UNIVERSIDAD NACIONAL DE COLOMBIA

PROYECTO – DISEÑO DE APLICACIÓN

ESTUDIANTES:

CARLOS ALBERTO CORTÉS RAMÍREZ

ANGEL GUILLERMO PEÑARREDONDA SILVA

JOSÉ LUIS RATIVA MEDINA

REINALDO TOLEDO LEGUIZAMÓN

ANDERSON CAMILO VARGAS ALEJO

PROFESOR:

Ph.D. JORGE EDUARDO ORTÍZ TRIVIÑO

UNIVERSIDAD NACIONAL DE COLOMBIA

MODELOS ESTOCÁSTICOS Y SIMULACIÓN EN COMPUTACIÓN Y COMUNICACIONES

GRUPO 2

Noviembre, 2023

Bogotá D.C – Colombia

Diseño de aplicación

El programa destinado a abordar el desafío del mundo de Wumpus fue desarrollado en Python 3, haciendo uso del paradigma de programación orientada a objetos para proporcionar una estructura eficiente y modular al código. Esta elección de diseño permitió una representación clara y coherente de las entidades del juego, como las casillas o celdas, lo que contienen las celdas, el agente y el Wumpus, mediante la creación de objetos con atributos y métodos específicos.

La implementación del paradigma orientado a objetos fue crucial para modelar de manera efectiva las interacciones dentro del ambiente del Wumpus. Cada componente del juego se abordó como un objeto independiente, lo que facilitó la gestión y comprensión de las complejidades del mundo virtual. Esta abstracción proporcionó una base sólida para representar la estructura del juego de manera lógica y comprensible.

Asimismo, la programación orientada a objetos permite encapsular la lógica del agente en clases dedicadas, lo que facilitó la implementación de estrategias y decisiones adaptativas. Cada instancia de la clase del agente tenía su propio estado y métodos especializados, permitiendo una toma de decisiones autónoma y eficiente en respuesta a las condiciones cambiantes del entorno.

