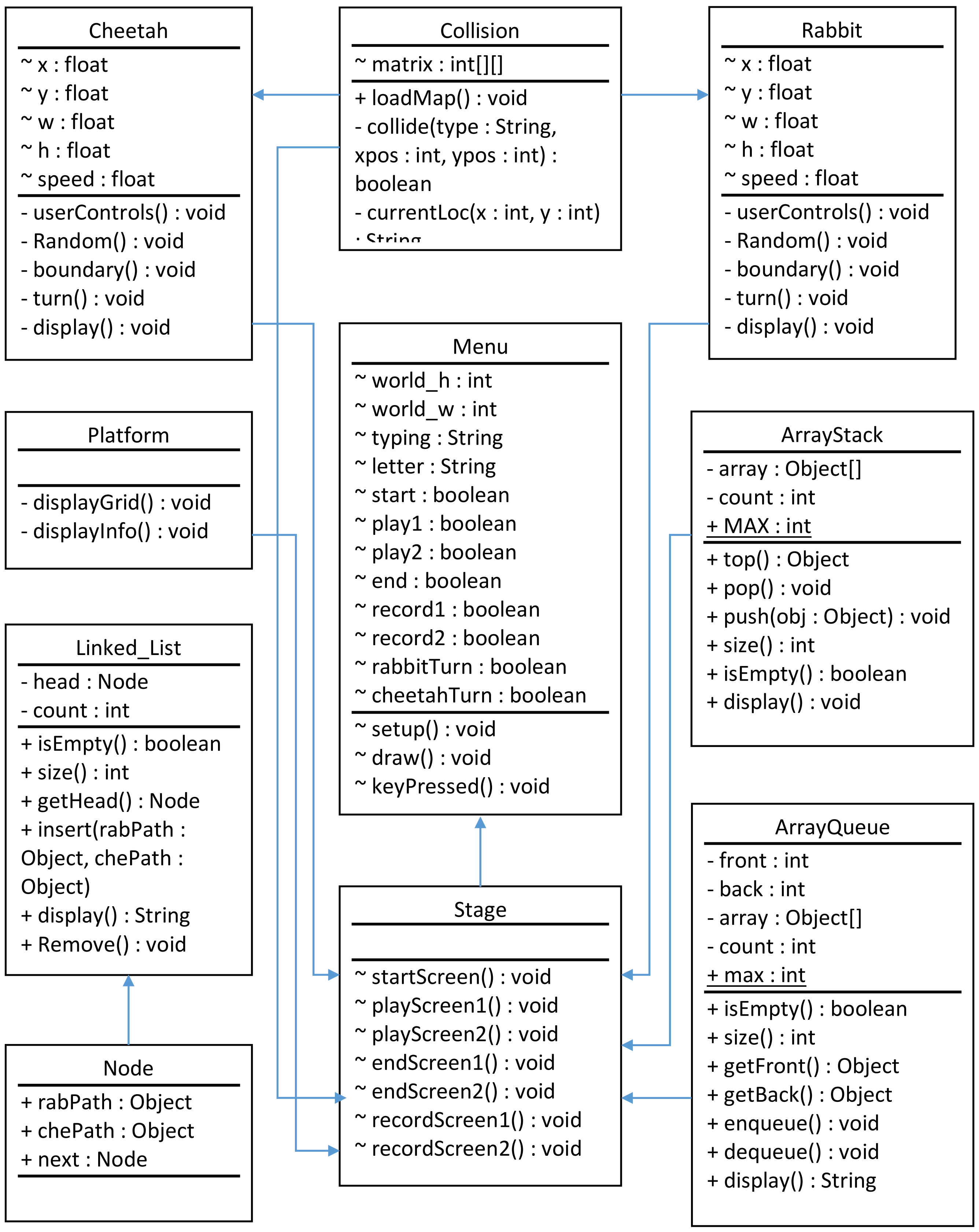
IS52038A Algorithms and Data Structures
Coursework 1 Part 2 2016

By
Anna Huang (ma304ah)
Angela Pham (ma302ap)

Contents

Algorithms:

