PROYECTO FLOW: PRIMERA ENTREGA

Análisis de Algoritmos
Departamento de Ingeniería de Sistemas
Pontificia Universidad Javeriana

Realizado por
Alejandro Morales Contreras

4 de octubre de 2022

Índice

1. Introducción

El presente pretende servir como documento de diseño y documentación sobre el proyecto del curso de análisis de algoritmos. El proyecto consiste en la implementación de un algoritmo que juegue "Flow". Este juego consiste en resolver rompecabezas *numberlink*. Cada rompecabezas consiste en conectar todos los pares de puntos del mismo color dibujando tuberías no intersectables hasta que todo el tablero está ocupado.

Para la primera entrega, se propone la elaboración de una interfaz para que un humano pueda interactuar y jugar. Para esta entrega, se propone una interfaz gráfica de usuario. En este documento se presenta entonces el diseño de esta inferfaz (sección 2), se muestran detalles de la implementación (sección 3) y una breve documentación (sección 4).

2. Diseño

2.1. Arquitectura del Sistema

Para implementar el juego mediante una interfaz gráfica de usuario, se hace necesario definir una arquitectura para diseñar el Sistema. En este caso, se va a utilizar el patrón MVC junto con un mediador de eventos para comunicar los componentes entre sí. En la figura 1 se presenta la arquitectura del Sistema.

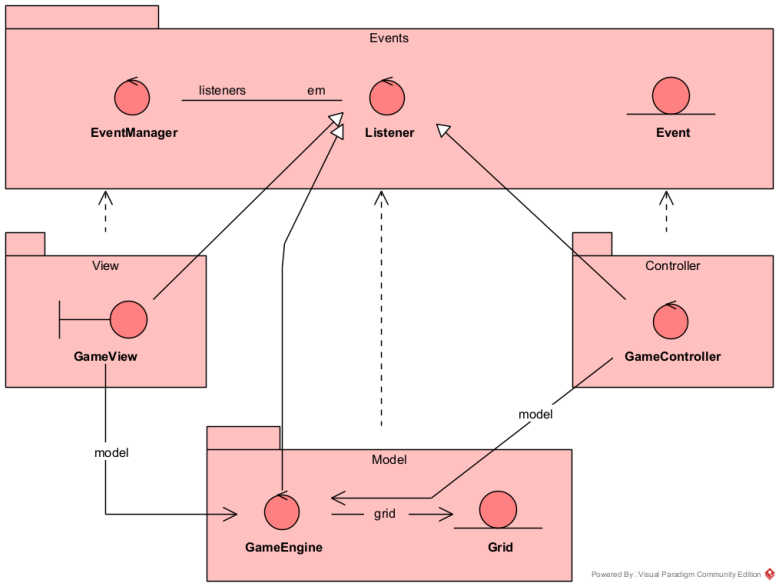

Figura 1: Arquitectura del Sistema

A continuación se presenta una breve descripción de cada uno de los componentes que hacen parte de esta arquitectura.

2.1.1. Model

El modelo guarda toda la lógica y estructura del estado del juego. Para lograr esto, se define un *GameEngine* que se encarga de almacenar el estado del juego, así como modificarlo acorde a los eventos de modificación del modelo. Nótese que en el modelo se tiene la clase *Grid*, la cual tiene toda la lógica de cómo funciona el juego.

2.1.2. View

La vista se encarga de representar el estado actual del juego gráficamente en la pantalla. *GameView* tiene toda la lógica asociada a pintar el modelo cada vez que se recibe un evento de actualización de pantalla.

2.1.3. Controller

El controlador se encarga de recibir todas las interacciones del usuario y llevarlas a modificar el modelo. *GameController* tiene la lógica asociada a recibir los eventos de interacción del usuario y transformarlos en eventos de modificación del modelo.

2.1.4. Events

Mediante los eventos se coordina toda la comunicación entre cada uno de los componentes. El *EventManager* se encarga de manejar la cola de eventos y notificar a todos sus *listeners asociados*. El *Listener* es notificado cada vez que se genera un evento, y también tiene la posibilidad de colocar eventos en la cola. Nótese que *GameEngine*, *GameView* y *GameController* heredan de este.

