UNIVERSITY OF SCIENCE – VNUHCM

Faculty of Information Technology

Class 22CLC07

----------oOo----------

PROJECT 2:
LOGICAL AGENT

LECTURERS:

Nguyễn Tiến Huy

Nguyễn Trần Duy Minh

Bùi Duy Đăng

Table of Contents

1. Member Introduction

MSSV	Name
22127103	Lê Thị Hồng Hạnh
22127278	Vũ Thu Minh
22127479	Lê Hoàng Lĩnh

2. Words of Appreciation

We would like to thank the lectures for their dedicated guidance, because this is the first time to do a project like this, so there may be mistakes, we hope that the lecturers can consider ignoring and commenting. Respect.

3. Work division and progress percentage

Tasks	MSSV	Progress percentage
Code		
UI	22127479	100%
Algorithm	22127103	100%
Generate map	22127278	100%
Report		
User interface	22127479	100%
Algorithm	22127103	100%
Demo, alignment report	22127278	100%

4. Requirements and level of completion

Requirements	Progress percentage
Finish problem successfully.	100%
GUI	100%
Generate at least 5 maps with difference structures such as position and number of Pit, Gold and Wumpus.	100%
Report algorithm, experiment with some reflection or comments	100%

5. Environment to compile and run code

Enviroment: python.

To complie and run code, read file “requirements.txt” that include libraries needed.

6. Approach method

Rules

Wumpus World's rules

- $W_{i,j}$: true if there is a Wumpus in room [i, j]
- $S_{i,j}$: true if there is a Stench in room [i, j]
- $P_{i,j}$: true if there is a Pit in room [i, j]
- $B_{i,j}$: true if there is a Breeze in room [i, j]
- $P\_G_{i,j}$: true if there is a Poisonus Gas in room [i, j]
- $H\_P_{i,j}$: true if there is a Healing Potions in room [i, j]
- $G_{i,j}$: true if there is a Gold in room [i, j]
- $W\_H_{i,j}$: true if there is a Whiff in room [i, j]
- G_L_{ij} : true if there is a Glow in room [i, j]

$$W_{i,j} \Leftrightarrow S_{i+1,j} \wedge S_{i-1,j} \wedge S_{i,j+1} \wedge S_{i,j-1}$$
$$S_{i,j} \Leftrightarrow W_{i+1,j} \vee W_{i-1,j} \vee W_{i,j+1} \vee W_{i,j-1}$$
$$P_{i,j} \Leftrightarrow B_{i+1,j} \wedge B_{i-1,j} \wedge B_{i,j+1} \wedge B_{i,j-1}$$
$$B_{i,j} \Leftrightarrow P_{i+1,j} \vee P_{i-1,j} \vee P_{i,j+1} \vee P_{i,j-1}$$
$$P\_G_{i,j} \Leftrightarrow W\_H_{i+1,j} \wedge W\_H_{i-1,j} \wedge W\_H_{i,j+1} \wedge W\_H_{i,j-1}$$
$$H_{P\,i,j} \Leftrightarrow G_{L_{i+1,j}} \wedge G_{L_{i-1,j}} \wedge G_{L_{i,j+1}} \wedge G_{L_{i,j-1}}$$

Our rules

- Safe cell is the cell that there is certainly no pit and wumpus in it
- When we are sure that there is a wumpus in adjacent cell, we shoot it immediately
- The agent only enters safe cells, if there are no safe cells that can be inferred, it will go back to start position and end exploring world
- Logic
- This project uses propositional logic to solve Wumpus World problem
- Inference rule approach: satisfiability
    - ***$\alpha \vDash \beta$ iff $\alpha \wedge \neg\beta$ is unsatisfiable***

A sentence is **satisfiable** if it is true in **some** models

A sentence is **unsatisfiable** if it is true in **no** models

7. User interface

UI functions will be visualized in class and written in UI.py

Libraries:

Pygame: used to build and manage the game's graphical interface, including drawing images and handling user events.
Tkinter: Used to create a file dialog for user interaction, allowing the selection of the input file from the file system.
Sys: Used to manage system operations, particularly to exit the program safely.

Game initialization:

The WumpusWorldApp class initializes the game by setting up the game window and configuring basic variables such as screen size, necessary images, and game state. It also loads the game grid from an input file if available.

Grid and size calculations:

Function calculates the size of the grid and cells within the grid based on the screen size and data from the input file. Space is allocated for the information panel.

Loading and display images:

Loading images and resize image to fit the cells in the game, these images include: wumpus, pit, agent, gold, poisonous gas, healing potion and various effect.

Update element and grid:

The load_grid function reads grid data from a file and converts the grid components into corresponding objects. The add_percepts function updates the percepts (such as smells and winds) for cells based on the components in the grid.

Drawing the user interface:

The update_display function and other drawing methods such as draw_grid, draw_elements, draw_agent, and draw_health_and_points render the game components on the screen. These methods help update the user interface with the grid, cell contents, agent's position, and information about health and points.

Event and Action Handling:

The perform_actions function executes actions from the output file and updates the game grid and display. The handle_events and run functions process user events such as button presses or agent movements, and update and redraw the screen in real-time.

Read output file function:

Output file nclude these elements: position, action, direction, health, point, remaining step
Example: (1, 1): TURN_LEFT: E: 75.0: 34260: 1
We split the ": " betweent eeach element to get those elements and return with tuple.