Diseño general de las clases usadas en el programa

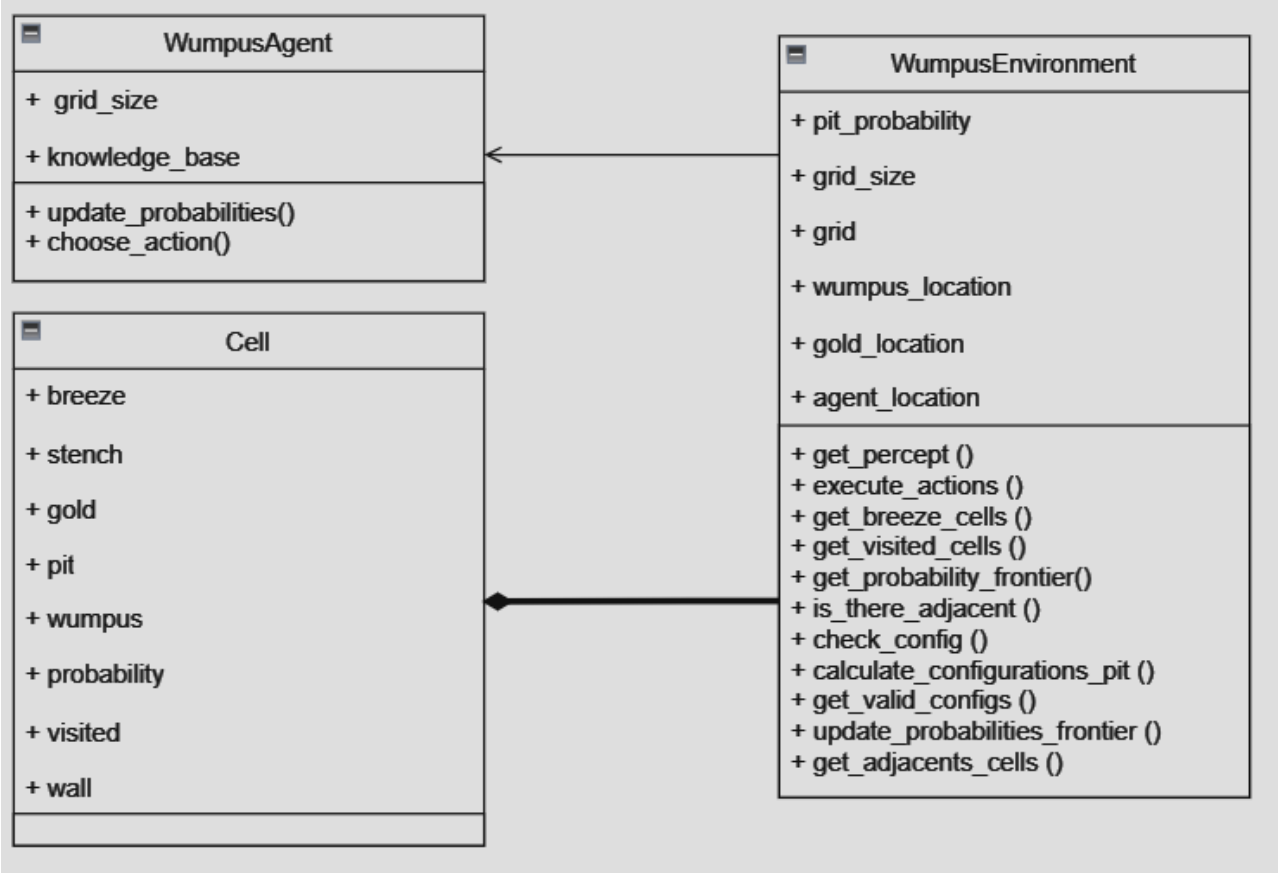

Figura 1. Diagrama de clases de la aplicación

1. La clase `**WumpusAgent**` parece ser una implementación de un agente inteligente diseñado para navegar y tomar decisiones en el contexto del juego del mundo de Wumpus. Aquí hay una descripción de los atributos y métodos de la clase:

- **grid_size**: Este atributo almacena la información sobre las dimensiones del tablero o cuadrícula del juego, como el número de filas y columnas. Puede ser esencial para que el agente se oriente y tome decisiones basadas en la estructura del entorno.
- **knowledge_base**: Este atributo podría representar la base de conocimientos del agente, donde se almacena la información que ha adquirido sobre el entorno del juego. Podría incluir información sobre la ubicación del agente, la presencia de obstáculos, y cualquier otro detalle relevante para la toma de decisiones.
- **update_probabilities**(): Este método probablemente esté destinado a actualizar las probabilidades asociadas con diferentes eventos o situaciones en el entorno del juego. Puede ser utilizado para reflejar las nuevas observaciones y conocimientos adquiridos por el agente, ajustando las probabilidades de eventos futuros.
- **choose_action**(): Este método posiblemente implementa la lógica de toma de decisiones del agente. Puede evaluar la información almacenada en la base de conocimientos y las probabilidades actualizadas para determinar la mejor acción a realizar en un determinado estado del juego. Las acciones podrían incluir moverse a una celda adyacente, disparar, recoger objetos, o cualquier otra acción permitida en el contexto del juego de Wumpus

2. La clase `**WumpusEnvironment**` parece ser responsable de gestionar y representar el entorno del juego del mundo de Wumpus. Del mismo modo, proporciona la infraestructura para modelar y simular el mundo de Wumpus, permitiendo que el agente interactúe con su entorno y tome decisiones basadas en percepciones y acciones específicas del juego.Aquí está una descripción de sus atributos y métodos:

   Atributos:

   - **pit_probability**: Un atributo que almacena la probabilidad de que haya un pozo en una celda específica. Puede ser utilizado para modelar la incertidumbre asociada con la presencia de pozos en el juego.
   - **grid_size**: Indica las dimensiones del tablero del juego, representando el número de filas y columnas en la cuadrícula.
   - **grid**: Un atributo que probablemente almacena la representación del tablero del juego. Puede ser una estructura de datos que mapea las coordenadas a instancias de la clase `Cell` o un tipo similar.
   - **wumpus_location**: Almacena la ubicación del Wumpus en el tablero. Este atributo es crucial para que el agente evite o interactúe con el Wumpus durante su exploración.
   - **gold_location**: Indica la ubicación del oro en el tablero. Puede ser crucial para que el agente decida su estrategia de búsqueda y recolección.
   - **agent_location**: Almacena la ubicación actual del agente en el tablero. Este atributo se actualiza a medida que el agente se mueve a través del entorno.

   Métodos:

   - **get_percept**(): Un método que obtiene la percepción de la celda en la que se encuentra el agente. Retorna un diccionario donde cada clave representa una

percepción (como `breeze`, `stench`, etc.) y su valor es un booleano que indica si la percepción es verdadera o falsa en la celda actual.

