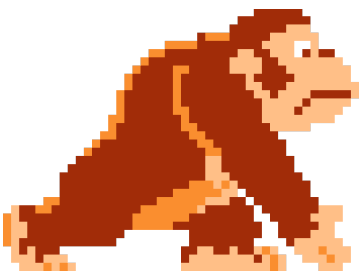
DONKEY KONG

Python - Pyxel



Sonsoles Molina Abad

Lorenzo Largacha Sanz

Doble Grado de Ingeniería Informática y ADE

ÍNDICE

INTRODUCCIÓN

Para el proyecto final de la asignatura de Programación del 1er curso del doble grado de Ingeniería Informática y ADE, hemos elaborado el juego de Donkey Kong, programado en Python y utilizando un motor de juegos llamado Pyxel. En el siguiente documento se encuentra un informe detallado que explica el código del programa y sus principales funcionalidades para su mejor entendimiento.

En este juego, Mario tendrá que rescatar a Pauline, pero no será tan sencillo, ya que tendrá que subir, a través de las escaleras, las plataformas que le separan de ella, evitando los barriles que Donkey Kong irá lanzando, saltando sobre ellos. Para ello tendrá tres intentos (o vidas) de llegar a la parte superior de la pantalla. Cada vez que choque con un barril, Mario morirá y comenzará desde el principio de las plataformas.

SPRINT 1: Objetos e interfaz gráfica

Para organizar nuestro programa, hemos dividido los elementos del juego en tres paquetes distintos: la carpeta “escenario”, que contiene los archivos con las clases “plataformas”, “escaleras”, “barriles” y “contadores”; la carpeta “personajes”, en la cual encontramos los archivos con las clases “dk” (Donkey Kong), “Mario”, y “Pauline”; y la carpeta “assets”, que contiene las imágenes utilizadas. Asimismo, disponemos de los ficheros “donkeyKong.py”, en el cual se encontrará todo el juego, “inicio.py”, “game over.py”, y otro para las constantes “constantes.py”. Todo ello está en una carpeta llamada “donkeyKong” y en cada fichero se importan los ficheros necesarios para cada clase. Para generar el tablero de juego tenemos otro fichero llamado “test” desde el cual llamamos a la clase “Juego” del fichero “donkeyKong.py”. Además, hemos creado mediante otra clase el texto del marcador (fichero “contadores”), en el cual se irán sumando 100 puntos cada vez que salte un barril, y para indicar las vidas restantes las iremos dibujando con pequeños Marios (para ello disponemos de un contador de vidas en la clase de Mario).

Para controlar en qué lugares de la pantalla pueden moverse los objetos del juego, hemos creado una matriz (a base de listas) que cuenta con una posición para cada pixel de la pantalla. De tal forma, almacenamos el número 1 en las posiciones en las que se encuentra la superficie de las plataformas, un 2 en las que corresponden a las escaleras, un 3 en la superficie de estas, un 4 en su centro, y un 0 en el resto de píxeles. Darle diferentes números a las posiciones de la matriz nos sirve para diferenciar en qué zona

se encuentran los objetos cuando se desplazan por la interfaz.

SPRINT 2: Movimiento de Mario

Para crear a Mario, tenemos una clase Mario en la cual encontramos sus características, es decir, su posición en x e y, su orientación o su estado, el cual usaremos para las animaciones. Su movimiento también estará definido en esta clase.

Hemos conseguido hacer que éste se mueva de derecha a izquierda presionando las flechas del teclado, teniendo en cuenta que algunas de las plataformas están inclinadas. Además podrá subir y bajar usando las escaleras, y bajará al final de las plataformas.

Dichos movimientos son posibles gracias a la matriz creada anteriormente. La clase recibirá como argumento la matriz y haremos que Mario solo pueda moverse por los números distintos de 0. Para moverse a cada lado haremos que su posición cambie según la flecha que hayamos presionado, sumando o restando píxeles a su posición actual (si presionamos a la derecha, sumamos a la posición x y se mantiene la misma y; a la izquierda, restamos a la x; hacia arriba, sumamos a la y; y hacia abajo restamos a la y). También puede subir y bajar escaleras usando las flechas (moviéndose por los 2, 3 y 4 de la matriz).

