Shiho Numakura

5/5/2018

CS 150

Professor Liew

Project 3: Maze Game

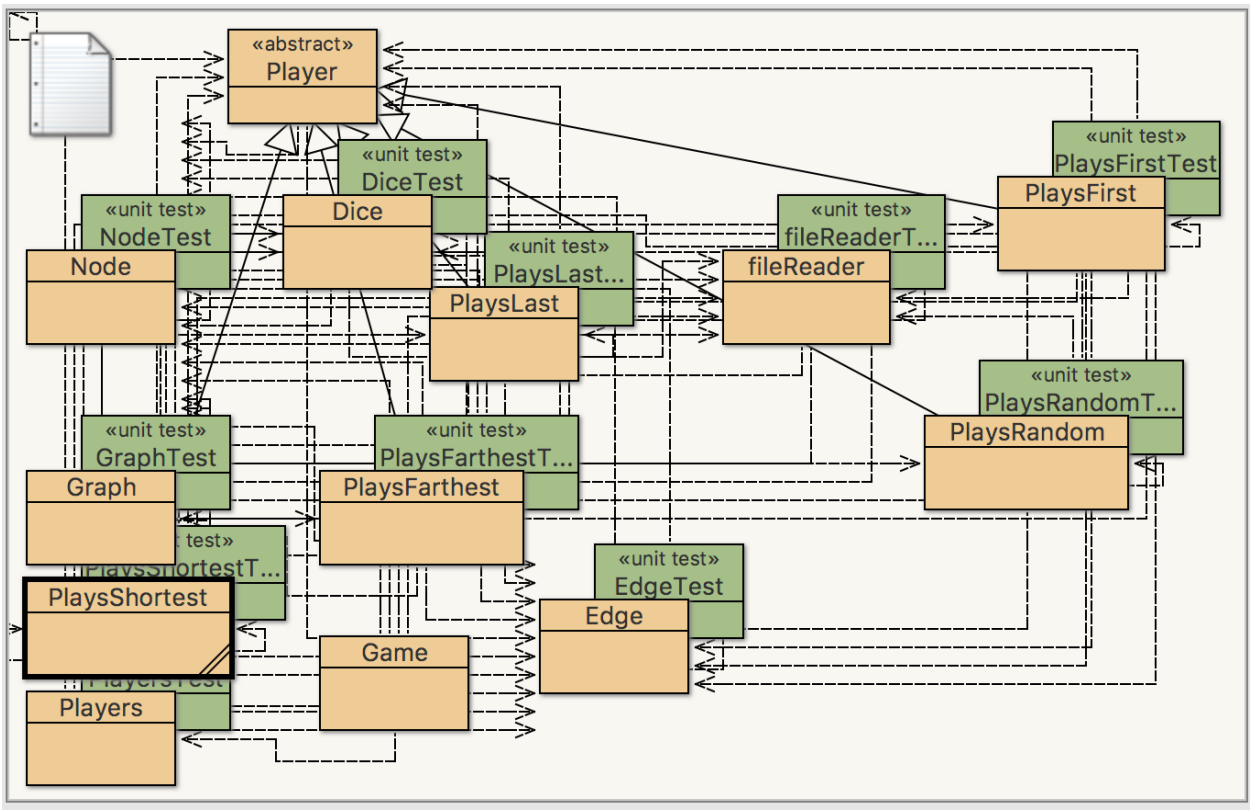
Introduction

This program will simulate a graph based maze game played by multiple players. Each player will be assigned a way to choose their direction in the game. There are nodes that connect to multiple edges which will direct to another node. Each edge will have different weights, representing the different steps that is required to take in order to reach a node. So, the different types of player will choose their edges differently. Each turn, the player will roll and dice and proceed to their direction. However, the first three types of player will proceed to move on to the exit node when the exit nodes are in their neighbor. The last two types of players will be slightly different. Using a given limit (in parameter), if the distance to closest path to an exit node using Dijkstra's algorithm (13.05.01 e-textbook) is within the limit, the fourth player will use the shortest path and the last player will use the longest path. The program will ask for the input for a prompt every round, unless it is commanded to continue without it. Throughout the game, the user will be able to see where all players are located. The goal for program is for one of the players to reach an exit node to finish the game.

