IFN680 Semester 2, 2019

Assignment 1
The Wumpus World – a Probability based Agent

Due Date: Sunday, 15/09/2019
Individual or group work: group size can be 1 – 3 people
Weight: 20%

~~~~~~

1. Introduction

Hunt the Wumpus -- is a well-known simple computer game which was originally written by Gregory Yob in BASIC in 1972. The game is in a cave consisting of many rooms connected by passageways. Lurking somewhere in the cave is a dreaded wumpus, a smelly, scary monster that eats anyone who enters its room. Typically, the game is to hunt the wumpus and kill it without getting killed. However, for this assignment, we will adopt a simplified version used in the Russell&Norvig AI text book, instead of killing the wumpus, the goal of the game is to find a heap of gold hidden somewhere in the cave without getting killed by the wumpus or falling into bottomless pits in the cave. Our goal is to write a program simulating an agent in the wumpus world to find the gold without being killed. One feature of the wumpus world is that the positions of the wumpus, the pits, and the gold are all random and the agent doesn't know where they are in the cave. The agent can only observe the room where it is and has no any idea about other rooms. Therefore, the wumpus world is partially observable. Moreover, the wumpus world is stochastic because what will happen when the agent moves into a room is uncertain, the agent could be eaten by the wumpus if the agent enters a room where the wumpus is sitting, or the agent falls into a bottomless pit, or the agent finds the gold, or there is nothing there. Luckily, the agent can perceive some valuable information in the room that it is in, which will help the agent to make decisions about whether entering or not entering any of its adjacent rooms. The wumpus world problem for this assignment is characterized below:

- The cave environment is a two-dimensional grid of squares, each square represents a room. There is only one wumpus, one piece of gold, and at least one pit, i.e., there could be multiple pits. Once they are deployed, they are static and never change positions. There is only one agent in the game. When the game starts, the agent is always sitting in the bottom-left room initially. The wumpus, the gold and the pits are randomly deployed in the environment when the game starts.
- The agent can perceive the following two kinds of information:
    - A breeze, if the agent's room is adjacent to a pit directly (not diagonally)
    - A stench, if the agent's room is adjacent to the wumpus directly (not diagonally).

Figure 1 depicts an example of the cave environment with two pits.

The agent can feel a breeze or smell a stench only in its current room if the room contains a breezy or a stench, and importantly, the agent will remember the breeze or stench information of all the rooms that it has visited so far. The breeze & stench
~~~~~~

information of all the visited rooms is crucial to the agent since the information warns the agent that danger is nearby, there must be some rooms (which contain a pit or the wumpus) adjacent to the breeze or stench rooms.

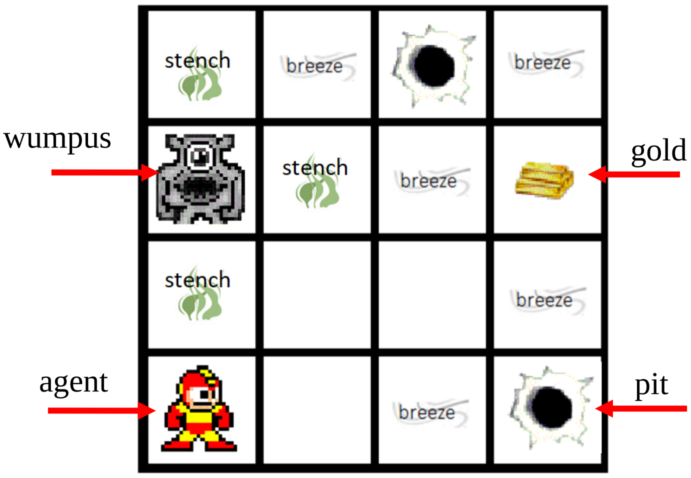


Figure 1 A 4x4 cave environment

For this assignment, we treat the wumpus and pits equally, meaning that a room is equally dangerous to the agent if the room contains a pit or the wumpus. Furthermore, a room contains a pit or the wumpus with the same probability 0.2, i.e., the prior probability of a room containing a pit or the wumpus is 0.2.