Processing Files:

Functions like read_input_file and read_output_file read data from files to provide information for the game and perform actions based on that data. The open_file_dialog function provides a user interface for selecting input file

8. Algorithm

Libraries:

- numpy: used to handle matrix operations
- sympy: used to present knowledge-based in propositional logic of agent

Components:

- Program: This class is responsible for building the map, reporting information about the elements in the cell and updating the map's status.
- Agent: The agent can only know the components of the cell where it is standing. The agent must go step by step and call the program to get information about the cell it is standing on from class program. From there, it makes percepts to find the direction of movement and infer objects based on the available information.
- Interface: This class is the communication between Program and Agent. Agent does not know the information on the whole map, it just can take the information of the current cell through Interface and the actions of agent change the components, Interface will handle these changes in map. There is a difference between representing the position of agent in map and the index of element in matrix, so this class is also responsible for converting between the position and the index.
- Cell: In some cases of map traversal, there is no further path so we must turn back. This class stores information of a cell, including its position, its parent's position (for find path to turn back), and its heuristic cost (for find path to return start).

Algorithms:

The main algorithm used to traversal the map is DFS (backtracking).

- Step 1: Get component in the current cell, update agent state (health, hp, point) and add knowledge to KB
- Step 2: Check if the agent is still alive. If agent is dead, end the explore world loop
- Step 3: Add the position of this cell to visited list
- Step 4: Get all neighbors of this cell that is not in visited list. The priority of neighbors list is based on the agent's direction. The neighbor in front of the agent has the highest priority and the neighbors behind it has the lowest priority.
- Step 5: From neighbors list, we will find all safe neighbors. The definition of safe cell is mentioned above. The safe cells are reversed (because DFS uses stack) and pushed to queue
- Step 6: A new cell is popped from stack. Check if this cell is the adjacent cell of the current cell. If two cells are not adjacent, we must go back from the current cell to the adjacent cell of the new cell by the way that just have all visited cell. Then from the current cell move to that new cell.
- Step 7: Repeat step 1 to 6 untill the agent dies or there is no more safe cell to explore
- Step 8: Check if the agent is still alive. If the agent is dead, the program ends. If the agent is still alive, it will find the shortest path (based on GBFS) from the current cell to the start. When go back to start, if the agent can be dead because of Poisonus Gas. Finally, notice the point and the live of the agent

Interface and Agent initialization:

The Interface is initialized to communicate between Agent and Program, handle file problem (read input and write output). The instance of Interface is an attribute of Agent when the agent is initialized.

Agent inference:

The library used to store and solve the propositional logic is sympy. To check if the satisfiability of a sentence, we use the Inference rule approach: $\alpha \models \beta$ ***iff*** $\alpha \wedge \neg\beta$ ***is unsatisfiable.*** The library sympy provides connectives (And, Or, Not, Implies, Equivalent) and satisfiable function to proving a sentence.

Update the state and KB of Agent:

When the agent enters a new cell, it has new percepts and new elements. If the current cell has Wumpus or Pit, the agent dies. If it has Stench, Breeze, Whiff or Glow, new KB is added to the agent's KB. If the current cell has Gold or Healing Potions, the agent grabs and heals (if necessary). If the current cell has Poisonus Gas, the agent's health decreases.

Check Wumpus or Pit:

Based on KB we can infer that a cell has Wumpus/Pit or not. The result of inference is True (Wumpus/Pit in cell), False (no Wumpus/Pit in cell) or None (not enough KB to infer)

Shoot Wumpus/ Grab Gold/ Grab Healing Potions:

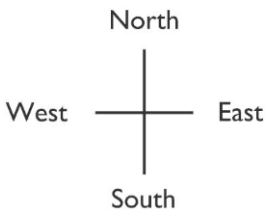
When the agent is sure that there is a wumpus in the adjacent cell, it will shoot it immediately. When the agent sees Gold or Healing Potions, it grabs immediately. The point of agent and the state of map will be updated.

Check two adjacent cells:

Based on the position of two adjacent cells, we can check if these cells are adjacent or not.

Turn action:

The position of the agent is (x, y) with $1 \leq x, y \leq size$, x is the horizontal axis and y is the vertical axis. From the position of the current cell and the next cell, we find the difference between two positions and the direction (N: North, S: South, E: East, W: West) of the next cell from the current cell. Therefore, we can turn the agent face to the next cell.



Move to the adjacent cell:

After doing turn action, the agent just goes forward to move to the next cell.

Find the shortest path only through the visited cell:

The algorithm used to find the shortest path is GBFS because the path cost between two cells is the same. Finding the shortest path to backtrack when there is no safer cell.

Move back:

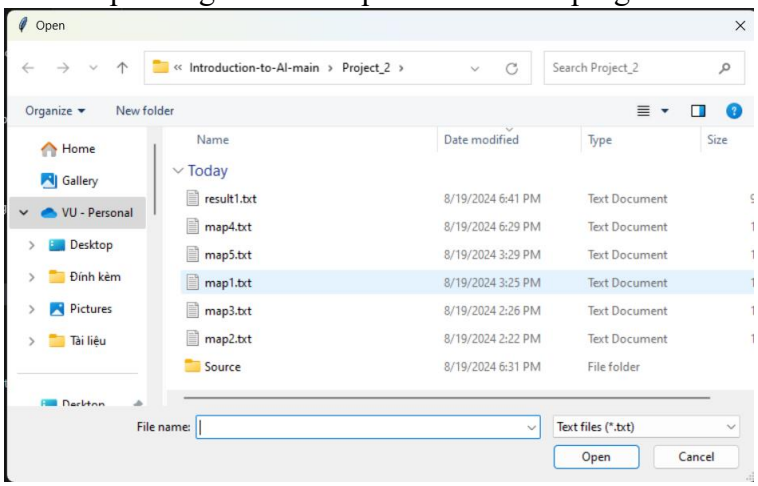
The find path algorithm returns a list of cells, we must go sequentially from cell to cell to record the action. Therefore, this function calls the move adjacent function many times until reach the destination cell.

