograrlo.

Primero que nada, existe un evento especial que usarás mucho en Game Maker que se llama *Step*. Este evento se ejecuta todo el tiempo mientras el objeto exista. Por ejemplo, si en este evento colocas la acción *Play sound* (reproducir sonido), creará un efecto muy molesto, ya que reproducirá el sonido una y otra vez, varias veces por segundo, sonando todos al mismo tiempo. ¡Puedes hacer la prueba! Pero te recomiendo bajar el volumen primero.

Segundo, hasta el momento solo hemos estado usando instrucciones que se ejecutan siempre que se produce el evento, sin importar nada más. Por eso que es momento de aprender un poco sobre los llamados "condicionales".

Los condicionales nos permiten controlar qué acciones se ejecutan o no dependiendo de la respuesta a una pregunta. Para entenderlo más fácilmente, analicémoslo con un ejemplo.

En nuestro caso necesitamos crear un evento s*tep* en el objeto "obj_controlador" para que verifique todo el tiempo la cantidad de manzanas. Para eso, necesitamos crear la condición "si hay cero manzanas hacer esto". En Game Maker es muy sencillo, usa la imagen 54 como guía.

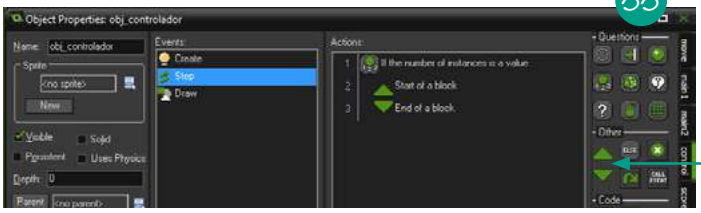


En el panel control de acciones hay acciones especiales, que en realidad son preguntas (los condicionales). En nuestro caso usaremos este , que nos permite preguntar por la cantidad que existe de un objeto. En la imagen de arriba puedes ver que donde dice "number:" hay colocado un "0", es decir que preguntará si la cantidad de manzanas es cero, justo lo que necesitamos.

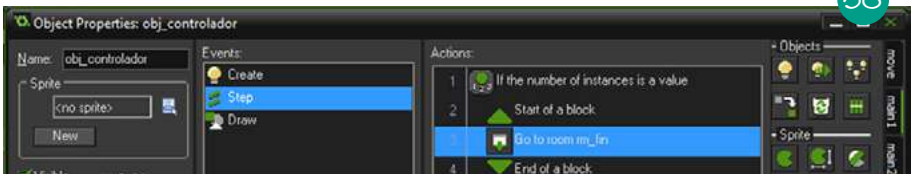
Lo que nos hace falta ahora es crear un lugar donde agregar las acciones que solo se ejecuten cuando la respuesta a la pregunta “¿hay cero manzanas?” sea “sí”. Para ello agregarás estas dos acciones de la manera que se ve en la imagen 55.

Ahora sí, todas las acciones que quieras que se ejecuten solo cuando haya cero manzanas van dentro de estas dos flechas, como muestra la imagen 56.

55



56



Llega el momento de lucirte. Debes crear una nueva room llamada “rm_fin” donde se muestre el puntaje del jugador y un botón que permita regresar al menú. Cuando la cantidad de manzanas sea cero debes hacer que el juego vaya a esta nueva room.

¡Felicitaciones! Has completado tu primer videojuego. En la siguiente sección, con lo aprendido aquí, haremos un juego de naves con disparos, *power ups* y ¡mucho más!

2.2.2. ¡Un juego de naves y disparos!

Ahora sí llego el momento de empezar con un proyecto más interesante. ¡Un juego de naves y disparos! Si no jugaste nunca un juego de este estilo, *Hydorah* es un ejemplo perfecto. Eso sí, es muy difícil.

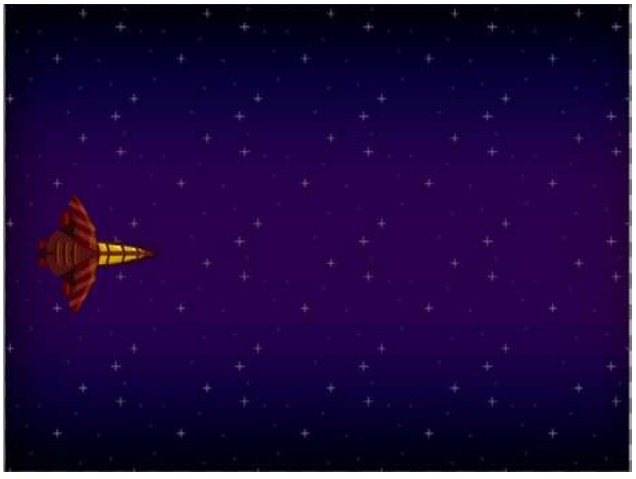
Hydorah es un juego de tipo *shooter* clásico basado en los antiguos estándares de diseño de videojuegos. No te sorprendas si pierdes muchas veces intentando pasar el primer nivel, es muy difícil.

Lo creó un desarrollador independiente apodado Loco Malito y puedes jugarlo gratis en el siguiente enlace: http://www.locomalito.com/hydorah.php

Para poder hacer este nuevo juego necesitarás las imágenes que se encuentran dentro de la carpeta de recursos (si quieres puedes usar otras, descargarlas de internet o dibujar las tuyas propias con un programa de edición de imágenes).

A diferencia del minijuego *Super atrapa manzanas*, este proyecto no va a estar explicado paso a paso ya que muchas de las cosas que se necesitan explicar se mostraron con el desarrollo del primer juego.

Todo juego de naves debe tener, por supuesto, una nave que pueda manejar el jugador. Para empezar debes crear un room llamado "rm_nivel", un objeto llamado "obj_nave", y un sprite llamado "spr_nave". También puedes agregarle un fondo para que no quede tan vacío. Luego agrega el "obj_nave" con el sprite "spr_nave" asignado al room, para que quede de esta manera:



Para poder manejar la nave con el teclado necesitamos usar los eventos correspondientes al teclado. Este evento 

se va a producir cuando se presione la tecla que seleccionaremos de la lista.

Empezaremos en este caso con la tecla UP (arriba). Lo que queremos lograr es que cuando se presione esta tecla la nave vaya hacia arriba. Para ello agregaremos la acción  con un valor, inicialmente, de 5. Guarda el juego y pruébalo. ¡Cómo puedes ver, la nave se dirige hacia abajo! Para hacer que vaya hacia arriba en la acción  que agregaste al evento *Keyboard up*, cambia el 5, por un -5. De esta manera se invierte el sentido y efectivamente irá para arriba. Guarda y pruébalo de nuevo.

Un problema que seguramente notaste es que la nave no frena cuando dejamos de presionar la tecla arriba. Una forma de arreglar esto es agregarle fricción a la nave para que frene lentamente. Es lo mismo que pasa con una bola de pool, que al ser empujada va frenando por la fricción que tiene con el paño verde de la mesa.

Por suerte, Game Maker cuenta con un mecanismo para incorporar fricción de forma sencilla. Para eso agrega un evento Create y usa la acción *Set friction*. Un buen valor para probar puede ser 0.5. Elige el valor que más te guste y pruébalo hasta que quedes satisfecho. Los valores más chicos harán que frene más lento y los más grandes que frene más rápido.