- The agent can move from one room to an adjacent room vertically or horizontally, but not diagonally. When the agent finds the gold, or enters a room which contains the wumpus or a pit, the game ends. If the agent has explored all available and safe rooms without finding the gold (and still alive), the agent will return to the original room empty handed but alive, then the game ends.

One approach to solve the wumpus world problem is to conduct a logical reasoning to determine whether or not an adjacent room is safe based on the breeze & stench information that the agent perceived so far. In fact, for this assignment, you are provided with a Python program which is an implementation of a logic-based approach for solving the wumpus world problem. The ‘agent’ in the provided program is called a logic-based agent. This agent is very cautious and never takes risk. It enters a room only when the agent is 100% sure that the room does not contain the wumpus or a pit. Therefore, the agent is safe and never be killed. However, the agent often misses opportunities to get the gold. For this assignment, your goal is to increase the agent’s chance to get the gold (and, at the same time, may increase the chance to get killed as well!). Your task is to implement a probability-based approach to allow the agent to explore a room which is probably unsafe but has a low probability of containing the wumpus or a pit.

2. A Probability-based Approach for the Wumpus World Problem

A proposition $PW_{i,j}$ is used to represent whether location (i, j) contains a pit/wumpus or not. $PW_{i,j}$ is a Boolean variable, true if (i, j) contains a pit/wumpus, false otherwise.

Similarly, a proposition $BS_{i,j}$ represents whether location (i, j) contains breeze/stench. Figure 2 depicts an instance of a cave environment.

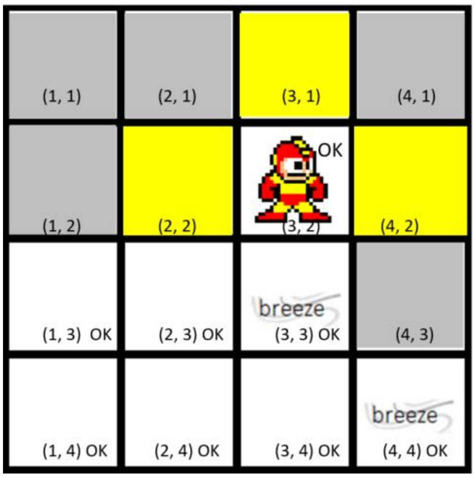


Figure 2
A cave environment instance

At any time during the game, the rooms in the cave can be divided into three parts, R_{known}, R_{query}, R_{others}. The three parts together cover all the rooms in the cave environment:

- R_{known}: a set of rooms that the agent has visited so far, e.g., the rooms marked with OK in Figure 2. These rooms are safe and the truth value of the pit_wumpus variables representing these rooms is 'False'. This part is known to the agent. Let PW_{known} and BS_{known} denote the known pit_wumpus and breeze_stench variables, respectively, for Figure 2, $PW_{known} = \{PW_{1,4} = False,\ PW_{2,4} = False, \dots\dots, PW_{3,2} = False\}$ and $BS_{known} = \{BS_{3,3} = True, BS_{4,4} = True,\ BS_{1,4} = False, \dots\dots, BS_{3,2} = False\}$
- R_{query}: a set of rooms that adjacent to the agent's current location and that have not been visited by the agent, e.g., the yellow rooms in Figure 2. This part is unknown to the agent, but they are accessible to the agent, it is the part that the agent will decide which room to explore for next move. We call the rooms in this part query rooms.
- R_{others}: a set of rooms that the agent has not visited and that are not adjacent to the agent's current location. These rooms are unknown to the agent, e.g., the grey rooms in Figure 2, and they are also not accessible to the agent. .

