Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876

Report For Hunt the Wumpus:

Summary and Program Functionality

The Hunt the Wumpus game is a text-based game created in the 1970s. This involves the player moving through a series of caves that are connected, in the shaped of a dodecahedron. The player's aim to is hunt a monster called the "Wumpus". In progression of the game, there are aspects that the player must attempt to avoid, such as "super bats" that may bring the player around the map, and bottomless pits. The player has "crooked arrows" which can be used to kill bats and the Wumpus. The game is won by the player shooting the Wumpus and is lost if the player is eaten by the Wumpus, falls into the bottomless pit, or has no more arrows.

To extend this base implementation of the game, further additional features were implemented. The most major of them being the implementation of a graphical user interface to interact with the program, so the game can be played as either a text-based version or using a Graphical User Interface.

Including this, the number of Wumpuses have been doubled, and the Wumpus has the ability to move (When the player smells the Wumpus, but misses shot), both to add an extra element of difficulty to the game. There is also a feature where there are arrows scattered around the map, and the player can pick these arrows up if they come across a cave that contains these arrows. The game will therefore not end if the player runs out of arrows, while there are still uncollected arrows in the cave system.

Major Design Decisions

Script Design

For classes, it was decided that entities that was cause actions themselves, would be required to have individual classes that would interact with other classes.

The Main class is what holds the main method and has an instance of every other class to be used. This is where the game is run, and all the logic is called from other classes.

The Game class is used to hold the methods that affect the gameplay and the aspects of the game that are not actions by other entities within the game. This includes features such as checking whether the player is near any of the warnings and displaying the warnings on screen.

The Player class is designed to hold all the methods that correlates to the actions of the player, that can influence the game, such the shootArrow method to contain the code logic for how the player shoots an arrow.

This is a similar way to how and why the Wumpus and the Bats classes have been implemented, as these are different entities, with their own actions that can influence the game. A Wumpus class was also necessary for the fact of having 2 Wumpuses, which would have its own individual location and death status.

In the game structure, various classes are assigned to entities responsible for actions. The Main class, housing the main method, orchestrates the game by utilizing instances of other classes.

The Game class manages gameplay-related methods and non-entity actions. It handles tasks like checking player proximity to warnings and displaying them on screen.

The Player class encapsulates player actions, influencing the game dynamics, including the shootArrow method for arrow-shooting logic.

Similarly, Wumpus and Bats classes represent distinct entities with their unique actions affecting the game. Multiple Wumpuses necessitated a dedicated Wumpus class to manage individual locations and status.

Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876

DrawMap

The DrawMap class is made for drawing the game map for graphical version of this game.

Key features of the DrawMap class include:

1. Constructor:

- Accepts a Renderer instance to work with.

2. Drawing Caves:

- Reads cave data from a JSON file ("NodeGenerator/dodecahedronCorrected.json").
- Iterates over the nodes in the JSON file, extracting information such as id, x, and y coordinates.
- Uses the Renderer to set the player's location and coordinates for each cave.

3. Drawing Edges:

- Reads edge data from the same JSON file.
- Iterates over the links in the JSON file, extracting source and target ids.
- Uses the Renderer to set line indexes for connecting edges.

4. Drawing the Map:

- Invokes the methods to draw edges and caves, facilitating the creation of the complete cave map.

5. Resetting the Map:

- Provides a method (resetMap) to clear the Renderer's stored coordinates and line indexes, enabling a clean slate for redrawing.

Tests

The Tests class incorporates UI tests for essential game functions. Due to its late addition in the production phase, some features became challenging to test using UI tests.

Renderer

The Renderer class facilitates GUI interactions, offering methods for rendering graphics and UI elements. Key features include:

Initialization and Singleton Design:
- Follows a singleton pattern (getInstance method).
- Initializes a JFrame for the GUI.

Circle and Line Drawing:
- Provides methods (drawCircles and drawLines) for rendering circles with labels and connecting lines.

Coordinate and Line Index Management:
- Methods (setCircleCoordinates and setLineIndexes) manage circle coordinates and line indexes.

UI Element Configuration:
- Methods (setLabel, setButton, setTextField) add and configure JLabels, JButtons, and JTextFields on the GUI.

UI Cleanup and Closure:
- Methods (erase, clearCoordinates, clearLineIndexes, windowClose) clear UI elements and close the window.

Painting Component:
- Overrides paintComponent for custom painting, drawing circles and lines.

Dynamic Label Removal:
- Method (removeLabel) dynamically removes JLabels from the GUI.

Focus Handling for TextFields:
- Configures text fields with focus listeners for managing placeholder text.

Button ActionListener Setup:
- Method (setButton) adds buttons with associated ActionListeners.

Magic Code Warning:
- A comment warns against altering specific code labeled as “MAGIC CODE”.

Developers can enhance UIs by integrating graphics, labels, buttons, and text fields using the Renderer class.

GameStarter

The GameStarter class provides introductory instructions and a textual representation of the game map. The starterInstructions method takes a parameter, showOrNot, to determine whether to display the instructions and map.