2.2. Modelo de interacción del Sistema

La interacción del Sistema y sus componentes está mediada por el manejador de eventos. En general, estos son los eventos de interacción importantes que analizar:

- Inicialización, generado al ejecutar por primera vez el juego y el cual se encarga de inicializar todos los componentes del Sistema.
- Tick, tick del reloj el cual se encarga de avisar a la Vista que debe actualizarse.
- Interacción del usuario, generado cuando el usuario interactúa con el Sistema y modifica el estado del modelo.

A continuación se presenta una breve representación de cada una de estas interacciones.

2.2.1. Inicialización y Tick

En la figura 2 se presenta el modelo de interacción de inicialización y tick del reloj. La inicialización es dirigida por el modelo, el cual crea el evento de inicialización para los otros componentes. Después, entra en un ciclo hasta que el juego acabe que genera continuamente los ticks. La Vista utiliza estos para recuperar el estado del modelo y actualizarse.

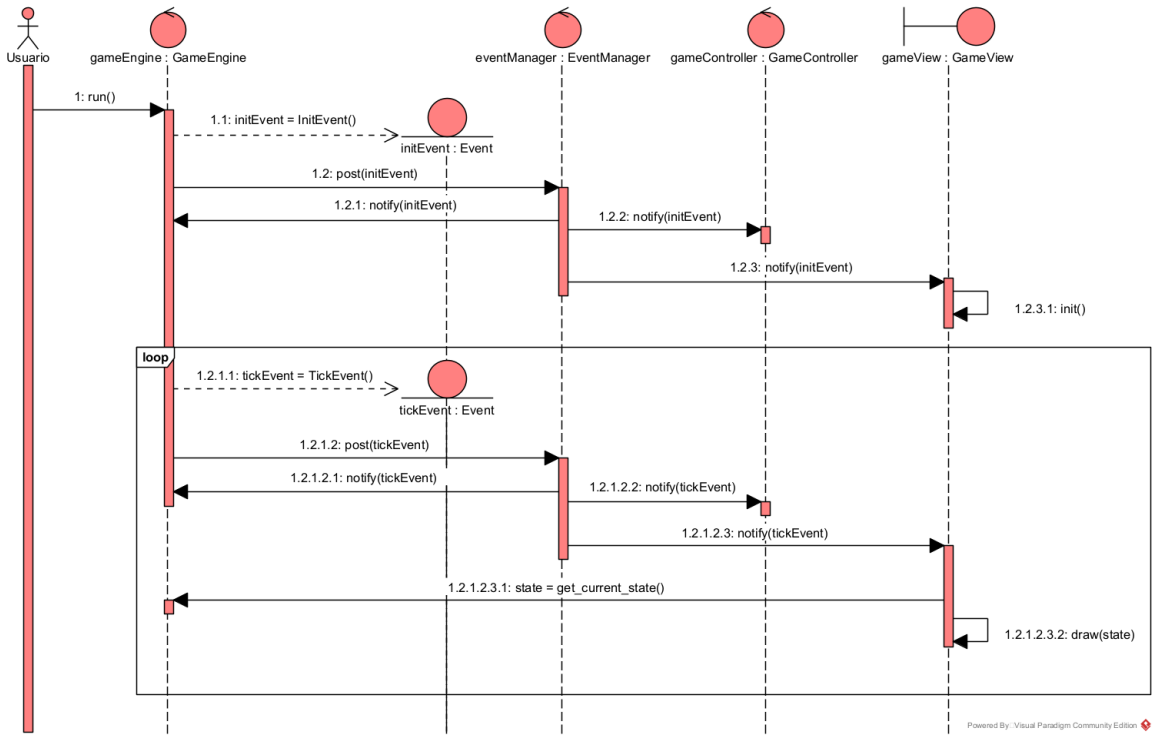

Figura 2: Interacción de inicialización y tick del reloj

2.2.2. Interacción del usuario

En la figura 3 se presenta el modelo de interacción del usuario. En general, el usuario interactúa con el controlador, el cual se encarga de traducir esta interacción en un evento de modificación del modelo (estado) del Sistema.