Explore world:

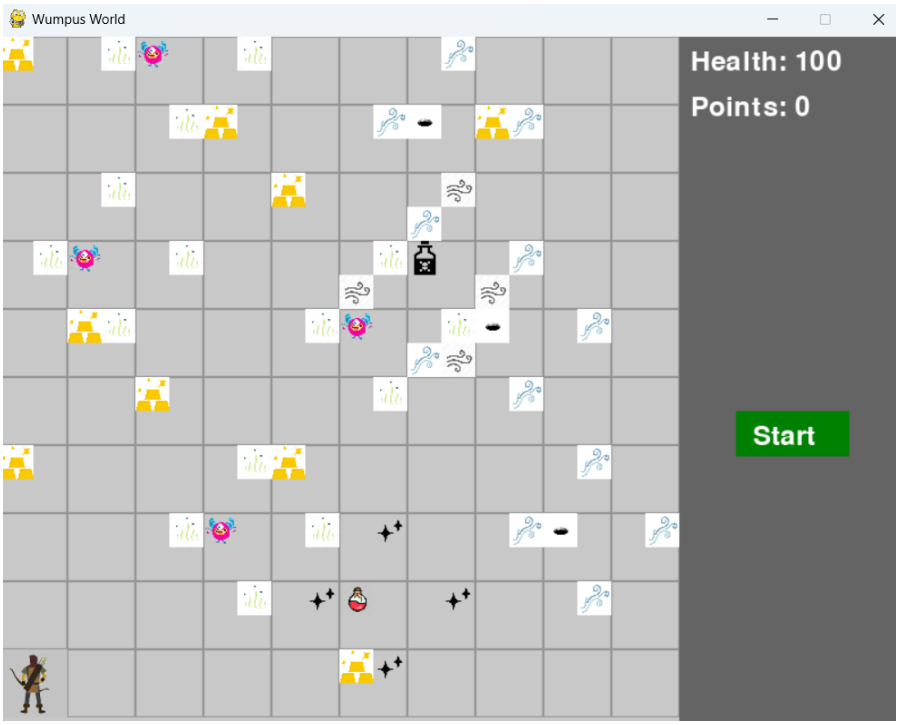
The main process of the agent is to explore the world. This function runs based on the algorithm mentioned above (DFS), and calls some helper functions mentioned above.

9. Demo

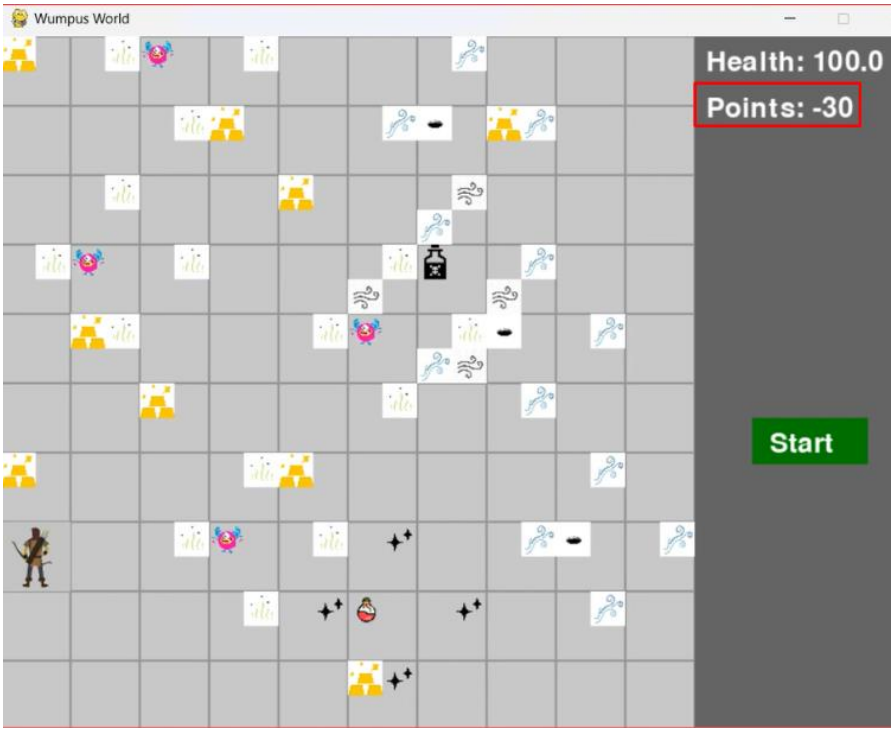
- Go to main.py file and click run. A window will appear to select the input file. Select the input file corresponding to each map to execute the program.



