Data Structures and Algorithm

# Data Structures

## List of Paths (pathList.json)

* **Description:** This data structure should be used to get a path from a path index
* **Composition:** List
* A path is defined to be a list of six xy-positions sequentially from minutes 1 through 6.
  + Ex: [[8689, 5078], [11583, 4778], [12916, 1850], [7264, 4877], [2476, 8344],[4415, 9147]]
    - [8689,5078] is the 1-minute position
    - [2576,8344] is the 5-minute position

## List of Nodes

* **Description:** This data structure should be used to get a node from a node index
  + A node is defined to be a position along a path
* **Structure:** Dataframe
  + *pos:* the position of the selected node on the map
  + *parent:* the prior node that the selected node branched off of
  + *pathIndices:* the indices of paths that are considered to be similar to the selected node

## Lookup Table

* **Description:** This table can be used to lookup the list of nodes corresponding to each minute
* **Structure:** Dataframe
  + *min:* the in-game minute
  + *nodeIndices:* the indices of nodes available at that minute

Here’s a tree diagram visualization of the nodes with some made-up data:



* Each node is represented by a square, and the node index is indicated by the number on the square.
* In nodeList, *Node 3* has a list of pathIndices for all of the paths that had a similar 2-minute position to the position of *Node 3*.
* *Node 3* is the parent of *Node 9*, *Node 9* has a list of pathIndices that had a similar 2-minute position to the position of *Node 3* and similar 3-minute position to the position of *Node 9*.
* *Node 9* is the parent of *Node 16*, *Node 16* has a list of pathIndices that had a similar 2-minute position to the position of *Node 3*, similar 3-minute position to the position of *Node 9*, and similar 4-minute position to the position of *Node 16*.
* nodeList parent value examples:
  + *Node 3:* null
  + *Node 9:* 3
  + *Node 15:* 7

# Algorithm

**Input:**

* Current minute, M: The current minute displayed on the graph
* User input, U: The node on the plot that was just selected by the user
* Parent node, P: The previous node selected by the user
  + Not applicable until the user is about to make their second input.

**Step 1: Does the user input correspond to a valid node?**

* Using the Lookup Table, check to see if U exists in the list of node indices corresponding to M.
* Using the List of Nodes, check to see if the parent node corresponding to U is equal to P.

**Step 2: If U is valid → Get list of path indices**

* Using the List of Nodes, get the list of path indices corresponding to U

**Step 3: Using the list of path indices → Get the next minute positions of these paths**

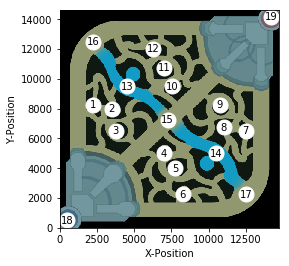
* Using the List of Paths, get the paths corresponding to the list of path indices obtained from Step 2.

**Step 4: Using the paths → Plot the next minute positions**

* Update M, M = M + 1
* Plot the position corresponding to the updated value of M for each path

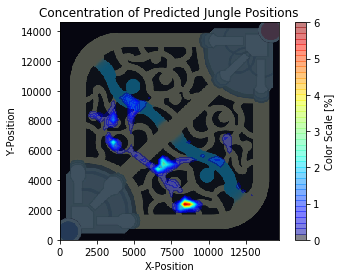
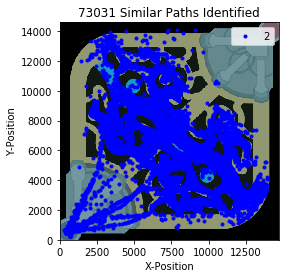
# Walkthrough

In order to construct these three data structures, I wrote a script that explored combinations of the coordinates shown below, and identified all of the paths composed of these coordinates that had at least 100 similar paths to the paths contained in pathList.



Throughout the interactive graphic, the contour map will most likely guide users to select positions near the coordinates shown above.

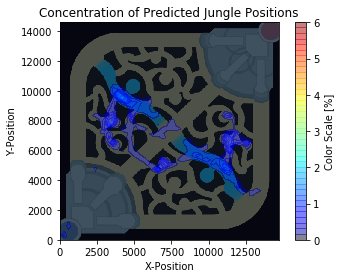
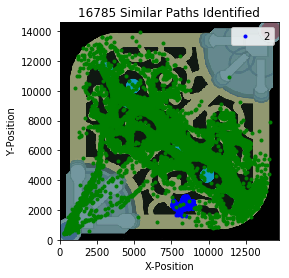
At the beginning of the interactive graphic, the user will be presented with a plot and a contourMap of the 2-minute positions from all of the paths at the start of the interactive graphic, and the user will select a starting 2-minute position for their path to begin.



The user should be drawn to selecting a position near the areas of high concentration on the heat map. If the user selected a position near *Node 5* (the only area with orange-red on the contour map), then the interactive graph would update and show a plot and contour map for the predicted 3-minute positions from all of the paths with similar 2-minute positions to that of *Node 5.*

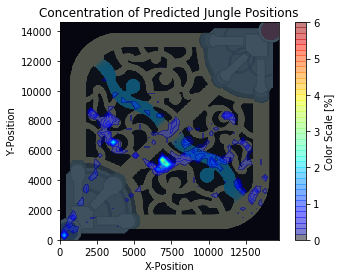
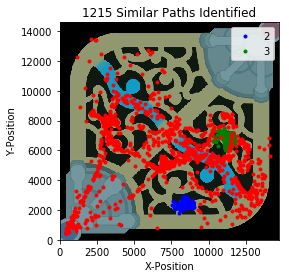
These path indices would be obtained by looking up the pathIndices for node 5 in the nodeList data structure, and the next minute positions for these paths would be obtained by looking up the 3-minute positions in the pathList data structure.

Once the next minute positions are obtained, the interactive graphic would update with the following plots:



The interactive graphic is now displaying 3-minute positions, and *Node 5* has been stored as the parent node. This time the user may select the position [11000, 6750]. The algorithm will use the lookupTable to identify the node indices available at minute 3, and then filter the nodeList by these node indices. The nodeList will then be filtered further by only looking at nodes with a parent value of 5 (*Node 5)* and a position near [11000, 6750]. From this, the algorithm should find that the user input corresponds to *Node 63*.

The same process will be undergone to find the next minute positions, and the interactive graph would update with the following plots:



For other paths at 4 minutes, there is typically not enough data (<100 similar paths) to allow the user to provide another input and generate 5-minute position predictions. However, in this example, I picked one of the most popular paths that still has an ample amount of data present at 4 minutes. Based on the contour map, the user may be tempted to select the following position: [7000,5000]. This position corresponds with *Node 93*, and the final update for the interactive graphic would look like this:

