VIETNAM NATIONAL UNIVERSITY

UNIVERSITY OF SCIENCE

TOPIC

Project 02
Wumpus logic world

MEMBERS

Lư Ngọc Liên	18127046
Vũ Công Minh	18127155

Course: Artificial Intelligence

Ho Chi Minh City– 2020

Contents

Assignment Plan

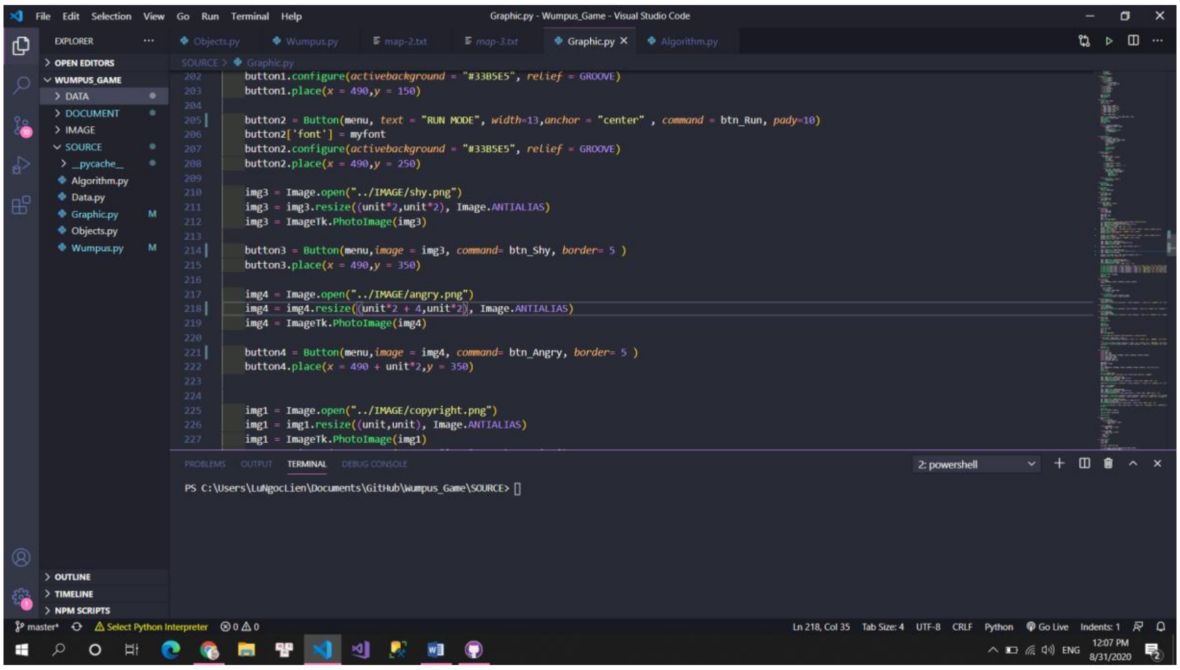
Student ID	Student Name	Tasks	% Complete
18127046	Lư Ngọc Liên	- Handle Input -Display text, buttons, images -Shy Agent - Collect gold - A* search - 5 static mazes	100%
18127155	Vũ Công Minh	- Create class objects -Aggressive Agent - Shoot wumpus -Handle clauses - Display objects - Random maze	100%

Self rating:

Project: 100% overall.