Key features of the GameStarter class include:

Introduction Text:

- A multi-line introductory text (textStarter) describing the player's role as a hunter in the caves.

Game Map:

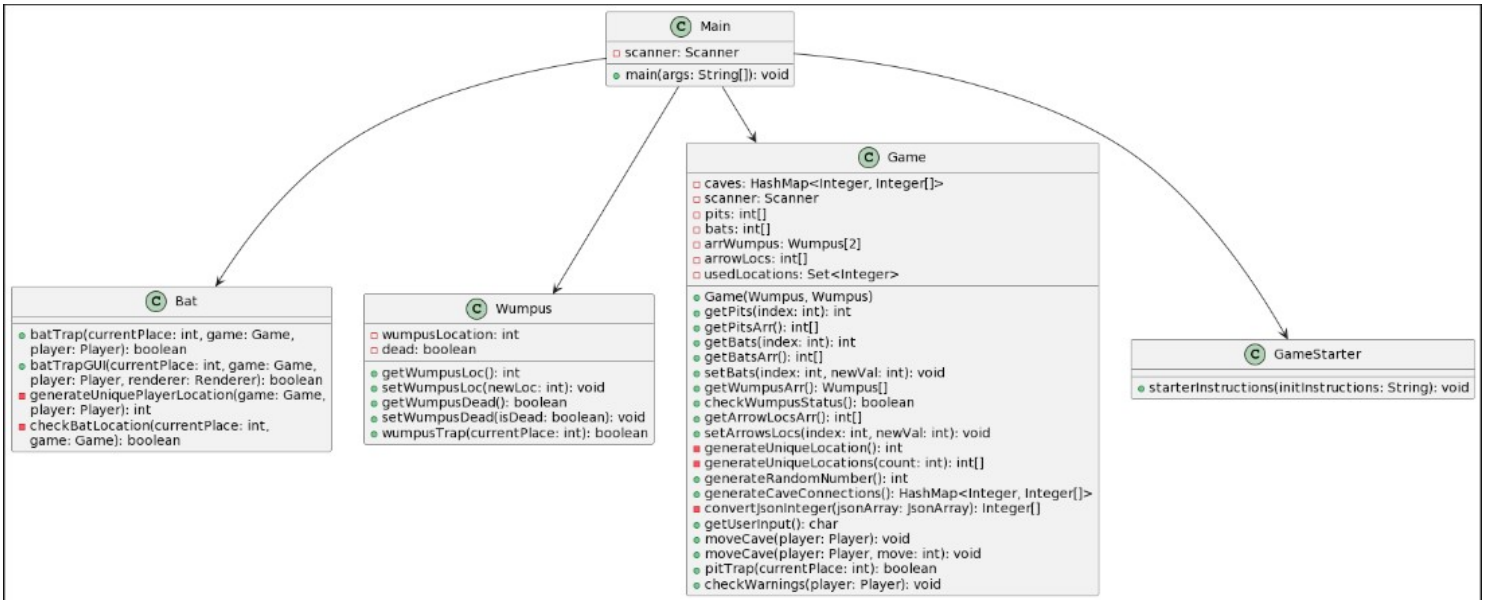
- A visual representation of the game map (map) with numbered rooms, tunnels, and hazard locations.

Conditional Display:

- The starterInstructions method displays the instructions and map only if the showOrNot parameter is set to "Y," "y," or "Yes."

