160006666 & 160009761 Practical 1 Wumpus Wednesday Feb 15

Requirements

Practical One, Hunt the Wumpus, had to include: one adventurer, one Wumpus, one exit, one treasure, a chosen number of pits, and a chosen number of superbats. The locations of the elements had to be random every time there was a new game. The basic objective included collecting the treasure and then continuing to the exit, which would only appear if the treasure was indeed in the adventurer's possession. The Wumpus did not have to be killed to win for the basic specification.

System Overview

For Hunt the Wumpus, the adventurer/player is in a cave system which is represented by a 20 x 20 grid and acts like a torus. This means the top edge is the same as the bottom edge and the left edge is identical to the right edge. On this grid, the player can move north, south, east or west. The player hunts the Wumpus using a bow and arrows. If the player misses the Wumpus when they shoot an arrow, it startles the Wumpus which moves to an adjacent cave. If they successfully hit the Wumpus, then the it perishes. There is no limit on arrows for the basic specification. If an arrow is shot, the arrow only travels one unit in any direction. If the player runs into the Wumpus, then the player dies and game over. The game also includes hazards. First, bottomless pits are randomly located death traps. If the player falls in, game over. A Wumpus has sucker feet which allows it to survive in a bottomless pit. Second, Superbats are also randomly distributed throughout the cave system. If a player meets a Superbat, it will randomly deposit the player at a different location in the caves. There are several rules on how the elements on the game interact. If the player is in an adjacent cave to a Wumpus, the player is notified by the smell of the Wumpus. If the player is adjacent to a bottomless pit, then the player can feel a breeze. If the player adjacent to the treasure, then they can see a shiny glitteringness. There is no notification if a Superbat is nearby. The player wins the game by collecting the treasure and reaching the exit without dying.

Design:

The Treasure, Exit, and Wumpus classes all extend Player because they need to access and use Player's attributes and methods when determining how to interact with Player. For example, the treasure needs to notify the adventurer with "shiny glitteringness" when it is adjacent, so it makes sense to be able to have Player's fields be visible to the Treasure class.

Treasure, Exit, and Wumpus classes all start with static variables x and y to represent their x and y coordinates in the torus cave system. The variables are static because since there is only one exit, one treasure, and one wumpus in the game, there needs to only be one instance for the entire class. It also made it easier to access the x and y fields from other classes because since there is only one x and y per class you did not have to worry about a specific instance. These static variables were randomly generated and were used to set a random location for the wumpus, exit, and treasure.

Treasure, Exit, and Wumpus also each had a setLocation() method. This was used to drop the different elements on the board in wumpusGame. The location of the treasure, exit, and wumpus was only dropped if the space that was randomly generated didn't contain another element. We achieved this by using a while loop with or statements to determine if that space on the game board was already filled. If it was already taken, then x and y would be randomly generated again. If not, then the elements would be dropped. For Exit, there was an extra if else conditional statement thrown in there, which allowed the player only to exit if they had killed the wumpus and collected the treasure.

All three of these classes, Treasure, Exit, and Wumpus contained getter methods which was helpful in accessing their x and y coordinates when implementing the percepts and notification methods. The classes also contained notification methods called notify() which took a 2D array parameter which was meant to represent the board of the game. The notify methods were written to help notify the player if they were adjacent to a element, and is ran in the Player class after every move. If the Player is adjacent to one of the elements that is supposed to notify, then a print statement will tell the player which element in nearby. If not, there is no notification.

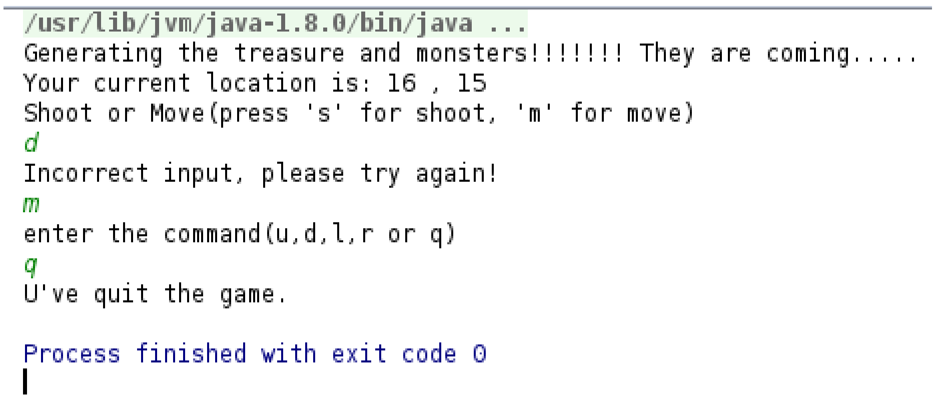
WumpusGame is the class that runs the entire game. It starts by declaring the game board, which we decided to make a 2D array with the size of 20 by 20. We found that a 2D array was a easy way to represent a grid. The class then sets the location of the elements using setLcoation() and prints the starting location for the player because it is randomly generated. We dropped the elements in this order and you can see in the setLocation() methods of each class how we avoid dropping one element on top of another, which I explained above and are in the comments. Finally, enterCommand() which is created in the player class runs Hunt the Wumpus until the player dies or chooses to exit the game uses the command “q”.