The agent needs to choose a room from R_{query} to explore. For the probability-based agent, the decision is made based on a conditional probability that each query room in R_{query} contains a pit/wumpus given the known information PW_{known} and BS_{known}. The conditional probability can be calculated using the following equation:

$$\boldsymbol{P}\left(P_q | PW_{known}, BS_{known}\right) \\ = \alpha \sum_{y \in R_{unknown}} \boldsymbol{P}(P_q, PW_{known}, BS_{known},\ y) \qquad (1)$$

where P_q is one query variable in R_{query} and $R_{unknown} = R_{others} \cup (R_{query}$ - $\{P_q\})$ containing all the unknown variables (i.e., the query variables and the inaccessible variables) excluding the particular query variable P_q. These unknown variables are the hidden variables to be enumerated. Based on product rule, we can get the following equation, where y is a truth assignment to all variables in $R_{unknown}$.

$$\boldsymbol{P}\left(P_q | PW_{known}, BS_{known}\right) \\ = \alpha \sum_{y \in R_{unknown}} \boldsymbol{P}(BS_{known} | P_q, PW_{known}, y) \times \boldsymbol{P}(P_q, PW_{known}, y) \qquad (2)$$

For this assignment, the code to compute $\boldsymbol{P}(BS_{known}|P_q, PW_{known}, y)$ is provided. You can call function `consistent(self,known_BS,event)` in `Robot` class to calculate the value, `known_BS` contains the known breeze_stench values to the visited rooms and `event` is a configuration to all rooms in the cave with the known truth values to the variables in PW_{known}. $\boldsymbol{P}(P_q, PW_{known}, y)$ is the joint probability distribution.

For this assignment, you are required to complete the following two functions in the given file `probability_based_move.py`:

- `PitWumpus_probability_distribution(width, height):` construct the joint probability distribution.
- `next_move_prob():` return a room location (`column, row`) based on the probability calculated using Equation (2) for the agent's next move.

If need, you can add more functions in `probability_based_move.py`. In this case, you need to add the links to these functions on top of the Class `Robot` in `the_wumpus_world.py`.

3. The Provided Python Package

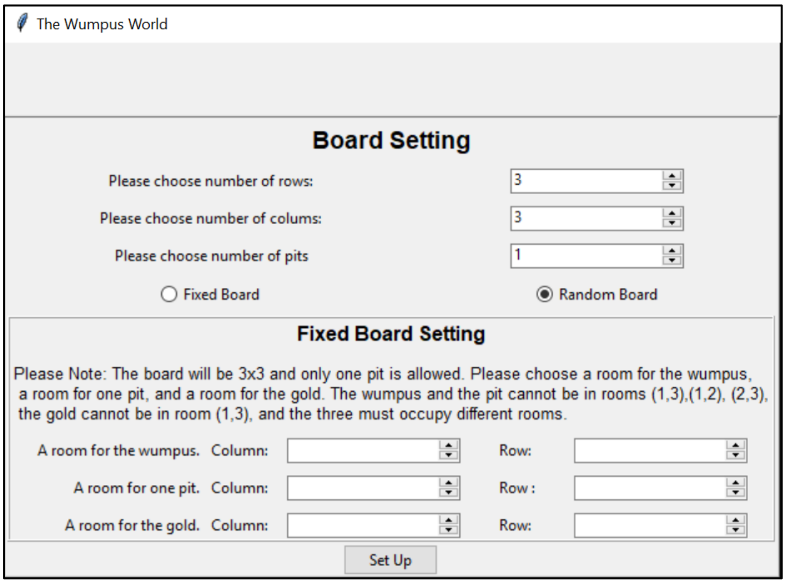
To facilitate the development of your program, a Python package is provided to you. In the package, you can find three Python files:

- `the_wumpus_world.py` : this file contains the main function. It provides all the facility code to define the board, the agent, set up the game and all the graphical user interfaces (GUI), and handle the agent's moves.
- `logic_based_move.py` : this is the logic-based agent which returns a location (column, row) for the agent to move by using the resolution proof method.
- `probability_based_move.py` : this is the file that you need to complete. It contains the two incomplete functions mentioned above.