Además, Mario puede saltar los barriles; para realizar este movimiento el usuario debe pulsar la tecla “espacio”, entonces Mario sube 2 píxeles y luego sube píxel a píxel progresivamente hasta llegar al máximo del salto (durante esta parte del proceso no se puede mover en otras direcciones). Cuando llega a la máxima altura que puede saltar si se puede mover lateralmente, y entonces caerá utilizando el mismo código que cuando se encuentra en un 0 de la matriz (caida al final de las plataformas p.e.).

Para este sprint hemos hecho uso de Mario estático, pero cada vez que cambia de sentido, su animación también lo hará, así como levantará el brazo al saltar.

SPRINT 3: Barriles

Para crear los barriles tenemos una clase barril que define objetos (barriles) con una serie de características como su posición en x e y, su orientación, su estado, o los aleatorios (probabilidad).

El método update sirve para actualizar las características de los barriles. Mediante la función pyxel.frame_count controlamos el lanzamiento de los barriles en función de los píxeles que recorren y actualizamos el estado (para animar el barril) usando el módulo, ya que cuando llegue al estado nº 4 volverá al estado nº 0 (4 mod 4 = 0). Además cada barril se mueve hacia derecha e izquierda automáticamente, teniendo en cuenta que algunas de las plataformas están inclinadas. Además puede bajar usando las escaleras (con una probabilidad del 25%, calculada mediante un número aleatorio del 1 al 4), o bajar cayendo por el final de las plataformas. Cuando el barril cae y entra en contacto con una nueva plataforma, este cambia su orientación (dirección) para seguir moviéndose hacia abajo.

Dichos movimientos son posibles gracias a la matriz creada anteriormente. La clase recibirá como argumento la matriz y cada barril tiene su movimiento restringido a los números distintos de 0, salvo cuando cae por el final de la plataforma. Para ello una serie de ifs crean en función de la situación una "nueva x" y una "nueva y" para la siguiente posición del barril, que se comprueba con otros ifs antes de modificar las "x" e "y" reales del barril. Además se obliga a los barriles a llegar hasta el final de la pantalla cuando caen por los bordes de las plataformas para que Mario no pueda esconderse de ellos (pegándose a los lados). Cuando los barriles llegan al borde inferior izquierdo desaparecen.

Por último el método draw pinta los barriles en la posición (x,y) indicada por el método update; teniendo en cuenta el estado en el que se encuentran (de frente cuando bajan por las escaleras o de lado cuando ruedan). Los barriles también están animados para dar la sensación de giro cuando se mueven.

SPRINT 4: Juego Básico

Para desarrollar el juego utilizaremos el fichero principal "donkeyKong.py" en el cual tenemos la clase "Juego". En el método inicial definimos todos los objetos que participan en el juego como la pantalla, la matriz, los personajes, la lista para almacenar los barriles… (creando objetos correspondientes a cada clase); y llamamos a pyxel.run para que se actualize el juego. Para añadir los valores a la matriz, tanto la clase "Escaleras" como la clase "Plataformas" tienen un método llamado "definir_matriz" que hace justamente eso.

El método "pintar_matriz" lo utilizamos cuando desarrollamos el programa para asegurar que se añadían los valores correctamente, pero este método no se utiliza en el

juego.

El método “choque” se ocupa de comprobar si mario choca contra un barril en función de su posición (x,y) y un margen alrededor de la posición (x,y) del barril. Cuando mario es tocado por un barril pierde una vida, y vuelve a su posición inicial (también se restablecen los barriles). Tiene un máximo de tres vidas por partida, y si se queda sin vidas se acaba el juego.

El método “lanzar_barril” crea objetos tipo barril y los añade a la lista de barriles. Está controlado por un if del método update para que lanze los barriles en función de los frames y nunca haya más del número de barriles máximo (10 barriles en pantalla).