Approach

To implement this program, there will be three stages: to create a graph, to create players. So, to first create the graph, we will need a class to read the input file, a class for the graph, a class for the nodes and edges in the graph. Then, since the players will all behave the same except for when they choose their edge, I created an abstract player class with five different

types of player class inheriting this abstract class. To create a wrapper class for the players, I also need to create a Player container class. I also needed another class to represent the dice, to see how many steps each player will take at that turn. Finally, to put all these classes together, I created an experimental controller class called Game class.



Approach; Figure 1. My Class Diagram.

As shown in Figure , I finalize my program with 13 classes. When the program is ran with the appropriate parameters, which are "the limit, the number of players, the max value for the dice, the graph description filename, a file with the starting node and the exit nodes"(C W Liew). Using these two input files, the fileReader class will go through the files creating a graph. My graph is implemented similarly to a LinkedList (LinkedList Class) where edges will direct to another node object. Although the edges are directed, there will always be an edge that will direct back from the directing node toward the current node. So once the first input file is

scanned through and successfully creates a Graph object, the second file will find the starting and exiting nodes in the graph.

In a Graph object, since we are restricted to use an Array, there is an ArrayList (ArrayList Class) which stores all the Nodes in the graph. Each Node class will store a node number which corresponds to the index +1 in the graph ArrayList. Each Node object will also store the edges that are directing out from the Node. Therefore, all Edge class will be stored in a Node and simply store the node that it is directing to and its weight. There will only be one Graph object that is created, for all players will be using the same graph to play the game.

Once my graph stage was completed, I began the Player abstract class. All players will all start from the given node, and store its current location on the graph in its instance variables, including its current node, current edge, and previous node. Since all players will behave the same when they are moving, there is an method that will change the location of the player each turn, while checking for its neighboring nodes to be an exit node. However, each player inheriting this class will have to implement its own algorithm to choose an edge when the player lands on a node. These classes include PlaysRandom class, PlaysFirst class, PlaysLast class, PlaysShortest class, and PlaysLongest class.

While the PlaysRandom object will choose its neighboring edges randomly when it lands on a node, the PlaysFirst object will choose its first option of edges. On the other hand, PlaysLast object will choose the last option of edges. As written before, these players will follow their given algorithm unless a exiting node is in their neighbor. PlaysShortest object will choose the least weighted edges from a node, and PlaysLongest objects will choose the most weighted edges. However, these players will be able to find existing nodes if the shortest path to them is within the limit given using the DijkStra's algorithm. All player class are restricted not to go to

the previous node that it came from, unless that is the only edge directing out. These players will be created with a specified starting node, and enter the graph from then. All these varying players in the game will all be stored into one Players object which will perform the turns on each players in the game.

Then, I created a Dice class which will give a random value every turn acting as a dice.

When all the classes were finalized, I created a Game class which will put all the objects needed for the game together. As it asks for the user for a prompt, the user will be given with four options: 'x' will exit the game, 'i' will continue a turn for all players, 'c' will stop asking the user for a prompt and run until the game is over. Finally, 'p' will print the current positions of the players.

Method

To obtain data, I used the same maze and since my dice uses a seed of the current time, it will generate different dice values every run. Using the results, I will see which player is most likely to win. I will then keep all parameters constant, but use different maze configurations. Doing so, I would like to observe the different behaviors with different mazes with varying maze size. Similarly, I will keep all parameters but change the starting nodes and the exiting nodes using the same maze and determine how it affects the result of the game. Lastly, I would like the obtain data with different maximum values that the dice can generate.