Both `the_wumpus_world.py` and `logic_based_move.py` are complete. You cannot make changes to these two files except for adding links to your functions in `the_wumpus_world.py.`

3.1 Graphical User Interfaces

The given package provides a graphical environment which allows you to configure the game board, select an agent to play the game, i.e., the logic-based agent or the probability-based agent, and define the maximum probability if the probability-based agent is chosen. You can start the game by executing the program in `the_wumpus_world.py`. When you start running the program, you will first see the GUI for initializing the game board as shown in Figure 3. You can change the board size and choose the number of pits. By default, the board is 3x3 and you only can choose 3 or 4 for the number of columns and rows. You can choose up to 3 pits. In addition, you can choose to generate a board with specific setting or choose to generate a random board. The 'Fixed Board' option is designed for the convenience of debugging your code. During the code development, you may like to use the same board to test your program in order to find problems. In this case, you can choose the fixed board option. A few specific locations cannot be used for the wumpus or the pits, which include the bottom-left location (i.e., (1, 3) for a 3x3 board, the first index is column, the second index is row), and the rooms at (2, 3) and (1, 2) for a 3x3 board. The reason is that, (1, 3) is the starting position of the agent, and when (2, 3) or (1, 2) is occupied by the wumpus or a pit, for the logic based agent, the agent will never move because its position (1, 3) will have breeze or stench.



The Wumpus World

Board Setting

Please choose number of rows: 3

Please choose number of colums: 3

Please choose number of pits 1

○ Fixed Board   ● Random Board

Fixed Board Setting

Please Note: The board will be 3x3 and only one pit is allowed. Please choose a room for the wumpus, a room for one pit, and a room for the gold. The wumpus and the pit cannot be in rooms (1,3),(1,2), (2,3), the gold cannot be in room (1,3), and the three must occupy different rooms.

A room for the wumpus. Column: ______ Row: ______

A room for one pit. Column: ______ Row : ______

A room for the gold. Column: ______ Row: ______

Set Up

Figure 3

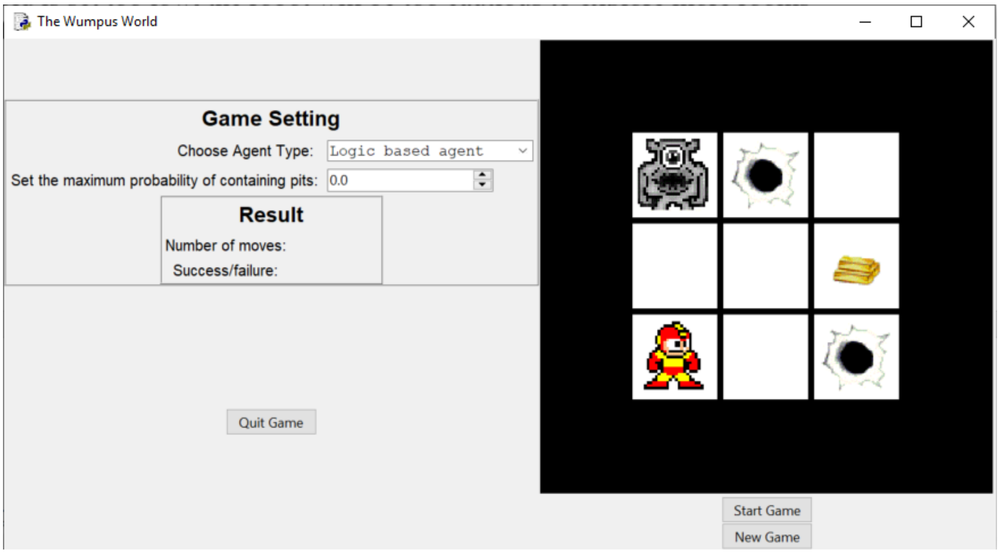
One thing that needs your specific attention is that, in this implementation, the position (1, 1) is the top-left room and the position (4, 4) is the bottom-right room in the 4x4 board. The coordinate layout for the 4x4 board is illustrated in Figure 4.

(1,1)	(2,1)	(3,1)	(4,1)
(1,2)	(2,2)	(3,2)	(4,2)
(1,3)	(2,3)	(3,3)	(4,3)
(1,4)	(2,4)	(3,4)	(4,4)

Figure 4

Once you click on the Set Up button, you will see the game board GUI as shown in Figure 5 which is a randomly generated board with 2 pits, the robot is always at the bottom left position initially.