Ahora es tu turno de hacer que la nave vaya hacia abajo usando lo que aprendiste hasta ahora.

Ya casi está listo el movimiento de la nave, solo falta un pequeño detalle, ¡que no se escape de la pantalla! Para evitarlo haremos uso nuevamente de los condicionales. Recuerda, los condicionales nos permiten ejecutar acciones dependiendo de si se cumple una condición o no.

En este caso lo que queremos hacer es que la nave no pueda ir hacia arriba pasado un cierto punto. Para lograr esto, antes examinaremos un poco cómo funcionan las posiciones de los objetos en pantalla.

La posición "x" de un objeto determina cuán a la derecha o cuán a la izquierda está, es decir su posición horizontal. El valor inicial a la izquierda de la pantalla es 0 (aunque puede tomar valores negativos) y no hay valor máximo. Mientras más a la derecha esté el objeto, más grande va a ser el valor en "x" (57).

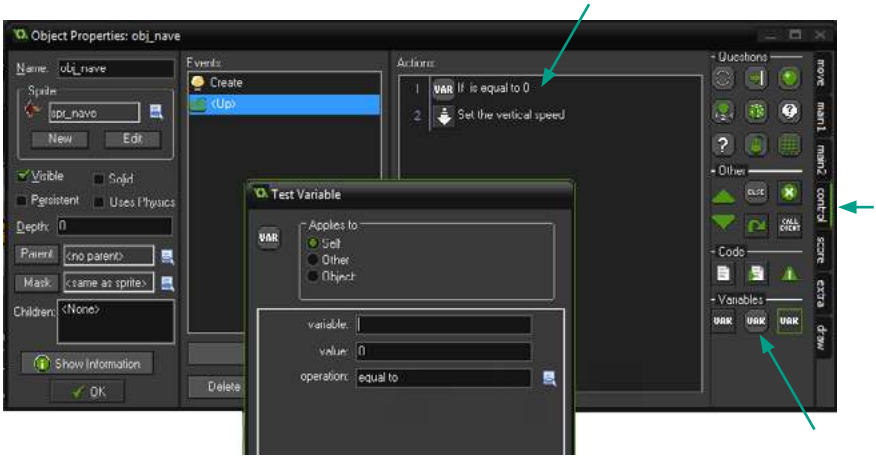
La posición "y", en cambio, determina cuán arriba o cuán abajo está el objeto, es decir su posición vertical. Cuanto más arriba está la nave, menor es el valor en "y" (58). La parte superior de la pantalla tiene valor 0. Sin embargo, pueden colocarse objetos más allá de esta línea y en ese caso la variable "y" toma valores negativos. Por ejemplo, en -50 la nave está 50 píxeles por arriba del inicio de la pantalla.

57

58

Habiendo explorado un poco acerca de cómo funcionan las posiciones X e Y, el paso siguiente es hacer que el jugador solo pueda mover su nave hacia arriba cuando la nave esté por debajo de la posición 0. Para eso, usarás el condicional VAR, que sirve para verificar el valor de una variable.

Cuando agregas esta condición dentro del evento *Keyboard Up*, te encontrarás con la siguiente ventana:



- *Variable*: el nombre de la variable. En este caso queremos saber la posición "y" (es importante que escribas "y" en minúscula).
- *Value*: el valor a verificar. Como queremos saber si la variable "y" es mayor que cero colocaremos 0.
- *Operation*: tiene tres valores posibles, *equal* (verifica si es igual al valor *value*), *smaller than* (verifica que la cantidad de objetos sea menor al valor *value*) y *greater than* (verifica que la cantidad de objetos sea mayor al valor *value*). En este caso queremos saber si la variable "y" es mayor a cero, por lo que colocaremos *greater than* (más grande que).

Recuerda encerrar entre los bloques  la acción  de setear la velocidad hacia arriba. De esta manera, si la condición se cumple, Game Maker ejecutará la acción que se encuentre dentro de las llaves. Finalmente deberá quedar así:

Guarda y pruébalo. Ahora es tu turno de hacer lo mismo con la tecla *Down* (abajo) para evitar que la nave se escape por debajo de la pantalla. Ya sabes todo lo que necesitas para hacerlo.

Si quieres puedes agregar que la nave se pueda mover horizontalmente, teniendo los mismos cuidados (usar los condicionales para que no se escape de la pantalla) y recordando que, para hacer movimiento horizontal, debes usar la variable "x".

Vamos a crear ahora, para darle un poco de diversión, un meteoro que el jugador deba esquivar para mantenerse con vida. Para eso, crea un sprite con un gráfico de meteoro y luego un objeto llamado "obj_meteoro".

Para que nuestro meteoro no se quede quieto, agrégale un evento *Create* (como recordarás, solo se produce al momento de creación del objeto) y asígnale velocidad horizontal con la acción .

Muy bien, ahora coloca en el nivel tu objeto “obj_meteoro”, unos píxeles por delante de la nave. Guarda y pruébalo.

Tal como pudiste observar, el meteoro se movió pero al chocar la nave... ¡no ocurrió nada!, ¡el meteoro atravesó al jugador y siguió de largo!

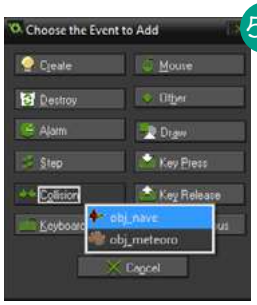
Lo que necesitamos hacer es detectar el momento en que el meteoro choca con el jugador y para ello, como habrás adivinado, necesitamos usar un evento nuevo.

El evento en cuestión se llama colisión y se produce cuando los objetos se chocan o se tocan.

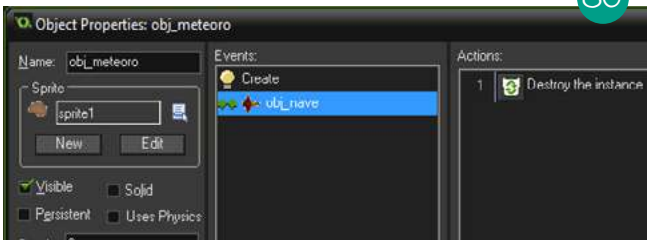
Dentro del objeto “obj_meteoro”, agregá un evento colisión contra el objeto “obj_nave”, tal como muestra la figura 59.

De esta manera se creará un nuevo evento que solo se disparará cuando los objetos se estén tocando.

Es el momento de intentar que reaccione nuestro meteoro, haciendo que, por ejemplo, se elimine el objeto meteoro con la acción , quedando como muestra la figura 60. Guarda y prueba.



59



60

¡Muy bien! Ahora sí, los meteoros se destruyen al chocar con el jugador. Es el momento de hacer lo mismo pero esta vez con la nave, es decir, crear un evento colisión con el objeto “obj_meteoro” dentro del objeto “obj_nave” y hacer que este se destruya.

Una vez que lo hayas hecho, podremos agregarle dificultad al juego haciendo que cada vez que el jugador choque un meteoro la partida se reinicie. Para lograrlo puedes utilizar la acción (*Game restart*) que se encuentra dentro de la solapa *main2*.