- **execute_action**(): Un método que probablemente implementa la lógica para ejecutar las acciones tomadas por el agente en el entorno del juego. Dependiendo de la acción, puede implicar movimientos, disparos, recogida de oro, etc.
- **get_breeze_cells() :** Este método devuelve las celdas del entorno que tienen una brisa. La presencia de una brisa indica la posibilidad de que haya un pozo cercano.
- **get_visited_cells() :** Retorna las celdas que han sido visitadas por el agente en el juego. Puede ser útil para rastrear el progreso y la exploración del agente en el entorno.
- **get_probability_frontier() :** Proporciona las probabilidades asociadas con las celdas en la frontera del entorno. Esto se refiere a las celdas que aún no han sido exploradas pero que están adyacentes a celdas visitadas.
- **is_there_adjacent () :** Verifica si hay alguna celda adyacente que cumple con cierta condición. El método devuelve un booleano indicando la presencia o ausencia de una celda que cumple con la condición.
- **check_config () :** Verifica la configuración actual del entorno. Está relacionado con la ubicación de elementos como pozos, Wumpus, oro, etc., en el entorno del juego.
- **calculate_configurations_pit () :** Realiza cálculos relacionados con las configuraciones posibles de pozos en el entorno. Implica determinar las configuraciones de pozos que son consistentes con las percepciones actuales.
- **get_valid_configs ():** Retorna las configuraciones válidas para el entorno del juego. Esto está relacionado con las combinaciones de elementos como pozos y Wumpus que son coherentes con las percepciones del agente.
- **update_probabilities_frontier () :** Actualiza las probabilidades asociadas con las celdas en el "frente" del entorno. Esto implica ajustar las probabilidades de presencia de elementos como pozos basándose en nuevas percepciones o acciones del agente, por lo cual es uno de los métodos más importantes pues es la que permite manejar la incertidumbre.
- **get_adjacents_cells () :** Retorna las celdas adyacentes a la celda dada en el entorno. Puede ser utilizado para determinar qué celdas son accesibles desde la celda dada

3. La clase `**Cell**` está diseñada para almacenar las propiedades de cada celda en el juego del mundo de Wumpus. Es llamada por la clase `**WumpusEnvironment**` para crear y gestionar el mundo en el que el agente se mueve. Cada instancia de la clase `Cell` representa una celda individual en el tablero del juego y almacena información esencial, como la presencia de corrientes de aire (`breeze`), hedor (`stench`), la existencia de oro (`gold`), la presencia de pozos (`pit`), la presencia de un Wumpus (`wumpus`), la probabilidad asociada con eventos o estados (`probability`), si la celda ha sido visitada (`visited`), y si es una pared (`wall`).

   Atributos de tipo booleano:

   - **breeze**: Indica si hay una corriente de aire en la celda, lo cual podría sugerir la presencia de un pozo cercano.
   - **stench**: Indica si hay un hedor en la celda, lo cual podría indicar la proximidad de un Wumpus.

- **gold**: Indica si hay oro en la celda, señalando la ubicación de un tesoro.
- **pit**: Indica si hay un pozo en la celda, representando un peligro potencial para el agente si decide moverse allí.
- **wumpus**: Indica si hay un Wumpus en la celda, lo cual puede ser crítico para la seguridad del agente.
- **visited**: Indica si la celda ha sido visitada por el agente. Este atributo puede ser útil para realizar un seguimiento del progreso y evitar repeticiones.
- **wall**: Indica si la celda es una pared o una barrera. Este atributo es crucial para el movimiento del agente y para evitar intentos de atravesar obstáculos.

Atributos de tipo Float:

- **probability**: Almacena un valor en punto flotante, posiblemente representando la probabilidad asociada con algún evento o estado en la celda. Este atributo puede ser utilizado para modelar la incertidumbre o confianza del agente en ciertas situaciones.

La interacción entre la clase `**WumpusEnvironment**` y la clase `**Cell**` permite la creación y gestión dinámica del mundo de Wumpus. La instancia de `**WumpusEnvironment**` utiliza instancias de `Cell` para construir el tablero del juego y rastrear el estado de cada celda a medida que el agente explora y toma decisiones. La información contenida en las instancias de `Cell` es crucial para que el agente tome decisiones informadas basadas en las condiciones del entorno. La relación entre las clases **WumpusEnvironment** y **Cell** es una relación de composición pues el ambiente no puede existir sin las celdas pero las celdas por el contrario si pueden existir, en otras palabras, el ambiente está compuesto por celdas.

En conjunto, estas clases trabajan en conjunto para proporcionar la estructura y la lógica necesarias para simular el mundo de Wumpus y permitir que el agente navegue y tome decisiones dentro de este entorno desafiante.