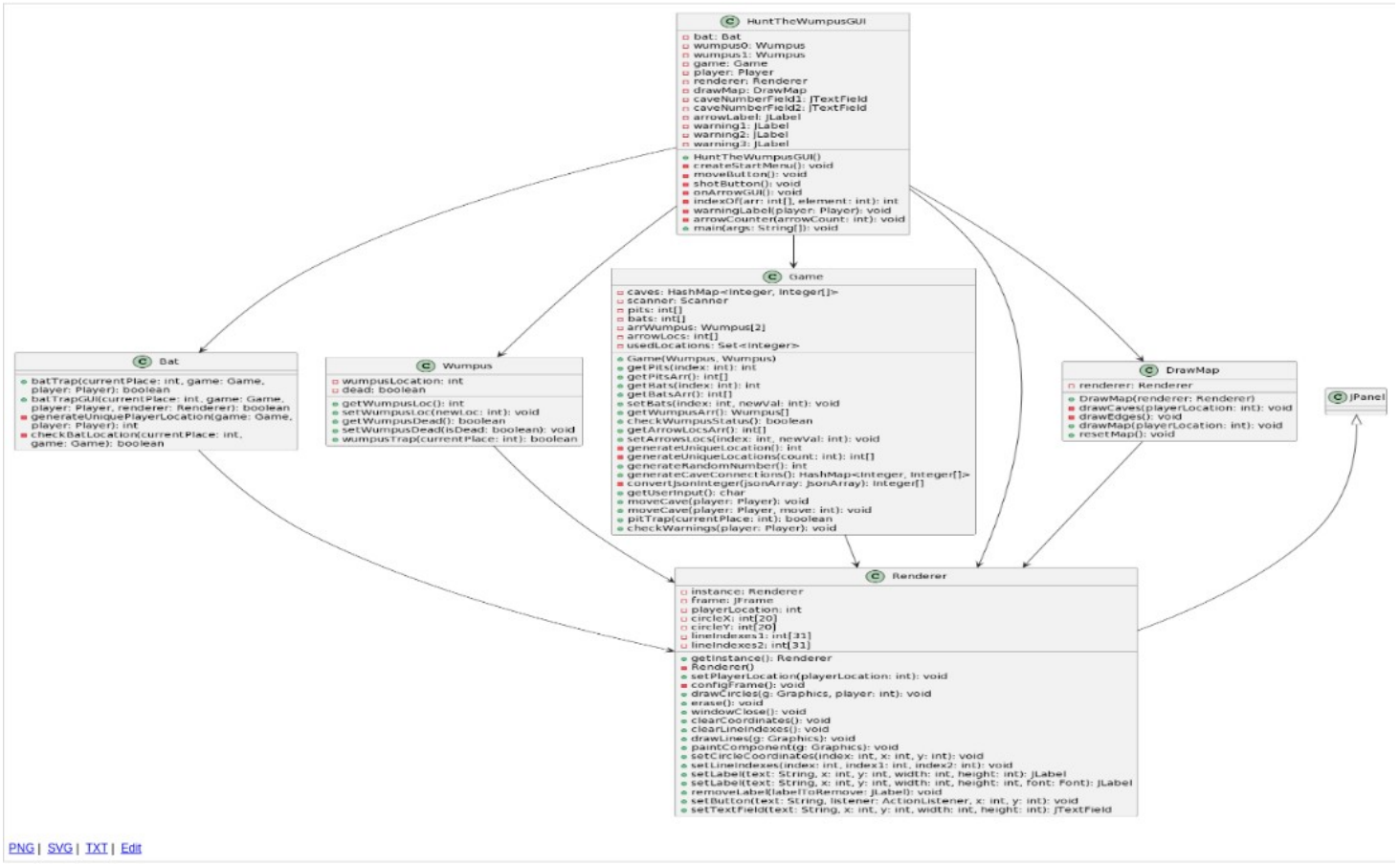
Developers can use the GameStarter class to provide players with essential information and an overview of the game environment before starting the gameplay.

Below is the UML table created to represent the final program. One UML program is for the text-based implementation of the game, while the second is to represent the necessary classes for the Graphical User Interface for the game.



Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876

Text based Hunt the Wumpus UML



GUI Based Hunt the Wumpus UML

Game Logic

The JSON file was used to store the cave connections and their locations generated by a Python script, NodeGenerator.py. The script utilizes the NetworkX library to create a dodecahedral graph representing the cave system in the game. The script then calculates and assigns positions to each node in the graph based on a specified structure.

Here's an overview of the Python script's functionality:

NetworkX Graph Generation:

- The script generates a dodecahedral graph (G) using NetworkX, representing the interconnected caves in the game.

Angular Offset Calculation:

- The script divides the circular layout into five parts, starting at 90 degrees. It iterates through four rings, each with a specified angular offset and set of nodes.

Node Position Computation:

- The script calculates the x and y coordinates for each node in the graph based on the ring structure, angular offset, and radii.

Node Attributes Assignment:

- The calculated x and y coordinates are assigned as attributes to the nodes in the graph.

Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876

Graph Visualization:

- The script visualizes the graph using Matplotlib, coloring nodes and edges, and annotating each node with its corresponding number.

JSON Serialization:

- Node positions and connections are serialized into a JSON format using the node_link_data method from NetworkX.
- The script adds a 'connections' key to the JSON data, storing the neighbours of each node.

File Saving:

- The script saves the generated JSON data to a file named 'dodecahedron.json'.
- Additionally, it saves a visual representation of the graph as a PNG file ('dodecahedron.png').
- Developers can use the generated JSON file to load the cave connections and positions in the game logic, allowing for dynamic and flexible cave configurations.

A major design choice that used was to represent the locations of the entities within the game in an array, with a different array for each separate entity. Therefore, in order to represent that an entity is no longer in a game (Wumpus or bat being killed), the location of said entity would be set to -1, as the entity would no longer “have a location”. Therefore, it will not interfere with any of the game continuation process, and in some cases, is used to define whether the game has been won (both Wumpus location is -1).

In reference to how the game works, it is all one large while loop, that runs as long as the attribute runGame remains true. Each run of the game is one loop of the Main class. This method of dealing with the game allows there to be a possibly infinite number of goes, until the game is won or lost.

Bugs or Defects

HuntTheWumpusGUI.java:

In HuntTheWumpusGUI.java, there are structural issues that deviate from the preferred design principles seen in Main.java. Due to time constraints, the class lacks proper refactoring and structuring.

Particularly, the warningLabel(Player player) method exhibits suboptimal coding practices with excessive nested if statements, compromising readability and maintainability. Refactoring this method to enhance clarity and adherence to coding standards would be beneficial for long-term code quality.

Test.java:

Tests.java file which contains unit tests use reflection elsewhere which is not a good practise, and this decision has been made only because of brief time frames we had for our project. Ideally, we should have refactored our code so we can test it properly without getting to use that java feature.

Renderer.java:

In the Renderer file, the number of circles and edges is currently hardcoded due to limitations in dynamically obtaining this information from a JSON file. Instead of dynamically determining the length of object arrays from the JSON file and passing that length, a fixed count of 20 nodes and 30 edges has been specified in the code. This approach simplifies the implementation by providing a predetermined structure for the graphical elements.

Version Control Explanation

The version control used to produce this project was a private repository on GitHub, where both members were added as contributors to the repository. The link to the repository is below:

https://github.com/Dod900ls1/HuntTheWumpus

The method to how the workflow was split is by classes and methods. The idea of functional abstraction was used, where it was confirmed what values and the datatypes of these were to be inputted and the return values were

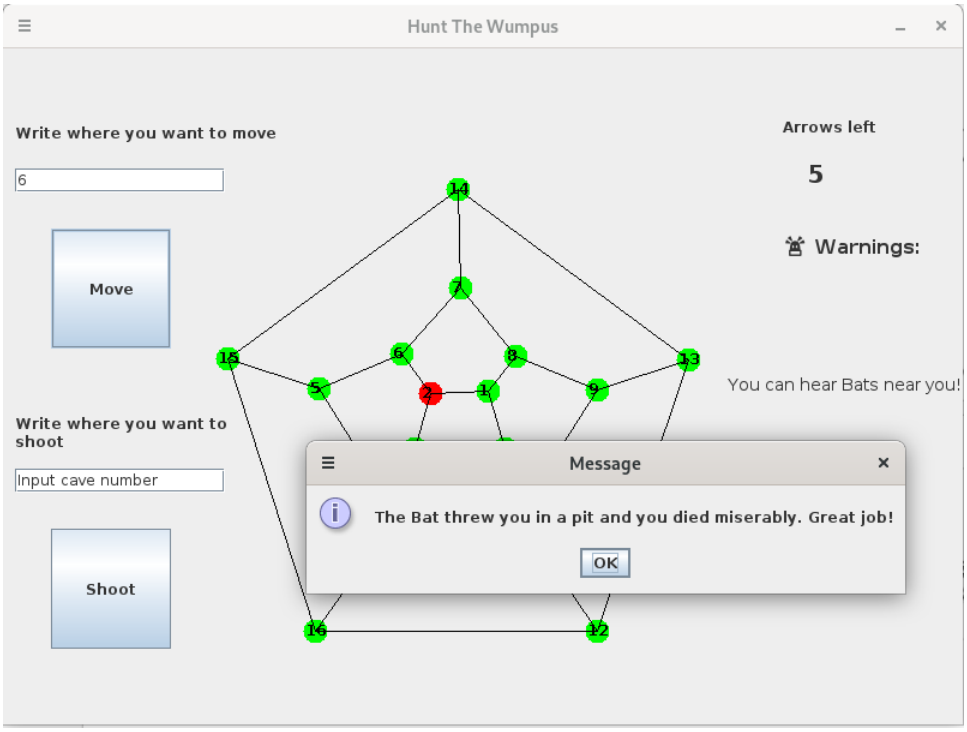
confirmed beforehand, while the actual logic behind how the method is implemented is not needed to be known by the other member. This allowed for seamless code development, where which part of the code to work on for each member was determined. The usage of Docstrings also helped to know about the input types and output types of each method that was to be used, without needing to read through the code thoroughly.

Manual and Unit Tests

Manual Tests

Graphical-Based Implementation

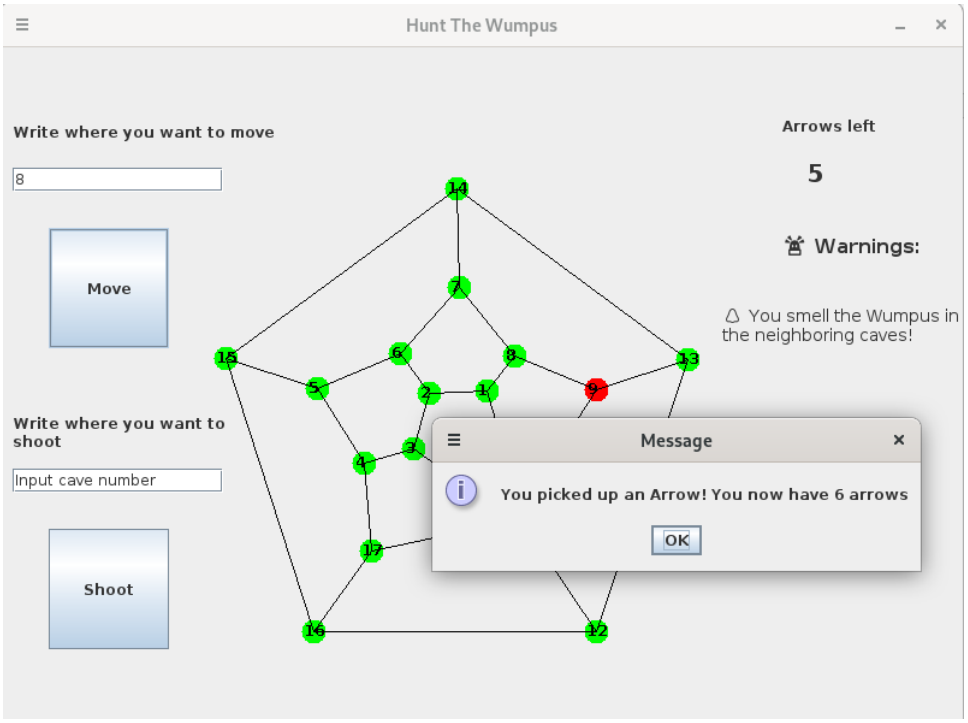
Case 1: Bat throw you in a pit.