Before you can start the game, you need to choose an agent. By default, the logic-based agent is chosen. If you choose the probability-based agent, you need to specify the maximum probability threshold that a room contains a pit/wumpus. If the threshold is set higher, the robot will have a higher chance to be killed by the wumpus or fall into a pit. If the threshold is set too low, the robot will be too cautious to explore more rooms and thus may not be able to find the gold. When the threshold is set to 0, the probability-based agent will perform the same as the logic-based agent.



The Wumpus World

Game Setting

Choose Agent Type: Logic based agent

Set the maximum probability of containing pits: 0.0

Result

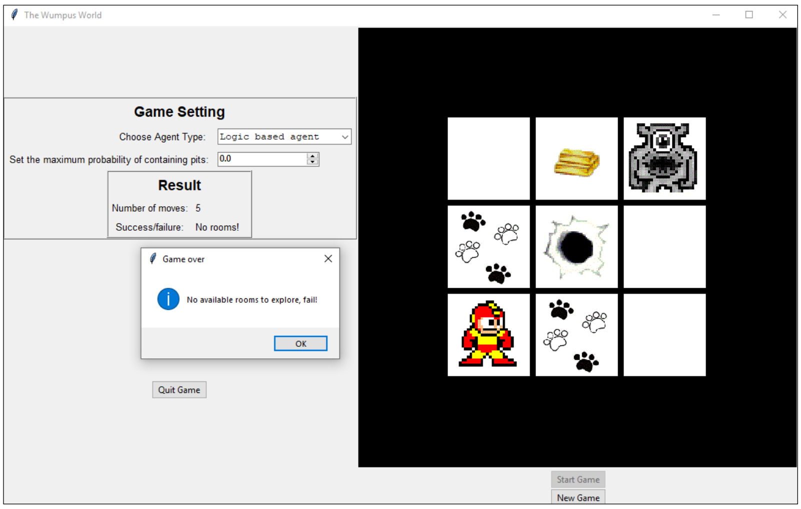
Number of moves:

Success/failure:

Figure 5

You can click on the Start Game button to start the game. Then the robot will explore the cave by itself without any interaction with the user. Once the game ends, you can click on the New Game button to start a new game with the same board setting, but may be in a different configuration if you have chosen the random board option.

A game ends with three possible outcomes: the robot found the gold, the robot returned to the starting position empty handed without finding the gold, or the robot was dead (i.e., killed by the wumpus or fell into a pit). But for the logic-based agent, there are only two outcomes, finding the gold or empty handed. The logic-based agent will never be killed. The result will be displayed on board as well, which includes the number of moves that the robot has made and the outcome of the game, success (i.e., finding the gold) or failure (i.e., dead or no room to explore). Also some message boxes will be popped out to indicate the outcomes. Figure 6 shows an example outcome where the logic agent failed to find the gold. This is a typical board setting that the logic-based agent will fail to find the gold, while the probability-based agent would be able to find the gold if the probability threshold is set properly.



The Wumpus World

Game Setting

Choose Agent Type: Logic based agent

Set the maximum probability of containing pits: 0.0

Result

Number of moves: 5

Success/failure: No rooms!

Game over

No available rooms to explore, fail!

Figure 6

3.2 Classes and useful functions

You don't have to understand the entire code in the_wumpus_world.py to do this assignment. But some functions and variables defined in the code may be useful.

- **Class Robot**

The class Robot defines the agent (called a robot in this code). The class provides the functions for the agent to get the surrounding information of its current location, conduct reasoning to determine its next move, and make the move to a new location. The provided code allows the agent to conduct a resolution based reasoning to determine next move mainly using three functions in the class, kb_add() to build its knowledge base, check_safety() to check whether or not a room is safe (i.e., there is no pit/wumpus in the room), and next_room() to determine a safe room (next_room() is in Python file logic_based_move.py). The provided code also allows the agent to conduct a probabilistic reasoning to choose a room based on probabilities. The function to complete the probabilistic reasoning is next_room_prob(), which is incomplete. Your task is to complete this function.

The following variables and functions in class Robot may be useful.