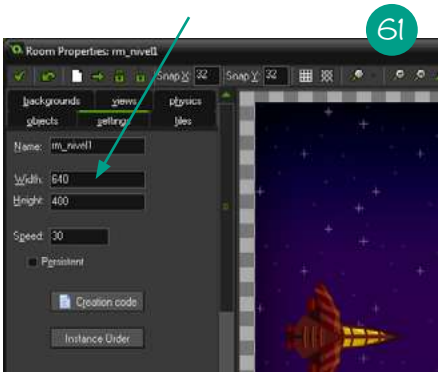
El nivel es apenas un pequeño cuadrado, ¿qué tal si lo agrandamos? Para hacerlo, tienes que ir al room del nivel y dentro de la solapa *settings* modificar

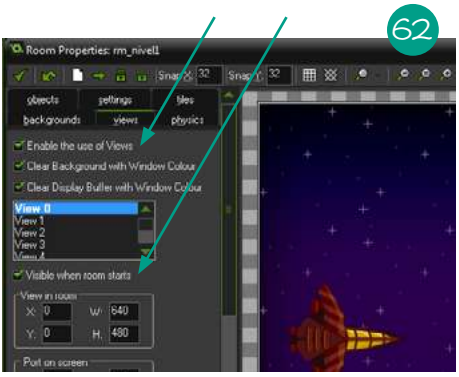
la propiedad *Width* (ancho) para hacerlo más largo, tal como muestra la figura 61. Puedes probar por ejemplo con el valor 2000. Guarda y pruébalo.

¿Qué ha pasado? La ventana del juego se estiró, quedando todo muy pequeño. Para arreglar eso haremos uso de las views.

Una view actúa como si fuera una pequeña ventana que nos permite ver solo una parte del nivel a la vez. Esto es justo lo que necesitamos.

Para ver tan solo la primera parte del nivel, ve al panel *view* del room y activa el uso de view tildando la opción *Enable the use of Views*. Haz lo mismo con la opción *Visible when room starts*, para que el uso de views esté activa desde el principio del juego. Si hiciste todo bien te debe quedar como muestra la imagen 62.





Guarda y prueba nuevamente. ¡Funciona perfecto!

Una de las últimas cosas que vamos a aprender es a hacer que nuestra nave dispare. Para eso primero necesitaremos crear un objeto "obj_disparo" y asignarle un sprite.

Lo primero es lograr que la nave pueda disparar presionando la tecla espacio, por lo que agregaremos el evento 

de la tecla espacio (*Space*). Ahora sí viene la parte nueva.

Para crear objetos, necesitamos usar la acción  *Create object moving*, es decir crear un objeto con movimiento. Al agregar esta acción al evento "Keyboard- Space" se abrirá la siguiente ventana:

En *object* nos pregunta por el objeto que queremos crear. En nuestro caso, el objeto "obj_disparo".

En donde dice *speed* coloca por ejemplo 10, que es la velocidad con la que moverá el disparo por la pantalla.

Por último, tilda la opción "relative" para que el disparo se cree en la misma posición que el jugador. Guarda y pruébalo.

Buenísimo, nuestra nave dispara... ¡pero tiene munición infinita! ¿Cómo podríamos evitar esto?

Una posible solución es crear una variable que limite la cantidad de disparos que puede hacer el jugador, es decir, un contador de munición.

Como habrás podido deducir, una variable es un lugar en donde se puede guardar información. Por ejemplo, como fue explicado antes, se puede almacenar la posición de la nave en las variables ya creadas X e Y.

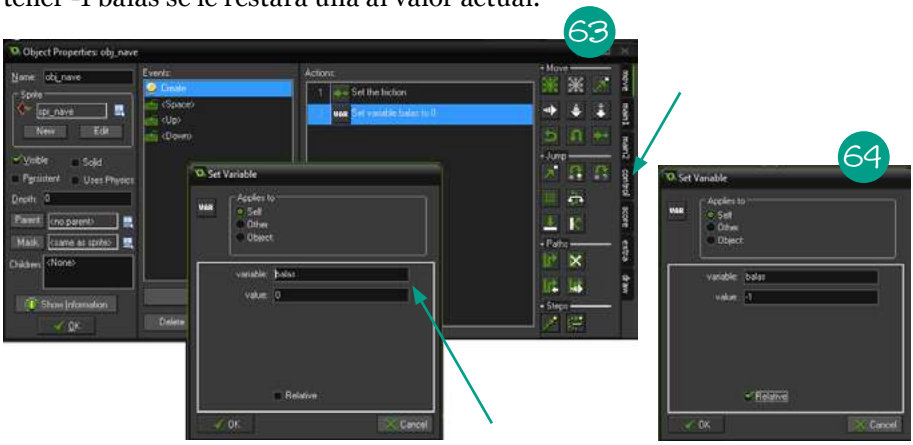
Desde un punto de vista más técnico, las variables son un lugar en la memoria de la computadora donde podemos guardar datos. Para usarlas en Game Maker, antes debemos declararlas (o inicializarlas) en el evento *Create* de un objeto y darles un nombre. Las variables creadas en un objeto (también llamadas variables locales) son propias de ese objeto.

Para el ejemplo que estamos siguiendo esta variable podría llamarse "balas", y podríamos inicializarla con un valor "10".

Lo siguiente sería agregar en el evento "se presiona la tecla espacio", es decir en donde se crea el objeto "obj_bala", que la cantidad de las mismas disminuya con cada disparo. Para eso deberíamos restarle 1 a la variable "balas" (63).

Para crear la variable, dentro del objeto "obj_nave", en su evento *Create*, debes agregar la acción *set var*. Donde dice "variable" coloca el nombre de la variable. En este caso sería "balas". En "value" coloca la cantidad de balas con las que te gustaría que empiece el juego, por ejemplo 10.

Ahora sí, para hacer que disminuya la cantidad de munición cuando dispara, debes agregar la acción *set var* al evento *key pressed space*, donde de valor le colocarás -1 y marcarás la opción *relative* (64). De esta forma, en lugar de tener -1 balas se le restará una al valor actual.



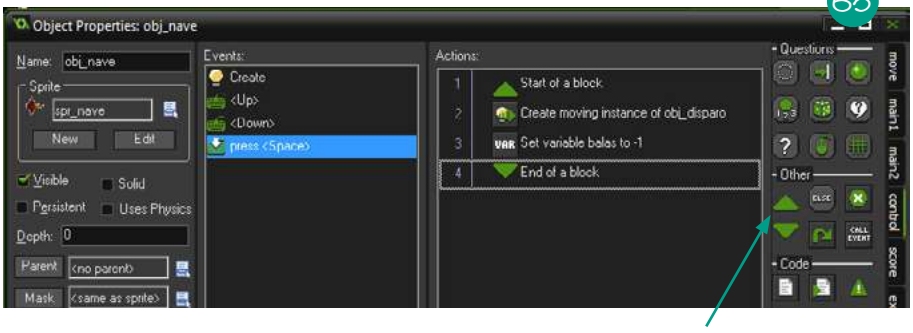
Guarda y prueba. ¿Qué ocurrió? La cantidad de balas disminuye con cada disparo pero la nave sigue disparando indefinidamente. Eso está bien porque no le dijimos que haga otra cosa.

Para evitar que siga disparando cuando se acabe la munición tenemos que utilizar los condicionales, tal como lo hicimos para limitar el movimiento de la nave.

