Ho Chi Minh City University

University of Science

Faculty of Information Technology.

LOGICAL AGENT

Project: LOGICAL AGENT

Course name: CSC14003 – Artificial Intelligence

Members

Võ Phương Nam (22127289)

Nguyễn Hoàng Khang

(22127178)

Trương Hà Huy Tâm (22127375)

Project advisor:

Nguyễn Trần Duy Minh

19th August 2024

Contents

Chapter 1

Overall Evaluation

1.1 Task

- The project is about the infamous Wumpus World game[5]. In this project, the game has been elevated to a new level by expanding the map to 10x10, as well as adding new elements such as a Poison Gas with Whiff surrounding, as well as a Potion with Glow surrounding.
- Overall task progress:

- The teammates have fully understood the task and perform the task relatively well. Our first approach is to use the Propositional Logic[2], which can be achieved using PySAT library[1]
- The jobs are divided fairly equally so that the members can perform as well as communicate with each other effectively.
- However, we have to deal with this project during the moments where there are two other ongoing projects from two other subjects, hence the progress of the whole team goes noticeably down.

1.2 Overall product

- The final product basically satisfies the requirements of the project.
- The agent tried his best to perform the task, and he made it back to the starting point.
- However, the agent's intelligence can still be further improved and he's capable of collection more gold, but with his current intelligence, it's not fully enough to get all of the available gold.

Chapter 2

Work Assignment and Evaluation

This project has been done by the following memers:

1. Vo Phuong Nam – Student ID: 22127289
2. Nguyen Hoang Khang – Student ID: 22127178
3. Truong Ha Huy Tam – Student ID: 22127375

The project is done based on a sample project in a much simpler version without the Potion and Poisonous Gas conditions [3]

All of the works are done in Visual Studio Code and compiled using Python.

Here is the table of work assignment of what each member has done

Work Assignment Table			
No.	Task description	Assigned to	Completion rate
1	Finish problem successfully	Truong Ha Huy Tam	100%
2	Graphical demonstration of each step of the running process.	Vo Phuong Nam	100%
3	Generate at least 5 maps with difference structures such as position and number of Pit, Gold and Wumpus	Vo Phuong Nam	100%
4	Report algorithms, experiment with some reflection or comments.	Nguyen Hoang Khang	100%

Chapter 3

Folder Structure and Execution Instructions

3.1 Folder structure

```
./AI-project2
├── assets/
│   ├── fonts/
│   │   └── CenturyGothic.ttf/
│   ├── images/
│   │   ├── agent_down.png/
│   │   ├── agent_left.png/
│   │   ├── agent_right.png/
│   │   ├── agent_up.png/
│   │   ├── arrow_down.png/
│   │   ├── arrow_left.png/
│   │   ├── arrow_right.png/
│   │   ├── arrow_up.png/
│   │   ├── background_game.png/
│   │   ├── breeze.png/
│   │   ├── cell.png/
│   │   ├── gas.png/
│   │   ├── gold.png/
│   │   ├── pit.png/
│   │   ├── potion.png/
│   │   ├── stench.png/
│   │   ├── visited_cell.png/
│   │   ├── wumpus_dead.png/
│   │   └── you_win.png/
│   ├── input/
│   │   ├── map_1.txt/
│   │   ├── ...
│   │   └── map_5.txt/
│   └── output/
│       ├── output_1.txt/
│       ├── ...
│       └── output_5.txt/
├── source/
│   ├── Algorithms.py/
│   ├── agent.py/
│   ├── cells.py/
│   ├── graphic.py/
│   ├── knowledge.py/
│   ├── main.py/
│   ├── map.py/
│   ├── object.py/
│   └── requirements.txt/
└── README.txt/
```

In details:

- **Fonts folder:** contains information about fonts used for the app.
- **Images folder:** includes information about images used for the app.
- **Input folder:** includes information about game's maps.
- **Output folder:** includes information about the logic for maps corresponding to the Input folder.
- **Source folder:** includes the necessary files to execute the program.
- **requirements.txt file:** includes the necessary libraries to be installed to python.
- **main.py file:** the main execution file of the project.

3.2 Execution instructions

To execute the program, follow these steps

- **Step 1:** Open the console at the same directory level as the *main.py* file.
- **Step 2:** Install the necessary packages listed in the *requirements.txt* file using the command:

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

- **Step 3:** To run the program, use one of the following commands: *py main.py* or *python main.py*.

Chapter 4

Function Description

4.1 Class Cell

4.1.1 Constructor `__init__` in class Cell

- **Input:**
  - `matrix_pos (tuple)`: The position in the matrix, represented as a tuple of coordinates (row, column).
  - `map_size (int)`: The size of the map, representing the dimensions (assumed to be square).
  - `objects_str (str)`: A string that represents objects present at this position, used for initialization.
- **Output:** None (Constructor function).
- **Purpose:** Initializes an instance of a class, setting up positional attributes in both matrix and map coordinates, calculating a unique index position, and initializing various properties related to exploration status and perception of the environment.
- **How it works:**
  1. Converts the matrix position `matrix_pos` to a map position `map_pos`, adjusting for different coordinate systems.
  2. Calculates a unique `index_pos` based on the map position, which can be used to uniquely identify each location in a linear fashion.
  3. Initializes the `explored` attribute to `False`, indicating that the position has not been explored yet.
  4. Initializes a `percept` list with 9 boolean values set to `False`, representing the absence of certain environmental features.
  5. Calls the `init_map` function to further initialize the map based on `objects_str`.
  6. Sets `parent` to `None` and initializes an empty `child_list` to store potential child nodes or related positions.

4.1.2 Function init_map in class Cell

- **Input:**
  - `map_str (str)`: A string representing the objects present in a specific map cell.
- **Output:** None.
- **Purpose:** To update the `percept` list by setting corresponding boolean values to `True` based on the objects found in the map cell, as represented by the input string.
- **How it works:**
  1. Iterates over each character in `map_str`.
  2. For each character, checks which object it represents using a series of conditional statements:
     - If the character matches `Object.GOLD.value`, sets the first element of `percept` to `True`.
     - If it matches `Object.PIT.value`, sets the second element of `percept` to `True`.
     - Similarly, other object types like `WUMPUS`, `BREEZE`, `STENCH`, `POISON_GAS`, `HEALTH_POT`, `WHIFF`, and `GLOW` are mapped to their respective indices in the `percept` list and set to `True`.
  3. Ignores certain objects like `AGENT` and `EMPTY`.
  4. If an unrecognized character is encountered, raises a `TypeError`.

4.1.3 Group of exist_ functions in class Cell

- **Consist of these function:** exist_gold, exist_pit, exist_wumpus, exist_breeze, exist_stench, exist_poison, exist_health_pot, exist_whiff, exist_glow.
- **Input:** None
- **Output:** bool: Returns True if object is present in the current cell, otherwise False.
- **Purpose:** Checks whether object is present in the current cell.

- **How it works:**

  1. Returns the value of the object's element in the percept list, indicating the presence of said object.

4.1.4 Function is_OK in class Cell

- **Input:**

  - health (int): The health status of the entity (typically represented as an integer).

- **Output:** bool: Returns True if the current cell is considered safe, otherwise False.

- **Purpose:** Determines if the current cell is safe based on the presence of specific hazards and the health status.

- **How it works:**

  1. Checks if there is no breeze in the cell (exist_breeze() returns False).
  2. Checks if there is no stench in the cell (exist_stench() returns False).
  3. Checks if there is no whiff in the cell or if whiff exists but health is greater than 1 (exist_whiff() returns False or health is not equal to 1).
  4. Returns True if all the above conditions are met, indicating the cell is safe; otherwise, returns False.

4.1.5 Function update_parent in class Cell

- **Input:**

  - `parent_cell (Cell)`: An instance of the `Cell` class representing the parent cell to be assigned.

- **Output:** None.

- **Purpose:** To update the `parent` attribute of the current cell, linking it to the provided `parent_cell`.