- **Map 1:**

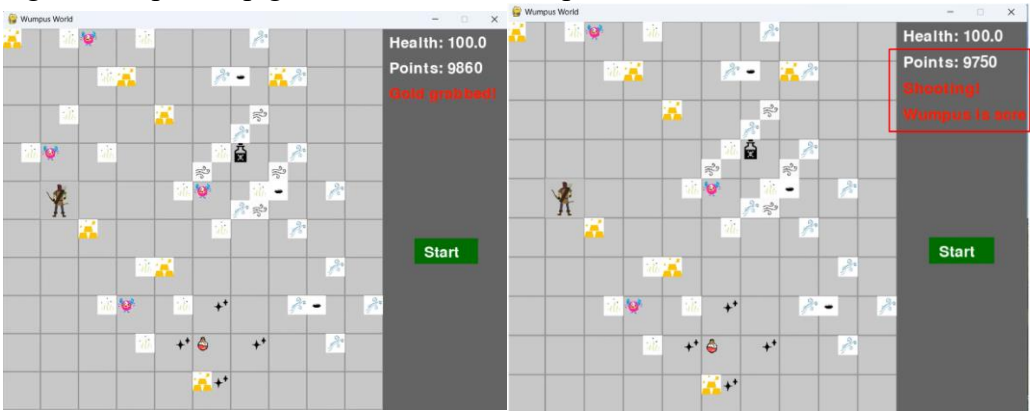


Click Start button to start.

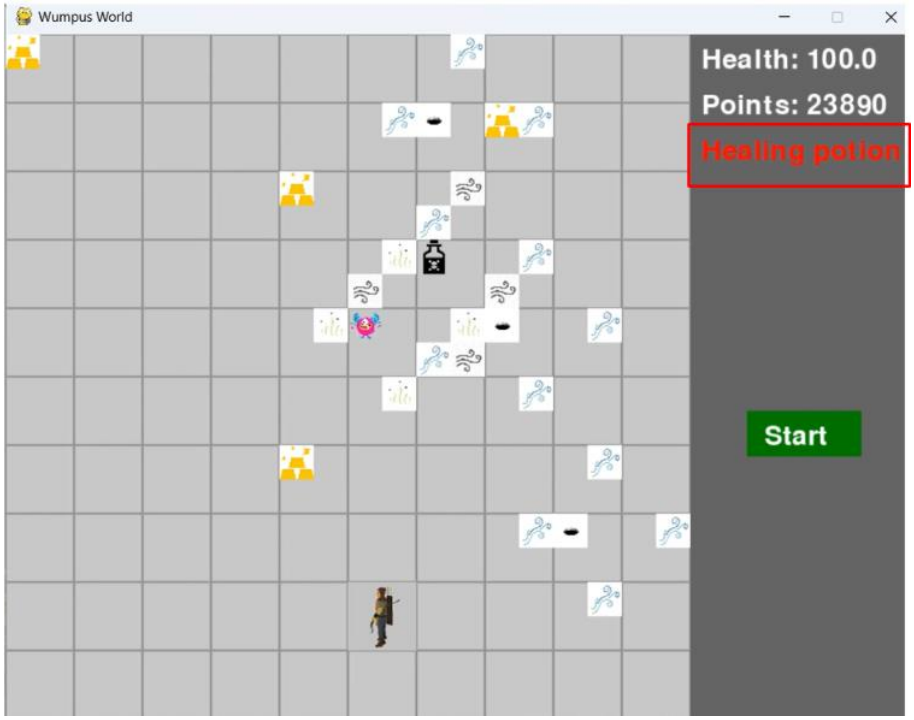


For each action such as turning left, turning right, moving forward, the agent loses 10 points.

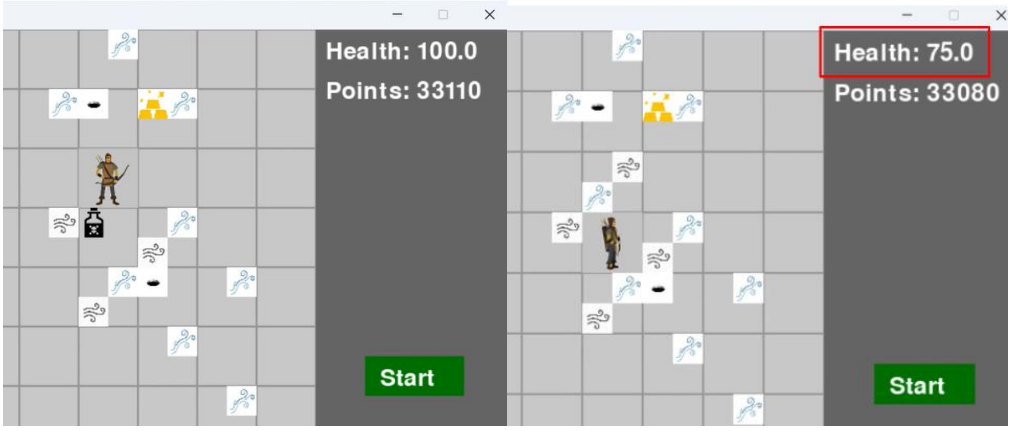
Agent who picks up gold will receive 5000 points.