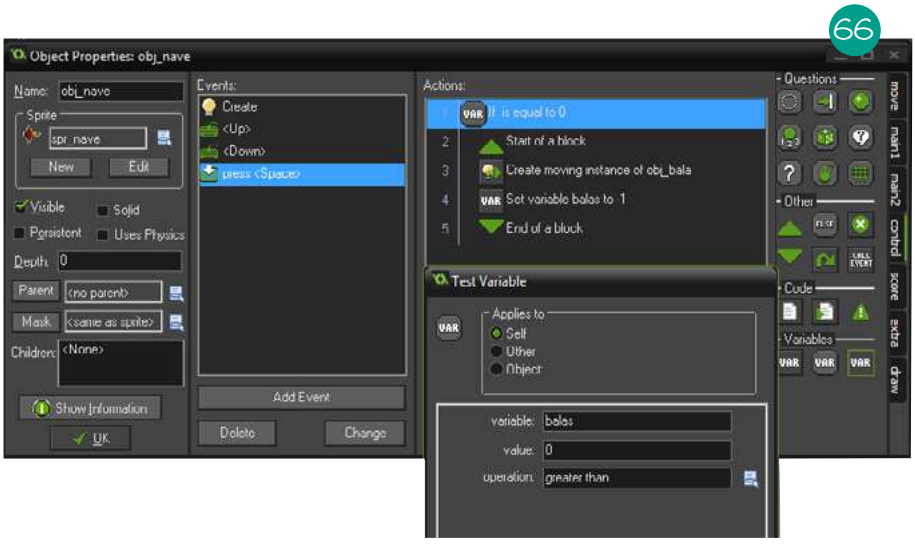
Para agregar esa condición debes colocar al inicio y al final de las acciones, dentro del evento *key pressed space*, las acciónes *start block* y *end block,* tal como muestra la imagen 65.

Luego agrega la condición “var” (66), pregunta por la variable “bala” y elige la opción *greater than* (queremos que solo dispare cuando tenga más que cero balas, es decir, ¡al menos una!). Como has notado, es muy similar a lo que has hecho con la posición “y” de la nave.

65



66



¡Ahora sí, la nave dispara hasta que se le termina la munición! ¿Y luego? Hace falta crear un objeto que le dé balas al jugador cuando este lo agarre.

Para eso tenemos que hacer uso del evento *Colisión* tal como has hecho con el meteoro y lograr que cuando la nave toque este objeto, se sume una cierta

cantidad de balas utilizando la acción *set variable* (recuerda marcar la casilla *relative* para que se sume a la cantidad actual de balas).

Por último, y aunque ya lo debes haber deducido, para hacer que las balas destruyan a los meteoros, debes agregar en el objeto "obj_meteoro", el evento *Colisión* con el "obj_bala" y finalmente agregar la acción "obj_destroy" al mismo.

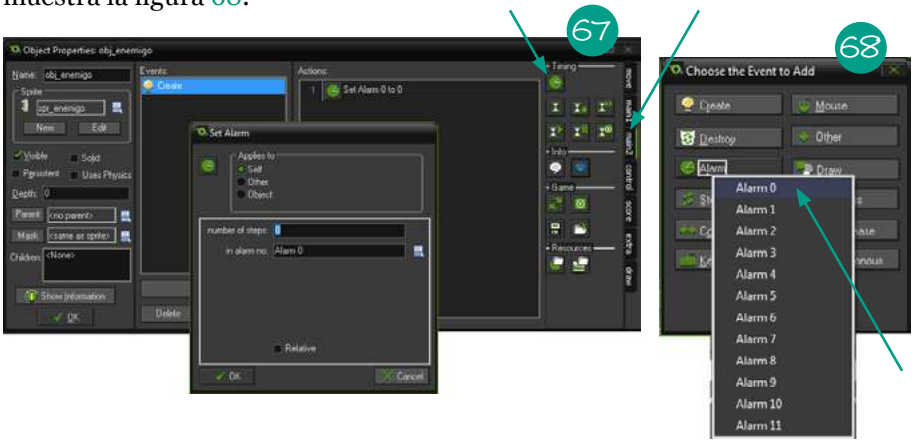
Bien, ya hay balas, meteoros, paquetes de munición y una nave. ¿Qué más faltaría? Agregar un enemigo que le dispare al jugador. Para eso, haremos uso de algo llamado "alarmas".

Las alarmas son propias de cada objeto y funcionan igual que la alarma de un reloj. Se señala en cuanto tiempo se quiere que la alarma "se dispare" y cuando lo hace se crea un evento *Alarm*. Son muy sencillas de utilizar.

Primero crea un objeto "obj_nave_enemiga" y luego en su evento *Create* colocas la acción *Set Alarm* (67).

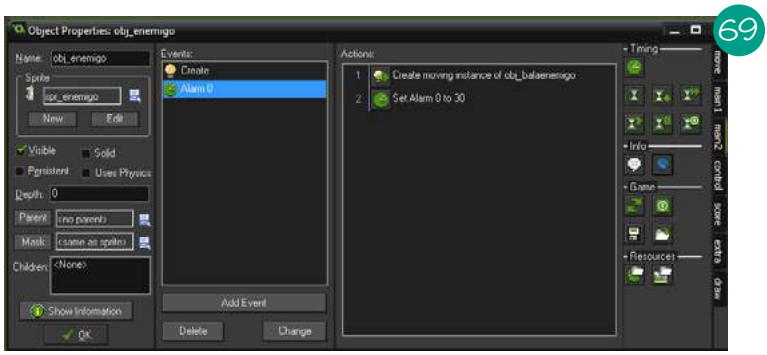
En el lugar donde dice "number of steps:", coloca el tiempo que quieres que espere la nave para comenzar a disparar (medido en cantidad de ciclos, en general 30 ciclos equivalen a un segundo). Prueba con un valor de 20. Luego en donde dice "in alarm no:" deja seleccionada la "Alarm 0" (Game Maker nos da la posibilidad de usar muchas alarmas en simultáneo).

Bien, pasados los 20 steps desde que se creó el objeto, se disparará el nuevo evento "Alarm 0". Para poder usarlo hay que agregar dicho evento como muestra la figura 68.



Ahora, tal como hiciste con el jugador y la tecla espacio, tienes que usar la acción *create object moving* para crear el objeto "obj_bala_enemiga" (previamente crearás el objeto "obj_bala_enemiga"). Una vez que lo hayas hecho, guarda y pruébalo (recuerda agregar el objeto enemigo al nivel, para poder verlo en acción).

¡La nave ha disparado solo una vez! La solución para que la nave dispare una y otra vez es hacer un truquito bastante simple. Dentro del evento "Alarm 0" coloca nuevamente la acción *Set Alarm* en 20 para que vuelva ponerse en funcionamiento la cuenta regresiva una y otra vez (69).



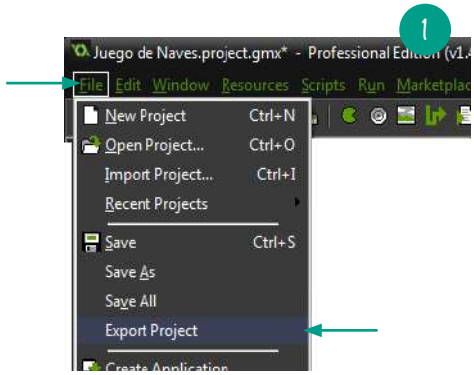
¡Por fin, hemos concluido! Cuentas con todo lo necesario para terminar tu propio juego de naves y lanzarte a la aventura de crear nuevos y mejores videojuegos.