- **How it works:**

  1. Assigns the `parent_cell` to the `parent` attribute of the current cell instance.

4.1.6 Function grab_gold in class Cell

- **Input:** None.
- **Output:** None.
- **Purpose:** To simulate the agent grabbing gold from the current cell, updating the cell's percepts.
- **How it works:**

  1. The function removes the health potion percept (index 0) from the current cell by setting it to `False`.

4.1.7 Function grab_health_pot in class Cell

- **Input:**
  - `cell_matrix (list[list[Cell]])`: A 2D list representing the grid of cells in the environment.
  - `kb (KnowledgeBase)`: The knowledge base that stores the logical clauses used for reasoning.
- **Output:** None.
- **Purpose:** To simulate the agent grabbing a health potion from the current cell, updating the cell's percepts and adjusting related percepts in adjacent cells.
- **How it works:**

  1. The function removes the health potion percept (index 6) from the current cell by setting it to `False`.
  2. It retrieves the list of adjacent cells to the current cell.

3. The function then iterates through these adjacent cells to manage the "Glow" percept (index 8), checking if any neighboring cells still have a health potion.
4. If no adjacent cells have a health potion, the "Glow" percept is removed from the adjacent cells, and corresponding clauses in the knowledge base are updated:
   - The positive glow literal is deleted from the knowledge base.
   - The negative glow literal is added to the knowledge base.
   - Implications involving the glow percept and adjacent health potions are removed.

4.1.8 Function kill_wumpus in class Cell

- **Input:**
  - `cell_matrix (list[list[Cell]])`: A 2D list representing the grid of cells in the environment.
  - `kb (KnowledgeBase)`: The knowledge base that stores the logical clauses used for reasoning.
- **Output:** None.
- **Purpose:** To handle the effects of killing a Wumpus in the current cell, including updating the percepts and knowledge base related to the "Stench" caused by the Wumpus.
- **How it works:**
  1. The function removes the "Wumpus" percept (index 2) from the current cell by setting it to `False`.
  2. It retrieves the list of adjacent cells to the current cell.
  3. The function iterates through these adjacent cells to manage the "Stench" percept (index 4). It checks if any neighboring cells still have a Wumpus.
  4. If no adjacent cells contain a Wumpus, the "Stench" percept is removed from the adjacent cells, and corresponding clauses in the knowledge base are updated:
     - The positive stench literal is deleted from the knowledge base.
     - The negative stench literal is added to the knowledge base.

– Implications involving the stench percept and adjacent Wumpus cells are removed.

4.1.9 Function get_adj_cell_list in class Cell

- **Input:**
  - `cell_matrix (list[list[Cell]])`: A 2D list representing the grid of cells in the environment.
- **Output:**
  - `adj_cell_list (list[Cell])`: A list of adjacent cells to the current cell.
- **Purpose:** To retrieve a list of adjacent cells surrounding the current cell within the boundaries of the grid.
- **How it works:**
  1. The function creates a list of possible positions for adjacent cells, including those to the right, left, above, and below the current cell.
  2. It iterates through these positions, checking if each one is within the valid range of the grid (i.e., not out of bounds).
  3. If a position is valid, the corresponding cell from the `cell_matrix` is added to the list of adjacent cells.
  4. The function returns the list of valid adjacent cells.

4.1.10 Function is_explored in class Cell

- **Input:** None
- **Output:** bool: Returns the current exploration status.
- **Purpose:** Checks if the current object has been explored.
- **How it works:**
  1. Returns the value of the explored attribute, indicating whether the object has been explored.

4.1.11 Function explore in class Cell

- **Input:** None
- **Output:** None
- **Purpose:** Marks the current object as explored.
- **How it works:**
  1. Sets the explored attribute to True, indicating that the object has been explored.

4.1.12 Function update_child_list in class Cell

- **Input:**
  - valid_adj_cell_list (list): A list of valid adjacent cells.
- **Output:** None
- **Purpose:** Adds valid adjacent cells to the current cell's child list and updates their parent to the current cell.
- **How it works:**
  1. Iterates through each cell in valid_adj_cell_list.
  2. If the parent attribute of an adjacent cell is None, it appends the cell to the child_list and calls update_parent on the adjacent cell to set the current cell as its parent.

4.1.13 Function get_literal in class Cell

- **Input:**
  - obj (Object): The type of object for which a literal is needed.
  - sign (str, optional): A string representing the sign of the literal, default is '+'.
- **Output:** int: A unique literal representing the object and its position, optionally negated.

- **Purpose:** Generates a unique literal based on the object type and position, which can be used in logical expressions.
- **How it works:**
  1. Maps the input obj to an integer value based on its type (e.g., Object.PIT is mapped to 1).
  2. Calculates the literal by combining the mapped integer with the cell's index position, adjusted by a factor based on the map size.
  3. If sign is '-', the literal is multiplied by -1.
  4. Returns the computed literal.

4.2 Class AgentBrain

4.2.1 Constructor __init__ in class AgentBrain

- **Input:**
  - map_filename (str): The filename of the map to be read and used for initialization.
  - output_filename (str): The filename where output will be stored.
- **Output:** None
- **Purpose:** Initializes an instance of the class by setting up various attributes and reading the map data.
- **How it works:**
  1. Sets the output_filename attribute.
  2. Initializes attributes related to the map (map_size, cell_matrix, init_cell_matrix) and agent (cave_cell, agent_cell, init_agent_cell).
  3. Creates a knowledgeBase instance and initializes lists for path and action_list.
  4. Initializes game-related attributes like score, health, max_health, and count_potion.
  5. Calls the read_map method with the map_filename to load and process the map data.

4.2.2 Function read_map in class AgentBrain

- **Input:**
  - map_filename (str): The filename of the map to be read and used for initialization.
- **Output:** None
- **Purpose:** Reads the map data from a file, initializes the cell matrix, and validates the map.
- **How it works:**
  1. Opens the file specified by map_filename and reads the map size from the first line.
  2. Reads the map data into a list of lists, where each cell's content is separated by a period (".").
  3. Initializes cell_matrix as a 2D list of None values, with dimensions based on map_size.
  4. Iterates over each row and column of the map data:
     - Creates a Cell object for each position in the matrix and stores it in cell_matrix.
     - If the cell contains the agent, sets agent_cell to this cell and updates its parent to cave_cell. A deep copy of this cell is stored in init_agent_cell.
  5. Creates a deep copy of cell_matrix and stores it in init_cell_matrix.
  6. Calls the is_valid_map method to validate the map:
     - If the map is invalid and no agent is found, raises a TypeError indicating that the map is invalid.
     - If the map is invalid and a specific invalid position is found, raises a TypeError indicating the row and column of the error.

4.2.3 Function is_valid_map in class AgentBrain

- **Input:** None
- **Output:**
  - bool: Returns True if the map is valid, otherwise False.

  - tuple or None: If the map is invalid, returns the position (row, column) of the invalid cell; if the map is valid, returns None.

- **Purpose:** Validates the map by ensuring the presence of required percepts around pits and Wumpuses, and checks if an agent is present on the map.

- **How it works:**

  1. Iterates through each row of cell_matrix.
  2. For each cell, retrieves the list of adjacent cells using get_adj_cell_list.
  3. Checks for the presence of pits:
     - If a pit exists in the cell, verifies that all adjacent cells contain a breeze.
     - If any adjacent cell does not contain a breeze, returns False and the position of the current cell.
  4. Checks for the presence of Wumpuses:
     - If a Wumpus exists in the cell, verifies that all adjacent cells contain a stench.
     - If any adjacent cell does not contain a stench, returns False and the position of the current cell.
  5. After validating all cells, checks if the agent is present on the map.
  6. If no agent is found, returns False and None.
  7. If all checks pass, returns True and None.

4.2.4 Function append_event_to_output_file in class AgentBrain

- **Input:**

  - text (str): The event text to be appended to the output file.

- **Output:** None

- **Purpose:** Appends a given text event to the output file, followed by a newline.

- **How it works:**

1. Opens the file specified by output_filename in append mode ('a').
2. Writes the provided text followed by a newline character to the file.
3. Closes the file automatically when the with block is exited.

