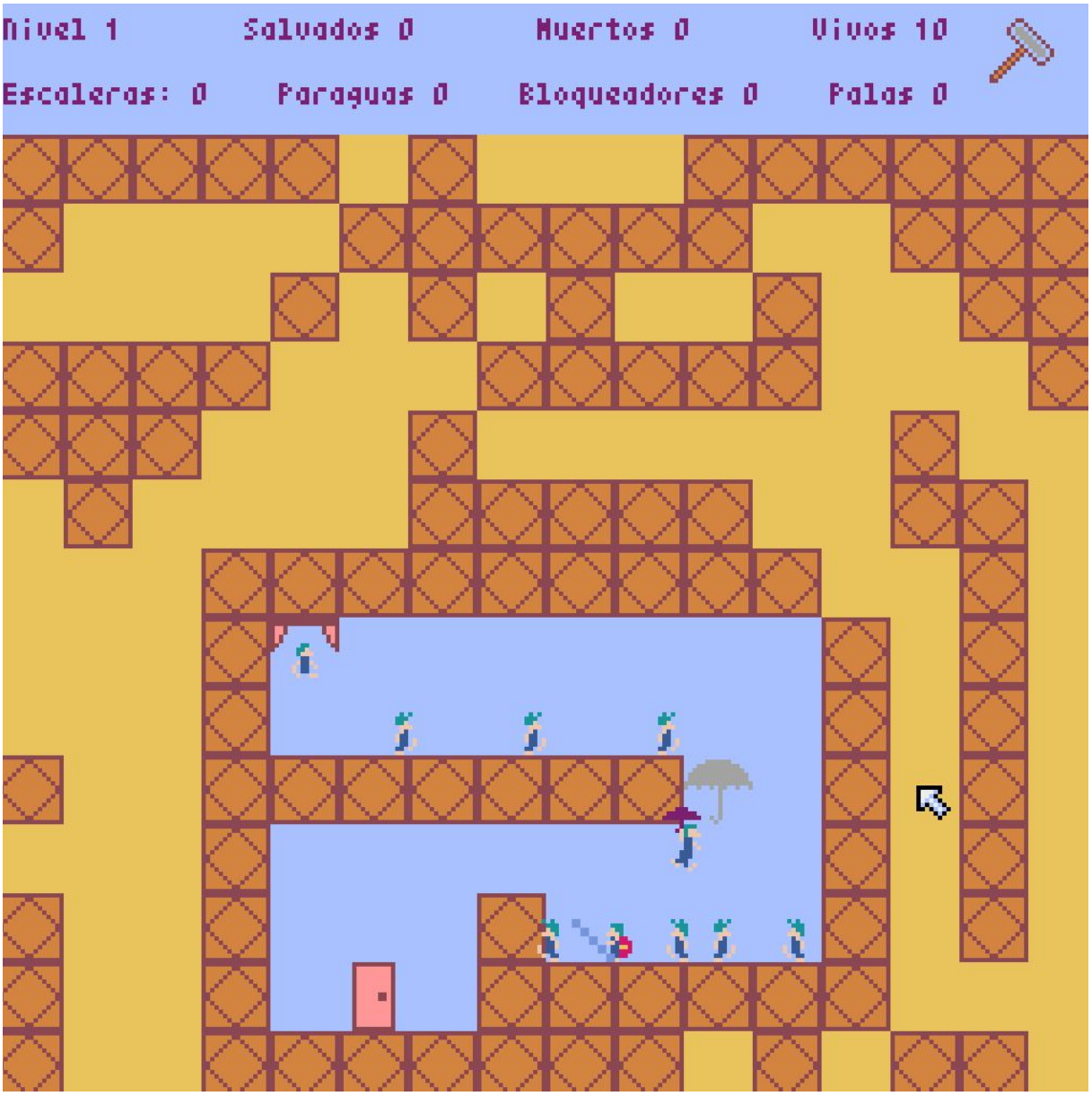
MEMORIA LEMMINGS



AUTORES

DANIEL FERNÁNDEZ OCAÑA (Grupo 81)

RAÚL ÁGREDA GARCÍA (Grupo 84)

Índice

Introducción

Lemmings se trata de un juego en el que unos personajes se mueven por la pantalla sin la posibilidad de ser controlados por el usuario. Sin embargo, el trabajo de este, es aplicarles unas tareas para que puedan llegar al destino, sin morir por una caída muy alta, o encontrarse con unos pinchos que les puedan llegar a matar.