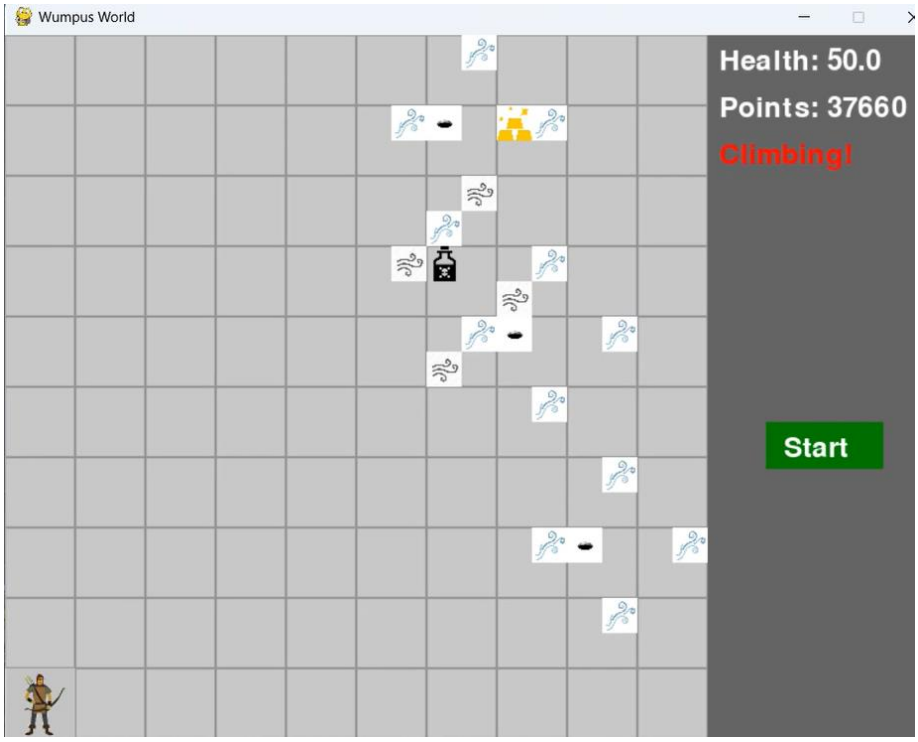
When shooting an arrow, the agent loses 100 points. When the Agent dies, the screen displays the message "Wumpus is screaming!" and the map is updated.



When picking up H_P the screen will display the message "Healing potion grabbed!".

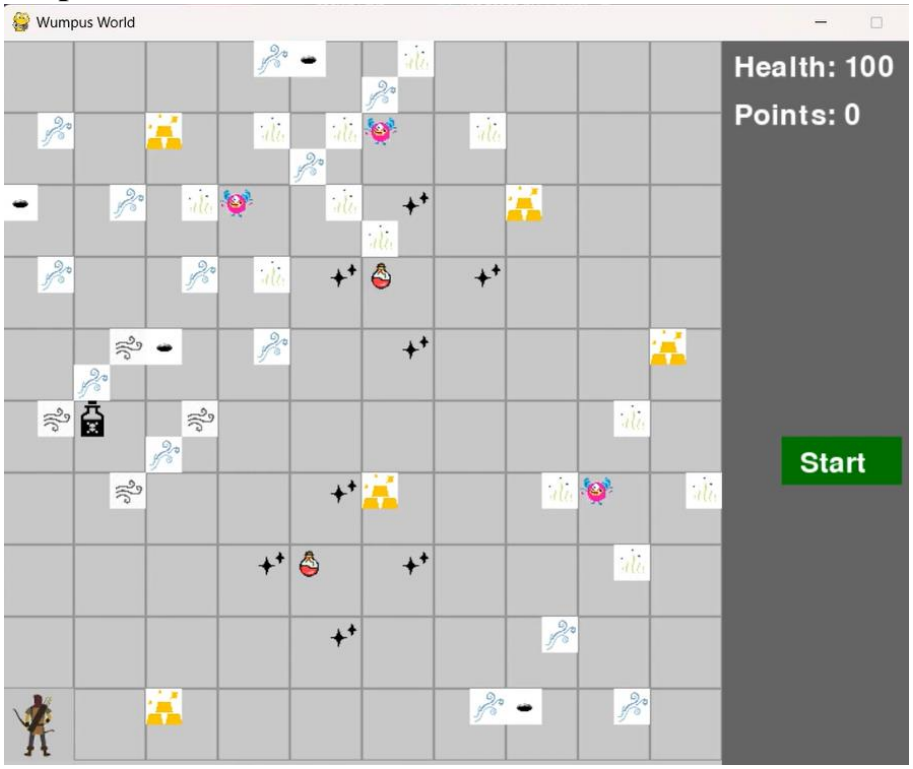


When the agent enters the P_G cell, the agent will lose 25% health.

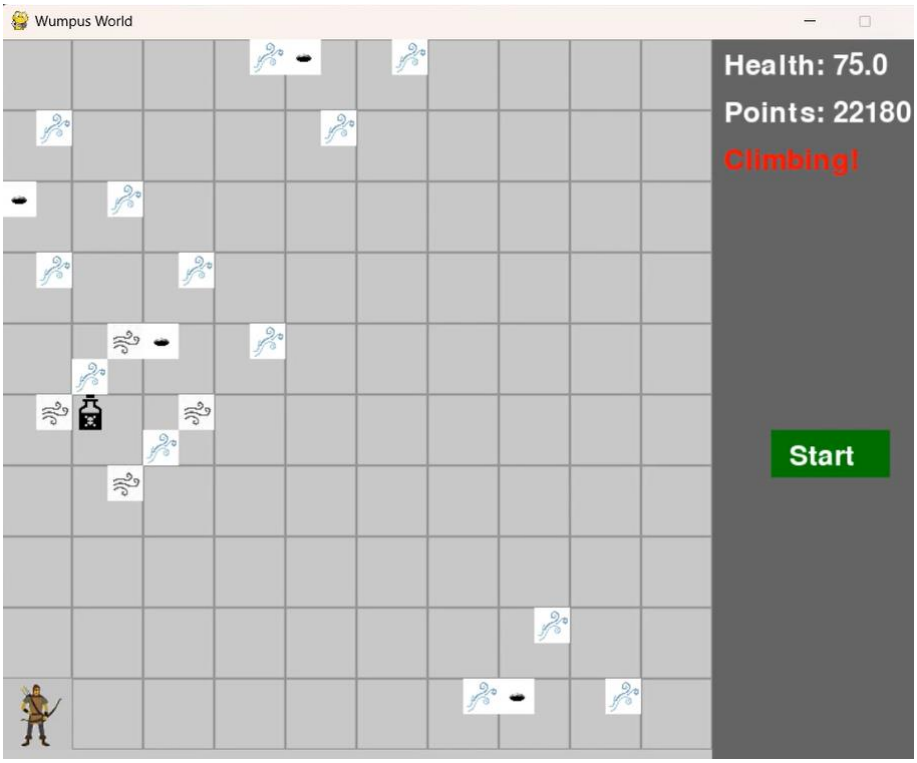


Agent returns to starting point and climbs out of cave.