Diseño general del flujo del programa

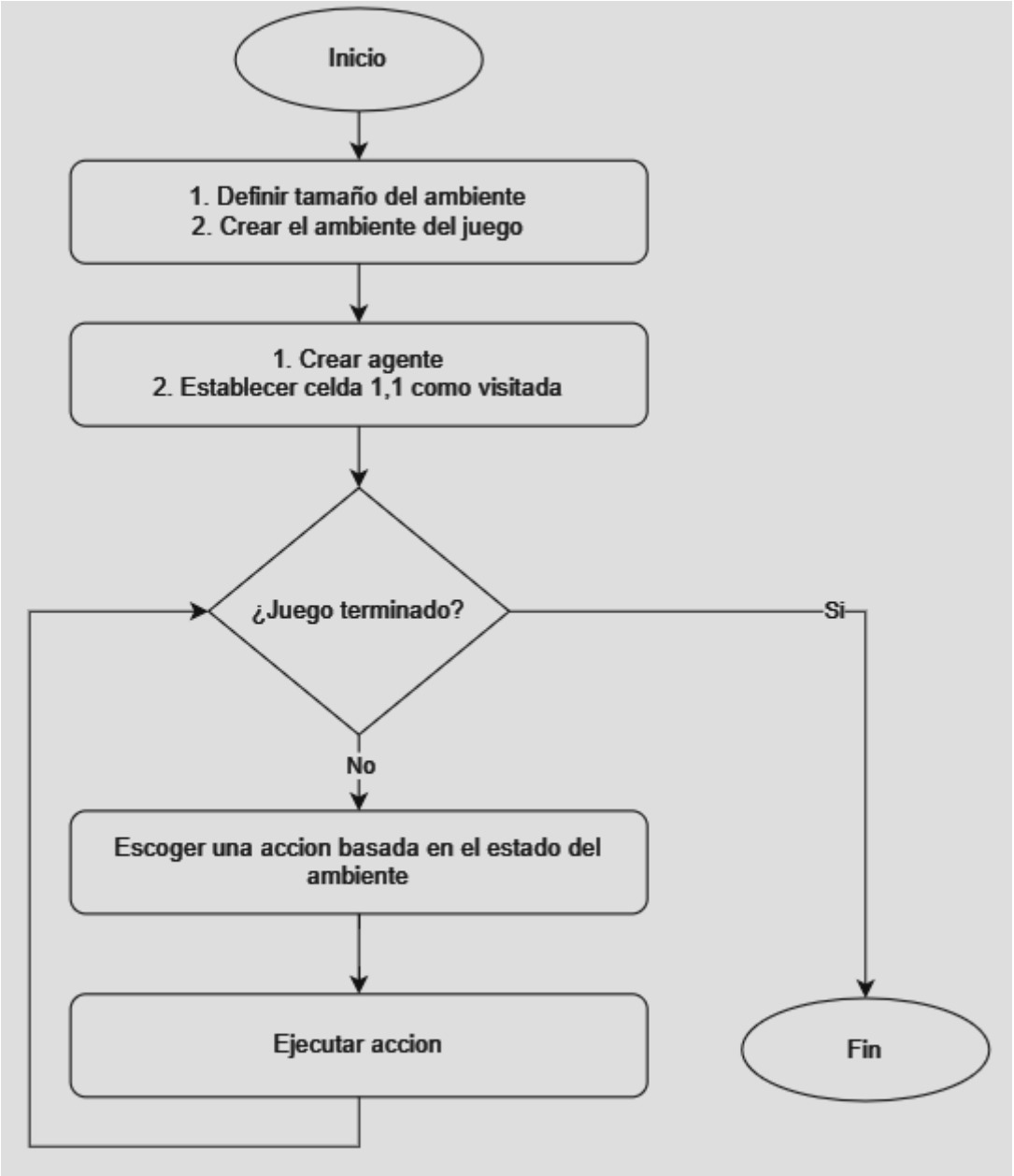

Figura 2. Flujo principal del programa desarrollado

En la figura 2 podemos observar el diagrama de flujo del juego de wumpus. Tenemos que primero se crea el ambiente definiendo su tamaño, del mismo modo, se establece el número de agujeros, la posición del Wumpus y la posición del oro aleatoriamente y después creamos el agente en la posición (1,1) de la grilla del ambiente, dado esto la celda (1,1) se establece cómo visitada. Se procede a preguntar si el juego ha terminado, la cual es la condición de parada. Esta condición de parada consiste en que si el jugador cayó en un Agujero o se encontró con el Wumpus pierde y si encuentra el oro, gana el juego. Teniendo claro esto, si el juego ha terminado, se termina la ejecución del programa. Si no ha terminado se procede a escoger una acción basada en las condiciones del ambiente, es decir, las celdas “seguras”, las que se perciben brisa y/o en las que se percibe hedor del

Wumpus. Una vez escogida la acción, se ejecuta y se vuelve a preguntar si el juego cumple alguna de las condiciones de fin del juego descritas anteriormente.