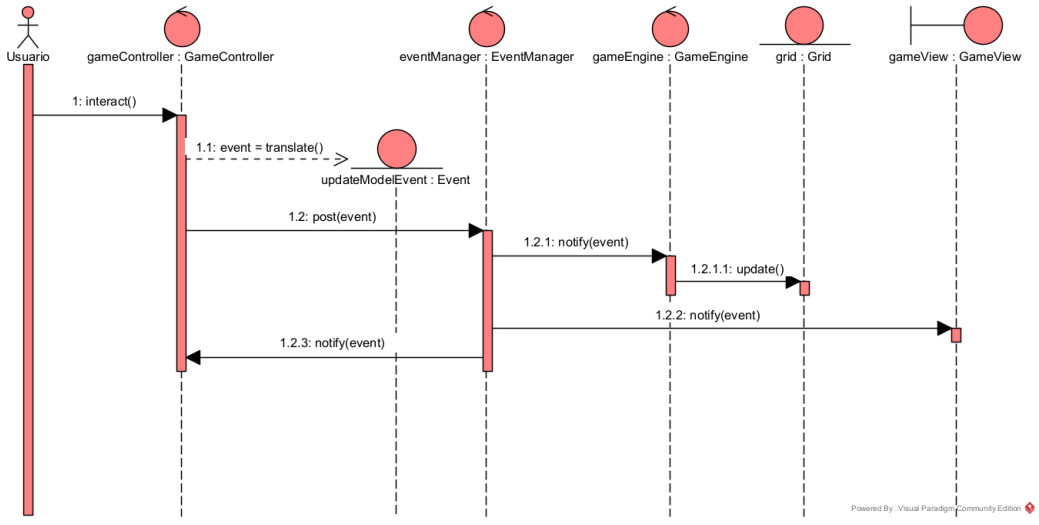

Figura 3: Interacción del usuario

2.3. Lógica del juego

Como se mencionó en la sección 2.1.1, la lógica de cómo funciona el juego está mediada por el *Grid*, el cual hace parte del estado del modelo. Como se vió en la sección 2.2.2, la modificación del modelo (y por ende, del estado del juego) está dirigida por las interacciones del usuario.

2.3.1. Representación del tablero

Para empezar, se define el prototipo que modela el *Grid*, presentado en la figura 4.

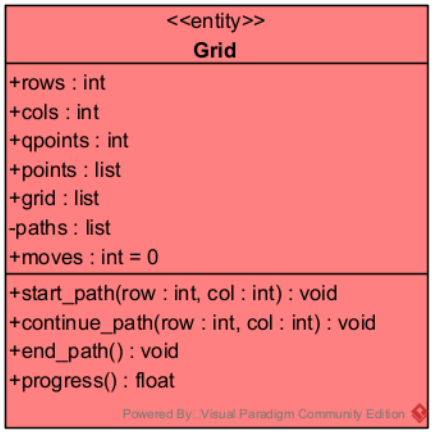

Figura 4: Prototipo de *Grid*

El tablero se modela a partir de una cantidad de filas (*rows*), una cantidad de columnas (*cols*), una cantidad de puntos (*qpoints*) y el posicionamiento (row, col) de cada par de puntos (*points*). A partir de esta información, es posible construir el tablero (*grid*): una matriz de $rows \times cols$. También se representan los caminos actuales (*paths*): un arreglo de (row, col) para cada punto disponible.

Cada celda del tablero es representada con un *estado*, acorde a su color (un número entero) y su interacción (o no) con las demás celdas (dos números enteros de posicionamiento). Para entender el posicionamiento de una celda, véase la figura 5.

