

Modelación y Computación Gráfica para Ingenieros

EDGEY

Juego realizado con pygame y PyOpenGL

Autor: Daniel Soto
Profesora: Nancy Hitschfeld K.
Auxiliares: Pablo Pizarro R.
Pablo Polanco
Ayudantes: Joaquín Torres
Rodrigo E. Ramos
Sergio Leiva
Fecha de realización: 27 de junio de 2017
Fecha de entrega: 28 de junio de 2017
Santiago, Chile

Índice de Contenidos

1	Introducción	1
2	Solución	1
3	Discusión de los Problemas	4
4	Resultados y Conclusiones	4

Lista de Figuras

1	Archivo que contiene información del mapa, y el resultado producido por la clase. . .	1
2	Ejemplo de nivel, con contador de prismas incluido.	2
3	Menús del juego.	3
4	Pantallas de victoria y pérdida	3

1. Introducción

El problema planteado trataba sobre realizar un clon del juego EDGE Extended. El jugador controla un cubo, el cual se puede mover en 4 direcciones rotando sobre sus aristas. Además el jugador puede escalar hasta un cubo de altura sobre su nivel actual. La cámara sigue al jugador mientras se mueve por el nivel. Este juego debía incluir prismas, plataformas que se caen luego de que el jugador las pise y plataformas que empujan al jugador cuando él camine frente a ellas. Cuando el jugador colecciona todos los prismas y se para sobre la meta, se termina el nivel.

2. Solución

El juego fue programado en Python 2.7, usando las librerías `pygame` y `PyOpenGL`. Además se usaron algunas funciones del paquete `pygltoolbox` creado por Pablo Pizarro R. Para almacenar las constantes del juego se utilizaron archivos `.json`. La tarea se desarrolló completamente usando `git` en el repositorio <https://github.com/danno-s/tarea-3-modelacion-grafica>. Los archivos `.py` principales son:

- `edgey.py`: Este archivo es el que se encarga de inicializar las librerías utilizadas, toda la lógica del juego y dibujar en pantalla los objetos del modelo. Junto a la clase `EdgeyCamera`, recibe input del teclado y define direcciones de movimiento del jugador dependiendo de la posición de la cámara.
- `edgey_camera.py`: Esta clase crea una instancia de una cámara a una altura fija con respecto al jugador, que puede rotar en intervalos de 90 grados en el plano x, y .
- `level_manager.py`: Esta clase se encarga de inicializar un mapa a partir de un archivo `.json` conteniendo las coordenadas de cada objeto en la matriz de tres dimensiones que representa el nivel. Un ejemplo simple y su resultado se muestran a continuación.

El archivo que codifica cada mapa usa números enteros para representar cada tipo de bloque, estos son:

- -1: Celda donde se ubica el jugador al comenzar el nivel.
- 0: Espacio vacío.
- 1: Bloque normal.
- 2: Prisma.
- 3: Bloque que cae.
- 4 al 7: Bloque que empuja para cada una de las 4 direcciones posibles.
- 8: Meta.
- 90 al 99: Bloque que permite al jugador elegir una opción.

Esta clase además incluye funciones para eliminar elementos en ciertas coordenadas, encontrar elementos en ciertas coordenadas y retornar lista de elementos de la misma clase.

- `presets.py`: Este archivo contiene la mayor cantidad de información de todos y define todos los tipos de bloques usados en el juego. Dentro de ésta se encuentran las clases:

```

{
  "levels ": [
    [
      [0, 0, 0, 0, 0],
      [0, 3, 3, 3, 0],
      [0, 3,-1, 3, 0],
      [0, 3, 3, 3, 0],
      [0, 0, 0, 0, 0]
    ]
  ],
  "options": [],
  "actions": [],
  "parameters": [],
  "tag": "game"
}

```

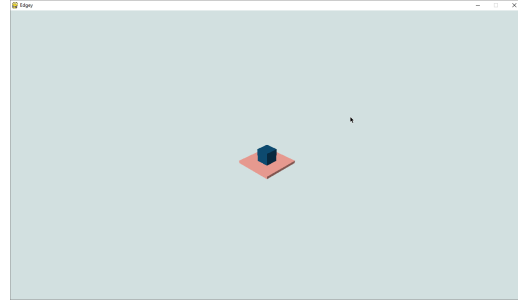


Figura 1: Archivo que contiene información del mapa, y el resultado producido por la clase.

- **Player:** Esta clase describe cómo debe moverse el cubo dependiendo de mensajes entregados a ella por otras clases. Por ejemplo, describe cómo debe ser el movimiento del cubo cuando el jugador es empujado por un bloque, o cuando se mueve sobre una casilla vacía.
- **Shard:** Esta clase es bastante simple y describe cómo deben ser dibujados los prismas que se deben coleccionar.
- **BasicTile:** Esta clase describe un cubo simple, al no interactuar con el nivel ni el jugador, no contiene ninguna función especial.
- **FallingTile:** Se describe una caída usando una variable booleana que se hace verdadera cuando el jugador se para sobre él. Si esta variable es verdadera se revisa cada frame si el jugador sigue parado sobre la plataforma y si no lo está, se comienza la caída de la plataforma. Luego de cierto punto, se elimina de la matriz del nivel el elemento.
- **PushingBlock:** Se usa un entero para describir la dirección en la que está orientado el bloque. Esta clase espera a que el jugador se pare frente a él y luego manda un mensaje de `slide(dirección)` al jugador para que se realice la animación del empujón. También se mueve parte del bloque junto al jugador para expresar mejor que está siendo empujado.
- **FinishTile:** Es prácticamente igual a un **BasicTile**, es de otro color y espera a que el jugador se pare sobre él con todos los prismas coleccionados para terminar el nivel.
- **OptionTile:** Este tipo de bloque al ser dibujado imprime un string en el nivel junto a una línea la cual indica el cubo relacionado. El jugador puede pararse sobre estos bloques y destacar la opción. Al presionar la barra espaciadora se consiguen los parámetros **action** y **param** los cuales definen la función ejecutada en el nivel y sus parámetros.