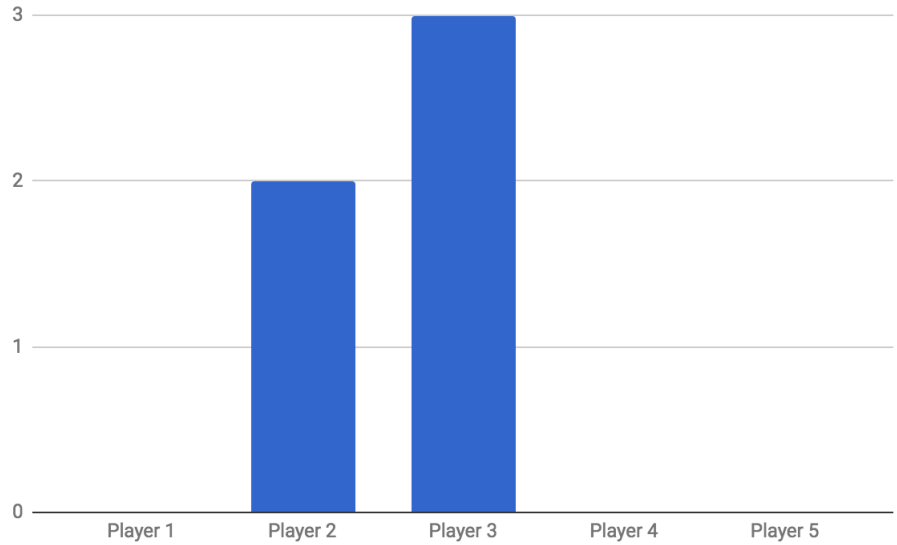
Data and Analysis

```
What would you like to do?
p
player 1: 6 on the Node
player 2: 6 on the Node
player 3: 6 on the Node
player 4: 6 on the Node
player 5: 6 on the Node
What would you like to do?
i
What would you like to do?
p
player 1: 6 to 3
player 2: 9 to 10
player 3: 6 to 24
player 4: 9 on the Node
player 5: 6 to 3
What would you like to do?
c
player 1: 6 to 3
player 2: 9 to 10
player 3: 6 to 24
player 4: 9 on the Node
player 5: 6 to 3
player 1: 3 to 10
player 2: 9 to 10
player 3: 10 on the Node
player 4: 9 on the Node
player 5: 6 to 3
Congradulations! player 3 has won on its 4th turn!
player 1: 3 49steps taken
player 2: 9 40steps taken
player 3: 10 38steps taken
player 4: 9 33steps taken
player 5: 6 25steps taken
```

Data and Analysis; Figure 1. My output with parameter of {"50","5","20","inputMaze1.txt","inputNodes.txt" }.

As shown in Figure 1 is my output for a run with parameters of {"50","5","20", "inputMaze1.txt", "inputNodes.txt" }. This tells that the limit is 50, with 5 players and the maximum value the dice will roll is 20. In this run, we see that player 3 has won. It will keep in track of the number of steps and the number of turns each player has played.

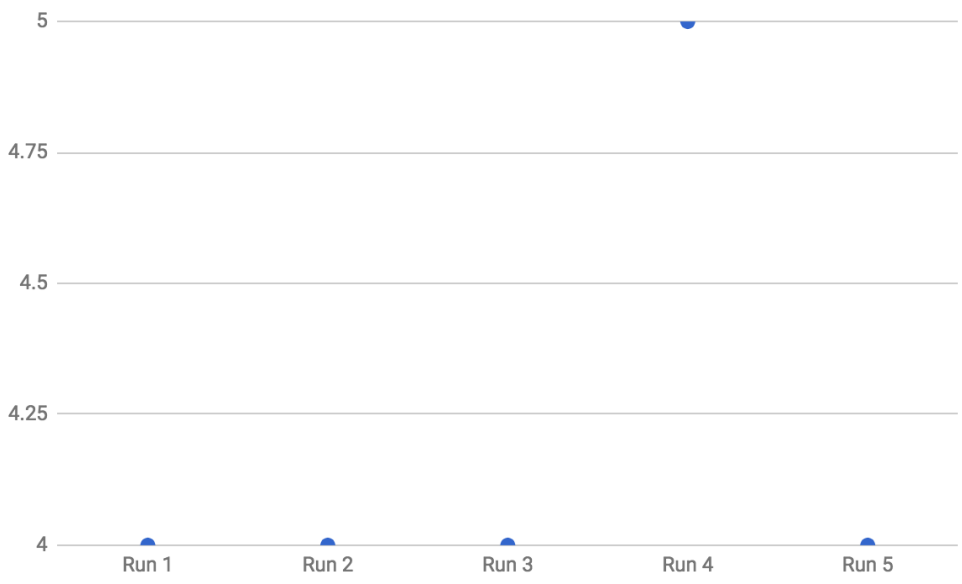
Using the same parameters, we see the results below of 5 runs.