4.2.5 Function add_action in class AgentBrain

- **Input:**
  - action (Action): The action to be added and executed.
- **Output:** None
- **Purpose:** Adds an action to the action list, updates the score, health, and other relevant attributes based on the action, and logs the action to the output file.
- **How it works:**
  1. Appends the action to the action_list.
  2. Prints the action and logs the action name to the output file using append_event_to_output_file.
  3. Updates the score, health, and other attributes based on the specific action:
     - For TURN_LEFT, TURN_RIGHT, TURN_UP, TURN_DOWN, and MOVE_FORWARD, deducts 10 points from the score.
     - For GRAB_GOLD, adds 5000 points to the score.
     - For GRAB_POTION, increases the count_potion by 1.
     - For SHOOT, deducts 100 points from the score.
     - For BE_EATEN_BY_WUMPUS and FALL_INTO_PIT, deducts 10,000 points from the score.
     - For SNIFF_GAS, reduces health by 1.
     - For HEAL, if potions are available and health is below the maximum, consumes a potion to restore health.
     - For CLIMB_OUT_OF_THE_CAVE, adds 10 points to the score.

4. Handles additional actions (KILL_ALL_WUMPUS_AND_GRAB_ALL_FOOD, DETECT_PIT, DETECT_WUMPUS, etc.) that currently pass without performing any operation.
5. Raises a TypeError if an unrecognized action is passed.

4.2.6 Function add_new_percepts_to_KB in class AgentBrain

- **Input:**
  - cell (Cell): The cell whose percepts are to be added to the knowledge base.
- **Output:** None
- **Purpose:** Adds new percepts from a given cell to the knowledge base (KB) and ensures logical consistency between the presence of hazards and percepts.
- **How it works:**
  1. Retrieves the list of adjacent cells using get_adj_cell_list.
  2. Adds clauses to the knowledge base for each percept detected in the cell:
     - For pits, Wumpuses, poison gas, health potions, breezes, stench, whiffs, and glow, the function adds literals to the KB based on whether the percept is present.
     - If both a pit and Wumpus are detected in the same cell, raises a TypeError since this is logically invalid.
  3. For each percept like breeze, stench, whiff, and glow, adds logical implications to the KB:
     - Adds implications from the percept to the presence of the corresponding hazard in one of the adjacent cells.
     - Adds implications from the presence of the hazard in adjacent cells to the percept being true in the current cell.
  4. If no percept is detected, adds clauses to the KB stating that the corresponding hazard does not exist in adjacent cells.
  5. Prints the updated KB and logs it to the output file using append_event_to_output_file.

4.2.7 Function turn_to in class AgentBrain

- **Input:**
  - next_cell (Cell): The next cell that the agent should turn towards.
- **Output:** None
- **Purpose:** Determines the direction the agent should turn to face the next_cell and adds the appropriate turn action to the action list.
- **How it works:**
  1. Compares the map_pos of next_cell with that of agent_cell to determine the direction:
     - If the x-coordinates (map_pos[0]) are the same, it checks the difference in y-coordinates (map_pos[1]) to determine whether to turn up or down.
     - If the y-coordinates (map_pos[1]) are the same, it checks the difference in x-coordinates (map_pos[0]) to determine whether to turn right or left.
  2. Calls add_action with the appropriate turn action (TURN_UP, TURN_DOWN, TURN_RIGHT, or TURN_LEFT).
  3. Raises a TypeError if next_cell is not directly adjacent to agent_cell (i.e., the cells do not share a row or column).

4.2.8 Function move_to in class AgentBrain

- **Input:**
  - next_cell (Cell): The cell that the agent should move to.
- **Output:** None
- **Purpose:** Directs the agent to turn towards and move to the specified next_cell.
- **How it works:**
  1. Calls the turn_to method to orient the agent towards the next_cell.
  2. Adds the MOVE_FORWARD action to the action list using add_action.

3. Updates the agent_cell to the next_cell, indicating the agent has moved to this new position.

4.2.9 Function backtracking_search in class AgentBrain

- **Input:** None
- **Output:**
  - bool: Returns True if the search completes successfully, otherwise False if the agent encounters a fatal hazard.
- **Purpose:** Explores the environment using a backtracking approach, identifying hazards, collecting items, updating the knowledge base (KB), and making decisions based on inferred information.
- **How it works:**
  1. Checks the current cell (agent_cell) for hazards:
     - If a pit is present, adds the FALL_INTO_PIT action, and the search terminates with False.
     - If a Wumpus is present, adds the BE_EATEN_BY_WUMPUS action, and the search terminates with False.
     - If poison gas is present, adds the SNIFF_GAS action and reduces health; if health is zero, the search terminates with False.
  2. If gold is present, adds the GRAB_GOLD action and removes the gold from the cell.
  3. Adds percepts such as breeze, stench, whiff, and glow to the action list if detected in the current cell.
  4. If the cell is not explored, marks it as explored and updates the KB with the cell's percepts.
  5. Retrieves a list of valid adjacent cells for further exploration.
  6. If the current cell is deemed unsafe, attempts to infer the presence or absence of hazards like pits, Wumpuses, and poison gas in adjacent cells by:

  - Using logical inference based on the KB.
  - Adding appropriate actions and updating the KB based on the inference results.
  - If hazards are confirmed or ruled out, updates the exploration path and KB.

7. Updates the list of unexplored child cells and recursively explores each child cell using backtracking.
8. If a path leads to a dead end, backtracks to the previous cell and continues searching.
9. Returns True if the search completes without encountering a fatal hazard.

4.2.10 Function solve_wumpus_world in class AgentBrain

- **Input:** None
- **Output:**
  - tuple: A tuple containing the list of actions performed (action_list), the initial agent cell (init_agent_cell), and the initial cell matrix (init_cell_matrix).
- **Purpose:** Solves the Wumpus World problem by exploring the environment, avoiding hazards, collecting items, and determining when the agent can safely exit the cave.
- **How it works:**
  1. Clears the output file by opening it in write mode and immediately closing it.
  2. Initiates the backtracking search to explore the environment and handle hazards, item collection, and updates to the knowledge base.
  3. Sets a victory_flag to True and checks if there is any remaining gold or Wumpus in the environment:
     - If no gold or Wumpus is left, the agent adds the KILL_ALL_WUMPUS_AND_GRAB_ALL_ action.
  4. Checks if the agent is back at the starting point (parent cell is the cave entrance) and, if so, adds the CLIMB_OUT_OF_THE_CAVE action.
  5. Returns a tuple containing:

- The list of actions performed (action_list).
- The initial agent cell (init_agent_cell).
- The initial state of the cell matrix (init_cell_matrix).

4.3 Class graphics

4.3.1 Constructor __init__ in class graphics

- **Input:** None
- **Output:** None
- **Purpose:** Initializes the Wumpus World game by setting up the Pygame environment[4], loading assets, and defining initial game state variables.
- **How it works:**

1. Initializes Pygame and its font module using pygame.init() and pygame.font.init().
2. Creates the game window with a resolution of 970x710 pixels and sets the window caption to "Wumpus World".
3. Initializes the game clock using pygame.time.Clock() to control the frame rate.
4. Initializes various game elements (map, agent, gold, potion, wumpus, pit, gas, arrow) to None.
5. Loads fonts for different text displays:
   - Main font (font) for general text.
   - Notification font (noti) for smaller messages.
   - Victory font (victory) for large victory messages.
6. Initializes a Pygame sprite group (all_sprites) to manage all game sprites.
7. Sets the initial game state to 'menu', with map_i initialized to 1 (indicating the first map).
8. Initializes the mouse to None.

9. Loads and scales the background image from the specified file path to fit the game window.
10. Sets the initial direction of the agent to 0 (default direction).

4.3.2 Function draw_button in class graphics

- **Input:**
  - surf (pygame.Surface): The surface on which the button will be drawn.
  - rect (pygame.Rect): The rectangle defining the position and size of the button.
  - button_colour (tuple): The RGB color tuple for the button background.
  - text_colour (tuple): The RGB color tuple for the text on the button.
  - text (str): The text to be displayed on the button.
- **Output:** None
- **Purpose:** Draws a button with specified text, colors, and position on the provided surface.
- **How it works:**
  1. Draws a rectangle on the specified surf surface using pygame.draw.rect, with the given button_colour and dimensions defined by rect.
  2. Renders the text with the specified text_colour using the main font (self.font).
  3. Calculates the center position for the text to align it within the button using the rectangle's center.
  4. Blits the rendered text onto the button on the main screen (self.screen) at the calculated position.