Please see my partner’s report for her design portion.

Testing

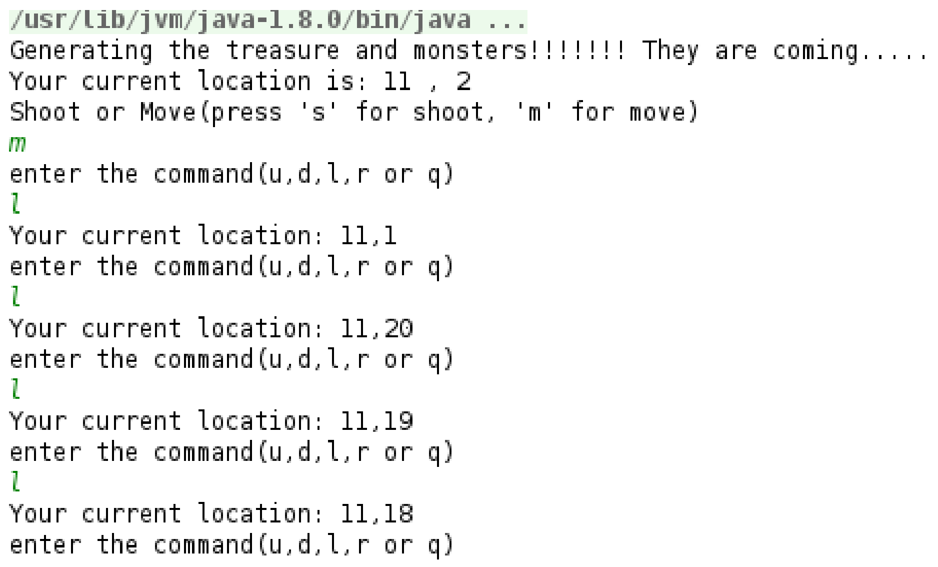
For the testing process of our game, we found that the best way to test our program was to simply play it and make sure that all the elements interacted with each other in the properly.

The first test we ran was to see if the game would quit if the player decided that did not want to play. The player had to type in “q” to quit the game. Our test was successful and did exit the game per request of the user. This test also dealt with making sure the user input was valid, if else the program would tell t he user to input another and valid command. This ran as expected.