Data and Analysis;

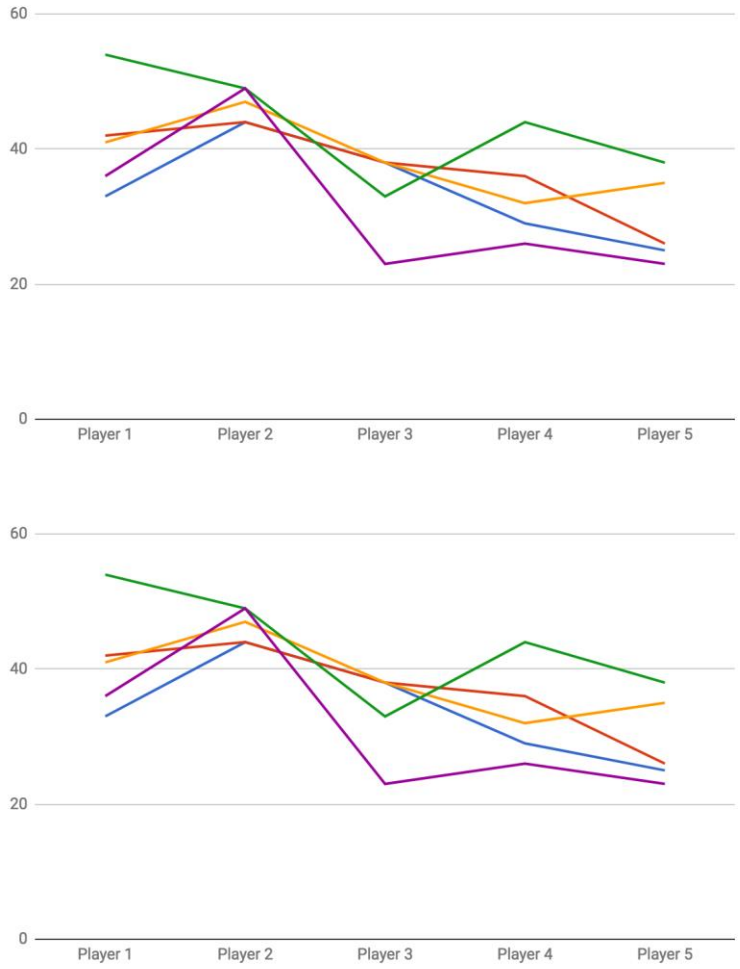
Figure 2. Result of 5 runs using parameters {"50","5", "20","inputMaze1.txt" ,"inputNodes.txt" }.

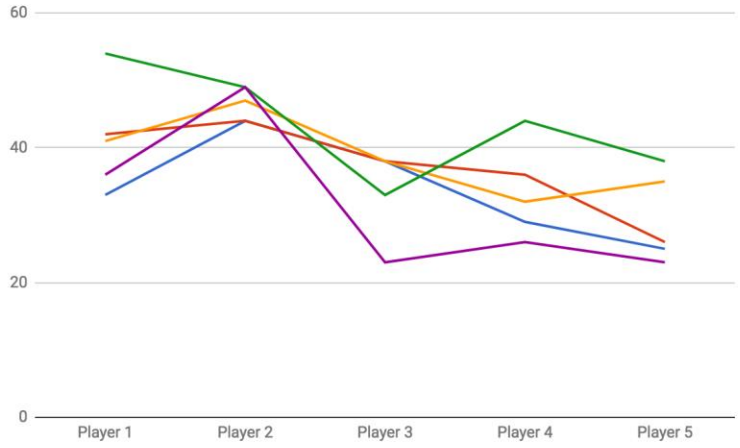
Figure 2 is the result of running the program 5 times using the same parameter and the maze. This was a surprising result for I assumed that player 4 had a great advantage of using the Dijkstra's algorithm.