El método “subir_puntos” incrementará en 100 cuando mario salte un barril.

El método “update” se ocupa de actualizar todas las variables del juego constantemente para que todo funcione. Primero comprobamos si estamos en la pantalla de inicio, en la pantalla de fin de juego, y si no en la pantalla principal del juego. En la pantalla de inicio espera a que el usuario pulse una tecla y en función de la respuesta recibida pasa a la pantalla principal. Aquí vamos llamando a los diferentes métodos para comprobar si mario cambia de posición, si gana, si muere. Además gestionamos el lanzamiento de los barriles y los eliminamos cuando llegan al final del recorrido; también comprobamos si se han chocado con mario antes, puesto que los barriles se desordenan al bajar.

El método “ganar” comprueba si mario llega a la plataforma de Pauline, analizando su posición (x,y) respecto a una referencia, con lo que ganaría el juego.

Por último el método “draw” pinta las imágenes obteniendo un recorte de los bancos de imágenes importados en el inicio y los sitúa en la posición adecuada. Para ello llama constantemente a los métodos draw de cada una de las otras clases, que se ocupan de pintar los objetos en función de las posiciones obtenidas de sus respectivos métodos update.

SPRINT 5: Elementos Extra

Como nuevas funcionalidades, hemos decidido añadir una pantalla de inicio, así como una de fin en la que se mostrará si se ha ganado o se ha perdido el juego y te dará la posibilidad de jugar otra vez. Cuando se muestren estas pantallas, deberás presionar la tecla enter para iniciar el juego (esto estará indicado en la misma). Asimismo, hemos

implementado otra nueva función para salir del juego y cerrarlo si pulsas la tecla Q ("quit"). Por otro lado, para que la presentación del juego no se haga pesada si no ganamos y para poder mostrar que funciona esta parte, hemos decidido añadir que, pulsando la tecla T, Mario se teletransporte a la posición en la que se gana el juego.

Además, hemos animado a Pauline para que salga en movimiento y pidiendo ayuda, hemos agregado nuevas imágenes, como el barril con el fuego en la esquina de la pantalla, y, para mejorar la calidad, hemos quitado los bordes negros de las imágenes para que salgan sin fondo, añadiendo un 0 en el comando que dibuja estos píxeles. Lo que hace esta función es quitar a la imagen el color que pongas (en este caso el 0 es el color negro). Por último, para ser más precisos a la hora de coger las imágenes, hemos hecho uso de una aplicación que cuenta los píxeles de las mismas (GIMP).

CONCLUSIONES

En resumen, hemos realizado todos los sprints indicados, incluyendo los elementos extra, de una forma organizada, sin código repetitivo y con métodos no excesivamente largos. Además, a la hora de realizarlo hemos ido añadiendo comentarios para un mejor entendimiento.

Al principio, nos supuso un problema el cómo dividir el trabajo y que luego encajase perfectamente, por lo que optamos por usar *git*, una plataforma en la que se puede realizar cambios en un mismo programa desde distintos ordenadores, y *teletype* para poder programar los dos a la vez en un mismo archivo. Asimismo nos hicimos cargo de que funcionase el juego en Anaconda. Otra de las dificultades que encontramos fue el movimiento de Mario, ya que disponía de una gran variedad de posibilidades.

Por otra parte, no nos resultó suficiente la materia dada en clase para poder implementar todas las funciones del juego. No obstante, gracias a los ejemplos proporcionados y a la ayuda del profesor pudimos desarrollarlo con éxito. Igualmete, aprendimos más funciones de Pyxel en la página https://github.com/kitao/pyxel/wiki .

El desarrollo de este proyecto nos ha parecido muy interesante y lo vemos como una forma muy entretenida de aprender a programar. Pensamos que le hemos dedicado mucho tiempo y esfuerzo y si hubiéramos dispuesto de más tiempo podríamos haber implementado la animación del inicio, la animación de la imagen de Mario corriendo o subiendo las escaleras, más niveles, etc.