Environment to compile and run the program

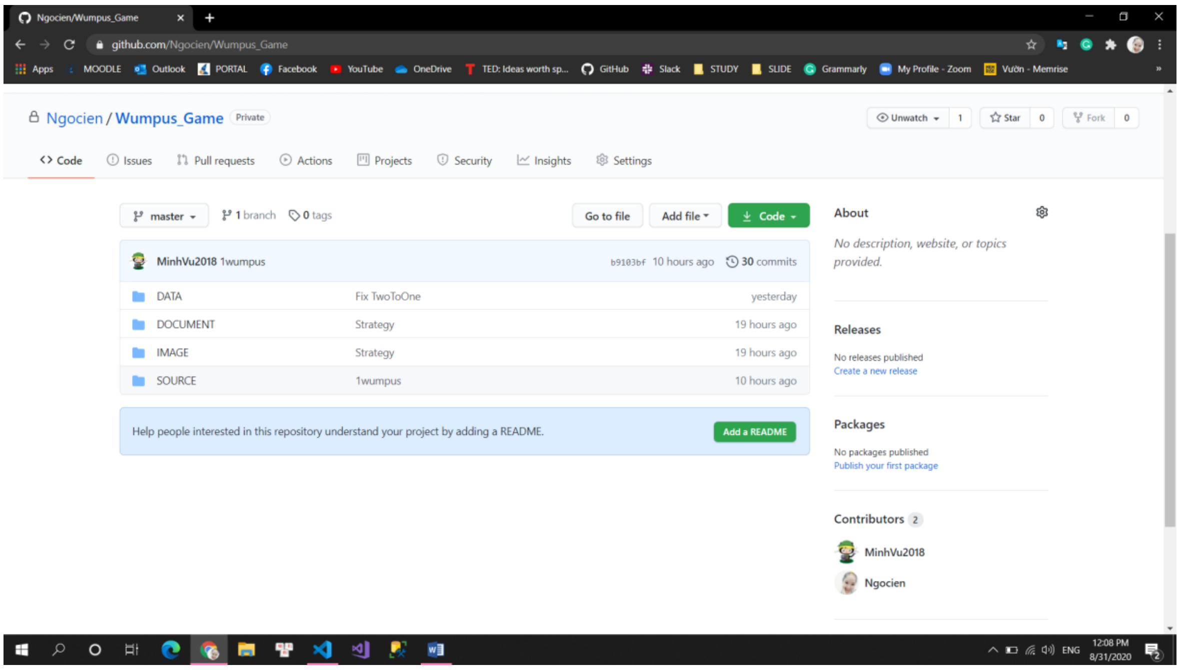
Environment to compile and run the program: Visual Code



```
        button1.configure(activebackground = "#33B5E5", relief = GROOVE)
        button1.place(x = 490,y = 150)

        button2 = Button(menu, text = "RUN MODE", width=13,anchor = "center" , command = btn_Run, pady=10)
        button2['font'] = myfont
        button2.configure(activebackground = "#33B5E5", relief = GROOVE)
        button2.place(x = 490,y = 250)

        img3 = Image.open("../IMAGE/shy.png")
        img3 = img3.resize((unit*2,unit*2), Image.ANTIALIAS)
        img3 = ImageTk.PhotoImage(img3)

        button3 = Button(menu,image = img3, command= btn_Shy, border= 5 )
        button3.place(x = 490,y = 350)

        img4 = Image.open("../IMAGE/angry.png")
        img4 = img4.resize((unit*2 + 4,unit*2), Image.ANTIALIAS)
        img4 = ImageTk.PhotoImage(img4)

        button4 = Button(menu,image = img4, command= btn_Angry, border= 5 )
        button4.place(x = 490 + unit*2,y = 350)


        img1 = Image.open("../IMAGE/copyright.png")
        img1 = img1.resize((unit,unit), Image.ANTIALIAS)
        img1 = ImageTk.PhotoImage(img1)
```

```
PS C:\Users\LuNgocLien\Documents\GitHub\Wumpus_Game\SOURCE>
```

Version control: GitHub



Ngocien / Wumpus_Game Private

MinhVu2018 1wumpus		b9103bf 10 hours ago 30 commits
DATA	Fix TwoToOne	yesterday
DOCUMENT	Strategy	19 hours ago
IMAGE	Strategy	19 hours ago
SOURCE	1wumpus	10 hours ago

Help people interested in this repository understand your project by adding a README.

About

No description, website, or topics provided.

Releases

No releases published
Create a new release

Packages

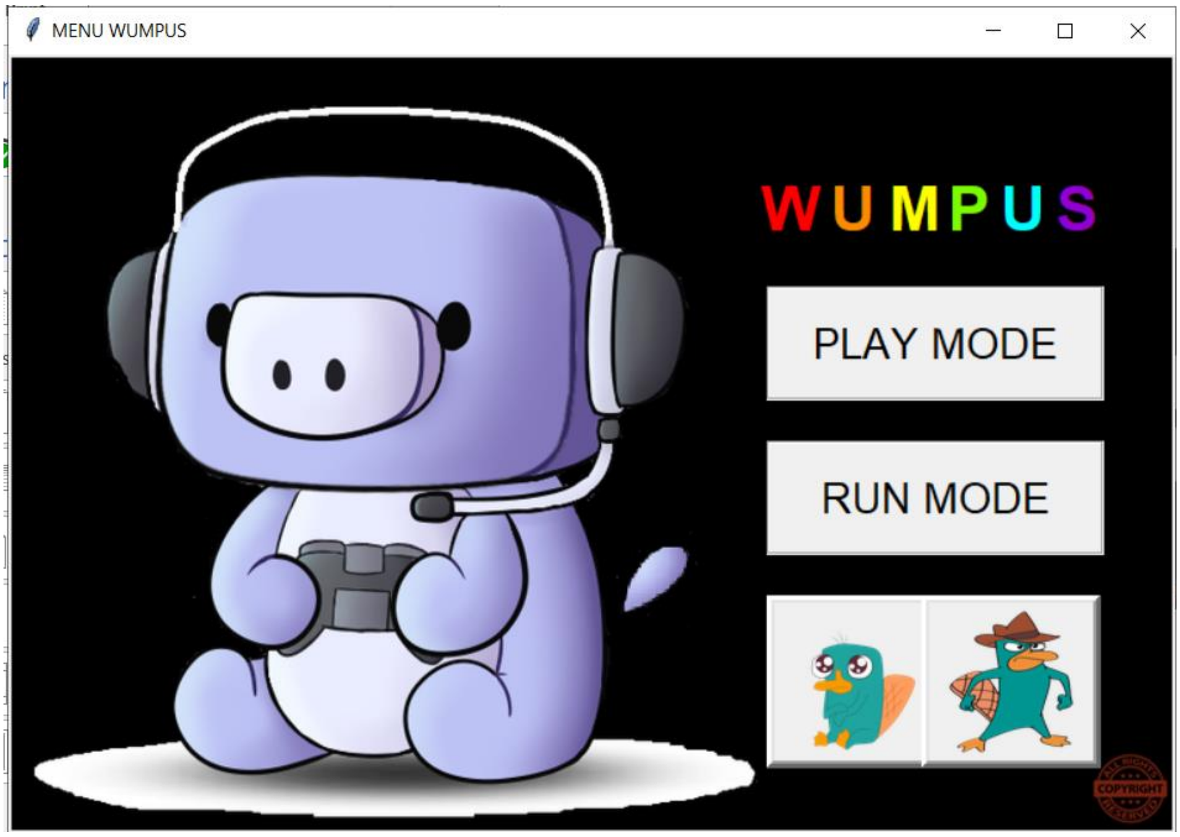
No packages published
Publish your first package