Data and Analysis; Figure 3. This shows the different turns that players took until the game was over.

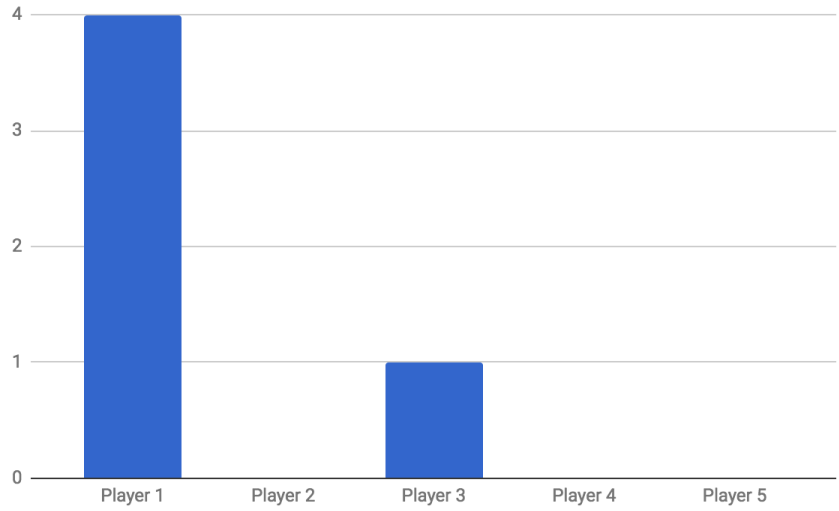
From figure 3, we can assume that constant configuration of the graphs and starting/exiting node will result in a similar result regardless of the multiple runs.

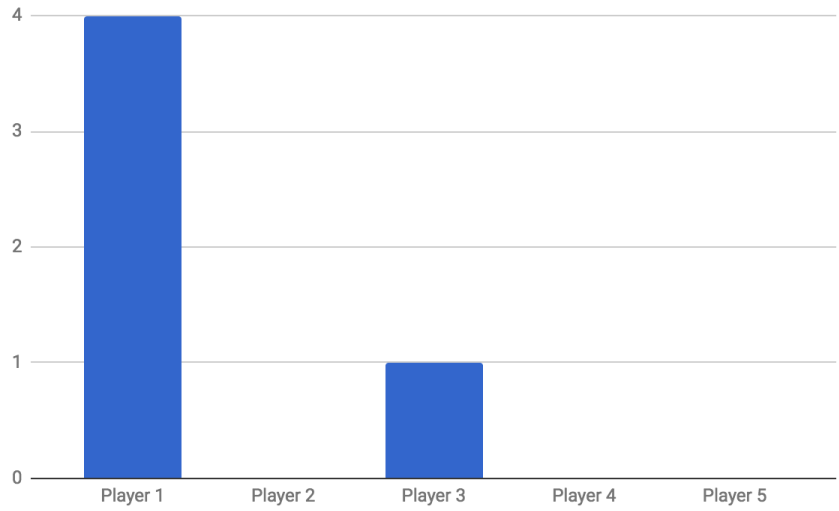




Data and Analysis; Figure 4. This is the different steps that each player took until the game was over.