- **Map 2:**

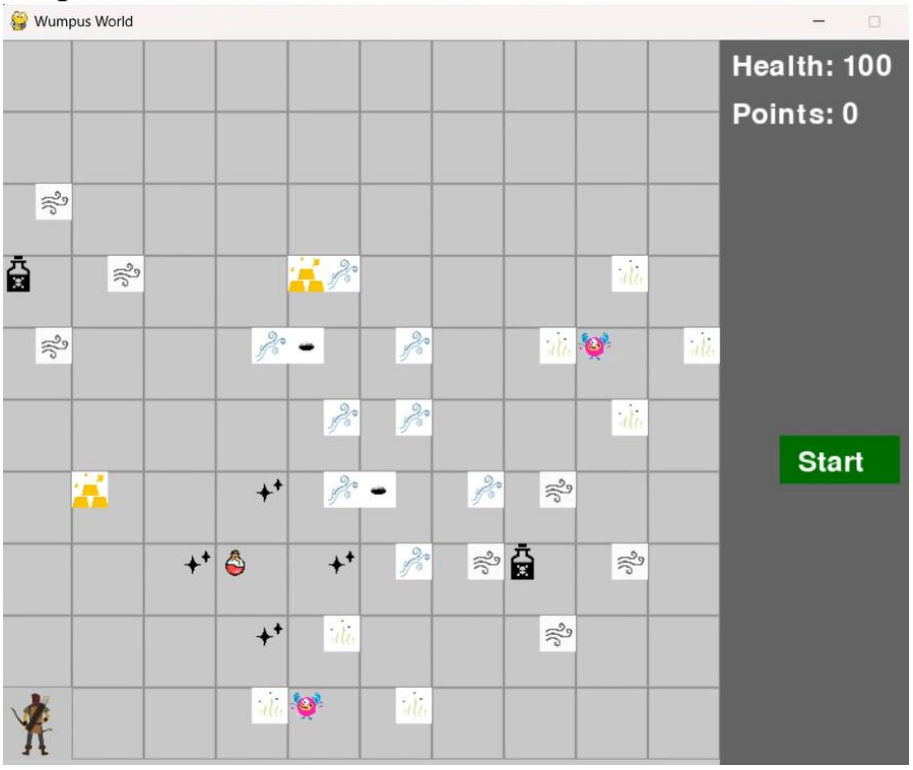


Result:

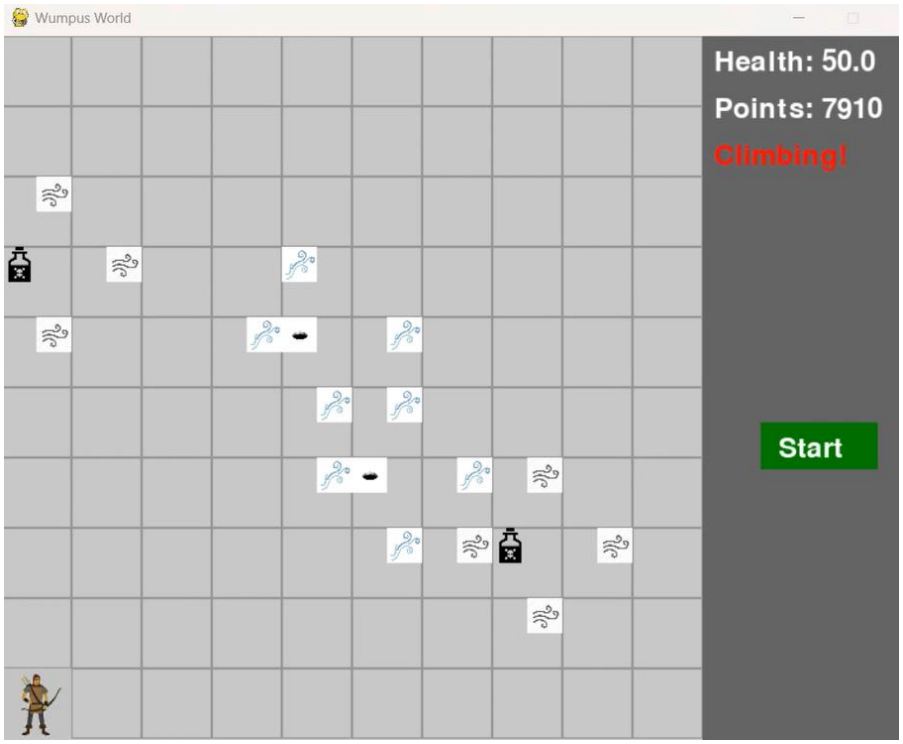


Agent returns to starting point and climbs out of cave.