El objetivo de este documento es explicar las fases de creación del juego, haciendo hincapié en las partes que puedan llegar a ser más complejas de entender.

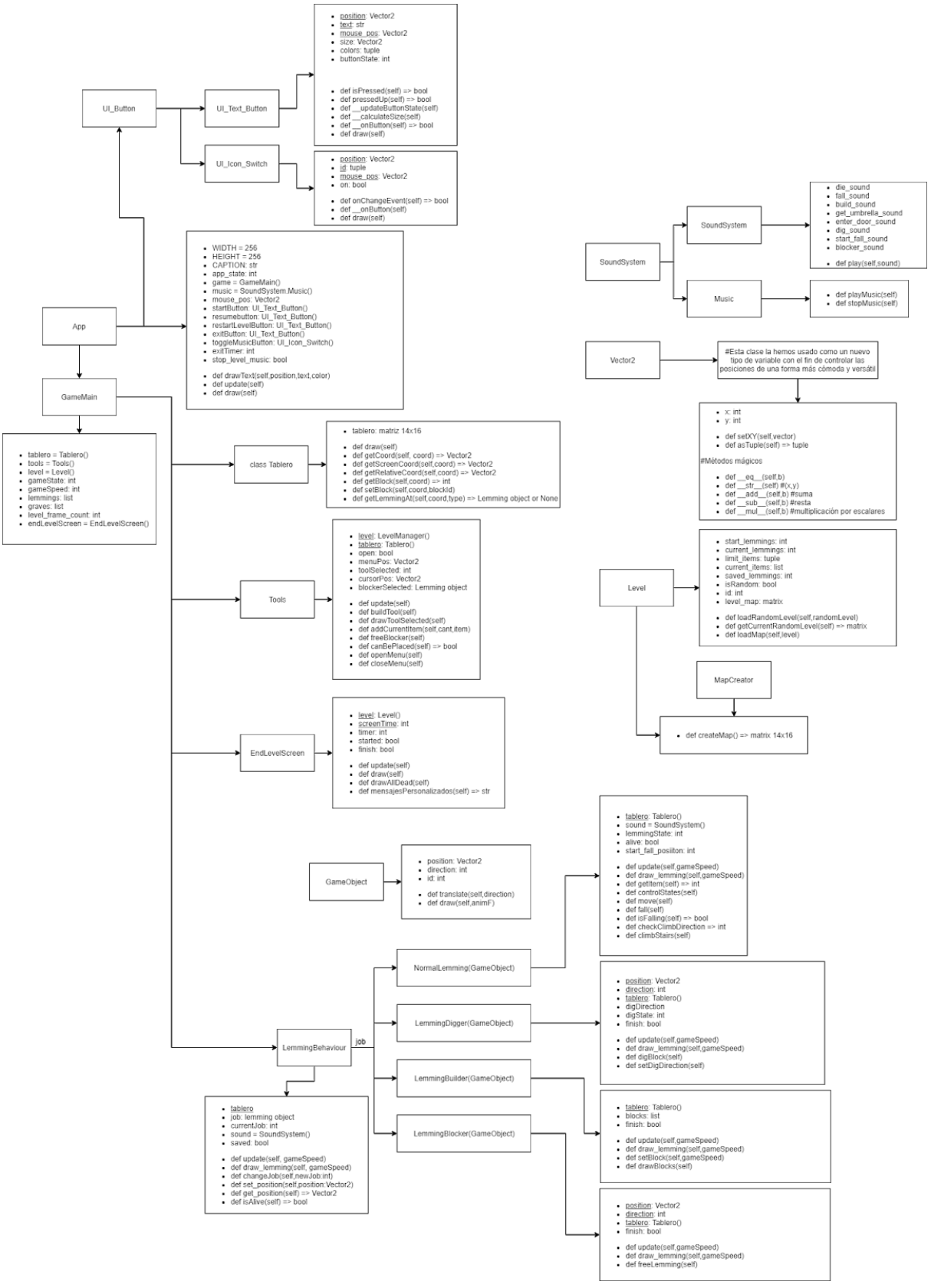
Instrucciones

Para empezar hay que extraer el archivo en una carpeta y abrir la consola de comandos en esta misma carpeta; después, para utilizar este juego solo hay que abrir el archivo “main.py” mediante la instrucción “python main.py” en el directorio del juego.