Contributors 2

MinhVu2018
Ngocien

Estimating the degree of completion level for each requirement

1. Finish problem successfully with 2 modes and 2 strategies for auto-run mode: 100%
2. Graphical demonstration of each step of running process: 100%

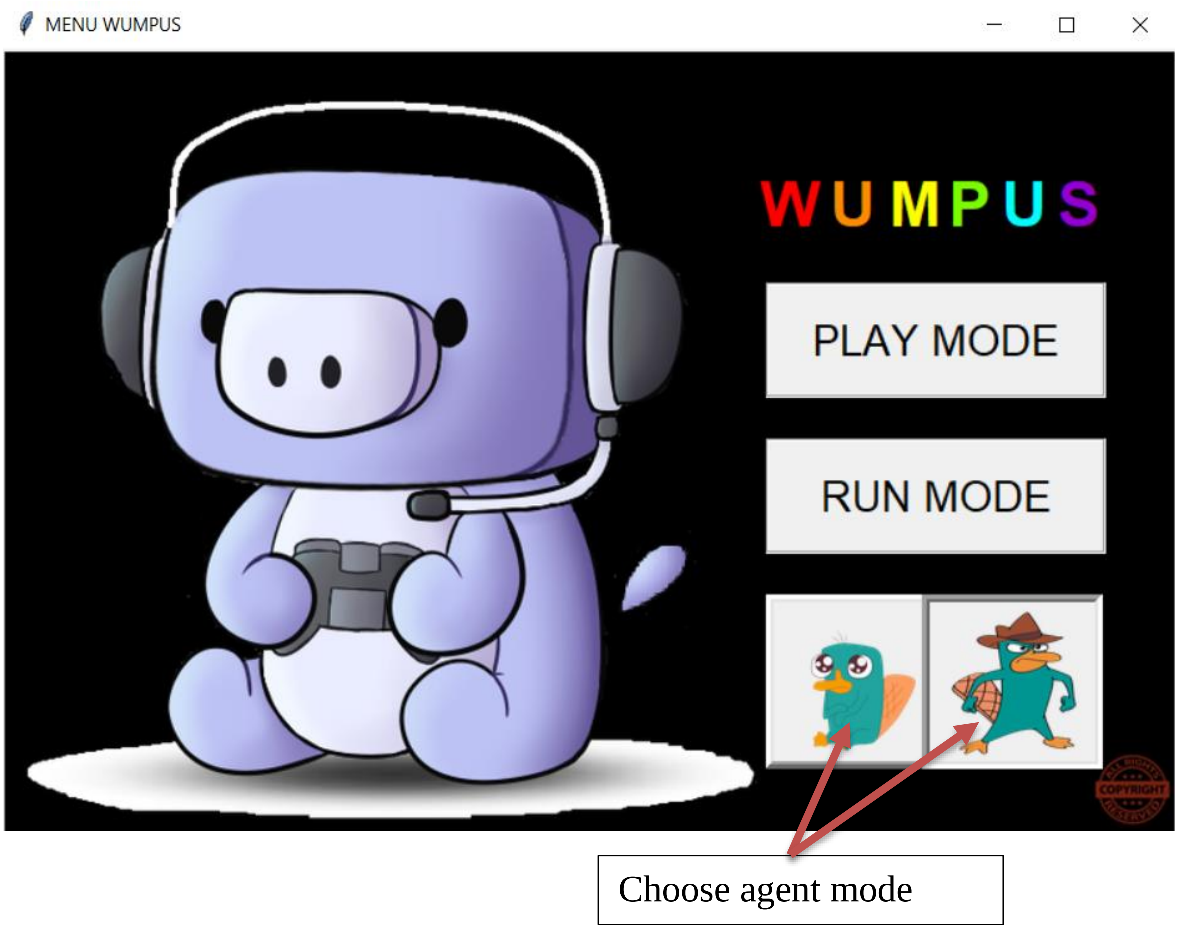


3. Generate 5 maps input and implementation generate Wumpus World randomly with difference structures such as position and number of Pit, Gold, Wumpus: 100%
4. Report our algorithm and compare the result of each strategy: 100%

Instruction

To play the game, you should follow steps below.

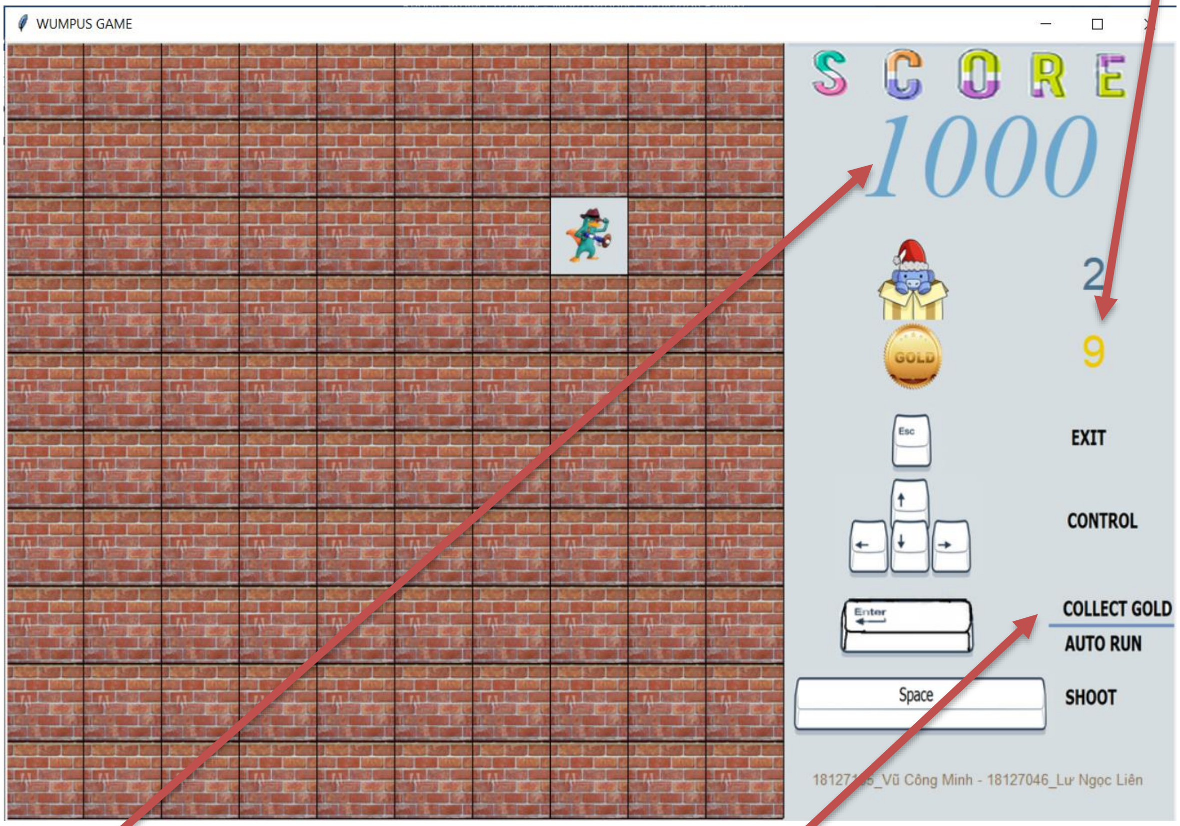
1. You have to choose what strategy mode (agent mode) you want to play by clicking one of 2 pictures below.
   - The left one is we call **shy agent**. The agent will try to collect all golds and remain score is as high as possible.
   - The right one is we call **aggressive agent**. The agent will kill wumpus as soon as they have a confident prediction.
2. After choosing strategy mode, you will choose the game mode by clicking one of 2 buttons. We have 2 for you.
   - The up one is the **play mode**. That means you will control the agent, kill wumpus and collect gold by yourself.
   - The down one is the **run mode**. That means the agent will move automatically, collect gold automatically and kill wumpus automatically.