- **Map 3:**

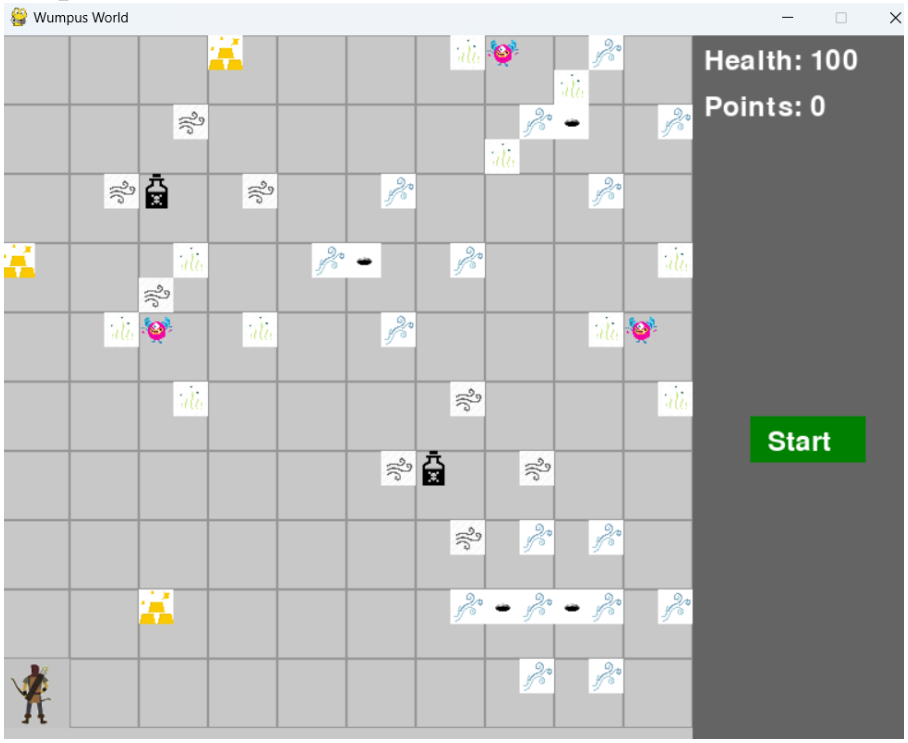


Result:

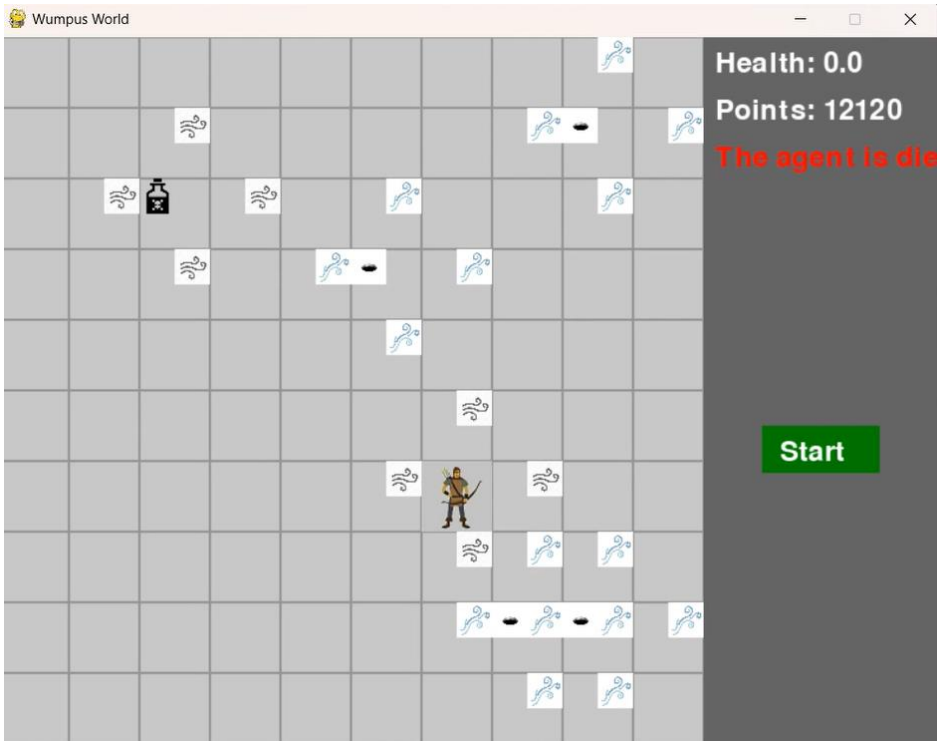


Agent returns to starting point and climbs out of cave.