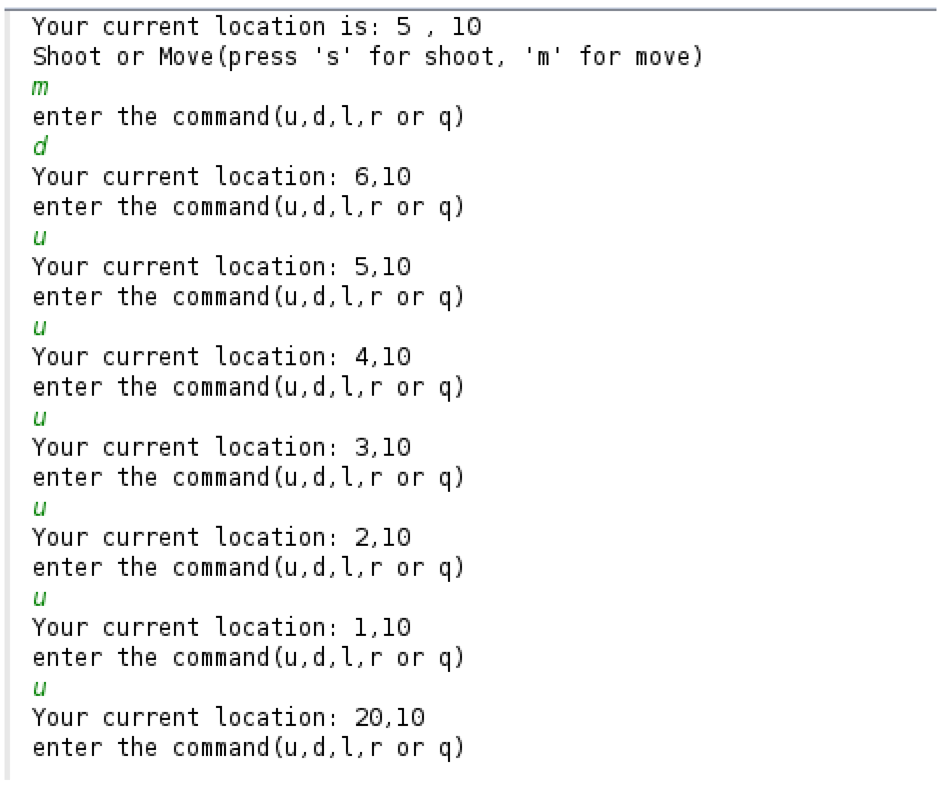


```
/usr/lib/jvm/java-1.8.0/bin/java ...
Generating the treasure and monsters!!!!!!! They are coming.....
Your current location is: 16 , 15
Shoot or Move(press 's' for shoot, 'm' for move)
d
Incorrect input, please try again!
m
enter the command(u,d,l,r or q)
q
U've quit the game.

Process finished with exit code 0
```

The second test made sure that our grid did indeed act like a torus. We tested this by going to the edge of the board and finding that it did indeed act like the top of the board was the same as the bottom and the left edge was the same as the right.

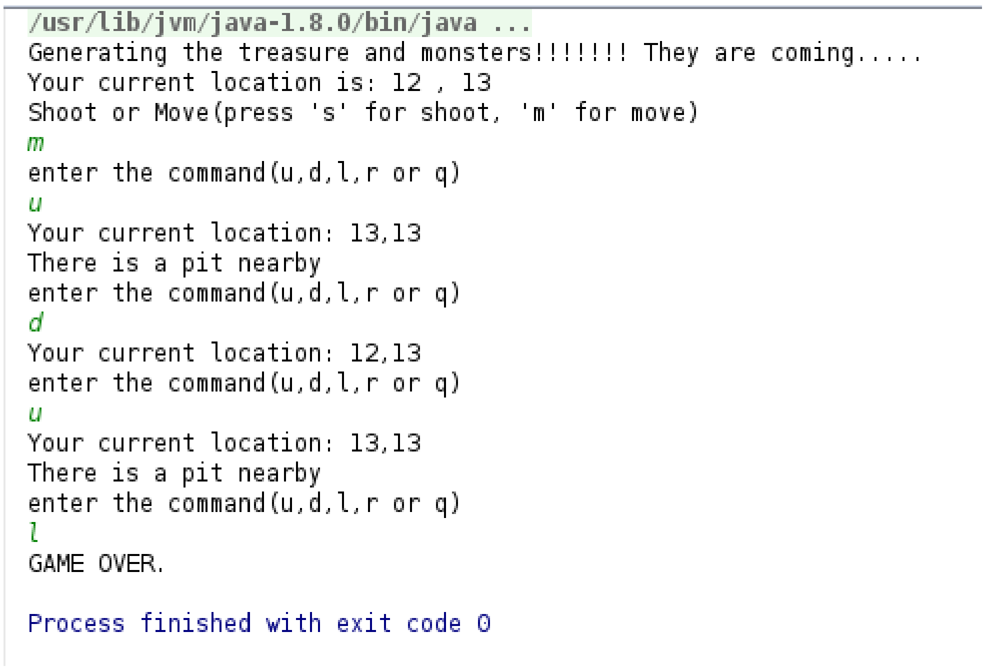


```
/usr/lib/jvm/java-1.8.0/bin/java ...
Generating the treasure and monsters!!!!!!! They are coming.....
Your current location is: 11 , 2
Shoot or Move(press 's' for shoot, 'm' for move)
m
enter the command(u,d,l,r or q)
l
Your current location: 11,1
enter the command(u,d,l,r or q)
l
Your current location: 11,20
enter the command(u,d,l,r or q)
l
Your current location: 11,19
enter the command(u,d,l,r or q)
l
Your current location: 11,18
enter the command(u,d,l,r or q)
```