Blue color: Number of wumpus

Yellow color: number of gold

3. Then you will see:



Initial score will be 1000

- Run mode: Press Enter to play
- Play mode:
  - Press Enter to collect gold
  - Esc to exit
  - Space to shoot

4. Finish Game: Congrats or Game Over

Algorithm explanation

What we use for algorithm:

- Current node: tuple (index, state, True/False)
  - If there is a Pit or Wumpus, it will be False
  - Else it will be True (Moveable)
- List Predicted: stored nodes at the current node with (⋀) operator
  - It will be update after current node add to List Visited
- List Visited: stored nodes visited
  - Add current node to list visited
- KB: stored list of [visited ⋀ predicted]

S		B	P
W	BS	P	B
S		B	
A	B	P	B

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

○ *Step 1:*

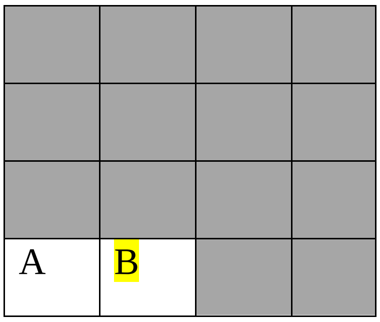
▨	▨	▨	▨
▨	▨	▨	▨
▨	▨	▨	▨
A	▨	▨	▨

- Current node: (3,’-’,True)
- List Predicted: [(3,’-’,True), (2,’-’,True), (7,’-’,True)]

  Update to Visited
- List Visited: [(3,’-‘,True)]

  Update to Predicted
- List Predicted: [(2,’-‘,True), (7,’-‘,True)]
- KB: (3,’-‘,True) ^ (2,’-‘,True) ^ (7,’-‘,True)

- *Step 2:*



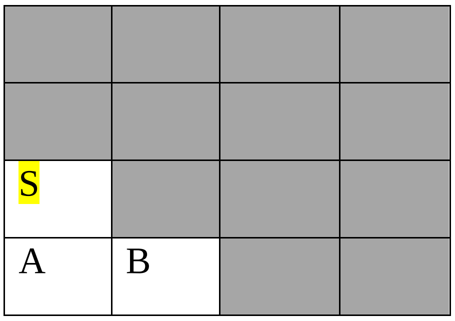
- Current node: (7, ’B’, True)

  Update to Visited
- List Visited: [(3,’-‘,True), (7,’B’,True)]
- List Predicted: [(2,’-‘, True)]

  Update to Predicted based on the last visited node

  --If last visited node is “Breeze” -> Predict “Pit” in its adjacency nodes
- Perceive: [(6,’P‘,False), (11,’P‘, False)]
- KB = (7,’B’,True) ^ (3,’-’,True) ^ (2,’-’,True)
- Update List Predicted: [(2,’-‘,True), (6,’P‘,False), (11,’P‘, False)]

- *Step 3:*



- Current node: (2, ’S’, True)

  Update to Visited

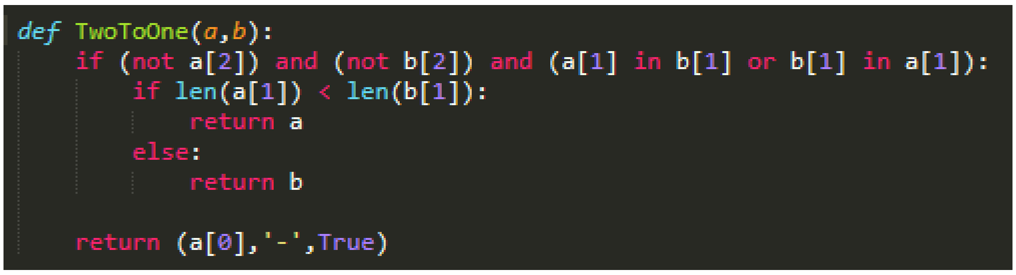
- List Visited: [(3,’-‘,True), (7,’B’,True), (2,’S’,True)]

- List Predicted: [(6,’P‘,False), (11,’P‘, False)]