4.3.3 Function game_draw in class graphics

- **Input:** None
- **Output:** None

- **Purpose:** Renders the game screen, displaying the map, the player's score, health, and the number of potions.
- **How it works:**
  1. Fills the entire screen with the color white (WHITE), clearing any previous content.
  2. Calls the draw method on self.map to render the game map onto the screen.
  3. Retrieves the player's current score using self.agent.get_score().
  4. Retrieves the player's current health using self.agent.get_health().
  5. Retrieves the number of potions the player has using self.agent.get_num_of_potion().
  6. Renders the score text using the main font (self.font) in black (BLACK) and centers it at coordinates (827, 25).
  7. Renders the health text using the main font in black and centers it at coordinates (827, 75).
  8. Renders the number of potions text using the main font in black and centers it at coordinates (827, 125).
  9. Blits (draws) all three pieces of text onto the screen at their respective positions.

4.3.4 Function win_event in class graphics

- **Input:** None
- **Output:** None
- **Purpose:** Handles events during the win screen, including quitting the game, updating the display, and transitioning the game state back to the menu.
- **How it works:**
  1. Iterates through the list of events captured by pygame.event.get().
  2. Checks if the event type is pygame.QUIT:
     - If true, quits Pygame using pygame.quit() and exits the program using sys.exit().

3. Updates the display using pygame.display.update() to ensure any visual changes are reflected.
4. Introduces a brief delay of 200 milliseconds using pygame.time.delay(200) to provide a smoother transition.
5. Changes the game state to 'menu', indicating that the game should return to the main menu after the win screen is displayed.

4.3.5 Function home_draw in class graphics

- **Input:** None
- **Output:** None
- **Purpose:** Renders the home screen by filling the screen with a white background.
- **How it works:**
  1. Fills the entire screen with the color white (WHITE), effectively clearing any previous content and preparing the screen for further drawing or interaction.

4.3.6 Function home_event in class graphics

- **Input:** None
- **Output:** None
- **Purpose:** Handles user interactions on the home screen, such as selecting a map to start the game or exiting the application.
- **How it works:**
  1. Iterates through the list of events captured by pygame.event.get().
  2. Checks if the event type is pygame.QUIT:
     - If true, quits Pygame using pygame.quit() and exits the program using sys.exit().
  3. Checks if the event type is pygame.MOUSEBUTTONDOWN:

- If the mouse click is within the coordinates of a map button, changes the game state to 'game' and sets the map index (map_i) to the corresponding map number.
- If the mouse click is within the coordinates of the exit button, quits Pygame and exits the program.

4. Updates the current mouse position using pygame.mouse.get_pos().
5. Changes the appearance of the buttons based on the mouse position:
   - If the mouse is hovering over a button, draws the button with a darker background (DARK_GREY) and red text (RED).
   - If the mouse is not hovering over a button, draws the button with a lighter background (LIGHT_GREY) and black text (BLACK).
6. Updates the display using pygame.display.update() to reflect any changes.

4.3.7 Function run in class graphics

- **Input:** None
- **Output:** None
- **Purpose:** Runs the main game loop, managing the game's state transitions, rendering, and handling events.
- **How it works:**

1. Enters an infinite loop that continuously checks and handles the current game state.
2. If the game state is 'menu':
   - Calls home_draw to render the home screen.
   - Calls home_event to handle user input on the home screen.
3. If the game state is 'game':
   - Changes the game state to 'try'.
   - Initializes the AgentBrain object and solves the Wumpus World problem for the selected map, retrieving the action list, cave cell, and cell matrix.

- Initializes the map, arrow, gold, potion, agent, and game objects (pit, Wumpus, poison) based on the cell matrix and sets up the sprite group.
- Draws the game screen using game_draw.
- Iterates over the action_list to execute and display each action:
  - Listens for quit events, allowing the player to close the game during the action sequence.
  - Calls display_action to perform and display the current action.
  - Updates the display using pygame.display.flip() and controls the frame rate with clock.tick(30).
  - If the action is KILL_ALL_WUMPUS_AND_GRAB_ALL_FOOD, sets the game state to 'win'.
  - If the action results in the agent's death (FALL_INTO_PIT, BE_EATEN_BY_WUMPUS or DIE_OF_GAS), sets the game state to 'gameover' and breaks the loop.
- Checks for quit events again to ensure the game can be closed after the action loop.

4. If the game state is 'win' or 'try':
   - Calls win_draw to render the win screen.
   - Calls win_event to handle events on the win screen.
5. Controls the overall frame rate of the game loop using clock.tick(60).

4.3.8 Function display_action in class graphics

- **Input:**
  - action (Algorithms.Action): The action to be displayed and executed within the game.
- **Output:** None
- **Purpose:** Executes and visually displays the result of various game actions, updating the game state, rendering changes, and managing animations.
- **How it works:**
  1. Based on the input action, the function performs the corresponding operations:

- TURN_LEFT, TURN_RIGHT, TURN_UP, TURN_DOWN: Adjusts the agent's direction, updates all sprites, redraws the game screen, and updates the display.
- MOVE_FORWARD: Moves the agent in the current direction, updates the map to discover the new cell, updates sprites, redraws the screen, and updates the display.
- GRAB_GOLD, GRAB_POTION: Executes the corresponding action, updates sprites, redraws the screen, and performs a short delay to show the animation.
- SHOOT: Handles the shooting action, including animating the arrow, and updates the display.
- KILL_WUMPUS: Marks the Wumpus as killed, updates the game state, and performs a short delay to show the animation.
- SNIFF_GAS: Handles the agent grabbing poison, updates the sprites, and performs a short delay to show the animation.
- BE_EATEN_BY_WUMPUS, FALL_INTO_PIT, DIE_OF_GAS: Handles the death of the agent, updates the game state to 'gameover', and updates the display.
- HEAL: Uses a health potion, updates the sprites, and redraws the screen.
- KILL_ALL_WUMPUS_AND_GRAB_ALL_FOOD: Sets the game state to 'win'.
- CLIMB_OUT_OF_THE_CAVE: Handles the agent climbing out of the cave, updates the screen, and performs a delay for the animation.
- DETECT_PIT, DETECT_WUMPUS, DETECT_NO_PIT, DETECT_NO_WUMPUS, DETECT_SAFE, DETECT_GAS, DETECT_NO_GAS: Handles detection actions, updates the sprites, and redraws the screen.
- INFER_PIT, INFER_NOT_PIT, INFER_WUMPUS, INFER_NOT_WUMPUS, INFER_SAFE, INFER_GAS, INFER_NOT_GAS: Handles inference actions without additional updates.
- PERCEIVE_BREEZE, PERCEIVE_STENCH, PERCEIVE_GLOW, PERCEIVE_WHIFF: Handles perceptive actions without additional updates.

2. For each action, the game display is updated to reflect the changes, and a delay is added where necessary to show animations.
3. If an action is not recognized, raises a TypeError with an appropriate error message.

4.4 Class knowledgeBase[1]

4.4.1 Constructor __init__ in class knowledgeBase

- **Input:** None
- **Output:** None
- **Purpose:** Initializes an instance of the class by setting up the knowledge base (KB) as an empty list.
- **How it works:**
  1. Initializes the KB attribute as an empty list (self.KB = []), which will be used to store knowledge or clauses for the class instance.

4.4.2 Static Method standardize_clause in class knowledgeBase

- **Input:**
  - clause (list): A list representing a clause, possibly containing duplicate elements.
- **Output:**
  - list: A sorted list with unique elements, representing the standardized clause.
- **Purpose:** Standardizes a clause by removing duplicate elements and sorting the remaining elements in ascending order.
- **How it works:**
  1. Converts the input clause list to a set using set(clause) to remove any duplicate elements.
  2. Converts the set back to a list and sorts it using sorted(list(set(clause))).
  3. Returns the sorted list as the standardized clause.