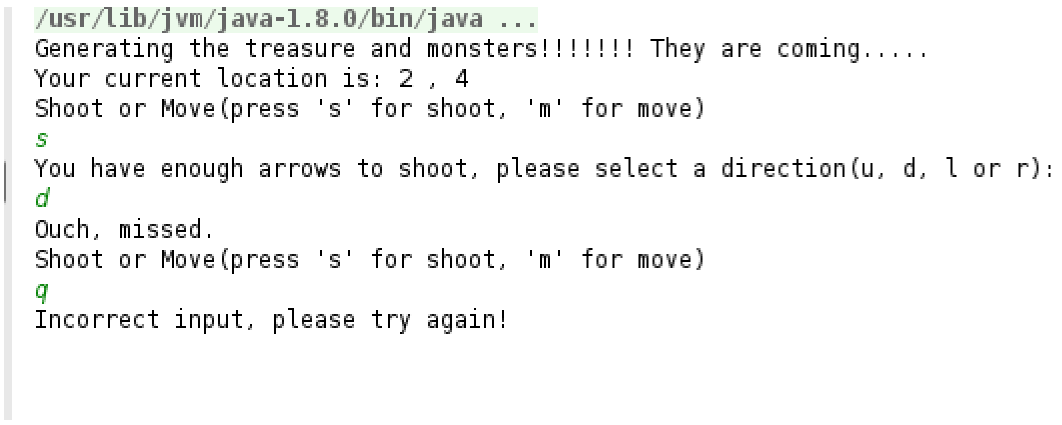
```
Your current location is: 5 , 10
Shoot or Move(press 's' for shoot, 'm' for move)
m
enter the command(u,d,l,r or q)
d
Your current location: 6,10
enter the command(u,d,l,r or q)
u
Your current location: 5,10
enter the command(u,d,l,r or q)
u
Your current location: 4,10
enter the command(u,d,l,r or q)
u
Your current location: 3,10
enter the command(u,d,l,r or q)
u
Your current location: 2,10
enter the command(u,d,l,r or q)
u
Your current location: 1,10
enter the command(u,d,l,r or q)
u
Your current location: 20,10
enter the command(u,d,l,r or q)
```

The third test we ran was to see if the percepts worked. We ran the game enough to run into one of the hazards. We then wanted to make sure that the player would die and game over under the right conditions. This was also successful as the player was adjacent to the pit, which notified the user, and then died when the player walked into the pit. We also did this



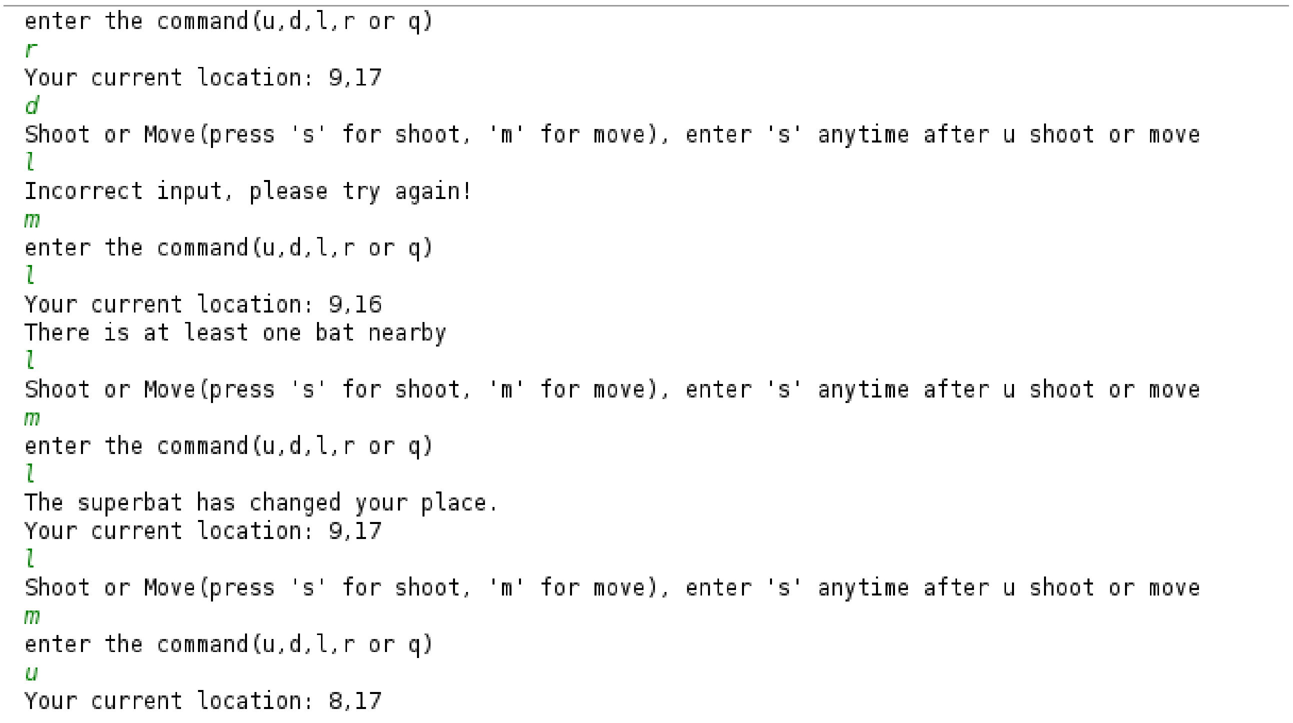
```
/usr/lib/jvm/java-1.8.0/bin/java ...
Generating the treasure and monsters!!!!!!! They are coming.....
Your current location is: 12 , 13
Shoot or Move(press 's' for shoot, 'm' for move)
m
enter the command(u,d,l,r or q)
u
Your current location: 13,13
There is a pit nearby
enter the command(u,d,l,r or q)
d
Your current location: 12,13
enter the command(u,d,l,r or q)
u
Your current location: 13,13
There is a pit nearby
enter the command(u,d,l,r or q)
l
GAME OVER.

Process finished with exit code 0
```