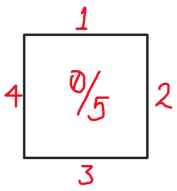

Figura 5: Posiciones de una celda

Las posiciones cardinales norte, este, sur y oeste de una celda se representan con 1, 2, 3 y 4 respectivamente. Así mismo, el centro de celda puede ser representando con un 0 (inicio / fin) o un 5 (fin parcial). Estas posiciones se utilizan para representar como se posiciona la celda en el tablero. El posicionamiento entonces se refiere a de dónde viene el camino a la celda, y a dónde va el camino desde la celda.

Para entender mejor este concepto, supóngase que se tiene un tablero 3×3 con dos colores (rojo y azul) representados por los enteros 1 y 2 (con 0 como vacío) como el que se presenta en la figura 6. Nótese en este que el posicionamiento de todas las celdas es $(0, 0)$, lo cual significa que no se han movido o no están conectadas por ningún camino.

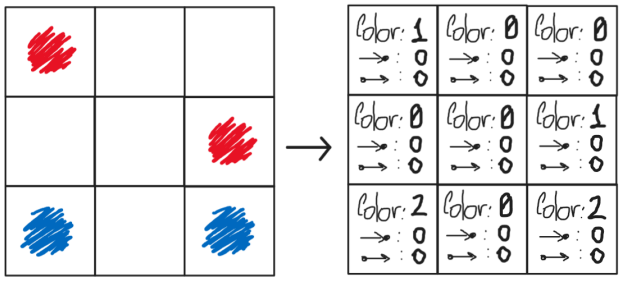

Figura 6: Ejemplo de un tablero inicial

Tracemos un primer camino para el color rojo, desde su punto ubicado en la esquina superior izquierda como el que se presenta en la figura 7. Nótese que el punto rojo ahora está representado como $(0, 3)$, indicando que es un punto inicial que va hacia el sur. La última celda del camino rojo es $(3, 5)$, indicando que un camino llega por el sur y se queda en el centro de la celda.

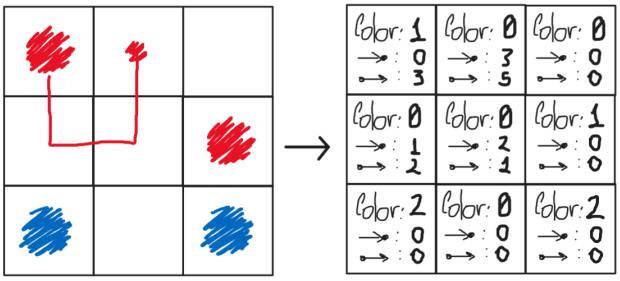

Figura 7: Ejemplo de un tablero primer movimiento

Un segundo camino para el color azul que conecta ambos puntos se presenta en la figura 8. Nótese que el punto inicial es $(0, 2)$ y el punto final es $(4, 0)$, representando este último que un camino llega desde el oeste y termina en este punto.

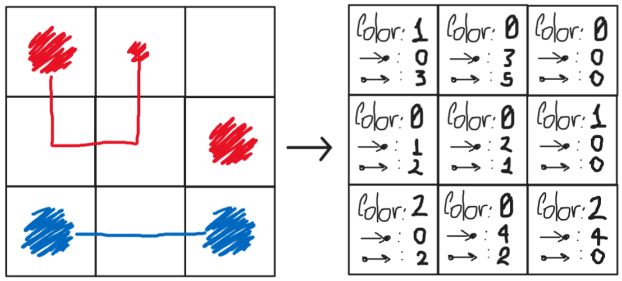

Figura 8: Ejemplo de un tablero segundo movimiento

2.3.2. Interacción con el tablero

La interacción del usuario con el tablero se compone de 3 operaciones principales:

- Iniciar un camino: el usuario hace un primer click sobre una celda que no está vacía. Esta interacción inicia un camino con el color de la celda sobre la que se para.
- Continuar un camino: el usuario continúa su click, moviendo el mouse por distintas celdas del tablero. Esta interacción continúa el camino del mismo color creado en el paso anterior.
- Terminar un camino: el usuario levanta el click. Esta interacción termina el camino que había sido iniciado.