Word as it supposed to work, program terminates after user presses OK button on a pop-up window.

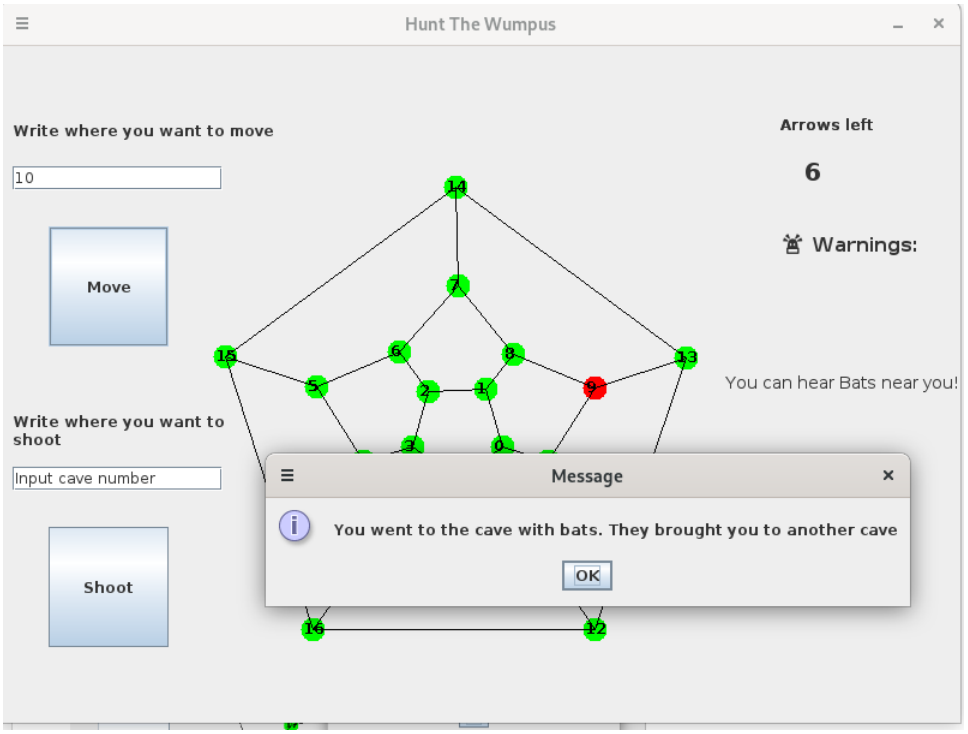
Case 2: Player picking up an arrow

Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876



Work as it is supposed to work, arrow counter shows 6 after user presses OK on a pop-up window.

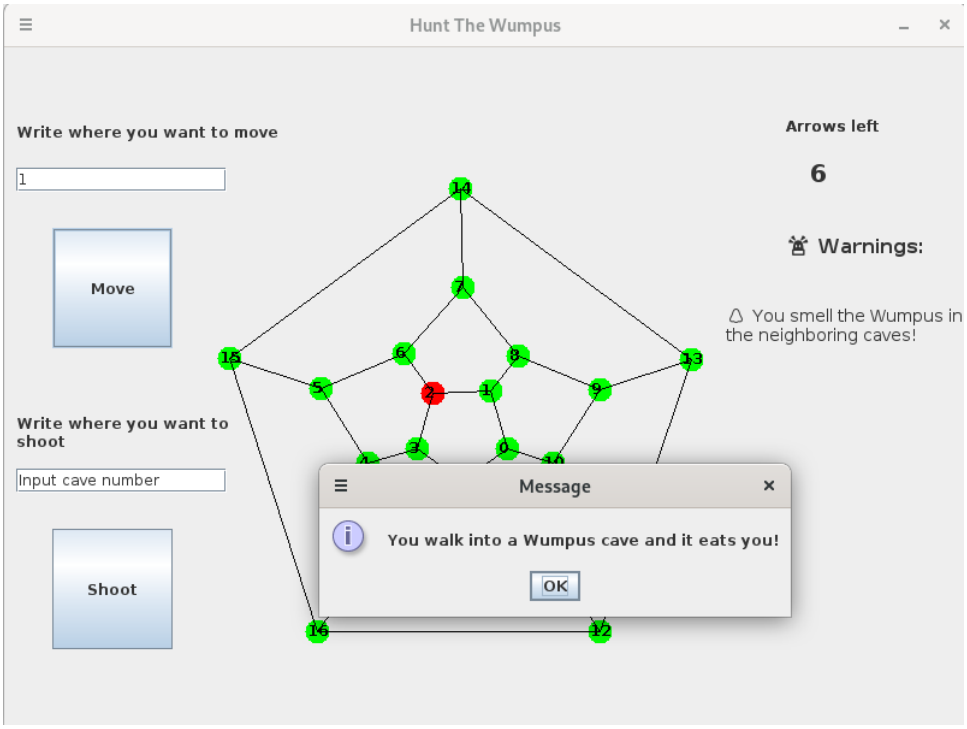
Case 3: Bat brings player to another cave