Recuerda que en este libro cuentas con las secciones Eventos en Game Maker y Acciones en Game Maker, donde podrás descubrir otras acciones y eventos interesantes que ofrece Game Maker y pueden resultarte muy útiles.

2.3. ¿CÓMO COMPARTIR EL JUEGO?

Para poder compartir tu juego es necesario que lo exportes desde Game Maker. Exportar significa, en realidad, transformar tu proyecto en un ejecutable para la plataforma que elijas. Como estamos usando la versión gratuita de Game Maker, solo es posible exportarlo a Windows.

Puedes usar la imagen 1 como guía: tras elegir la opción *File* → *Export Project*, aparecerá una ventana de dialogo que te permitirá ponerle nombre a tu proyecto.



Ahora sí el archivo que puedes compartir es el que tiene de nombre “Tu-juego.exe”.

A la hora de compartir tu juego cuentas con muchas opciones: crear una cuenta en un servicio como Dropbox para subir archivos, usar Google Drive o bien enviarlo como archivo adjunto por mail o en mensajes de Facebook.

También existen sitios especializados como Game Jolt, en donde no solo puedes subir tu propio juego sino también imágenes, descripciones, ¡y hasta tener fans!

Sea cual sea la opción que elijas, recuerda que siempre es bueno acompañar tu link de descarga con una breve descripción de tu juego y algunas instrucciones mínimas acerca de cómo se juega. Eso sí, no te extiendas demasiado en el texto ya que en general a los jugadores no les gusta leer mucho.

Debes incluir obligatoriamente algunas imágenes del juego y asegurarte de que estas sean las más vistosas y representativas. De esta manera vas a atraer a los jugadores para que se animen a probarlo.

También puedes usar programas como CamStudio para grabar un video y subirlo a YouTube, e incluso editar este video usando programas como Windows Movie Maker en Windows y OpenShot en Linux.

A continuación van algunas consideraciones que debes tener en cuenta antes de subir tu juego a internet:

Usar recursos sobre los cuales tengas los derechos para usarlos. En muchos casos puedes encontrar muchos recursos gratuitos a través de internet que lo único que te piden a cambio de poder usarlos es darles crédito en tu juego.

Que tu nombre esté en los créditos del juego.

Comprobar que el juego funcione bien y no tenga errores graves que impida que se pueda jugar. Es muy común hacer cambios a último momento antes de subir el juego, solo para descubrir que estos cambios son los responsables de errores importantes.

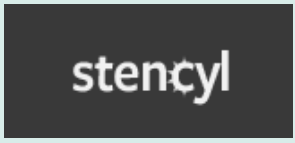
También puedes subir o compartir tu juego en sitios de comunidades de desarrolladores de juegos, como por ejemplo www.duval-vg.com, de Argentina. Cuando publiques tu juego allí (o en cualquier otro sitio de este estilo) pide consejo a otros desarrolladores, sé humilde y aprovecha cada crítica que recibas ya que te ayudarán a mejorar tu juego actual o futuros.

2.4. ALTERNATIVAS A GAME MAKER

Si por alguna razón Game Maker no te ha gustado o quieres buscar otras opciones similares que te dejen hacer juegos de forma fácil y rápida, a continuación hay una lista de donde puedes elegir y probar el que más te guste.

Game Develop: es una alternativa –completamente gratuita– para hacer los juegos en HTML5, lo que permite jugarlos en el navegador, en smartphones y en PC. Está disponible en idioma inglés, tanto para Windows como para Linux. Link al sitio oficial: http://www.en.compilgames.net/index.php

Construct 2: es muy sencillo de utilizar y dispone tanto de una versión paga como de una versión gratuita bastante limitada, pero que igualmente permite hacer juegos pequeños y grandes experimentos. En cuestión de minutos puedes tener un videojuego funcionando con este programa. Está disponible solo para Windows y puede exportar juegos en HTML5, al igual que Game Develop. Tiene muchísimas plataformas preparadas para exportar, desde smartphones hasta consolas. Link al sitio oficial: https://www.scirra.com/



Stencyl: funciona diferente a Game Develop y a Game Maker, pero dispone también de muchas facilidades a la hora de crear juegos. La versión gratuita permite exportarlos para que sean ejecutados en un navegador web. Está disponible tanto para sistemas Windows, Mac y Linux. Cuenta con una traducción al español. Link al sitio oficial: http://www.stencyl.com/

Pilas Engine: es un desarrollo argentino hecho por Hugo Ruscitti, se encuentra construido en el lenguaje interpretado Python y completamente en español. Lamentablemente no cuenta con una interfaz visual ni con editor de niveles, pero es un buen punto de partida para aquellos a quienes les interese más la programación de los videojuegos. Está disponible tanto para Linux como para Windows y permite exportar a ambas plataformas. Link al sitio oficial: http://pilas-engine.com.ar/

2.5. EVENTOS EN GAME MAKER

A continuación se encuentran los eventos más importantes en Game Maker con su descripción:

CREATE (CREACIÓN)

Este evento se produce una sola vez por objeto y se produce al momento de su creación. Si el objeto fue colocado en el editor de niveles, se produce en el mis-

mo momento en que empieza el nivel. Si el objeto es instanciado, por ejemplo un objeto "obj_bala", se produce en el momento en que se crea.

DESTROY (DESTRUCCIÓN)

Este evento se produce una sola vez por objeto y se produce al momento en que se destruye una instancia con la acción Instance destroy  .

ALARM (ALARMAS)

Las alarmas se pueden producir muchas veces por objeto. Se disparan en el momento en que se acaban la cantidad de steps que fueron configurados con una acción *set alarm*  .

Un ejemplo del uso de alarmas: para lograr que un objeto desaparezca a los 60 steps de ser creado, en el evento *create* se debe agregar la acción *set alarm*  y configurar a la alarm 0 en 60 step. Luego se debe agregar el evento *Alarm 0* y dentro del evento la acción *destroy*  .

STEP (PASO)

Este evento se produce todo el tiempo que el objeto existe. Es útil para ejecutar acciones usando condicionales.

COLLISION (COLISIÓN)

Este evento se produce cuando el objeto entra en colisión con otro objeto elegido. Se puede utilizar por ejemplo para hacer que el personaje pueda agarrar moneda.

Para ello, en el objeto "obj_moneda" se debe agregar el evento *colisión* con "obj_personaje", y agregar las acciones *destroy*  para eliminar la moneda y la acción *set score*  para cambiar el puntaje.

KEYBOARD (TECLADO)

En *Keyboard* están los eventos relacionados con el teclado. Estos eventos se producen siempre que la tecla elegida se encuentre presionada. Por ejemplo, se puede usar para mover al personaje al presionar la tecla "arriba".

MOUSE

Mouse contiene varios eventos que corresponden al estado del mouse.

- *Left button* (izquierdo), *Right button* (derecho) y *Middle button* (del medio): se producen siempre que el cursor del mouse esté sobre el objeto y se presiona alguno de estos tres botones.
- *No button* (ningún boton): cuando no se presiona ningún botón y el cursor está sobre el objeto.