4.4.3 Function add_clause in class knowledgeBase

- **Input:**
  - clause (list): A list representing a clause to be added to the knowledge base.
- **Output:** None
- **Purpose:** Adds a standardized clause to the knowledge base (KB) if it is not already present.
- **How it works:**
  1. Calls the standardize_clause method to remove duplicates from the clause and sort it.
  2. Checks if the standardized clause is not already in the knowledge base (KB).
  3. If the clause is not present, appends it to the KB.

4.4.4 Function del_clause in class knowledgeBase

- **Input:**
  - clause (list): A list representing a clause to be deleted from the knowledge base.
- **Output:** None
- **Purpose:** Removes a standardized clause from the knowledge base (KB) if it is present.
- **How it works:**
  1. Calls the standardize_clause method to remove duplicates from the clause and sort it.
  2. Checks if the standardized clause is present in the knowledge base (KB).
  3. If the clause is found in the KB, removes it from the list.

4.4.5 Function infer in class knowledgeBase

- **Input:**
  - not_alpha (list): A list of clauses representing the negation of a proposition (alpha) to be tested for inference.

- **Output:**
  - bool: Returns True if the negation of the proposition (not_alpha) is unsatisfiable, meaning the proposition is logically inferred from the knowledge base. Returns False otherwise.
- **Purpose:** Determines whether a proposition can be logically inferred from the knowledge base (KB) by checking the satisfiability of its negation using a SAT solver.
- **How it works:**
  1. Initializes a Glucose3 SAT solver instance (glu).
  2. Creates a deep copy of the current knowledge base (clause_list).
  3. Adds each clause from clause_list to the SAT solver.
  4. Adds each clause from neg_alpha (the negation of the proposition) to the SAT solver.
  5. Solves the SAT problem using glu.solve():
     - If the SAT solver finds a solution (sol is True), the negation of the proposition is satisfiable, so the function returns False.
     - If the SAT solver finds no solution (sol is False), the negation is unsatisfiable, meaning the proposition can be inferred, so the function returns True.

4.5 Class Map

4.5.1 Constructor __init__ in class Map

- **Input:**
  - init_agent_pos (tuple): A tuple representing the initial position of the agent in the format (row, column).
- **Output:** None
- **Purpose:** Initializes an instance of the class by setting up game elements related to the map, including cell sizes, images, and the discovery status of cells.
- **How it works:**

1. Initializes the spacing between elements (self.space) and the size of the map (self.size) to 10.
2. Sets the size of each cell in the map to 60 pixels (self.cell_size).
3. Loads the image for a generic cell from the specified file path and stores it in self.cell.
4. Loads the image for a pit from the specified file path and stores it in self.pit.
5. Initializes a 10x10 grid (self.pit_discover) with all values set to False, representing undiscovered pits.
6. Loads the image for a discovered cell from the specified file path and stores it in self.discover_cell.
7. Initializes a 10x10 grid (self.is_discover) with all values set to False, representing undiscovered cells.
8. Marks the initial agent position (init_agent_pos) as discovered by setting the corresponding cell in self.is_discover to True.

4.5.2 Function draw in class Map

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which the map will be drawn.
- **Output:** None
- **Purpose:** Renders the map onto the screen, displaying either a generic cell, a discovered cell, or a pit based on the current game state.
- **How it works:**
  1. Initializes x and y coordinates to self.space to set the starting position for drawing.
  2. Iterates over each cell in the 10x10 grid (self.size):
     - If the cell is marked as discovered (self.is_discover[i][j] is True), blits the discovered cell image (self.discover_cell) onto the screen at the current (x, y) position.
     - If the cell is not discovered (self.is_discover[i][j] is False) but contains a pit (self.pit_discover[i][j] is True), blits the pit image (self.pit) onto the screen at the current (x, y) position.

  - If the cell is neither discovered nor contains a pit, blits the generic cell image (self.cell) onto the screen at the current (x, y) position.
  - After drawing each cell, updates the x coordinate by adding the cell size and spacing to position the next cell horizontally.

3. After completing a row, resets x to self.space and updates the y coordinate by adding the cell size and spacing to position the next row vertically.

4.5.3 Function discover_new_cell in class Map

- **Input:**
  - i (int): The row index of the cell to be marked as discovered.
  - j (int): The column index of the cell to be marked as discovered.
- **Output:** None
- **Purpose:** Marks a specific cell on the map as discovered.
- **How it works:**

1. Sets the value of self.is_discover[i][j] to True, indicating that the cell at position (i, j) has been discovered.

4.5.4 Function discovered in class Map

- **Input:** None
- **Output:**
  - list: A 2D list representing the discovery status of all cells on the map (self.is_discover).
- **Purpose:** Returns the discovery status of all cells on the map.
- **How it works:**

1. Returns the self.is_discover 2D list, which indicates which cells have been discovered.

4.5.5 Function agent_climb in class Map

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display the message.
  - font (pygame.font.Font): The font to use for rendering the text.
- **Output:** None
- **Purpose:** Displays a message indicating that the agent has climbed out of the cave, along with a score increase.
- **How it works:**
  1. Renders the text "Agent climbed out!" using the provided font in black color (0, 0, 0).
  2. Centers the text at coordinates (830, 100) and blits it onto the screen.
  3. Renders the text "+ 10" using the same font and color.
  4. Centers the text at coordinates (850, 150) and blits it onto the screen.

4.5.6 Function pit_detect in class Map

- **Input:**
  - i (int): The row index of the cell where the pit was detected.
  - j (int): The column index of the cell where the pit was detected.
- **Output:** None
- **Purpose:** Marks a specific cell on the map as containing a pit.
- **How it works:**
  1. Sets the value of self.pit_discover[i][j] to True, indicating that a pit has been detected at the cell position (i, j).

4.6 Class Pit

4.6.1 Constructor __init__ in class Pit

- **Input:**
  - x (list): A list of x-coordinates (row indices) where pits are located.
  - y (list): A list of y-coordinates (column indices) where pits are located.
- **Output:** None
- **Purpose:** Initializes an instance of the Pit class by setting up the pit image, its positions on the map, and the notification grid for nearby cells.
- **How it works:**
  1. Loads the pit image from the specified file path and scales it to 150x300 pixels.
  2. Initializes the is_discovered attribute to None.
  3. Sets the map size to 10x10 cells.
  4. Initializes a 10x10 grid (noti) with all values set to False, representing cells that should notify the player of nearby pits.
  5. Initializes a 10x10 grid (pit_pos) with all values set to False, representing the presence of pits on the map.
  6. Sets the appropriate cells in pit_pos to True based on the provided x and y coordinates, indicating the location of pits.

4.6.2 Function pit_discovered in class Pit

- **Input:** None
- **Output:** None
- **Purpose:** Marks the pit as discovered.
- **How it works:**
  1. Sets self.is_discovered to True, indicating that the pit has been discovered.

4.6.3 Function pit_notification in class Pit

- **Input:** None
- **Output:** None
- **Purpose:** Updates the noti grid to mark cells adjacent to pits as notifying the player of a nearby pit (breeze effect).
- **How it works:**
  1. Iterates over the pit_pos grid, and for each pit, marks the adjacent cells (up, down, left, right) in the noti grid as True.

4.6.4 Function update in class Pit

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display notifications.
  - font (pygame.font.Font): The font to use for rendering the text.
  - is_discovered (list): A 2D list indicating which cells have been discovered by the player.
- **Output:** None
- **Purpose:** Displays a "Breeze" notification on the screen for cells that are adjacent to a pit and have been discovered by the player.
- **How it works:**
  1. Iterates over the noti grid and checks if a cell is marked as notifying (True) and has been discovered.
  2. If both conditions are met, renders the text "Breeze" using the provided font and displays it on the screen at the appropriate coordinates.
  3. Updates the display to show the notifications.