- **Map 4:**

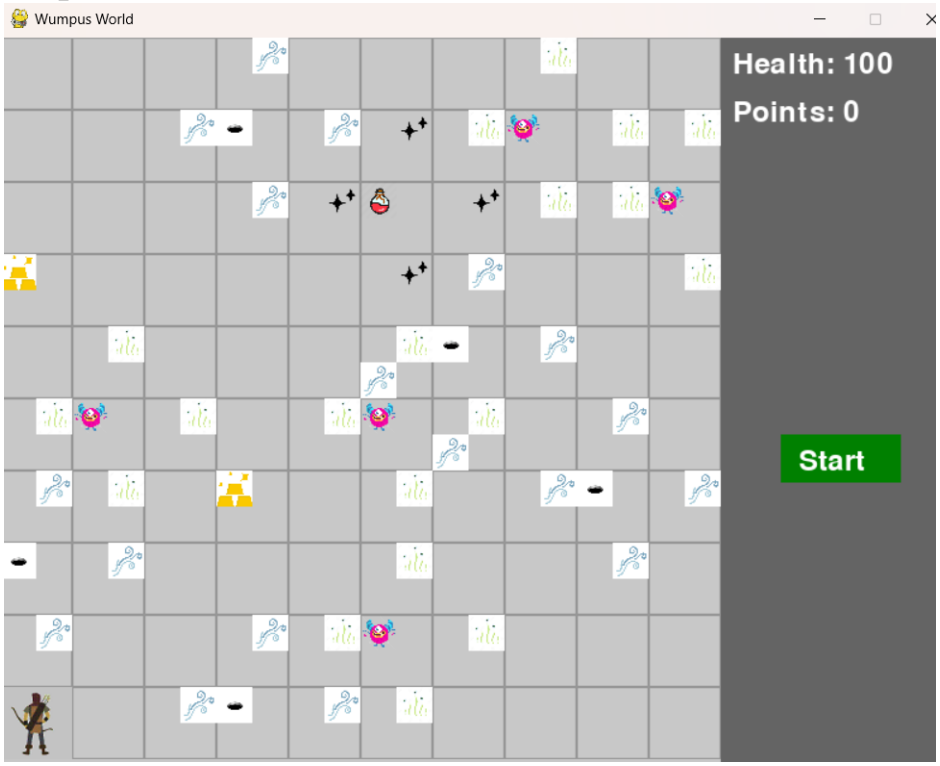


Result:

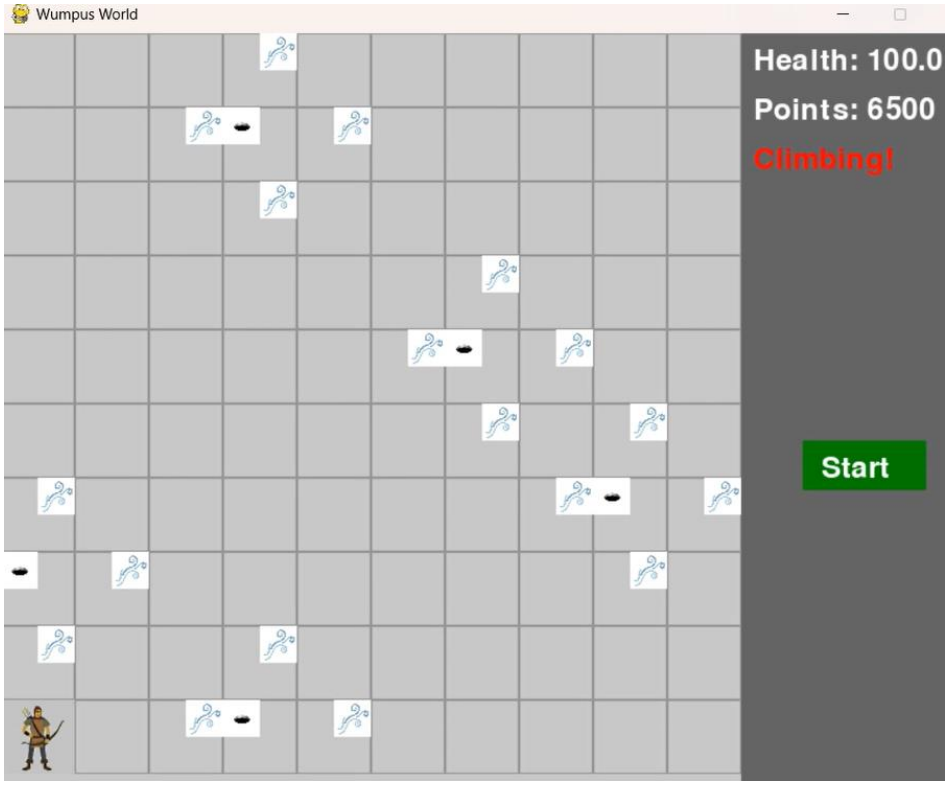


The program ends when the agent's health is 0 and prints the message "The agent is dead" to the screen.

- **Map 5:**



Result:



Agent returns to starting point and climbs out of cave.

- **Output file content**
  The structure of output:
  position: action: direction: health: point: hp

result1.txt

```
1   (1, 1): MOVE_FORWARD: N: 100: -10: 0
2   (1, 2): TURN_RIGHT: E: 100: -20: 0
3   (1, 2): TURN_RIGHT: S: 100: -30: 0
4   (1, 2): MOVE_FORWARD: S: 100: -40: 0
5   (1, 1): TURN_LEFT: E: 100: -50: 0
6   (1, 1): MOVE_FORWARD: E: 100: -60: 0
7   (2, 1): MOVE_FORWARD: E: 100: -70: 0
8   (3, 1): TURN_RIGHT: S: 100: -80: 0
9   (3, 1): TURN_RIGHT: W: 100: -90: 0
10  (3, 1): MOVE_FORWARD: W: 100: -100: 0
11  (2, 1): MOVE_FORWARD: W: 100: -110: 0
12  (1, 1): TURN_RIGHT: N: 100: -120: 0
```

- **Conclusion:**

The agent can grab most of the Gold in the safe cells that can be inferred, so the point is quite high. Because it just enters the safe cells, it will not die due to Wumpus or Pit.

The DFS (backtracking) algorithm is safe but not optimal, the agent must go around so many times and lose points. This is a reasonable trade-off to avoid the agent being kill by Wumpus and Pit.

10.References

https://www.steamforvietnam.org/blog/lap-trinh-pygame-t-rex-jump-bang-python

https://codelearn.io/sharing/lap-trinh-game-co-ban-voi-pygame-p2

https://devops.vinahost.vn/Programming-Langguage/Python/Python-Pygame01/#tao-cua-so-game

https://t3h.edu.vn/tin-tuc/python-tkinter-lap-trinh-gui-bang-tkinter-trong-python