Robot variables

- **self.cave**: it is an object of Cave, it is the cave board that the agent is in ('self' means this is an instance variable of this class). The two functions that you are required to complete are methods of Robot class. This means that, in the two methods you can get access to the cave via variable **self.cave.**
- **self.visited_rooms**: it is a set object, it contains a set of (column, row) pairs of all the rooms visited by the agent so far.
- **self.jdp_PW**: it is an object of JointProbDist. It provides the joint probability distribution of pit/wumpus in the given cave environment.
- **self.max_pit_probability**: it is a floating number, it is the maximum probability threshold specified by the user.
- **self.current_position**: it is a tuple (column, row), it is the location of the room that the agent is in currently.

Robot functions

- **self.check_safety(self, column, row):** return True if the room (column, row) is safe (i.e., there is no pit/wumpus in the room), otherwise False.
- **self.consistent(self,known_BS,event):** return 1 if the truth values in known_BS are consistent with the values in event, 0 otherwise
  - known_BS: a dict containing the visited rooms with their corresponding truth value for breeze/stench;
  - event: a dict containing the rooms each with an instantiated truth value for pit/wumpus.
- **self.observation_pits(self,observed_locations):** return a dict containing a set of var: val pairs indicating whether these rooms contain a pit or not. var is the variable name of a room, val is a truth value.
  - observed_locations: a set of (column, row) pairs, which have been visited.

- **`self.observation_breeze_stench(self,observed_locations):`** return a dict containing a set of `var`: `val` pairs indicating if each of the rooms has a breeze/stench. `var` is the variable name of a room, `val` is a truth value.
  - `observed_locations`: a set of (column, row) pairs, which have been visited.

- **Class `Cave`**

This class defines the board for the 2-dimensional cave environment. Each room in the cave is specified by a pair of column and row, (column, row) as showed in Figure 4.

- **`self.getsurronding(column, row):`** You can use this function via an object of `Cave` to return the adjacent rooms of a specific location `(column, row)`, e.g., `self.cave.getsurrounding(x,y)` returns a list of adjacent rooms of `(x,y)` in cave `cave`, each room is a pair of (column, row).

You can use the variables defined in this class such as `self.WIDTH` and `self.HEIGHT` which are the number of columns and rows in the cave. But you cannot use variables `_goldCoor`, `_wumpusCoor`, and `_pitCoors`, which are protected variables and cannot be used in other classes.

5. Requirements

You can complete this assignment individually or in a team of up to 3 students.

1) Your task is to complete the two required functions in `probability_based_move.py.`
2) You must insert your name and your student ID at the beginning of both files `probability_based_move.py` and `the_wumpus_world.py.` Fail to do so will cause mark deduction. For a team, all team members' name and ID should be added.
3) Your code must implement a probability based approach to choose a room for the next move. You are encouraged to implement the approach explained in Section 2 (explained in week 5 lecture as well). You can implement other probability based approaches. But it will not bring any extra marks (it may cause mark deduction if it is not correctly implemented).
4) Again, in your code, you cannot use any of the three protected variables defined in class `Cave`: `_goldCoor`, `_wumpusCoor`, and `_pitCoors`
5) Your code must be well-presented and easy to understand. Concise inline comments are required to explain the purpose of significant code segments and a brief header comment is required for each of your methods.
6) You are required to write a report which contains the following content:

   a) A statement of completeness
      i. Provide a list of functions completed in your `probability_based_move.py` file
      ii. If you have modified any part of the code which was not allowed to modify, list the changed part/parts and provide your justification about why you need to do so.

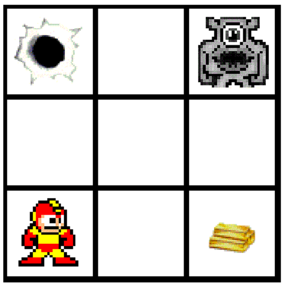
iii. A workload distribution over team members in the case of a team work.

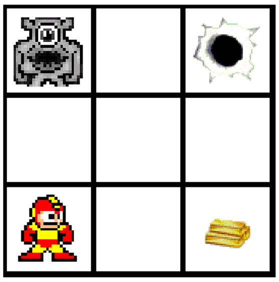
b) Probability-based approach

Indicate which probability-based approach has been implemented in your code, the approach explained in Section 2 or some other approaches. Provide a brief description to the approach that you have implemented. You need to cite at least one reference for the approach if you didn't implement the approach in Section 2. A half page (at most one page) should be sufficient for describing your approach.