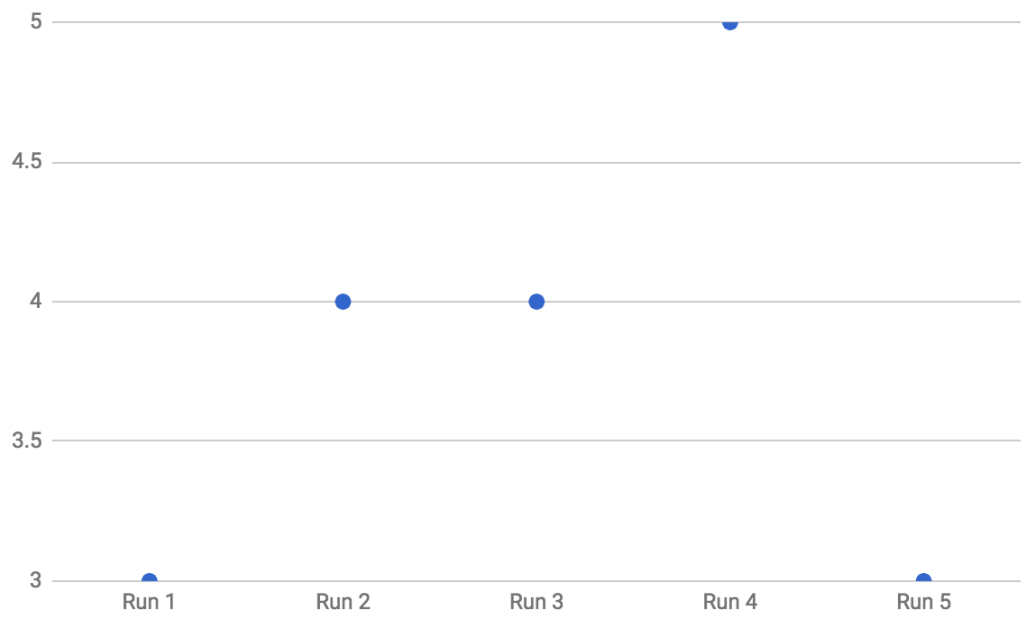
Since the results were so similar, I became suspicious that my dice seed were the same. However, since it takes in the System.nanoTime() (System Class), it shouldn't be. Therefore, I concluded that the time I ran this program were too close that my seed did not vary enough.