- *Left pressed*, *right pressed*, *middle pressed*: se producen una sola vez, en el momento en que el cursor está sobre el objeto y se comienza a presionar el botón. Para que vuelva a producirse debe soltarse y volverse a apretar el botón
- *Left released*, *right released*, *middle released*: se producen una sola vez, cuando se deja de presionar el botón y el cursor está sobre el objeto.
- *Mouse enter*: se produce cuando el cursor entra por primera vez al objeto.
- *Mouse leave*: se produce cuando el cursor sale por primera vez del objeto.
- *Mouse Wheel up*: la ruedita del mouse sube y el cursor está sobre el objeto.
- *Mouse Wheel down*: la ruedita del mouse baja y el cursor está sobre el objeto.
- En *global mouse*, se encuentran los eventos generales del mouse. Es decir, cuando se producen todos estos eventos anteriores, pero el mouse puede estar o no sobre el objeto.

OTHER (OTROS)

En *Other* se encuentran varios eventos agrupados.

- *Outside room*: se produce cuando un objeto se sale del nivel. Es útil para eliminar objetos que ya no van a volver a aparecer en pantalla.
- *Intersect boundary*: se produce cuando un objeto está en el borde del nivel.
- *Views -> Outside View 0..11*: se produce cuando el objeto está fuera del área visible de esa view.
- *Views -> Boundary View 0..11*: se produce cuando el objeto está en el borde del área visible de esa view.
- *Game Start*: se produce cuando comienza el juego.
- *Game end*: se produce justo antes de salir del juego.
- *Room start*: se produce cuando empieza el nivel.
- *Room end*: se produce cuando termina el nivel.
- *No more lives*: se produce cuando se acaban las vidas.
- *No more health*: se produce cuando se acaba el health.
- *Animation end*: se produce cuando termina la animación del sprite.

DRAW (DIBUJADO)

Este evento se produce todo el tiempo, al momento en que el juego dibuja cosas en pantalla. Es útil para ejecutar las instrucciones que tienen el borde naranja. Cuando se coloca una acción dentro del evento *Draw*, el sprite del objeto deja de dibujarse automáticamente; si se necesita dibujar el sprite además de ejecutar dicha acción, se debe agregar la acción *draw self*  .

Dentro del evento *Draw* hay tres opciones:

- *Draw*: se produce todo el tiempo al momento en que el objeto dibuja cosas en pantalla.
- *Draw GUI*: se produce todo el tiempo luego de que los objetos dibujen sus sprite en pantalla. Es útil para mostrar puntajes y textos sin que sean tapados por otros objetos.

- *Resize*: se produce cuando se cambia el tamaño de la ventana del juego.

KEY PRESS (SE PRESIONA UNA TECLA)

Se produce una sola vez, cuando se presiona la tecla elegida. Puede ser útil, por ejemplo, para hacer que un personaje dispare cuando se presiona la tecla espacio. Aunque se mantenga presionada esa tecla, no se crean más disparos ya que el evento solo se produce una vez. Se volverá a producir si se suelta la tecla y se la presiona nuevamente.

KEY RELEASE (SE SUELTA UNA TECLA)

Se produce una sola vez, cuando la tecla elegida se suelta. Para que se vuelva a producir debe presionarse la tecla nuevamente.

2.6. ACCIONES EN GAME MAKER

En este mini manual vas a encontrar un resumen de las acciones más importantes.

Las acciones marcadas con un borde naranja solo tienen efecto cuando son colocadas en el evento *draw*.

Las acciones con un borde hexagonal son condicionales.

Move (mover): en este panel de acciones se encuentran todas las acciones que afectan a la posición del objeto en pantalla.

Move fixed (movimiento fijo): permite darle al objeto velocidad hacia uno o más direcciones utilizando para eso el cuadro de flechas en la sección *Directions* y eligiendo un valor en la opción *Speed*. Si está marcada la opción *Relative*, se suma la velocidad en esa dirección a la velocidad y dirección actual del objeto. Si se elige el botón del centro (sin marcar *Relative*) se detiene el objeto en ese lugar.

Move free (movimiento libre): permite especificar en grados la dirección del objeto. Tiene dos parámetros: *Direction* (el ángulo) y *Speed* (la velocidad).

Move towards (moverse hacia): permite que el objeto se dirija hacia un punto específico. Tiene tres parámetros: X (la posición X donde se desea que vaya), Y (la posición Y donde se desea que vaya) y *Speed* (la velocidad).

Horizontal Speed (velocidad horizontal): permite cambiar la velocidad horizontal del objeto. Los valores positivos hacen que vaya hacia la derecha y los negativos hacia la izquierda. Su único parámetro es *Hor. Speed* (velocidad horizontal).

Vertical Speed (velocidad vertical): permite cambiar la velocidad vertical del objeto. Los valores positivos hacen que vaya hacia abajo y los valores negativos hacia arriba. Su único paraámetro es *Ver. Speed* (velocidad vertical).

Gravity (gravedad): permite asignarle gravedad a un objeto. Crea una fuerza que atrae a un objeto hacia un ángulo determinado. Tiene dos parámetros, *Direction* (la dirección de la fuerza de gravedad) y *Gravity* (la fuerza).

Reverse horizontal (invertir horizontal): invierte la velocidad horizontal. Ejemplo: si estaba yendo hacia la izquierda, al ejecutar esta acción se moverá hacia la derecha.

Reverse Vertical (invertir vertical): invierte la velocidad vertical. Ejemplo: si estaba yendo hacia la arriba, al ejecutar esta acción se moverá hacia la abajo.

Friction: agrega fricción que hará frenar al objeto. Su único parámetro es *friction* (la fuerza con que se frena). Típicamente se utilizan valores pequeños como por ejemplo 0.3. Cuanto más alto este valor más se frena el objeto.

Jump to (saltar hacia): coloca al objeto en una posición X e Y determinada por los parámetros.

Jump to start (saltar hacia el inicio): vuelve a colocar al objeto en la posición X e Y donde fue creado.

Jump to random (saltar hacia un punto aleatorio): pone al objeto en una posición aleatoria en el nivel.

Warp screen (reaparecer al salir de la pantalla): permite que el objeto, al salir de los límites del nivel, ingrese desde la posición opuesta. Tiene un único parámetro con 3 valores posibles:

- *Horizontal*: solo permite que el objeto reaparezca si se escapa de la pantalla desde la derecha o izquierda.
- *Vertical*: solo permite que el objeto reaparezca si se escapa de la pantalla desde arriba o abajo.
- *Both*: ambos.

Main 1 (principal 1): en este panel se encuentran las acciones que permiten crear objetos, cambiar sprites, reproducir sonidos y cambiar el nivel.

Create instance (crear instancia): permite crear una instancia de un objeto. Tiene tres parámetros:

- *Object*: una lista desplegable que muestra los objetos disponibles. Este es el objeto que va a crearse.
- *X*: la posición X donde va a crearse.
- *Y*: la posición Y donde va a crearse.

Si se marca la opción *Relative*, el objeto se crea en el mismo lugar donde se encuentra el objeto que ejecuta la acción.

Create moving (crear con movimiento): permite crear una instancia de un objeto. Tiene cinco parámetros:

- *Object*: una lista desplegable que muestra los objetos disponibles. Este es el objeto que va a crearse.
- *X*: la posición X donde va a crearse.
- *Y*: la posición Y donde va a crearse.
- *Speed*: velocidad.
- *Direction*: el ángulo de dirección del objeto.