Work as it supposed to. Player location changes after they press OK button on pop-up window.

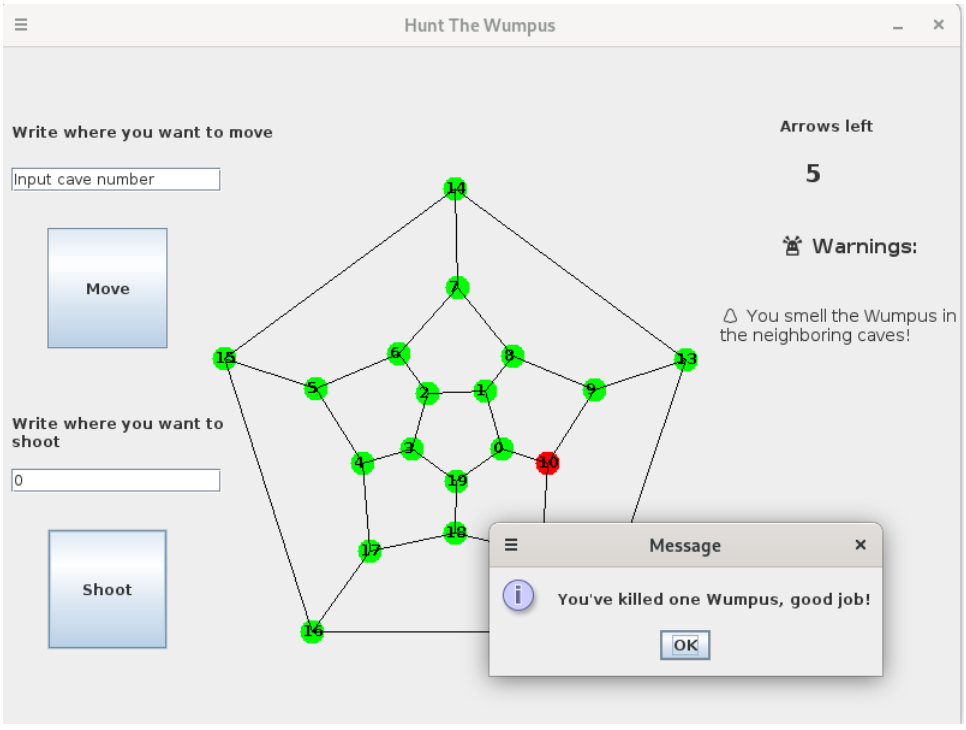
Case 4: Wumpus eat the player

Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876

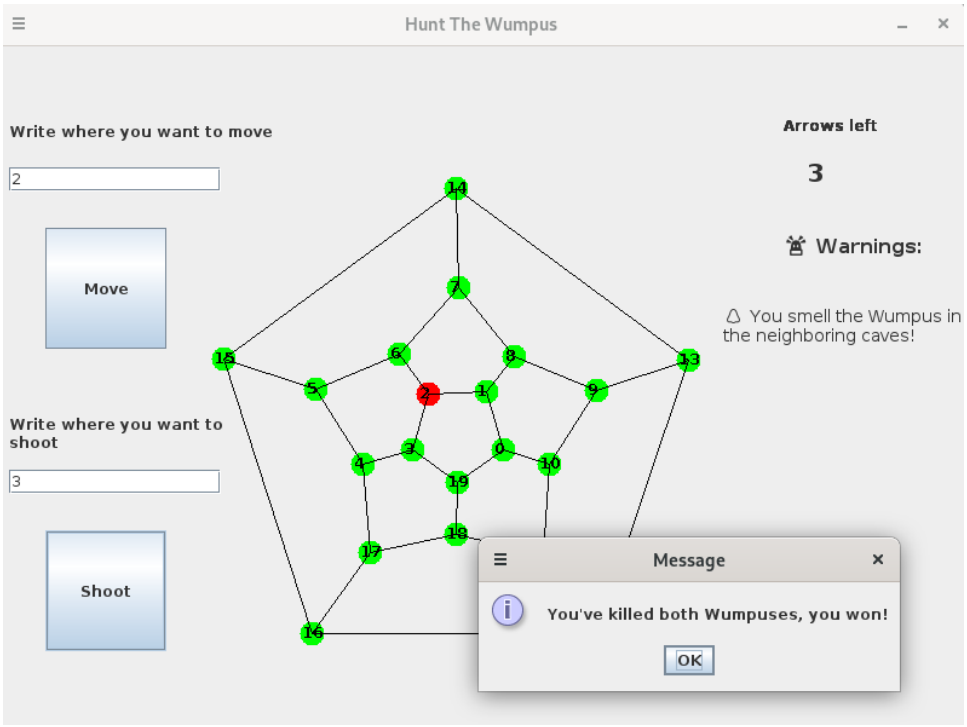


Work as it supposed to. Program terminates after player presses the OK button on a pop-up window.