4.7 Class Wumpus

4.7.1 Constructor __init__ in class Wumpus

- **Input:**
  - x (list): A list of x-coordinates (row indices) where Wumpuses are located.
  - y (list): A list of y-coordinates (column indices) where Wumpuses are located.
- **Output:** None
- **Purpose:** Initializes an instance of the Wumpus class by setting up the Wumpus image, its positions on the map, and the notification grid for nearby cells.
- **How it works:**
  1. Loads the Wumpus image (alive) from the specified file path and scales it to 100x200 pixels.
  2. Sets the map size to 10x10 cells.
  3. Sets the position for the notification text to (835, 100).
  4. Initializes is_discovered to None.
  5. Initializes a 10x10 grid (noti) with all values set to False, representing cells that should notify the player of nearby Wumpuses (stench effect).
  6. Initializes a 10x10 grid (wumpus_pos) with all values set to False, representing the presence of Wumpuses on the map.
  7. Sets the appropriate cells in wumpus_pos to True based on the provided x and y coordinates, indicating the location of Wumpuses.

4.7.2 Function wumpus_kill_notif in class Wumpus

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display notifications.
  - font (pygame.font.Font): The font to use for rendering the text.

- **Output:** None
- **Purpose:** Displays a notification on the screen when a Wumpus is killed and changes the Wumpus image to reflect its death.
- **How it works:**
  1. Loads the Wumpus image (dead) from the specified file path.
  2. Renders the text "A wumpus is killed!" using the provided font in black color (0, 0, 0).
  3. Centers the text at the position specified by self.pos and blits it onto the screen.
  4. Blits the dead Wumpus image at (775, 200) on the screen.
  5. Updates the display to show the notification and image change.

4.7.3 Function wumpus_notification in class Wumpus

- **Input:** None
- **Output:** None
- **Purpose:** Updates the noti grid to mark cells adjacent to Wumpuses as notifying the player of a nearby Wumpus (stench effect).
- **How it works:**
  1. Iterates over the wumpus_pos grid, and for each Wumpus, marks the adjacent cells (up, down, left, right) in the noti grid as True.

4.7.4 Function wumpus_killed_notification in class Wumpus

- **Input:**
  - i (int): The row index of the cell where the Wumpus was killed.
  - j (int): The column index of the cell where the Wumpus was killed.
- **Output:** None

- **Purpose:** Removes the Wumpus from the specified cell and updates the noti grid to stop notifying the player of stench in the adjacent cells.
- **How it works:**
  1. Sets self.wumpus_pos[i][j] to False, indicating that the Wumpus in that cell has been killed.
  2. Sets the noti values for the adjacent cells (up, down, left, right) to False, removing the stench notification.

4.7.5 Function update in class Wumpus

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display notifications.
  - font (pygame.font.Font): The font to use for rendering the text.
  - is_discovered (list): A 2D list indicating which cells have been discovered by the player.
- **Output:** None
- **Purpose:** Displays a "Stench" notification on the screen for cells that are adjacent to a Wumpus and have been discovered by the player.
- **How it works:**
  1. Iterates over the noti grid and checks if a cell is marked as notifying (True) and has been discovered.
  2. If both conditions are met, renders the text "Stench" using the provided font and displays it on the screen at the appropriate coordinates.
  3. Updates the display to show the notifications.

4.7.6 Function stench_found in class Wumpus

- **Input:**
  - i (int): The row index of the cell to check for stench.

  - j (int): The column index of the cell to check for stench.

- **Output:**

  - bool: Returns True if the cell at position (i, j) has a stench notification, otherwise False.

- **Purpose:** Checks whether a given cell is marked as having a stench, indicating the presence of a nearby Wumpus.

- **How it works:**

  1. Returns the value of self.noti[i][j] to indicate whether the cell at (i, j) is marked with a stench.

4.8 Class Gold

4.8.1 Constructor __init__ in class Gold

- **Input:** None

- **Output:** None

- **Purpose:** Initializes an instance of the Gold class by setting up the gold image and its display position.

- **How it works:**

  1. Loads the gold image from the specified file path and scales it to 150x300 pixels.
  2. Sets the position for the notification text to (835, 100).

4.8.2 Function grab_gold in class Gold

- **Input:**

  - screen (pygame.Surface): The Pygame surface on which to display notifications.
  - font (pygame.font.Font): The font to use for rendering the text.

- **Output:** None

- **Purpose:** Displays a notification on the screen when the player finds gold, along with the corresponding score increase.
- **How it works:**
  1. Renders the text "You found gold!" using the provided font in black color (0, 0, 0).
  2. Centers the text at the position specified by self.pos and blits it onto the screen.
  3. Blits the gold image at (750, 200) on the screen.
  4. Renders the text "Score + 5000" using the provided font in black color.
  5. Centers the score text at (900, 600) and blits it onto the screen.
  6. Updates the display to show the notification and the gold image.

4.9 Class Gas

4.9.1 Constructor __init__ in class Gas

- **Input:**
  - x (list): A list of x-coordinates (row indices) where poisonous gas is located.
  - y (list): A list of y-coordinates (column indices) where poisonous gas is located.
- **Output:** None
- **Purpose:** Initializes an instance of the Gas class by setting up the gas image, its positions on the map, and the notification grid for nearby cells.
- **How it works:**
  1. Loads the gas image from the specified file path and scales it to 100x200 pixels.
  2. Sets the map size to 10x10 cells.
  3. Sets the position for the notification text to (835, 100).
  4. Initializes is_discovered to None.
  5. Initializes a 10x10 grid (noti) with all values set to False, representing cells that should notify the player of nearby gas (whiff effect).

6. Initializes a 10x10 grid (gas_pos) with all values set to False, representing the presence of gas on the map.
7. Sets the appropriate cells in gas_pos to True based on the provided x and y coordinates, indicating the location of gas.

4.9.2 Function grab_poison in class Gas

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display notifications.
  - font (pygame.font.Font): The font to use for rendering the text.
- **Output:** None
- **Purpose:** Displays a notification on the screen when the player encounters poisonous gas, along with the corresponding health decrease.
- **How it works:**
  1. Renders the text "You sniffed poisonous gas!" using the provided font in black color (0, 0, 0).
  2. Centers the text at the position specified by self.pos and blits it onto the screen.
  3. Blits the gas image at (750, 200) on the screen.
  4. Renders the text "Health - 25" using the provided font in black color.
  5. Centers the health decrease text at (900, 600) and blits it onto the screen.
  6. Updates the display to show the notification and the gas image.

4.9.3 Function gas_discovered in class Gas

- **Input:** None
- **Output:** None
- **Purpose:** Marks the gas as discovered.

- **How it works:**

  1. Sets self.is_discovered to True, indicating that the gas has been discovered.

4.9.4 Function gas_notification in class Gas

- **Input:** None
- **Output:** None
- **Purpose:** Updates the noti grid to mark cells adjacent to gas as notifying the player of nearby gas (whiff effect).
- **How it works:**

  1. Iterates over the gas_pos grid, and for each gas location, marks the adjacent cells (up, down, left, right) in the noti grid as True.

4.9.5 Function update in class Gas

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display notifications.
  - font (pygame.font.Font): The font to use for rendering the text.
  - is_discovered (list): A 2D list indicating which cells have been discovered by the player.
- **Output:** None
- **Purpose:** Displays a "Whiff" notification on the screen for cells that are adjacent to gas and have been discovered by the player.
- **How it works:**

  1. Iterates over the noti grid and checks if a cell is marked as notifying (True) and has been discovered.
  2. If both conditions are met, renders the text "Whiff" using the provided font and displays it on the screen at the appropriate coordinates.
  3. Updates the display to show the notifications.

4.10 Class Potion

4.10.1 Constructor __init__ in class Potion

- **Input:** None
- **Output:** None
- **Purpose:** Initializes an instance of the Potion class by setting up the potion image and its display position.
- **How it works:**
  1. Loads the potion image from the specified file path and scales it to 150x300 pixels.
  2. Sets the position for the notification text to (835, 100).

4.10.2 Function grab_potion in class Potion

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display notifications.
  - font (pygame.font.Font): The font to use for rendering the text.
- **Output:** None
- **Purpose:** Displays a notification on the screen when the player finds a potion.
- **How it works:**
  1. Renders the text "You found potion!" using the provided font in black color (0, 0, 0).
  2. Centers the text at the position specified by self.pos and blits it onto the screen.
  3. Blits the potion image at (750, 200) on the screen.
  4. Updates the display to show the notification and the potion image.