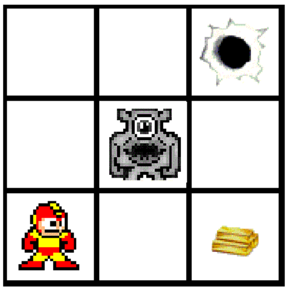
c) Test cases

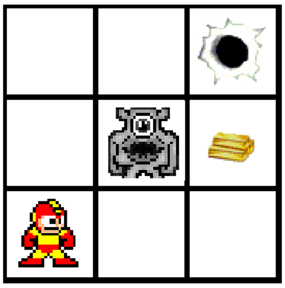
Provide four example scenarios to show that the logic-based agent fails to find the gold, but your probability-based agent can. The scenarios can be 3x3 or 4x4 board, one pit or more pits. You need to include the screenshots of the cave boards to show the comparison for each of the examples. The four board configurations must be substantially different. For example, the following two boards are actually identical given that we treat pits and the wumpus the same.





The following two boards are also very similar since the pit and the wumpus positions are identical.





Your code will be tested using some board settings. The four example boards provided by you will also be used to test your code.

6. Submission

Submit your assignment on Blackboard. If you are in a team, only one of the team members needs to submit the assignment. Make sure you list all the team members in your report and the code as well. The feedback will be given to the person submitting the assignment and is expected to be shared with the other team members.

One single zip file containing the following files should be submitted:

1. A report in pdf format.
2. Your Python files, `probability_based_move.py` and `the_wumpus_world.py`.

7. Marking criteria

Name	Student ID	Total Marks
		/20

Report Quality **4 marks**

4 marks	3 marks	2 marks	1 mark	0 mark
• Professionally very well presented report with appropriate paragraphs and sentence structure. • Easy to understand • Include all required content. • Meet all requirements specified in Section 5. • Use fluent language with correct grammar, spelling and punctuation. No errors.	• Report is very well written and easy to understand. • Include all required content. • A few insignificant language errors or inappropriate sentences	• Report is generally written well and understandable • A few language issues make some content unclear • Missing some required information, e.g., workload allocation, team information	• Large parts of the report are poorly written, making many parts difficult to understand • Missing required information • Fail to meet some of the requirements, e.g., change the provided code	• The entire report is poorly written and hard to understand • Missing important required content • Fail to meet the requirements
Sub-total mark	**/4**			

<u>Function Implementation</u> 8 marks

Functions	**Criteria**	**Marks**	**comments**
PitWumpus probability distribution	Use appropriate functions in the provided package	/1	
	Correctly calculate probability distribution	/1	
Next-move prob	Reasonable algorithm/idea	/2	
	Correctly implement the algorithm to calculate the probability	/2	
	Use appropriate functions in the provided package	/2	
Sub-total mark	**/8**		

<u>Code Quality</u> 4 marks

4 marks	**3 marks**	**2 marks**	**0-1 mark**
• Code is easy to understand • Clear, uncluttered layout • Appropriate inline comments • Informative header comments for each method • Meaningful variable names and constants • Correct use of parameters	• Code is easy to understand • Insufficient comments • Use of meaningless variables	• Code is roughly understandable • Missing method header comments • Many meaningless variables	• Code is hard to understand • No comments at all
Sub-total mark	**/4**		

<u>Test Cases</u> 4 marks

4 marks	**3 marks**	**2 marks**	0-1 mark
• Pass all tests • Valid self-defined cases	• A few of the self-defined cases are invalid, e.g., too similar • Fail one test	• Fail half of the tests • Most of the self-defined cases are invalid	• Fail most of the tests • All self-defined cases are invalid • 0 if fail all tests
Sub-total mark	**/4**		

Final Remarks

- Do not underestimate the workload. Start early. You are strongly encouraged to ask questions during the practical sessions.
- Email questions to yue.xu@qut.edu.au

THE END