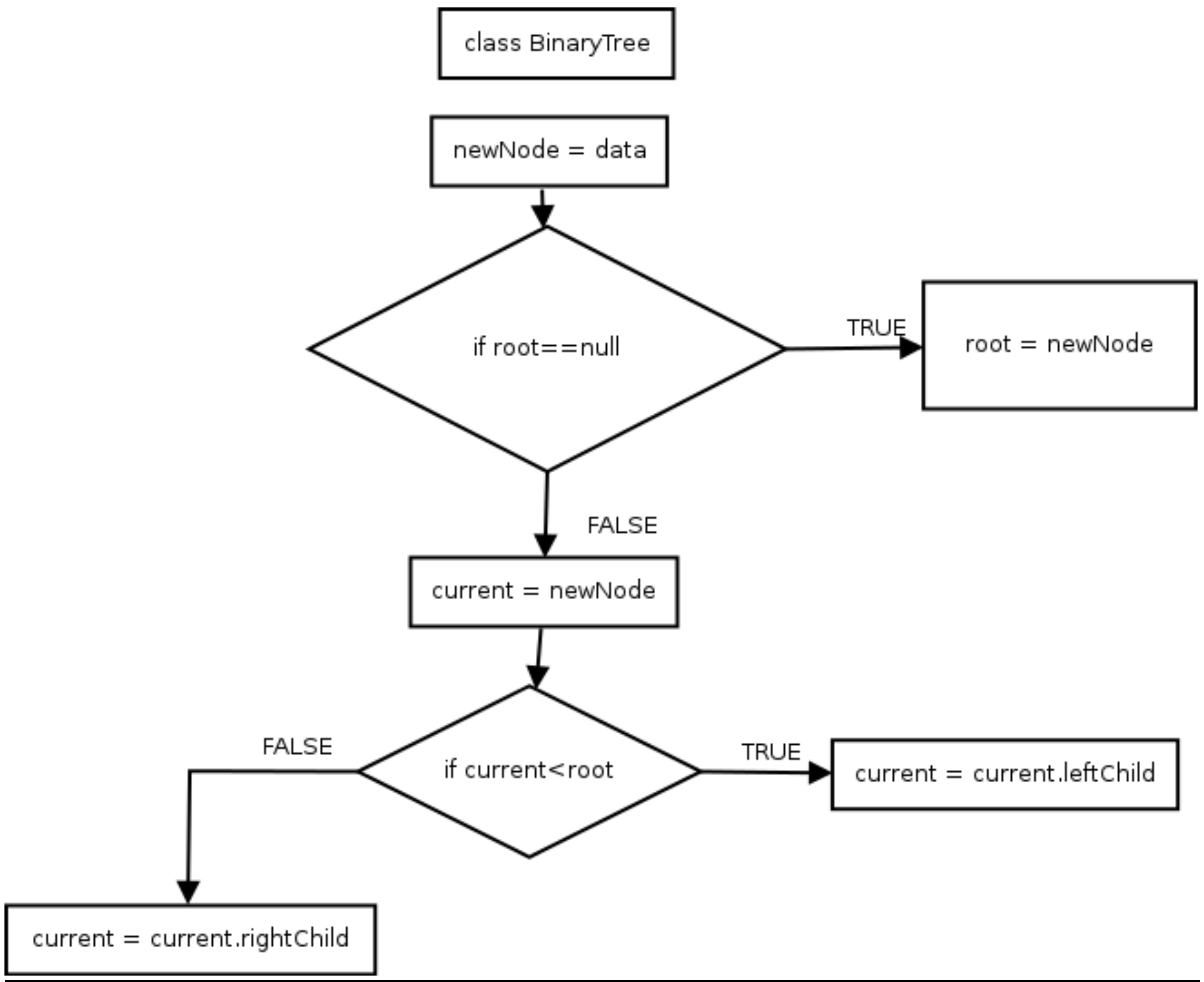
UML



Flowchart

One of the algorithms we've used from other sources are linked list [1], binary tree [2] and the method to allow user to input [3].

Flow chart of the binary tree design



Implementation

When extending our program further we decided to create a switch case that moved the rabbit to a random direction in order for the program to be played by one player. This was the easiest way to create an AI interaction but meant it wasn't always moving in a direction that was beneficial to the rabbit.