Si se marca la opción *Relative*, el objeto se crea en el mismo lugar que el objeto que ejecutó esta acción.

Create random (crear aleatorio): de entre 4 posibles objetos, crea uno al azar. Tiene por parámetros:

- *Object 1, 2, 3, 4*: el objeto que puede llegar a crear.
- *X*: posición X.
- *Y*: posición Y.

Destroy (destruir): destruye el objeto.

Change Sprite (cambiar sprite): permite cambiar el sprite que usa el objeto. Tiene tres parámetros:

- *Sprite*: el sprite a usar.
- *Sub image*: si está animado, qué cuadro debe mostrarse en pantalla.
- *Speed*: velocidad de la animación.

Transform Sprite (transformar sprite): permite cambiar el tamaño del sprite. Tiene cuatro parámetros:

- *X Scale*: el tamaño en ancho (1 es el tamaño normal, 2 el doble, 0.5 la mitad).
- *Y Scale*: el tamaño en alto (1 es el tamaño normal, 2 el doble, 0.5 la mitad).

- *Angle*: el ángulo de rotación del sprite.
- *Mirror*: si está espejado o no el sprite, y en qué dirección.

Color Sprite (colorear el sprite): le da color al sprite. Tiene dos parámetros.
- *Color*: el color (blanco deja al sprite sin cambios).
- *Alpha*: transparencia (1 es totalmente opaco, 0 es totalmente transparente).

Play sound (reproducir sonido): reproduce un sonido de los recursos de audio agregados al proyecto.
Sound: el sonido a reproducir.
Loop: si se tiene que repetir indefinidamente.

Stop sound (parar sonido): detiene un archivo de audio que se está reproduciendo.
Sound: el sonido a detener.

Previous room (volver al nivel anterior)

Next room (ir al nivel siguiente)

Restart room (reiniciar el nivel)

Diferent room (ir a un nivel diferente): ir a un nivel particular que se especifica en el parámetro *new room*.

Main 2 (principal 2): en este panel están las acciones que permiten usar las alarmas, reiniciar, detener y guardar el juego, y otras más.

Set alarm (configurar alarma): Permite configurar una alarma para que se dispare en X cantidad de steps (pasos).
Steps: número de pasos.
Alarm: la alarma a disparar (de Alarm0 a Alarm1).

Display message (mostrar mensaje): muestra una ventana con un mensaje de texto.

Open URL (abrir URL): abre una página web determinada por el parámetro URL.

Restart Game (reiniciar juego): reinicia el juego.

End Game (terminar juego): sale del juego.

Save game (guardar el juego): guarda el estado del juego en un archivo cuyo nombre se especifíca en el parámetro "file name".

Load Game (cargar el juego): carga el estado del juego desde un archivo cuyo nombre se especifíca en el parámetro "file name".

Control: en este panel se encuentran las opciones que permiten hacer condicionales.

Colisión free (libre de colisiones): verifica que un punto (X, Y) se encuentre libre de colisiones. Es decir, que no haya ningún objeto allí.
X: posición X.
Y: Posición Y.
Objects: tiene dos valores posibles, *Solid* (solo detecta los objetos que tengan marcada la opción solid) y *All* (verifica todos los objetos).

Check Object (chequear objeto): verifica que un objeto particular se encuentre en un punto X, Y.
X: posición X.
Y: Posición Y.
Objects: el objeto que debe encontrarse en ese lugar.

Test instance count (chequear la cantidad de objetos): verifica la cantidad que hay de un objeto en particular.
Object: el objeto a verificar.
Count: la cantidad que usa de referencia.
Operation: tiene tres valores posibles, *equal* (verifica si es igual a la cantidad count), *smaller than* (verifica que la cantidad de objetos sea menor a la cantidad count) y *greater than* (verifica que la cantidad de objetos sea mayor a la cantidad count).

Test change (verificar la oportunidad): permite arrojar un "dado virtual" de X cantidad de caras (sides). Cuanta más cantidad de caras tenga, menos probable es que se ejecute la acción determinada por este condicional.

Question (pregunta): hace una pregunta y dependiendo de su resultado (si el jugador contesta YES) ejecuta la acción determinada por este condicional.

Check mouse (verificar mouse): verifica si un botón del mouse se encuentra presionado.
Button: el boton puede ser *none* (ninguno), *left* (izquierdo), *right* (derecho), o *middle* (el del medio).

Start block (comenzar bloque): permite comenzar a marcar un grupo de acciones.

End block (terminar bloque): permite cerrar un grupo de acciones.

Comment (comentario): permite agregar un comentario en el código.

Else (de lo contrario): si la condición anterior no se ejecutó, se ejecutará la siguiente o el siguiente grupo de acciones.

Repeat (repetir): se repite una cantidad fija de veces el siguiente bloque de acciones.

Set var (asignarvariable): le asigna un valor "value" a una variable, que tiene de nombre "variable". Si se pone en el evento *create*, crea esta variable, y si se marca la opción *relative*, se suma (o resta dependiendo del signo) el valor "value" a la variable.

Check var (verifica la variable): verifica el valor de una variable.
Variable: el nombre de la variable.
Value: el valor a verificar.
Operation: tiene tres valores posibles, *equal* (verifica si es igual al valor *value*), *smaller than* (verifica que la cantidad de objetos sea menor al valor *value*) y *greater than* (verifica que la cantidad de objetos sea mayor al valor *value*).

Draw var (dibujar variable): dibuja el valor de una variable en una posición X e Y determinada. Solo tiene efecto si se coloca en un evento *draw*.

Score (puntaje): en este panel se encuentran todas las acciones relacionadas con el puntaje.

Set score (asignar puntaje): le asigna un valor al puntaje. Si la opción *relative* está marcada, se suma (o resta, dependiendo del signo) este valor al puntaje previo.

Test score (verifica el valor del puntaje): es un condicional que verifica el valor del puntaje.
Value: el valor de a verificar.
Operation: tiene tres valores posibles, *equal* (verifica si es igual al valor *value*), *smaller than* (verifica que la cantidad de objetos sea menor al valor *value*) y *greater than* (verifica que la cantidad de objetos sea mayor al valor *value*).

Draw Score (dibujar puntaje): dibuja el puntaje en una posición X e Y determinada. El parámetro *caption* permite agregarle una etiqueta, por ejemplo: "Puntaje:" o "puntos:". Está acción solo tiene efecto si se coloca en un evento draw.

Draw (dibujado): aquí están todas las acciones que tienen que ver con el dibujado de imágenes y textos. Para tener efectos deben estar colocados en un evento *draw*.

Draw self (dibujarse a sí mismo): dibuja el sprite asignado al objeto. Cada vez que se utiliza el evento *draw*, el sprite deja de dibujarse en pantalla automáticamente. Para hacerlo, se debe usar específicamente esta acción.

Draw Text (dibujar texto): permite dibujar el texto *text* en una posición X e Y determinada.

Este conjunto de acciones permite dibujar figuras, sprites y backgrounds.

Set full screen (configurar pantalla completa): cambia el modo en que se ve el juego.
Tiene tres valores posibles: *window* (el juego se muestra en una ventana), *fullscreen* (el juego se muesta en pantalla competa) o *switch* **(**intercambia entre modo window y fullscreen).