Case 5: Player kills two Wumpuses



Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876



Work as it supposed to. Program terminates after player wins.

All basic features work.

Text-Based Implementation

Below is the gameplay testing of the Text-Based Implementation of the Game:

Gameplay 1 - Fall into a pit:

```
Do you want to see instructions and a game map? (Y/N)
N
You're in the cave number, 11.
Available caves are: 10, 12, 18.

WARNING:
You can feel the blowing of wind. Pit is near you!

Shoot = S, Walk to another cave = W
W

You're in the cave number, 11.
Available caves are: 10, 12, 18.
10

HAHAHA! YOU FALL IN A PIT AND DIE!
```

Result: Works as Intended

Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876

Gameplay 2 - Get Moved by Bat + Pick Up Arrow:

```
Do you want to see instructions and a game map? (Y/N)
N
You're in the cave number, 1.

Available caves are: 0, 2, 8.
You picked up an Arrow! You now have 6 arrows.

WARNING:
You can hear Bats near you!

Shoot = S, Walk to another cave = W
W
You're in the cave number, 1.
Available caves are: 0, 2, 8.
8

You have gone to the cave with bats. They brought you to the cave number 1.

---------------------------------------------------------------------

N
You're in the cave number, 1.

Available caves are: 0, 2, 8.

WARNING:
You can hear Bats near you!

Shoot = S, Walk to another cave = W
W
You're in the cave number, 1.
Available caves are: 0, 2, 8.
8

Shoot = S, Walk to another cave = W

---------------------------------------------------------------------
```

Result: Works as Intended.

Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876

Gameplay 3 - Wumpus Double Kill (Win Game)

```
Do you want to see instructions and a game map? (Y/N)
N

You're in the cave number, 18.
Available caves are: 17, 19, 11.
Shoot = S, Walk to another cave = W
W

You're in the cave number, 18.
Available caves are: 17, 19, 11.
17
------------------------------------------------------------------------
You're in the cave number, 17.
...
...
...

You're in the cave number, 4.
Available caves are: 3, 5, 17.

WARNING:
You smell the Wumpus in one of neighbour caves!

Shoot = S, Walk to another cave = W
S
You can shoot in caves number 3, 5, 17
5
You've killed one of the Wumpus!
------------------------------------------------------------------------
You're in the cave number, 4.
...
...
...

You're in the cave number, 8.
Available caves are: 7, 9, 1.

WARNING:
You smell the Wumpus in one of neighbour caves!

Shoot = S, Walk to another cave = W
S
You can shoot in caves number 7, 9, 1
9

You've killed one of the Wumpus!
You've killed both Wumpuses. Very well done!
------------------------------------------------------------------------
```

Result: Works as Intended. (... used to signify large portion of unneeded game logic)

Gameplay 4 - Wumpus eats Player:

```
Do you want to see instructions and a game map? (Y/N)
N

You're in the cave number, 18.
Available caves are: 17, 19, 11.
Shoot = S, Walk to another cave = W
W

You're in the cave number, 18.
Available caves are: 17, 19, 11.
17
------------------------------------------------------------------------
You're in the cave number, 17.
```

- Uses reflection to set the Wumpus location.
- Invokes the private method "killWumpus" from the Player class.
- Asserts that the result is false, indicating the successful kill.

2. testDontKillWumpus:
- Similar to the first test but with an empty assertion message.

3. testKillBat:
- Checks if the player can kill a bat.
- Uses reflection to set the bats' locations.
- Invokes the private method "killBat" from the Player class.
- Asserts that at least one bat has been killed.

4. playerFinishItsArrows:
- Checks if the player loses the game when running out of arrows.
- Uses reflection to set the player's arrow count.
- Invokes the private method "arrowCounter" from the Player class.
- Asserts that the result is false, indicating the loss.

5. testScareWumpus:
- Verifies if the Wumpus relocates after the player shoots near it.
- Uses reflection to set the Wumpus location.
- Invokes the public method "scareWumpus" from the Wumpus class.
- Asserts different outcomes based on the method result.

6. testWalking_ValidUserInput:
- Checks if the player can move to a valid cave.
- Sets up a game and player using reflection.
- Invokes the private method "isRightStep" from the Player class.
- Asserts that the result is true.