For Part 2, we have made several attempts at making our program more advanced. Our first attempt was to allow the cheetah access more neighbours and to move diagonally rather than just up, down, left, right. This was done easily by adding more key statements and changing the x/y direction.

For 2b, we were able to write a method that could store the previous walks of each character as a record and display the co-ordinates taken. One of the reasons why we used queues is that it can store the order in which it first occurred such as when we stored the walks so that we could easily find the shortest path taken (e.g. rabQ2, cheQ2) and display it easily using linked list.

For 2c, we attempted to add a model walk using previous walks that were taken. We stored the co-ordinates in two separate ArrayList. One for x (arrayX) and the other y co-ordinate (arrayY). The reason why we chose to use this is because it doesn't have a defined size as the array can grow dynamically. We ran into a few problems and it didn't work as how we intended it to be in which, it only displays the last co-ordinates of cheetah or rabbit.

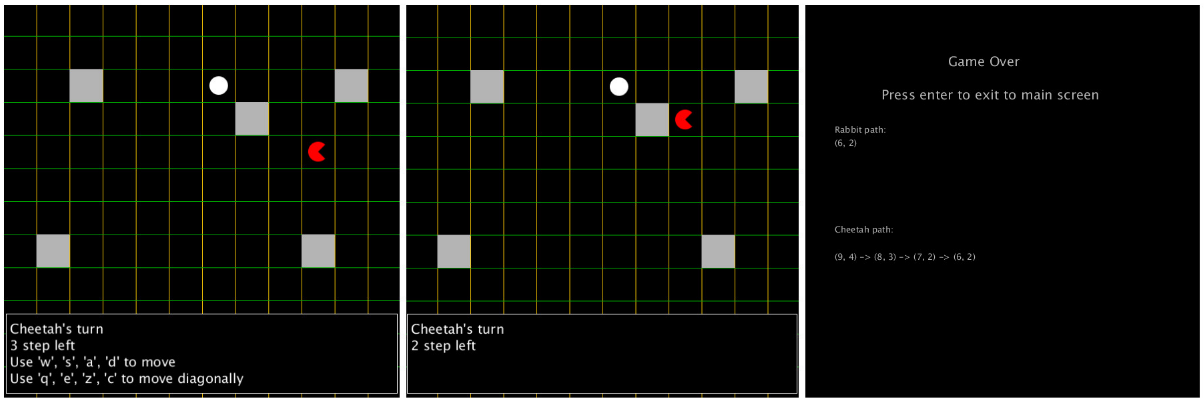
For 2d, another feature added was to store each of the walks in a binary tree. Using this type of data structure to store the lengths of each path taken is beneficial, as it would sort the path in terms of order (smallest to largest) and would be faster to search for a certain length route. However, we didn't have time to finish this.

Testing

As said before in part 1, we kept having issues with the built in draw function as it kept looping and causing infinite loops. We had to make another turn method (turn2) for rabbit because the first turn method is within keyPressed method and the rabbit would only move if user presses a key; which is not what we wanted so made turn2 and declared that inside play screen which will be called in side draw method.

However, this caused a few problems where it would look as if rabbit has not moved at all because animation of it moving is so fast. Not to mention the fact it some reason, the rabbit would go off screen or crash into a block and kill the program.