4.10.3 Function use_potion in class Potion

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display notifications.
  - font (pygame.font.Font): The font to use for rendering the text.
- **Output:** None
- **Purpose:** Displays a notification on the screen when the player uses a potion.
- **How it works:**
  1. Renders the text "Potion used" using the provided font in black color (0, 0, 0).
  2. Centers the text at the position specified by self.pos and blits it onto the screen.
  3. Updates the display to show the notification.

4.11 Class Arrow

4.11.1 Constructor __init__ in class Arrow

- **Input:** None
- **Output:** None
- **Purpose:** Initializes an instance of the Arrow class by loading and storing images of the arrow in different directions.
- **How it works:**
  1. Initializes an empty list self.img_list to store the arrow images.
  2. Defines a list direction containing the four possible directions: 'left', 'down', 'right', and 'up'.
  3. Iterates over the direction list:
     - Constructs the file path for each arrow direction image using the direction name.
     - Loads the image from the specified file path and converts it for Pygame compatibility.
     - Appends the loaded image to self.img_list.

4.11.2 Function shoot in class Arrow

- **Input:**
  - direct (int): The direction in which to shoot the arrow (0 = left, 1 = down, 2 = right, 3 = up).
  - screen (pygame.Surface): The Pygame surface on which to display the arrow.
  - y (int): The row index of the arrow's starting position.
  - x (int): The column index of the arrow's starting position.
- **Output:** None
- **Purpose:** Shoots the arrow in the specified direction by calling the appropriate method.
- **How it works:**
  1. Checks the value of direct and calls the corresponding method (shoot_left, shoot_down, shoot_right, or shoot_up) to shoot the arrow in that direction.

4.11.3 Function shoot_left in class Arrow

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display the arrow.
  - x (int): The column index of the arrow's starting position.
  - y (int): The row index of the arrow's starting position.
- **Output:** None
- **Purpose:** Shoots the arrow to the left from the specified position.
- **How it works:**
  1. Calculates the arrow's position on the screen based on the grid coordinates.
  2. Blits the arrow image (facing left) onto the screen at the calculated position.
  3. Updates the display to show the arrow.

4.11.4 Function shoot_down in class Arrow

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display the arrow.
  - x (int): The column index of the arrow's starting position.
  - y (int): The row index of the arrow's starting position.
- **Output:** None
- **Purpose:** Shoots the arrow downward from the specified position.
- **How it works:**
  1. Calculates the arrow's position on the screen based on the grid coordinates.
  2. Blits the arrow image (facing down) onto the screen at the calculated position.
  3. Updates the display to show the arrow.

4.11.5 Function shoot_right in class Arrow

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display the arrow.
  - x (int): The column index of the arrow's starting position.
  - y (int): The row index of the arrow's starting position.
- **Output:** None
- **Purpose:** Shoots the arrow to the right from the specified position.
- **How it works:**
  1. Calculates the arrow's position on the screen based on the grid coordinates.
  2. Blits the arrow image (facing right) onto the screen at the calculated position.
  3. Updates the display to show the arrow.

4.11.6 Function shoot_up in class Arrow

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display the arrow.
  - x (int): The column index of the arrow's starting position.
  - y (int): The row index of the arrow's starting position.
- **Output:** None
- **Purpose:** Shoots the arrow upward from the specified position.
- **How it works:**
  1. Calculates the arrow's position on the screen based on the grid coordinates.
  2. Blits the arrow image (facing up) onto the screen at the calculated position.
  3. Updates the display to show the arrow.

4.12 Class agent

4.12.1 Constructor __init__ in class agent

- **Input:**
  - x (int): The initial row position of the agent on the map.
  - y (int): The initial column position of the agent on the map.
- **Output:** None
- **Purpose:** Initializes an instance of the Agent class, setting up the agent's attributes, including its position, health, score, and image.
- **How it works:**
  1. Calls the __init__ method of the pygame.sprite.Sprite superclass to initialize the sprite.
  2. Initializes the agent's score (self.score) to 0.

3. Sets the maximum health points (self.MAX_HP) to 4.
4. Initializes the count of potions (self.count_potion) to 0.
5. Sets the agent's current health (self.health) to 4.
6. Loads the agent's initial image (facing right) from the specified file path and stores it in self.image.
7. Initializes an empty list self.img_list to store potential future images.
8. Calculates the agent's screen coordinates based on the provided x and y positions:
   - self.y is calculated as 40 + (x - 1) * 70, positioning the agent vertically on the screen.
   - self.x is calculated as 40 + (y - 1) * 70, positioning the agent horizontally on the screen.
9. Retrieves the rectangular area of the agent's image (self.rect) and centers it at the calculated (self.x, self.y) coordinates.
10. Sets the spacing between cells on the grid to 70 pixels (self.spacing).
11. Initializes self.i and self.j to represent the agent's current grid position, calculated as x - 1 and y - 1, respectively.

4.12.2 Function load_image in class agent

- **Input:** None
- **Output:** None
- **Purpose:** Loads and stores the agent's images facing different directions into the img_list attribute.
- **How it works:**

1. Appends the agent's current image (self.image) to the img_list attribute.
2. Defines a list temp containing the directional names 'left', 'down', and 'right'.
3. Iterates over the temp list:
   - Constructs the file path for each directional image using the direction name.

  - Loads the image from the specified file path and converts it for Pygame compatibility.
  - Appends the loaded image to self.img_list.

4.12.3 Function appear in class agent

- **Input:**
  - screen (pygame.Surface): The Pygame surface on which to display the agent.
- **Output:** None
- **Purpose:** Renders the agent's image onto the screen at its current position.
- **How it works:**
  1. Adjusts the agent's position by subtracting 30 pixels from both the x and y coordinates to properly center the image.
  2. Blits (draws) the agent's image onto the provided screen at the adjusted position (self.x - 30, self.y - 30).

4.12.4 Group of get_ functions in class agent

- **Consist of these function:** get_score, get_health, get_num_of_potion, get_position.
- **Input:** None
- **Output:**
  - int/tuple: The current selected attribute of the agent.
- **Purpose:** Returns the agent's current selected attribute.
- **How it works:**
  1. Returns the value of the selected attribute of the agent.

4.12.5 Function move in class agent

- **Input:**
  - direct (int): The direction in which to move the agent (0 = left, 1 = down, 2 = right, 3 = up).
- **Output:** None
- **Purpose:** Moves the agent in the specified direction by calling the appropriate movement method.
- **How it works:**
  1. Checks the value of direct and calls the corresponding method:
     - If direct is 0, calls self.move_left() to move the agent left.
     - If direct is 1, calls self.move_down() to move the agent down.
     - If direct is 2, calls self.move_right() to move the agent right.
     - If direct is 3, calls self.move_up() to move the agent up.

4.12.6 Group of move_ functions in class agent

- **Consist of these function:** move_up, move_down, move_left, move_right.
- **Input:** None
- **Output:** None
- **Purpose:** Moves the agent in the selected direction on the grid, adjusting its position and score accordingly.
- **How it works:**
  1. Adjust the agent's position (self.x/self.y) by the value of self.spacing (70 pixels) to move it on the screen.
  2. Decreases the agent's score (self.score) by 10 points to account for the movement.
  3. Checks if the agent's current row/column index (self.i) is greater than 0 and less than 9:
     - If true, decreases/increase self.i by 1 to update the agent's grid position.

4.12.7 Group of turn_ functions in class agent

- **Consist of these function:** turn_left, turn_right.
- **Input:** None
- **Output:**
  - int: Returns 0/1 to indicate the direction the agent is now facing (left/right).
- **Purpose:** Rotates the agent to face left/right, changes its image accordingly, and decreases the score.
- **How it works:**
  1. Decreases the agent's score (self.score) by 10 points as a consequence of turning.
  2. Updates the agent's image to the left/right-facing version by setting self.image to self.img_list[1]/[2].
  3. Returns 0/1 to indicate that the agent is now facing left/right.