Una vez hecho esto ya se puede disfrutar del juego. Para que empiece, hay que pulsar el botón “Start”. Si quieres desactivar la música, reiniciar el nivel o salir del programa pulsa la letra “M” o la “P” para abrir el menú y seleccionar la función deseada.

Para abrir el menú de herramientas hay que pulsar click derecho en cualquier parte de la pantalla y sin soltarlo, arrastrar el ratón hacia la herramienta que se desea utilizar, una vez seleccionada, pasas el ratón por la casilla en la que utilizar la herramienta y si está tachada significa que no se puede usar en esa casilla, en el caso de que esté permitido, basta con hacer click para poner la herramienta. Si la herramienta no la ha usado ningún lemming aún, puede ser eliminada con la herramienta del martillo.

Diseño de clases



Método y algoritmos más relevantes

Clase principal

La clase App se encarga de poner todo en marcha, desde los botones hasta el tablero, inicializando todas las clases necesarias para que el juego funcione.

Dentro de esta clase, los dos métodos principales son el “update” y el “draw”, funciones de pyxel ejecutadas por la función pyxel.run() y que hemos tratado de dar a cada método su función:

En la función update se van a realizar todas las operaciones necesarias para que el programa funcione, como el movimiento del lemming o detectar el click del botón, en esta llamaremos a la función update de cada clase, por ejemplo la clase lemming tendrá una función update donde su posición será actualizada en cada frame y detectaremos la acción que debe hacer en cada momento, entre otras cosas.

La función draw, sin embargo, es la encargada de dibujar en la pantalla del juego mediante los métodos de pyxel, desde los lemmings hasta el tablero e incluso las herramientas. De la misma forma que en el update, en esta función también llamaremos a la función draw de todas las clases.

Clase Vector2

Esta clase la usamos como herramienta, crear un Vector2 es mucho más rápido que crear dos variables para cada coordenada, o incluso más cómodo que hacer una lista o tupla, además podemos utilizar operaciones directas entre dos vectores mediante operadores como suma, resta o multiplicación por escalares, esto se consigue gracias a los métodos especiales de python.

La clase Vector2 incluye muchas funciones útiles para el programa, como “magnitude”, la cual te devuelve el módulo del vector, o “distance” que te da la distancia entre un vector y otro mediante el teorema de pitágoras.

Clase Game Manager

La clase Game Manager es la encargada de controlar los niveles. A través de la función start, la cual toma por parámetro el nivel que se desea empezar. Cada vez que empieza un nuevo nivel, vuelve a crear los objetos necesarios como el tablero con un nuevo nivel cargado o reiniciar la lista de lemmings por ejemplo. Para controlar cuando ha terminado o no un nivel, se comprueba si están todos muertos o han finalizado el nivel, es entonces cuando se llama de nuevo a la función start con parámetro level+1, para que empiece el siguiente nivel.

Para spawnear a los lemming, hemos hecho uso de la función __spawn_lemming(), la cual, genera lemmings cada vez que pasa un tiempo determinado (para ello utilizamos pyxel.frame_count),

y se instancian siempre y cuando la lista que guarda los lemmings, no supera al límite predefinido de lemmings de cada nivel.

Clase Level Manager

La clase Level Manager es la encargada de cargar los niveles a partir de un índice el cual es pasado por parámetro. Además va guardando la información del nivel actual como el número de herramientas usadas, los lemmings que quedan, los que se han muerto, los que se han salvado (los que han llegado a la puerta) etc. Nosotros hemos hecho un número determinado de niveles, una vez terminan estos, los niveles empiezan a ser aleatorios. Para crear los niveles utilizamos la función loadLevel(self,level_id); lo primero que hace esta función es comprobar si el nivel va a ser aleatorio o no, haciendo uso de la clase LoadMap, donde guardamos las matrices de cada nivel que obtenemos mediante la función getMap(level); y los parámetros iniciales (lemmings al inicio, herramientas disponibles…) con la función getAtributes(level).