La implementación de la clase **OptionTile** permite crear menús dentro de una escena del juego. Esto se hace inicializando los bloques con la acción **load**, la cual le dice al loop principal del juego,

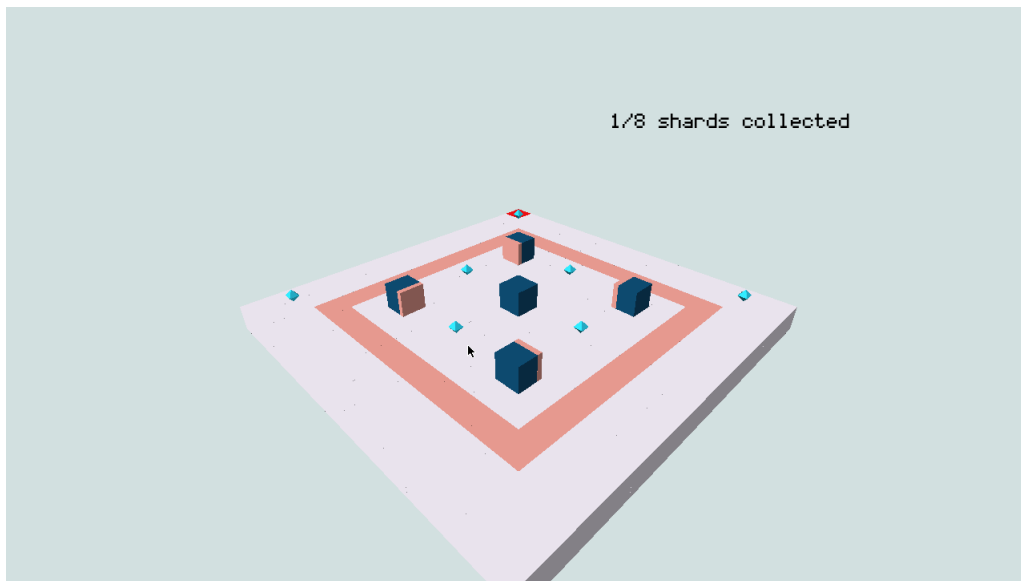


Figura 2: Ejemplo de nivel, con contador de prismas incluido.

que cuando el jugador presiona la barra espaciadora, se debe cargar el nivel almacenado en la variable `param`. La carga del nivel se hace con la función `load_level`, definida en `edgey.py`. A continuación se muestran un par de imágenes del menú principal:

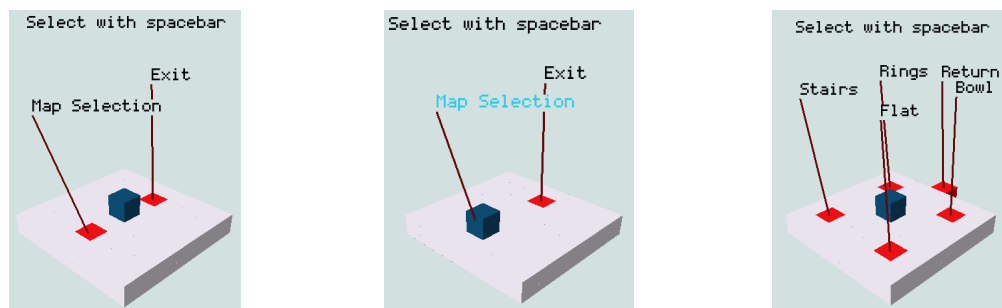


Figura 3: Menús del juego.

Estas implementaciones también permiten la inclusión de una pantalla de fin de nivel y otra para cuando el jugador se cae del nivel. Las cuales se muestran a continuación:

3. Discusión de los Problemas

Durante el desarrollo del juego se encontraron problemas principales:

- Rotar el cubo que controla el jugador fue bastante complejo, pues hacer rotaciones alrededor de un eje local para cierto objeto no es parte de las funciones de `OpenGL` y además se debe recordar el progreso de cada rotación dado que la función `Rotatef` no mantiene el cambio entre fotogramas. Esto fue solucionado dibujando el cubo del jugador último en cada ciclo del



Figura 4: Pantallas de victoria y pérdida

juego y moviendolo hacia el origen de las coordenadas. El progreso de la rotación se solucionó usando una variable auxiliar para almacenarla.

- Crear el menú fue bastante difícil, dado que al usar `OpenGL`, dibujar texto a pantalla se debe hacer a través de texturas creadas con funciones de `pygame`. Debido a esto, hacer un menú tradicional se complica bastante. Esto se solucionó con la idea de incluir el menú principal del juego como otro mapa más, agregando un tipo de bloque nuevo que describe funciones y parámetros a ejecutar cuando se presiona un botón.

4. Resultados y Conclusiones

La aplicación logra funcionar como se esperaba, cuando el número de cubos supera los 200 el juego comienza a correr más lento. Gracias a la implementación de la clase `level_manager` y su constructor a base de archivos `.json` permite que diseñar nuevos niveles sea muy simple y flexible. La clase `OptionTile` también entregó una manera creativa e inmersiva para hacer los menús del juego, pues así el jugador nunca pierde el control del cubo.