4.12.8 Function update in class agent

- **Input:** None
- **Output:** None
- **Purpose:** Updates the agent's position and score if the agent is near the boundaries of the grid, and re-centers the agent's image accordingly.
- **How it works:**
  1. Checks if the agent's horizontal position (self.x) is greater than 670 (indicating it is near the right boundary):
     - If true, decreases self.x by the value of self.spacing (70 pixels) to move the agent left.
     - Increases the agent's score (self.score) by 10 points as a consequence.
  2. Checks if the agent's horizontal position (self.x) is less than 40 (indicating it is near the left boundary):
     - If true, increases self.x by the value of self.spacing to move the agent right.

   - Increases the agent's score by 10 points as a consequence.
3. Checks if the agent's vertical position (self.y) is less than 40 (indicating it is near the top boundary):
   - If true, increases self.y by the value of self.spacing to move the agent down.
   - Increases the agent's score by 10 points as a consequence.
4. Checks if the agent's vertical position (self.y) is greater than 670 (indicating it is near the bottom boundary):
   - If true, decreases self.y by the value of self.spacing to move the agent up.
   - Increases the agent's score by 10 points as a consequence.
5. Updates the agent's rect.center to the current (self.x, self.y) coordinates, re-centering the agent's image on the screen.

4.12.9 Group of point-updating functions in class agent

- **Consist of these function:** shoot, wumpus_or_pit_or_poison, grab_gold, climb.
- **Input:** None
- **Output:** None
- **Purpose:** Handles the reduction and the increment of the agent's score.
- **How it works:**
  1. Decreases/increase the agent's score (self.score) each time said events occur.

4.12.10 Function grab_poison in class agent

- **Input:** None
- **Output:** None
- **Purpose:** Handles the effect of the agent encountering poisonous gas, reducing the agent's health.

- **How it works:**

    1. Decreases the agent's health (self.health) by 1 point when the agent encounters poison.

4.12.11 Function grab_potion in class agent

- **Input:** None
- **Output:** None
- **Purpose:** Handles the action of the agent collecting a potion, increasing the count of potions held.
- **How it works:**

    1. Increases the count of potions (self.count_potion) by 1 each time the agent collects a potion.

4.12.12 Function use_potion in class agent

- **Input:** None
- **Output:** None
- **Purpose:** Uses a potion to restore the agent's health if a potion is available.
- **How it works:**

    1. Checks if the agent has any potions available (self.count_potion > 0).
    2. If a potion is available:
        - Decreases the potion count (self.count_potion) by 1.
        - Increases the agent's health (self.health) by 1, ensuring that it does not exceed the maximum health (self.MAX_HP).

Chapter 5

Program Execution

After executing the command to run the program, we get the following result:

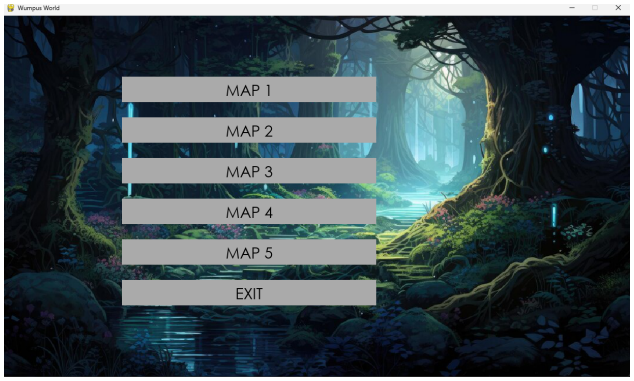

Figure 5.1: Program Menu.

After selecting the map, the corresponding results for each case will be obtained as follows:

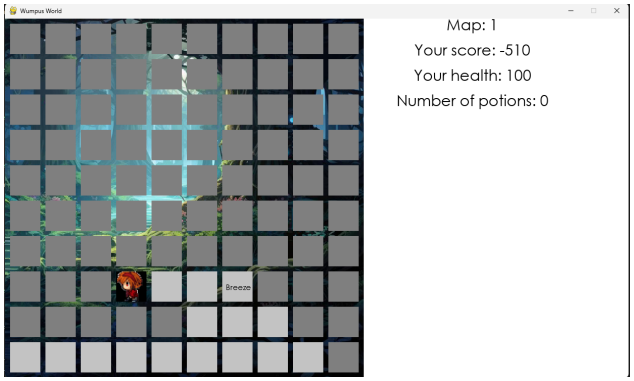

Figure 5.2: Map 1.

Figure 5.3: Map 2.

Figure 5.4: Map 3.

Figure 5.5: Meap 4.

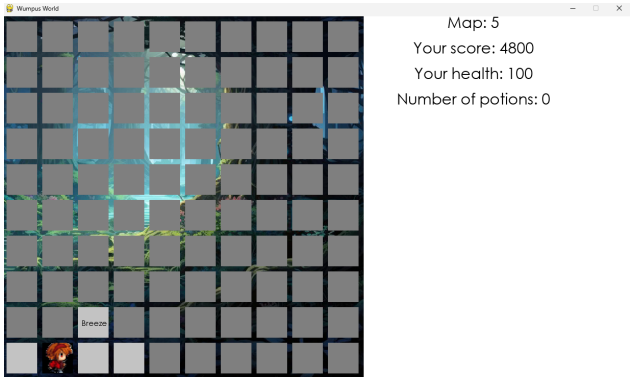

Figure 5.6: Map 5.

After completing the program run, we get the following:

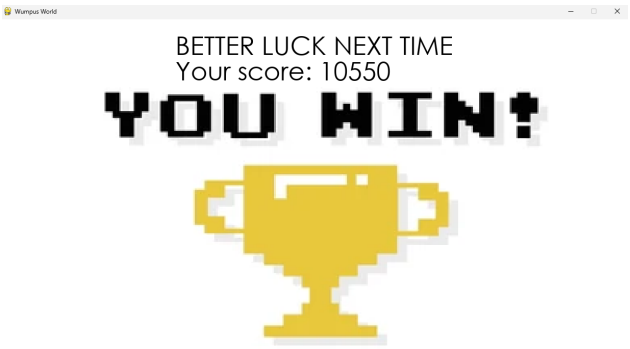

Figure 5.7: Finish.

After that, it will return to the menu.

Chapter 6

Comments and Conclusion

6.1 Comments

There is a special case that: if the initial room where the Agent starts is adjacent to the Wumpus, the Agent will still climb out of the cave. If there is also a Stench in that room, the Agent will still not be able to infer anything and will then try to shoot at any adjacent rooms until the current room no longer has a Stench. The last adjacent room the Agent shoots at is the safe room, and he can move to that room.

6.2 Conclusion:

After completing this project, we can learn the following:

1. **Understanding of Logical Search Agents:** We will gain experience in designing and implementing a logical search agent in a partially observable environment, which is crucial in artificial intelligence.

2. **Application of Propositional or First-Order Logic:** We will understand how to use Propositional Logic or First-Order Logic (or both) to make decisions in uncertain environments.

3. **Problem-Solving in Complex Environments:** We will learn to navigate a complex environment like the Wumpus World, which includes handling unpredictable elements such as pits and the Wumpus itself.

4. **Algorithm Development and Optimization:** Through experimentation and reflection, we will improve our ability to develop algorithms that maximize performance based on specific metrics like score in the Wumpus World.

5. **Team Collaboration:** Since this is a group assignment, we will also develop teamwork and collaboration skills, essential for working in professional environments.

6. **Practical Programming Skills:** We will improve our Python programming skills, particularly in the context of AI, and possibly gain experience with graphical demonstrations.

7. **Project Management:** We will learn to plan, manage time, and deliver a project within a set deadline while meeting all specified requirements.

Chapter 7

References

[1] Antonio Morgado Alexey Ignatiev Joao Marques-Silva. *PySAT-solvers [English]*. URL: https://pysathq.github.io/docs/html/api/solvers.html.

[2] Curtis Franks. *Propositional Logic [English]*. URL: https://plato.stanford.edu/entries/logic-propositional/.

[3] Trần Đình Sang Kiều Công Hậu. *The sample Wumpus World project*. URL: https://github.com/kieuconghau/ai-wumpus-world/tree/master.

[4] Dan Lawrence. *Pygame GUI Quick start Guide [English]*. URL: https://pygame-gui.readthedocs.io/en/latest/quick_start.html.

[5] Wikipedia. *Wumpus World [English]*. URL: https://en.wikipedia.org/wiki/Wumpus_world.