Función MapCreator

Esta función crea los mapas de manera aleatoria, para ello hace uso de distintos algoritmos. Para empezar crea un tablero lleno de ceros (sin bloques), después convierte la última fila y las columnas laterales en 1 (bloque por defecto). Lo siguiente que hace es repetir durante 7 veces (7 plataformas a crear) el siguiente algoritmo: escoger una fila al azar, luego decidir el tamaño de la plataforma y donde va a comenzar, para posteriormente rellenar estos bloques también con unos.

Una vez hecho eso, comprueba que en las columnas laterales no haya ningún 1, y en el caso de que lo haya lo sustituye por un 2 (bloque de arena). Para ello utiliza la función changeIn(lista, x, y), la cual cambia “x” por “y” en “lista”. Por último pone la entrada y la salida; para poner la entrada empieza a comprobar desde arriba a la izquierda, y va comprobando bloque por bloque hacia la derecha y hacia abajo si es posible poner ahí la entrada; para la salida hace el mismo procedimiento, pero empezando desde abajo a la derecha, y hacia la izquierda y hacia arriba.

Clase Tablero

La principal función de la clase tablero es interactuar con la matriz que se encarga de guardar que bloque hay en cada posición a lo largo del tiempo.

La primeras funciones a destacar son getBlock(), que nos indica la id del bloque en las coordenadas indicadas; y setBlock() que cambia la id del bloque deseado en unas coordenadas

concretas. La función getCoord() devuelve las coordenadas de la pantalla, en coordenadas de la matriz.

Clase Game Object

Es la clase de la cual heredan tanto la clase lemming como todas las encargadas de hacer sus trabajos(LemmingBuilder, LemmingBlocker, etc). Esta clase guarda la posición del lemming, la dirección de su movimiento y el id del dibujo, el cual varía dependiendo de varios factores. También se encarga de dibujarlo con su animación mediante la función draw.

Clase LemmingBehaviour

La clase LemmingBehaviour es la encargada de realizar las tareas básicas del Lemming, desde su movimiento hasta controlar los trabajos que debe hacer al coger una herramienta.

La función principal de esta clase es controlar los estados en los que se encuentra el Lemming, mediante su propiedad job, que toma el valor de otra clase que hereda de GameObject, y que se especializa en un trabajo específico. Para ello, mediante la función changeJob(self,newJob), el objeto asignado a job se cambia a uno de las siguientes clases: la clase NormalLemming, para el movimiento; LemmingBuilder, encargado de construir la escalera; LemmingBlocker, el lemming que bloquea el paso a los demás lemming; y LemmingDigger encargado de cavar/picar bloques.

Otras funcionalidades

Como elementos adicionales hemos añadido el trabajo de cavar o picar bloques con la clase LemmingDigger, que se encarga de que los lemmings destruyan unos bloques de tierra para abrirse paso; la música y los sonidos de los lemmings mediante la clase SoundSystem; también hemos añadido un menú que facilita algunas funciones del programa, como quitar la música, cerrarlo o incluso reiniciar el nivel; por último hemos añadido una pantalla de puntuaciones al final de cada nivel, la cual indica alguna información interesante sobre como se ha desarrollado cada partida.

Otra implementación adicional son los niveles predefinidos que creamos mediante otro programa en python que se encuentra en la carpeta de assets, el cual nos permite dibujar con el ratón un nuevo nivel e imprimir por consola la matriz resultante, para así pegarla en la lista de la clase LoadMap donde almacenamos los niveles.

Conclusión

Este trabajo ha sido un proyecto muy entretenido, nos ha permitido aplicar todo lo que hemos visto en clase, y desarrollar alguna cosa más. Sin embargo, hemos encontrado algunos baches en el desarrollo del juego, por lo que, a continuación, vamos a mencionar los más relevantes y la manera en la que los hemos resuelto.