Our fourth test was to made sure that the player could shoot arrows.



```
/usr/lib/jvm/java-1.8.0/bin/java ...
Generating the treasure and monsters!!!!!!! They are coming.....
Your current location is: 2 , 4
Shoot or Move(press 's' for shoot, 'm' for move)
s
You have enough arrows to shoot, please select a direction(u, d, l or r):
d
Ouch, missed.
Shoot or Move(press 's' for shoot, 'm' for move)
q
Incorrect input, please try again!
```

Our fifth test had to do with the superbats to make sure that it put you in a random location if you ran into them.



```
enter the command(u,d,l,r or q)
r
Your current location: 9,17
d
Shoot or Move(press 's' for shoot, 'm' for move), enter 's' anytime after u shoot or move
l
Incorrect input, please try again!
m
enter the command(u,d,l,r or q)
l
Your current location: 9,16
There is at least one bat nearby
l
Shoot or Move(press 's' for shoot, 'm' for move), enter 's' anytime after u shoot or move
m
enter the command(u,d,l,r or q)
l
The superbat has changed your place.
Your current location: 9,17
l
Shoot or Move(press 's' for shoot, 'm' for move), enter 's' anytime after u shoot or move
m
enter the command(u,d,l,r or q)
u
Your current location: 8,17
```

Because we felt as though we interacted with most of the elements successfully, we suspended testing our code any further. If you notice small changes in the text of the game for the different screen shots its because we changed some stuff around, but this did not affect the success of the tests.

Critical Evaluation:

In order to achieve a intermediate deliverable we added a few elements to the basic specification. For example for the player we included a life span. The player only has a certain amount of moves to find the treasure and leave the cave successfully. We did this by creating a variable called countMoves, which is incremented up every time the player moves in a direction. We incremented the variable by placing it in the move direction methods. The countMoves variable would only increase if the input was valid. If the number of maximum moves was reached, then game over.

We also wanted to bring back the original point of the game and included the feature of having to kill the Wumpus and collect the treasure in order to win the game. We did this by using an if conditional statement in the setLocation method in the Exit class that only set the location of the exit if the treasure was collected and the wumpus was dead. We used booleans to see if these statements were true. SetLocation needed to be ran every time the player moved because we needed to continuously see an update on the status of treasure collection and the wumpus' life status

Individual Contribution:
Implemented the Exit, Superbats, pits, half of the player class and the main class. In the player part, have done the shooting, shootormove, and the torus system. Pass the value through the project. In the extension, I have done the status design and the limit number of arrows. Debug the entire program and do all the testing.

Have done the part that collect the input from the player and check the input validity.