<u>Demonstration</u>

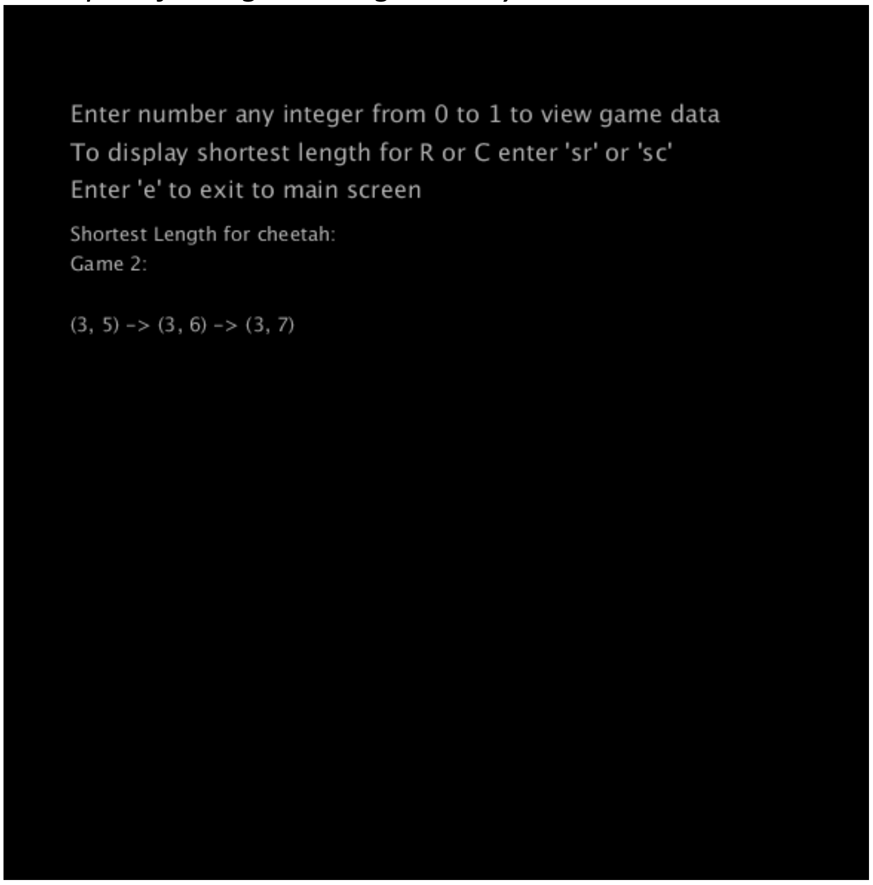


```
Cheetah's turn
3 step left
Use 'w', 's', 'a', 'd' to move
Use 'q', 'e', 'z', 'c' to move diagonally
```

```
Cheetah's turn
2 step left
```

```
Game Over

Press enter to exit to main screen

Rabbit path:
(6, 2)

Cheetah path:

(9, 4) -> (8, 3) -> (7, 2) -> (6, 2)
```

Example of using the diagonal keys to catch the rabbit. Co-ordinates are given.



```
Enter number any integer from 0 to 1 to view game data
To display shortest length for R or C enter 'sr' or 'sc'
Enter 'e' to exit to main screen

Shortest Length for cheetah:
Game 2:

(3, 5) -> (3, 6) -> (3, 7)
```

Example of our shortest path

Discussion

Time complexity

The average case for queues and linked list are O(n). Depending on what operations these are doing, the worst case scenario could be O(n) if n is not 1. Stacks however have time complexity of O(1). Although if a search function is implemented, the time complexity would be O(n). For binary tree, the average case is O(log n) and the worst case is O(n)

Limitations and Successes

When implementing our model walk, we later realised that we should use the matrix 2d array. It would have been easier to store the co-ordinates as it can store both x and y co-ordinate.

When creating the method to store the paths length in the binary tree, we were successfully able to implement the binary tree class but had problems with adding the nodes, due to it requiring an int to be stored rather than a string. When storing the node as a string, it meant that it sorting the paths by length wasn't possible.

If time was not an issue, we would have changed the binary tree class to solve this problem as well as been able to research other path-finding algorithms such as BFS or A* to further improve our program and make the rabbit "smarter".

Also, doing further reading and implementation of graph search specifically "pursuit-evasion" could have given us a better outcome.

What we have learnt

We have learnt to use the data structures more confidently and have a better understanding on time complexity and role it plays in how well a program performs.

Reference:

Banas, D. (3 Mar 2013) Linked List In Java [online] < http://www.newthinktank.com/2013/03/linked-list-in-java/ > [Accessed on 24 Feb 2016]

Banas, D. (28 Mar 2013) Binary Tree In Java [online] < http://www.newthinktank.com/2013/03/binary-tree-in-java/ > [Accessed on 27 Feb 2016]

Shiffman, D. Learning Processing [online] < http://learningprocessing.dreamhosters.com/examples/chapter-18/example-18-1/ > [Accessed on 26 Feb 2016]