  Update to Predicted based on the last visited node

  --If last visited node is “Stench” -> Predict “Wumpus” in its adjacency nodes

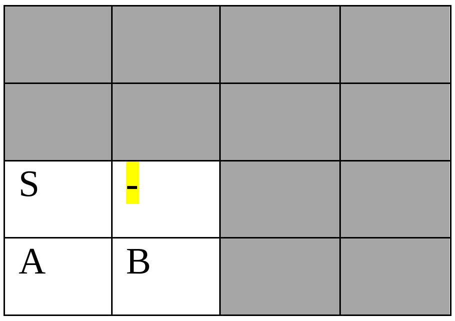
- Perceive: [(6,’W‘,False), (1,’W‘, False)]
- KB = (3,’-‘,True) ^ (7,’B’,True) ^ (2,’S’,True) ^ (6,’P‘,False) ^ (11,’P‘, False)
  There are 2 nodes at index 6 so we use our TwoToOne function to handle:
  Clause: (“W” in “P”) or (“P” in “W”) is False so it makes function return a safe node (index,‘-‘, True).
  We can imagine that a node predicted both “Pit” and “Wumpus” is impossible, so it could be a safe node, no “Pit” no “Wumpus” here.



```python
def TwoToOne(a,b):
    if (not a[2]) and (not b[2]) and (a[1] in b[1] or b[1] in a[1]):
        if len(a[1]) < len(b[1]):
            return a
        else:
            return b

    return (a[0],'-',True)
```

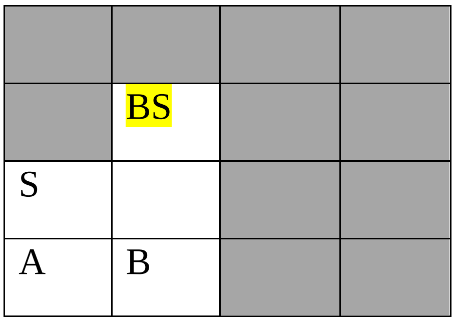
- Update List Predicted: [(1,’W‘,False), (11,’P‘, False)]

○ *Step 4:*

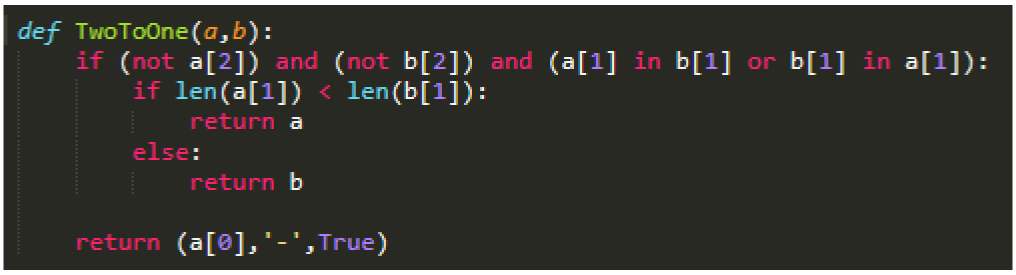


- Current node: (6,’-’, True)
  Update to Visited
- List Visited: [(3,’-’,True), (7,’B’,True), (2,’S’,True), (6,’-’, True)]
- List Predicted: [(1,’W’,False), (11,’P’, False)]
- Perceive: [(5,’-’,True), (10,’-’,True)]
- KB = (3,’-’,True) ^ (7,’B’,True) ^ (2,’S’,True) ^ (6,’-’, True) ^ (1,’W’,False) ^ (11,’P’, False)
- Update List Predicted: [(1,’W‘,False), (11,’P‘, False), (5,’-’,True), (10,’-’,True)]

- *Step 5:*



- Current node: (5,’BS’, True)
  Update to Visited
- List Visited: [(3,’-’,True), (7,’B’,True), (2,’S’,True), (6,’-’, True), (5,’BS’, True)]
- List Predicted: [(1,’W‘,False), (11,’P‘, False), (5,’-’,True), (10,’-’,True)]
  Update to Predicted based on the last visited node
  --If last visited node is “Stench” and “Breeze”
  -> Predict (“Wumpus” or “Pit”) in each its adjacency nodes
- Perceive: [(1,’PW’,False), (4,’PW’,False), (9,’PW’,False)]
- KB = (3,’-’,True) ^ (7,’B’,True) ^ (2,’S’,True) ^ (6,’-’, True) ^ (5,’BS’, True) (1,’W‘,False) ^ (11,’P‘, False) ^ (5,’-’,True) ^ (10,’-’,True)
  There are 2 nodes at index 6 so we use our TwoToOne function to handle
  +Clause: (“W” in “WP”) or (“WP” in “W”) is True so it makes function return a Wumpus node (index, ‘W‘, False).
  +We can imagine that “WP” is (W v P)
  “W” is (W)
  +The clause would be W ^ (Wv P) so it will be W ➔ Wumpus node is here



```
def TwoToOne(a,b):
    if (not a[2]) and (not b[2]) and (a[1] in b[1] or b[1] in a[1]):
        if len(a[1]) < len(b[1]):
            return a
        else:
            return b

    return (a[0],'-',True)
```