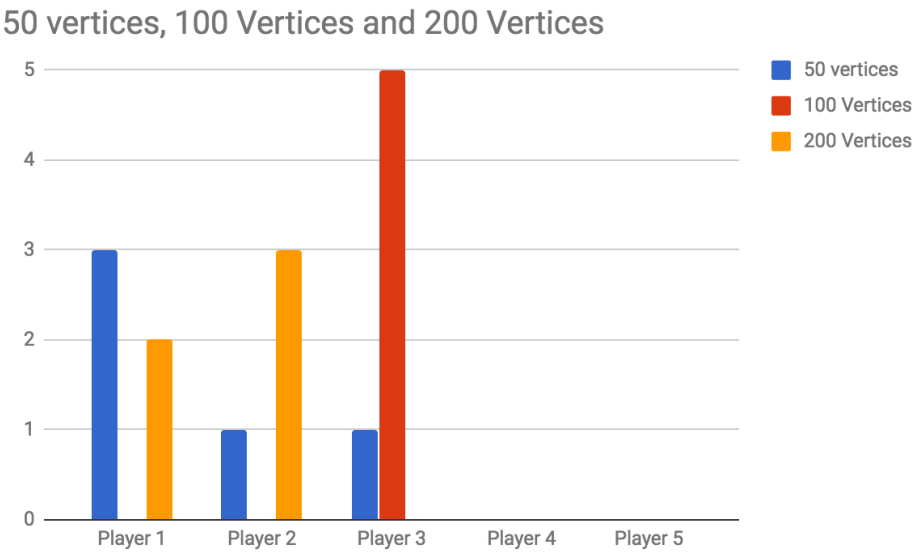


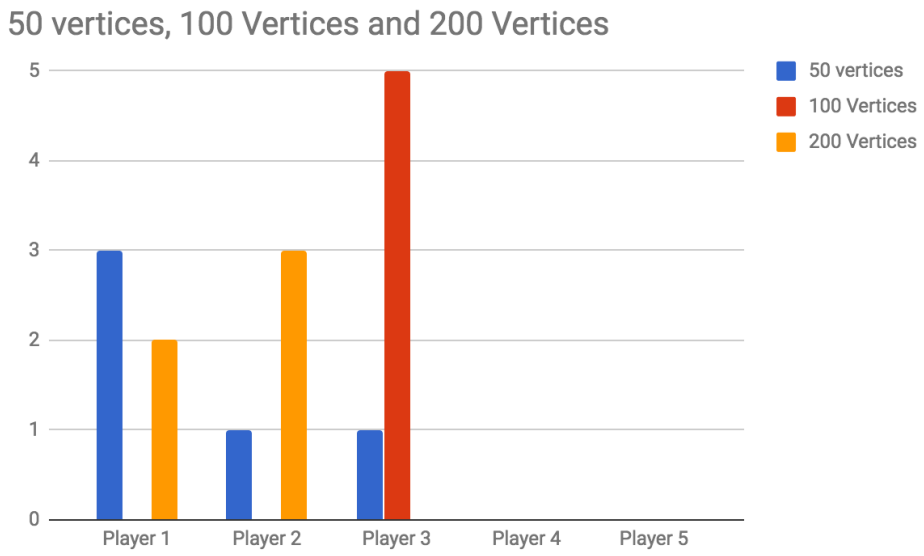
Data and Analysis; Figure 5. Winners of 5 runs using a different maze.

Figure 5 shows the results for parameters {"50","5","20", "inputMaze2.txt", "inputNodes.txt" }, which uses a different maze than the first one. As shown, you can tell that it is very likely that the first player will win with this configuration. Although the first one had 50 vertices, where as this includes 200. However, it shows that the numbers of turns that players took to reach a exit node did not change as significantly as I assumed shown in Figure 6.



Data and Analysis Figure 6. Number of turns the winner played to win the game.

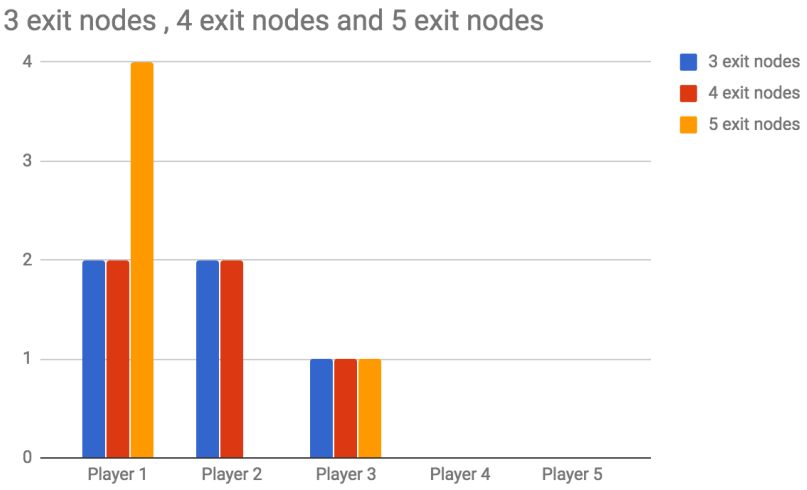


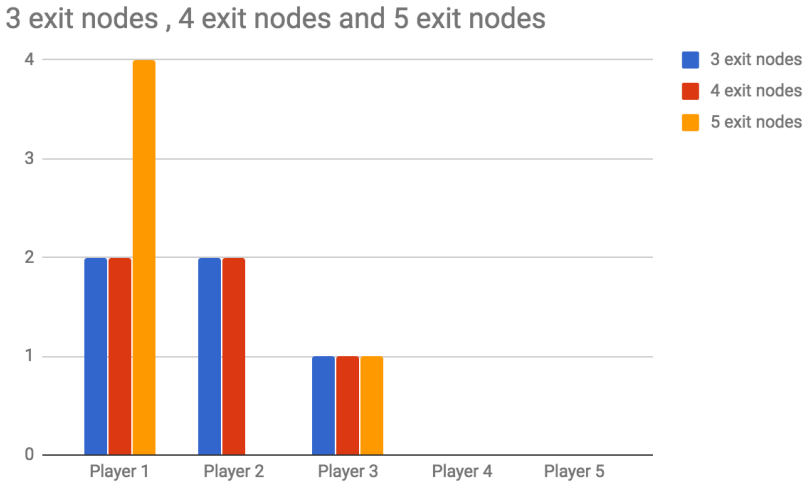


Data and Analysis;