2.3.3. Iniciar un camino

En la figura 9 se presenta un flujograma con la lógica del juego al iniciar un camino.

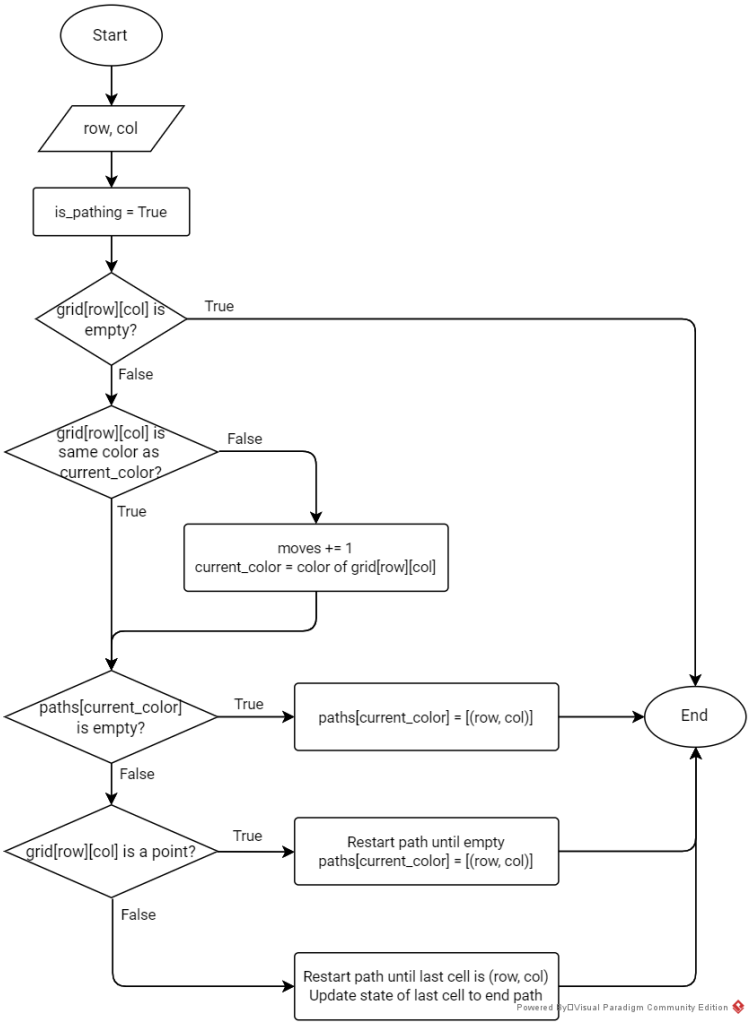

Figura 9: Flujograma de iniciar un camino

2.3.4. Continuar un camino

En la figura 10 se presenta un flujograma con la lógica del juego al continuar un camino.