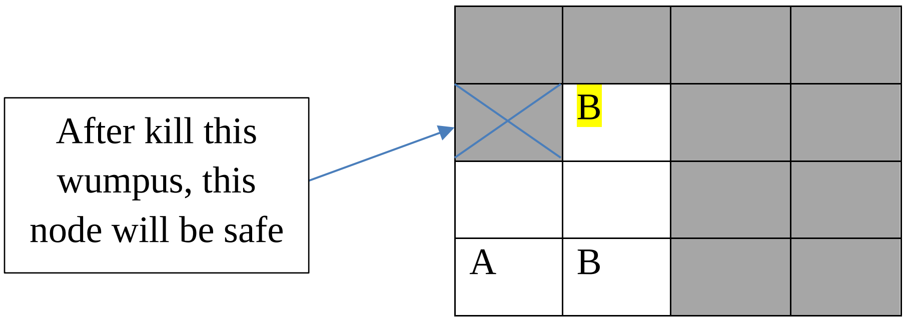
- Update List Predicted: [(1,’W‘,False), (11,’P‘, False), (5,’-’,True), (10,’-’,True), (4, ‘WP’, False), (9,’WP’, False)]
- Now, we have checked node 1 is Wumpus 2 times. So it must be wumpus here.

- In shy_mode: Agent don't kill this Wumpus and choose another safe node in predicted to visit
- In aggressive_mode: Agent kill this Wumpus and request Game to update its ListVisited and ListPredicted:

+ ListVisited: [(3,'-',True), (7,'B',True), (2,'-',True), (6,'-', True), (5,'S', True)]

+ List Predicted: [(1,'-',True), (11,'P', False), (5,'-',True), (10,'-',True),

(4, 'P', False), (9,'P', False)]

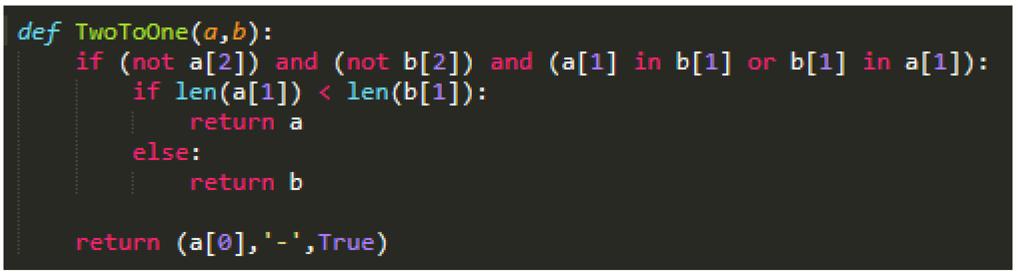


Some explanations:

- When we predicted all the next node at the current node is wumpus or pit. We will **use A* search** to find the shortest path from current node to the safe node **in List Visited** to remain score as high as possible. **(shy agent)**
- When we predicted the next node at the current node is a wumpus for sure, kill wumpus and this wumpus node will be safe node **(aggressive agent)**. Else use A* search to find the shortest path from current node to the safe node in List Visited.
- When at the current node has gold, agent **requests to collect the gold.**
- When we shoot the wumpus, all stench at top, down, left, right of the wumpus will **be requested to delete and update** to the List Predicted.
- After update visited list and predicted list, the agent **sends a signal** to graphic request its display.
- Agent has a list wumpus predicted to handle at the **second addition** of this node, we will kill it because its could be monster. If it's the first time, agent don't need to kill because shooting decrease our score by 100. If it's the third or fourth for

sure, agent may have a long path before going back to its Stench node around and each moving step decrease our score by 10. So that, we think **2 times is the best choice**.

- Because shooting decrease our score 100, Agent shouldn't shoot anymore. If Agent shoot, there would be open more rooms and we can collect more golds. Each step costs 10 and collect gold gives 100 but the number of golds is limited. So, Agent have to **exchange more steps to find 1 or 2 gold**. **It doesn't help Agent to have the highest score when we choose Aggressive_Mode**. We can see the comparison at **page 16-17** below.
- After kill Wumpus, Agent **predicts it like a safe node and choose next node to visit again**. So, in some cases Agent kill Wumpus at the left but visit the right one because two safe nodes there has the same minimum path from Agent.
- When Agent's ListPredicted is full of False nodes, Agent call ClimbOut function **to return back the initial room and escape the maze**.
- **TwoToOne** function to handle a node which has two prediction.



```python
def TwoToOne(a,b):
    if (not a[2]) and (not b[2]) and (a[1] in b[1] or b[1] in a[1]):
        if len(a[1]) < len(b[1]):
            return a
        else:
            return b

    return (a[0],'-',True)
```

Case 1: If we have one True node, the clause will return False and it will be predicted as safe node

Case 2: Both a and b are ("W", False) or ("P", False) the clause will return True and it will be predicted as the same with a and b