7. testWalking_InvalidUserInput:
- Validates that the player cannot move to an inaccessible cave.
- Sets up a game and player using reflection.
- Invokes the private method "isRightStep" from the Player class.
- Asserts that the result is false.

8. testPitTrap_PlayerFallsInPit:
- Verifies that the player loses the game when falling into a pit.
- Uses reflection to set the pit locations.
- Invokes the public method "pitTrap" from the Game class.
- Asserts that the result is false.

Student ID: 230026622 THIS IS A GROUP SUBMISSION Partner ID:230022876

9. testPitTrap_PlayerDoesNotFallInPit:
   - Checks that the player does not lose the game when not falling into a pit.
   - Uses reflection to set the pit locations.
   - Invokes the public method "pitTrap" from the Game class.
   - Asserts that the result is true.

10. testWumpusTrap_PlayerEntersWumpus:
   - Verifies that the player dies when entering a cave with the Wumpus.
   - Uses reflection to set the Wumpus location.
   - Invokes the public method "WumpusTrap" from the Wumpus class.
   - Asserts that the result is false.

We also made some tests manually. We have checked that player can find an arrow in some caves and that they can pick that arrow in both txt and GUI versions. We also ensured that all basic features such as walking, shooting, and dying from all possible reasons works on both versions.

Additional Features

To expand on the base game required by the Course Specification, there were more additional features added to the development of the game.

One such feature was the addition of two Wumpuses to the game in comparison to the original one Wumpus. The reason for this was that the game would be much more interesting to play, with the threat of being eaten (and so losing the game) being doubled. This would result in more riveting gameplay and had to be implemented by created two instances of the Wumpus and changing certain gameplay decisions (such as when you win), to accommodate the fact that there are two Wumpuses (such as only allowing a game to be won, if both Wumpuses have been killed).

To add another degree of difficulty to the game, the Wumpus was given the ability to move around. In specifics, if the player shot an arrow when the Wumpus warning was shown on screen, and the arrow misses, the Wumpus would hear an arrow, and relocate to another cave. This relocation mechanic also included the Wumpus being able to relocate to the players current cave, adding another element of danger to where the player chooses to shoot their arrow.

Due to having these two Wumpuses and its newfound ability to move, we concluded that the preset number of arrows, 5, would be far too little to be able to play this game with a degree of comfort. Due to this we decided to scatter some arrows around the caves, which the player can collect and use to kill the Wumpuses and win the game. This led to minor changes in a game ending scenario. Now the player can run out of arrows, but if there are still arrows left uncollected in the cave system, the game will not end, allowing the player still with the chance to win the game.

To enhance the player's experience, we decided to implement a graphical user interface (GUI) for our Hunt the Wumpus game. Three additional classes were introduced to our basic implementation, contributing to improved scalability and readability for other developers. These classes, designed for GUI functionality, played a key role in creating a more engaging and user-friendly gaming environment."

Individual Contribution

I was responsible for creating the GUI, specifically working on the Renderer.java, DrawMap.java, and HuntTheWumpusGUI.java files. Additionally, I assisted my partner in developing the UI for a text-based version, which we implemented in the GameStarter.java file. I also identified and corrected some code in the NodeGenerator.py file, focusing on automatically generating nodes, edges, numbers, and their coordinate locations in a JSON file.

In the initial stages, when it was uncertain whether we would be working in pairs, I developed a basic version of the game featuring a moving Wumpus. However, the structure was not optimal, with everything confined to just two files and lacking clear organization. Consequently, my partner undertook the task of refactoring and improving the structure.

Conclusion

Overall, the game produced is a successful implementation of the original “Hunt the Wumpus program”, with several aspects of additional functionality that has multiple different difficulty levels.

In terms of how things could have been improved, there could have been a method to incorporate both the GUI implementation of the game and the text-based implementation of the game in the same class, allowing one singular class’s main method needing to be called (and the choice of which implementation to be chosen from there), rather than calling the main method of the separate class depending on which implementation of the game to play. This would decrease the any amount of confusion in the running of the game for the user, while allowing a seamless transition between both the text-based implementation of the game, and the GUI implementation of the game, enhancing the player experience.

From this development process, the idea of functional and procedural abstraction was developed and understood, with the idea that two developers don’t need to know how the other developer’s program works if the return type and arguments passed in are explicitly stated and confirmed. This allows for a more seamless workflow, where both developers can work at their own pace and independently, while the development of code continues to occur.

Bibliography

Java docs: https://docs.oracle.com/javase%2F7%2Fdocs%2Fapi%2F%2F/javax/swing/package-summary.html

Button example: https://www.javatpoint.com/java-jbutton

How to Write an Action Listener: https://docs.oracle.com/javase%2Ftutorial%2Fuiswing%2F%2F/events/actionlistener.html#:~:text=We%20can%20do%20this%20by,the%20action%20listener's%20actionPerformed%20method.

Ascii paint: https://github.com/Herobread/ascii-paint

UML diagram generator: https://www.planttext.com/

Singleton pattern: https://www.baeldung.com/java-singleton

Singleton pattern2: https://www.youtube.com/watch?v=KUTqnWswPV4