Draw effect (dibujar efecto): permite crear un efecto de partículas. Es la única acción del panel *draw* que se puede utilizar fuera del evento *draw*:

- *Type*: el tipo de efecto. Puede ser explosión, ring, ellipse, firework, etc.
- *X*: posición X.
- *Y*: posición Y.
- *Size*: el tamaño, puede ser *small* (pequeño), *medium* (medio), *large* (grande).
- *Color*: el color.
- *Where*: tiene dos valores, *below* (se dibuja debajo de los objetos) o *above* (se dibuja arriba de los objetos).

El juego como herramienta de expresión

Los videojuegos, además de ser divertidos, tienen la capacidad de transmitir ideas y sentimientos. Más allá de las historias de ficción que puedan llegar a contar a través de textos, personajes y cinemáticas, existe una narrativa que es propia de los juegos, en la que el único protagonista es el jugador y su relación con el juego.

Distintos desarrolladores han utilizado los videojuegos para transmitir sus opiniones y su manera de ver el mundo utilizando juegos.

El investigador uruguayo Gonzalo Frasca creó un juego llamado *12 de septiembre*, donde el jugador puede eliminar terroristas lanzado misiles sobre una ciudad afgana. Sin embargo, cada misil que el jugador utiliza hace que las personas que viven en la ciudad se horroricen por tales actos (generalmente, muchos civiles terminan muertos por estos ataques) y terminen convirtiéndose en terroristas.

En el juego *Passage*, Jason Rohrer intenta reflejar la vida de una persona como un recorrido en el cual uno no puede deshacer las decisiones que toma.

Los videojuegos se diferencian de otros medios artísticos en que no solo pueden mostrar sentimientos sino también generarlos a partir de las acciones y decisiones que toman los jugadores.

Los temas sobre los que se pueden hacer videojuegos no tienen límites. No estás obligado a hacer el próximo juego de zombis o sobre *aliens* espaciales. Puedes hacer un juego para dedicárselo a un amigo o amiga, sobre las cosas que te pasan en tu vida diaria o, simplemente, sobre aquello que te guste.

A continuación podrás ver varios juegos únicos que te ayudarán a convencerte de que puedes hacer juegos acerca de lo que quieras, sin necesidad de restringirte a géneros conocidos

- TODAY I DIE: es un juego hecho por el desarrollador argentino Daniel Benmergui, en donde el jugador construye un poema interactuando con lo que sucede en el juego.

- BACK HOME: es un juego que sintetiza, y extiende de alguna manera, un libro clásico como *El Principito*. Lo desarrolló el estudio de desarrollo santafesino KillaBunnies.

- PEWMA: creado por el estudio argentino Chimango Games. El jugador debe manejar un ave mística tratando de lograr su récord. Está ambientado e inspirado en leyendas de la cultura mapuche.

- ASYLUM: lo creó el estudio argentino Senscape, es un juego de terror psicológico inspirado en las novelas de H.P. Lovecraft y ambientado en una gigantesca y decadente institución mental.

Como pudiste notar, además de ser variados, todos estos juegos son desarrollos argentinos. Sí, ¡se pueden hacer juegos en Argentina! Si tienes acceso a una computadora y a internet, ya no tienes excusas para empezar a experimentar.

3.1. JUEGOS COMO PROTESTA, JUEGOS EDUCATIVOS, DE REFLEXIÓN, PERSONALES, O SIMPLEMENTE DIVERTIDOS

Como ya vimos anteriormente, los temas de los que pueden tratar los videojuegos son muchos y muy variados pero también existen diversos motivos por los cuales los juegos son creados.

Existe una categoría de juegos conocida como "juegos serios" que engloban a aquellos juegos que tienen un fin más allá de solo divertir. A continuación una descripción de las cuatro categorías más significativas.

Juegos educativos: intentan introducir conocimientos de diversas áreas a través de un videojuego. Tienen la particularidad de ser muy difíciles de diseñar ya que deben hacer coexistir dos tipos de reglas que son, en general, distintas: las del juego y las de los conceptos que se quieren transmitir. Esta es la razón por la cual no hay muchos juegos educativos que sean buenos y divertidos. Un buen juego educativo, por lo tanto, es aquel en el que el conocimiento y los patrones que el jugador descubre y aprende a través del dominio de sus dinámicas son semejantes a ciertos conceptos de la vida real que el juego pretende enseñar.

Juegos publicitarios: se los conoce como advergames y son creados para hacer publicidad a un producto. Su objetivo es, además de entretener, presentarles ese producto a los potenciales clientes. Estos juegos muestran publicidad de un solo producto ya que fueron creados especialmente para ello. Existen muchos juegos que muestran publicidad, por ejemplo los juegos de Smartphone, pero estos no son advergames ya que aparecen muchos productos y los juegos no tienen relación con los que se publicita en estos anuncios genéricos (conocidos como *ads*).

Juegos artísticos: intentan mostrar una visión de los creadores acerca de ellos mismos o del mundo que los rodea. Suelen tratar temáticas diversas y en

general intentan transmitir un mensaje que se entremezclan con las mecánicas de los juegos. Un ejemplo es el ya mencionado *Passage*, de Jason Roher.

Juegos de entrenamiento: rozan lo que se conoce como simuladores y se enfocan en entrenar a las personas para desarrollar determinada tarea, por ejemplo manejar una máquina compleja, resolver situaciones de emergencia, etc.

Sea cual fuere la temática, y fuera cual fuera tu motivo para hacer tu juego, lo que hemos visto en este pequeño libro te servirá para embarcarte en la aventura.

Sobre el autor

ALEJANDRO ADRIÁN IGLESIAS es licenciado en Sistemas de Información por la Universidad Nacional de Luján, donde trabaja como docente y como administrador de sistemas. Es jefe de trabajos prácticos del Departamento de Tecnología de la UNIPE e integra, en esa misma universidad, el equipo de investigadores del Laboratorio de Investigación y Formación en Nuevas Tecnologías Informáticas Aplicadas a la Educación. Participó en el desarrollo de videojuegos independientes y sus intereses están centrados en el uso de tecnologías de forma creativa, como herramienta de expresión y de empoderamiento ciudadano. Es autor del libro *Diseño y construcción de objetos interactivos digitales: experimentos con la plataforma Arduino*, publicado por UNIPE: Editorial Universitaria en 2015.

unipe: editorial universitaria

HERRAMIENTAS
SERIE TIC

Tras haber transitado un largo camino, los videojuegos se han afianzado como una forma de expresión que combina narrativa, diseño gráfico, música y conocimientos de informática. En *¡Quiero hacer un videojuego! Un e-book para niños y niñas de 8 a 99 años*, Alejandro Iglesias hace un breve recorrido por la historia y los elementos básicos que los componen para luego sumergirnos en la experiencia de crearlos. A través de una detallada y amena guía visual, aquí podrán incursionar en el desarrollo de un videojuego de la mano de ejercicios y experimentos básicos. Lejos de promover el viejo antagonismo entre aprendizaje y ocio, este libro propone pensar la creación de un juego electrónico como un puntal para la alfabetización digital activa de niños, jóvenes y adultos.

ISBN 978-987-3805-20-2

9 789873 805202

editorial.unipe.edu.ar