Diagrama de flujo cálculo de probabilidades de movimiento

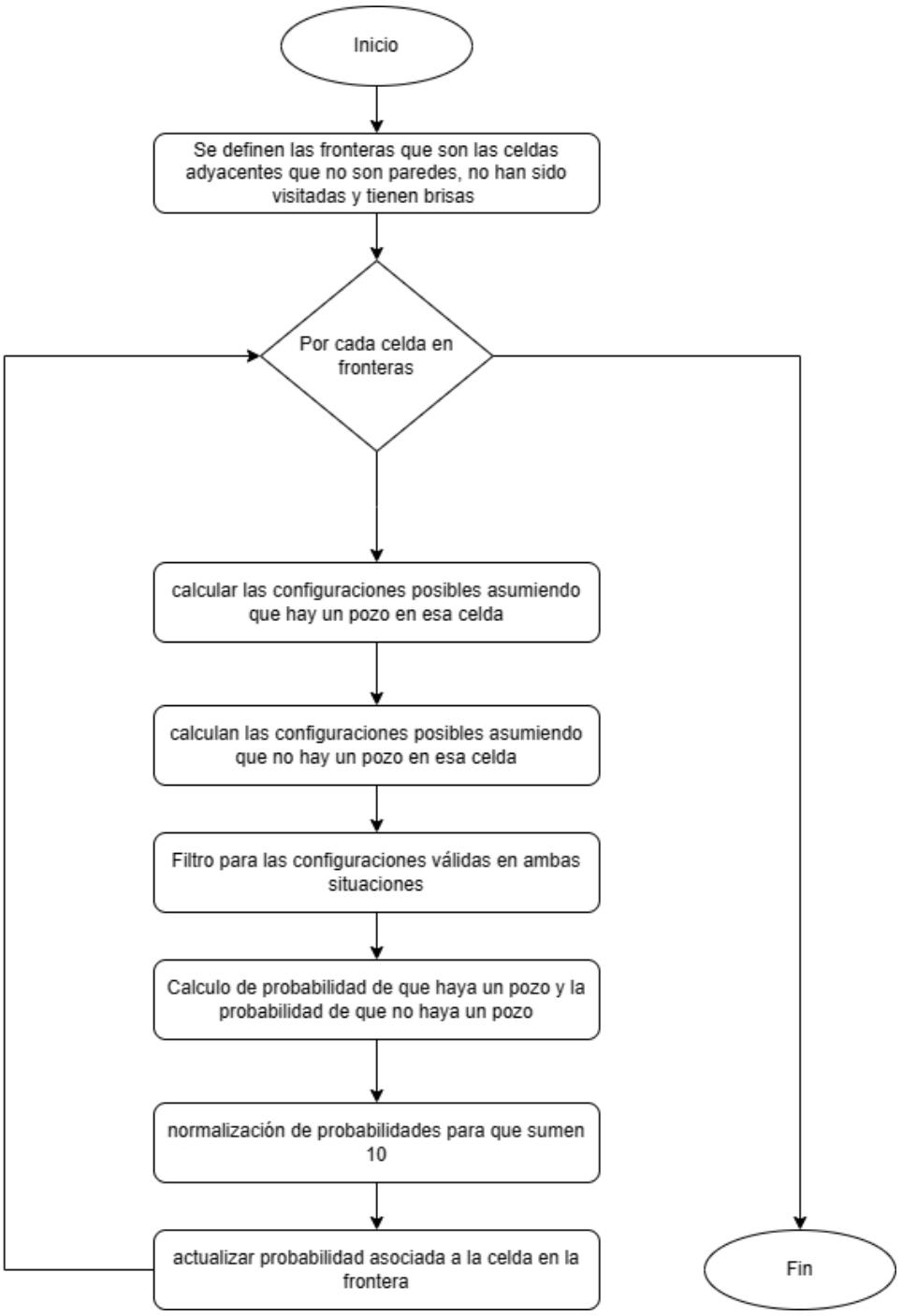

Figura 3. Flujo principal de la función del cálculo de probabilidades

En el flujo se puede observar la función que maneja el cálculo de la probabilidad para el movimiento de nuestro agente, el cual realiza este cálculo basándose en las brisas que genera los pozos y en el olor que genera el Wumpus.