Problemas:

Colisiones

Nada más empezar el proyecto, nuestro mayor problema fue pensar cómo íbamos a hacer el sistema de colisiones. Al principio pensamos que sería buena idea detectar el color en un píxel determinado (situado a los pies del lemming por ejemplo), mediante la función pget() de pyxel, para ver si tiene un suelo debajo. Pronto descubrimos que esta manera nos iba a dar problemas, en el momento en el que nos dimos cuenta de que solo hay 16 colores para dibujar en la pantalla, lo cual limitaría en gran medida los elementos que podemos dibujar.

Nuestra solución fue cambiar por completo el sistema, nuestra clase Tablero nos permite crear una matriz de un tamaño determinado, (14x16 como se indica en la guía), la cuál va a guardar en cada celda, una id que representará el bloque del archivo.pyxres donde guardamos los dibujos. Después, tenemos varias funciones que nos permiten modificar esa matriz y obtener sus valores. En concreto para el caso de las colisiones, la función más relevante es getBlock() la cuál se le puede introducir por parámetro las coordenadas del lemming, y devolverte la id del bloque donde se encuentra, además tiene otro parámetro especial llamado “offset” a modo de vector para comprobar que bloque tiene abajo, izquierda, derecha, o arriba del lemming. Por ejemplo, offset = (1,0) te devuelve la id a la derecha del lemming.

Escaleras

Cuando empezamos a crear el lemming el movimiento de izquierda a derecha, y la caída fue relativamente fácil, el problema estuvo a la hora de implementar el movimiento de las escaleras, ya que todo se volvían condiciones de si detectaba este bloque en esta posición y este otro no para así cambiar su movimiento en diagonal, lo cual daba más problemas, porque según subía la escalera, el lemming detectaba otros bloques, un caos.

La solución fue algo muy útil llamado máquina de estados. Una máquina de estados consiste en utilizar una variable (tipo entero en este caso) a la que se le indica el estado del lemming. Por ejemplo, si es 0 el lemming se mueve de izquierda a derecha, 1 cae, 2 cae pero con paraguas, 3 sube escaleras a la izquierda, 4 sube hacia la derecha…

De esta forma todo el sistema se simplifica ya que para cada estado sólo tienes que preocuparte de ciertos factores, y no de todo a la vez.

Trabajo

El siguiente problema estuvo en los trabajos. Nuestra clase Lemming se estaba volviendo muy larga (la idea era que todos los archivos como mucho tuvieran poco más de 200 líneas), y añadirle más trabajos solo alargaba el código y lo volvía más difícil de leer y de implementar nuevas funcionalidades.
Para ello creamos la clase LemmingBehaviour, que se encargaría de controlar qué trabajo estaba realizando el lemming en un determinado momento, para ello hacemos uso de la propiedad job, un objeto que va cambiando de clase según el trabajo que tome el lemming. NormalLemming es la clase con la que empieza, la que se encarga de andar, caer, subir escaleras etc. Cuando este coge una herramienta, job cambia a una de estas clases según lo que ha cogido (LemmingBlocker, LemmingBuilder y LemmingDigger). Todas las clases tienen su update y draw, de modo que simplemente en LemmingBehaviour hay que poner job.update() y job.draw() para que funcionen automáticamente. Las tres últimas mencionadas en concreto también tienen la propiedad finish, que indica cuando han terminado su trabajo para que vuelva a ser de tipo NormalLemming.

En definitiva, estos problemas a parte de algún que otro dolor de cabeza, nos han permitido encontrar soluciones ingeniosas que, a parte de funcionar, simplificaban y hacían el código mucho más modular, por ejemplo, la solución del trabajo de los lemmings nos permite fácilmente crear una nueva clase donde programemos un nuevo trabajo y añadirlo. Por lo que consideramos que ha sido una buena forma de poner a prueba nuestras capacidades además de que es muy satisfactorio ver cómo todo funciona.
Como crítica al trabajo en sí, consideramos que es un juego bastante difícil y que toma mucho tiempo de desarrollar, nosotros nos hemos apañado bien porque ya teníamos experiencia programando, pero como proyecto de primer curso, pensamos que hay otras opciones más sencillas.