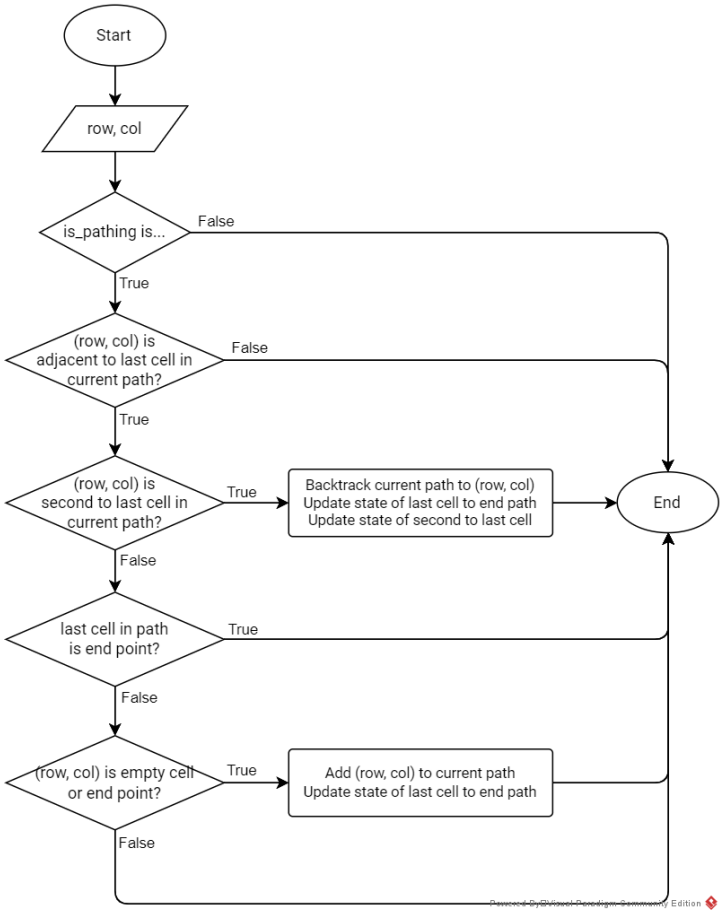

Figura 10: Flujograma de continuar un camino

2.3.5. Terminar un camino

En la figura 11 se presenta un flujograma con la lógica del juego al terminar un camino.

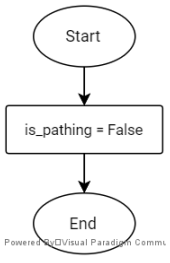

Figura 11: Flujograma de terminar un camino

3. Implementación

El juego se desarrolla en Python v3.10. Para la implementación de las interfaces gráficas, se hace uso de la librería PyGame v2.1.2, la cual cuenta con múltiples módulos para facilitar la creación de videojuegos en Python. La implementación se hace acorde a la arquitectura definida en la sección 2.1, con el uso de la librería PyGame en la vista y el controlador.

3.1. Vista previa del juego

Una vista previa del juego se presenta en la figura 12.

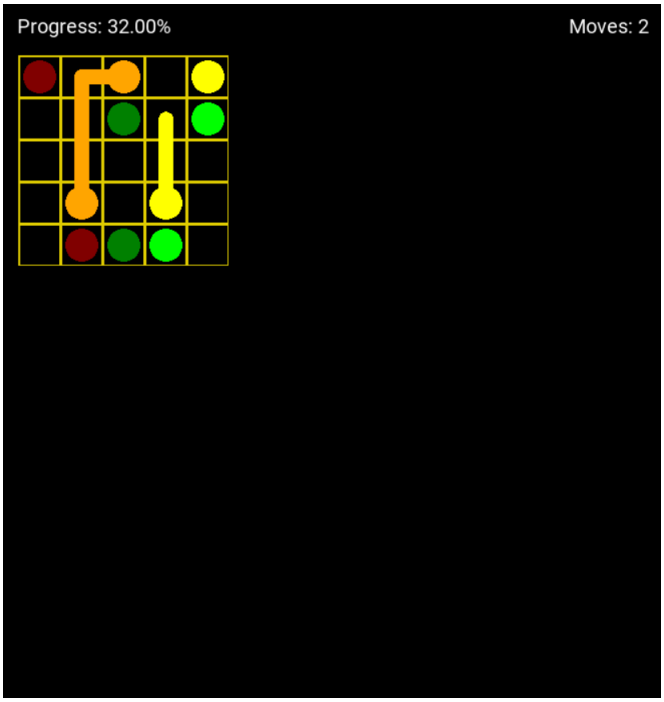

Figura 12: Vista previa del juego

3.2. Funcionalidades del Sistema

Antes de empezar, el usuario tiene la posibilidad de:

- Iniciar un juego con un tablero de niveles predeterminados
- Iniciar un juego con un tablero proporcionado en un archivo
- Iniciar un juego con un tablero creado al azar

Durante el juego, el usuario puede:

- Crear caminos manteniendo presionado con el click derecho o izquierdo del mouse
- Reiniciar el tablero presionando la tecla R
- Ver en todo momento el progreso y cantidad de movimientos

3.3. Restricciones del Sistema

Se definen las siguientes restricciones:

- La cantidad mínima de puntos es 2 y la cantidad máxima es 16
- El tablero mínimo es de 2×2 y el máximo es de 15×15

3.4. Configuración de los tableros

Como se mencionó en la sección 3.2, existen múltiples formas de iniciar un juego. Todas estas dependen de una configuración del tablero, la cual se representa como un diccionario en Python que contiene los mismos datos para representar un tablero (ver sección 2.3.1). Por ejemplo, el mismo tablero inicial de la figura 6 es representado por la siguiente configuración:

```
{
   "rows": 3,
   "cols": 3,
   "qpoints": 2,
   "points": [
        [
            [0, 0], [1, 2]
        ],
        [
            [2, 0], [2, 2]
        ]
   ]
}
```

Así mismo, esta configuración puede venir de niveles predeterminados, de un archivo o ser creada al azar.

3.4.1. Tableros de niveles predeterminados

El juego viene con unos niveles predeterminados definidos dentro de un archivo JSON. Estos niveles aseguran que es posible ganar y existe una única solución. Refiérase a la sección 4.2.1 para ver como ejecutarlos.

3.4.2. Tableros por archivos

Es posible definir una configuración similar a la presentada anteriormente dentro de un archivo JSON para proceder a jugar dicho tablero. Sin embargo, no es posible asegurar que se puede ganar. Refiérase a la sección 4.2.2 para ver como ejecutarlos.

3.4.3. Tableros al azar

Es posible definir una cantidad arbitraria de filas, columnas y puntos para que el juego genere un tablero al azar y poder proceder a jugarlo. Sin embargo, no es posible asegurar que se puede ganar. Refiérase a la sección 4.2.3 para ver como ejecutarlos.

4. Documentación

4.1. Compilación

4.1.1. Requisitos previos

La máquina debe contar con la versión de Python 3.10 o superior.

4.1.2. Windows

Clonar el repositorio donde se aloja el código

```
> git clone https://github.com/amoralesc/flow.git
> cd flow
```

Crear un entorno virtual para instalar las dependencias

```
> python -m venv venv
```

(Opcional) Activar los permisos de ejecución de scripts

```
> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Activar el entorno virtual

```
> venv\Scripts\activate
```

Instalar las dependencias

```
> pip install -r requirements.txt
```

4.1.3. Linux / Mac

Clonar el repositorio donde se aloja el código

```
$ git clone https://github.com/amoralesc/flow.git
$ cd flow
```

Crear un entorno virtual para instalar las dependencias

```
$ python -m venv venv
```

Activar el entorno virtual

```
$ source venv/bin/activate
```

Instalar las dependencias

```
$ pip install -r requirements.txt
```

4.2. Ejecución

Para ver todas las opciones de ejecución posibles:

```
$ python main.py -h
```

4.2.1. Tableros de niveles predeterminados

Para visualizar las configuraciones de tablero de los niveles disponibles:

```
$ cat data/levels.json
```

Para ejecutar un tablero de un nivel predeterminado:

```
$ python main.py -l LEVEL
```

donde LEVEL es un entero que representa la posición del nivel en el arreglo de niveles disponibles (indexado desde 1).

4.2.2. Tableros por archivos

Configurar previamente el tablero siguiendo los lineamientos de 3.4 dentro de un archivo. Para ejecutar el tablero desde un archivo:

```
$ python main.py -f FILE
```

donde FILE es la ruta hasta el archivo que contiene la configuración del tablero.

4.2.3. Tableros al azar

Para ejecutar un tablero al azar:

```
$ python main.py -r ROWS COLS POINTS
```

donde ROWS, COLS y POINTS son 3 enteros representando la cantidad de filas, columnas y puntos respectivamente.

4.3. Documentación del código

El código se encuentra completamente documentado, y es posible referirse a este para entender las distintas partes de la implementación.