Figure 7; Results using different vertices in a maze

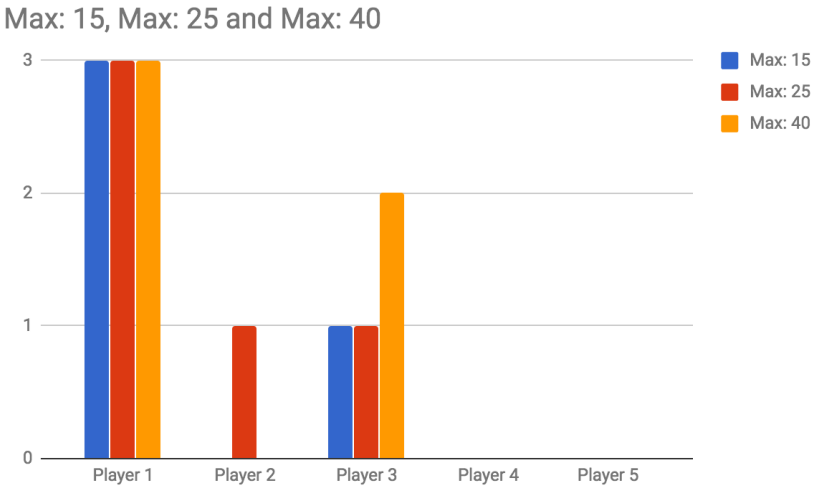
As shown in figure 7, it is apparent that player 4 and 5 has not have a win despite the configuration of the maze. Each number of vertices, 50, 100, and 200 were each ran 5 times using the same parameters for others. Figure 7 shows that the number of vertices are most likely not to be connected with the player that is likely to win.





Data and Analysis; Figure 8; Results using different number of exiting nodes on a 50 vertices maze.

Figure 8 shows the results using different numbers of exiting nodes on a 50 vertices maze. I used the same 50 vertices maze since it had the most varying on their results when the game is played. Every run, the node numbers of the exiting nodes were changed and each numbers of exiting nodes, 3, 4, and 5, were run 5 times each. According to the previous results, it is apparent that player 4 and player 5 has a significant disadvantage somewhere. Printing out the results of my Dijkstra distances to all nodes, it was obvious that it had an error since the distances were ridiculously low.



Data and Analysis; figure 9; Results using different maximum values the dice can generate.

Like my other experiments, figure 9 shows my results using the different maximum values the dice can generate. The different values includes 15, 25, and 40. Here, a constant 50 vertices maze was used and each were ran 5 times. However, according to figure 9, it does not give a significant pattern to the winners other than player 4 and 5 unable to win.

Data and Analysis; Figure 10; Results using different limits

To actually determine whether my Dijkstra's algorithm was working, I decided to do an experiment altering the limits, shown in figure 10. However, player 4 and 5 failed to win a single game despite the increase in limit theoretically should give them a much higher advantage of winning. Therefore, I conclude that my method for Dijlstra's algorithm has an error.

Conclusion

Throughout data analysis, though it was successfully running before, I realized that my program has an error in its Dijkstra's algorithm, for the varying experimental set ups does not change the results of the game. It also shows that the starting node and exiting nodes greatly affects the results as well. However, throughout this project, I learned the complications of video games. Although this asks for user prompts, it is still a simulation of players. I learned the various boundary cases, and even the characteristics that are generated randomly are coded specifically to run randomly.

References

(2017, December 19), Retrieved May 5, 2018, Class ArrayList<URL>, from https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html

(2017, December 19), Retrieved May 5, 2018, Class LinkedList<E>, from https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html

(2017, December 19), Retrieved May 5, 2018, Class Scanner, from https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Retrieved May 5, 2018, CS150 Data Structures and Algorithms, from https://canvas.instructure.com/courses/1171720

C W Liew, Retrieved May 5, 2018, CS150: Project3 - Maze Game, from https://moodle.lafayette.edu/pluginfile.php/391530/mod_resource/content/4/p3.pdf