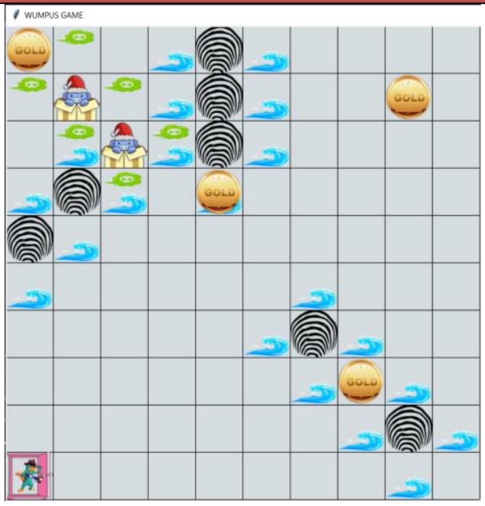
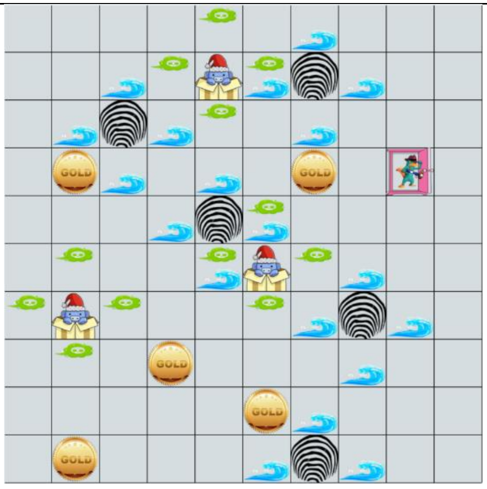
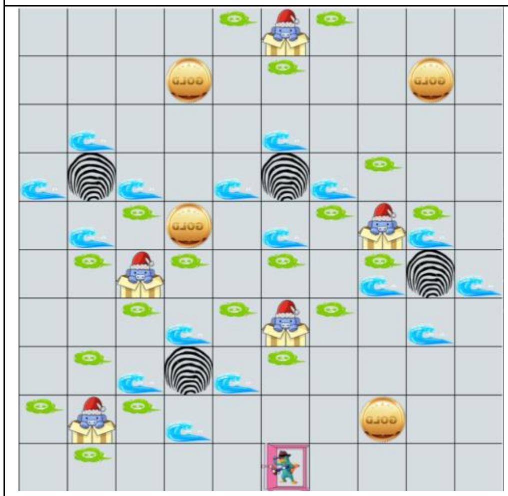
Case 3: If one of them is ("WP", False) the clause will return True and it will be predicted as the another one.

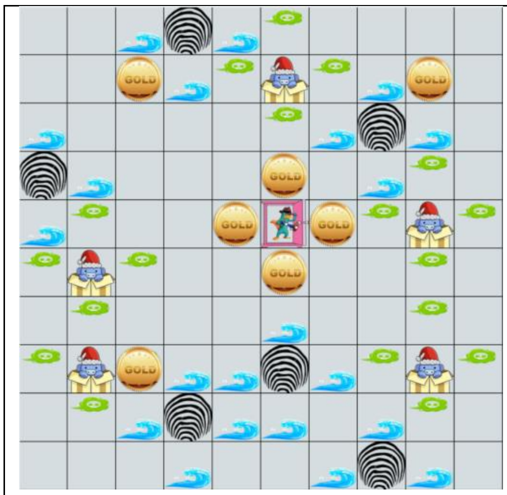
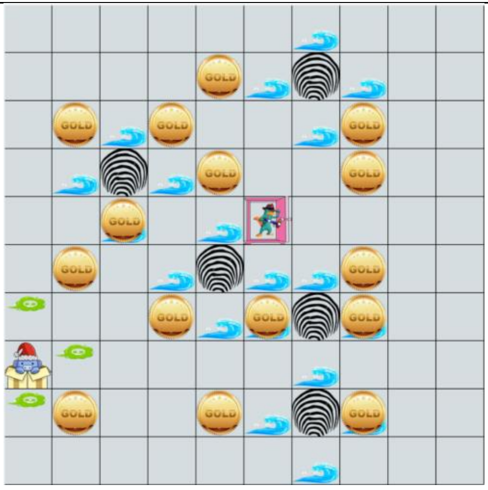
+ "WP" in "WP" is True ➔ That node still be predicted as Wumpus or Pit

+ "W" in "WP" is True ➔ That node would be predicted as Wumpus

+ "P" in "WP" is True ➔ That node would be predicted as Pit

Comparison between 2 strategy mode

Maze	Shy agent	Aggressive agent
 WUMPUS GAME	250	-260
	610	-360
	710	-700

	1000	-200
	1330	1200

References

PL and FOL from lectures

https://www.seas.upenn.edu/~cis391/Lectures/Propositional-Logic.pdf

Create Transparent Image: remove.bg

Images: from google search and self-draw with Paint, Photoshop…