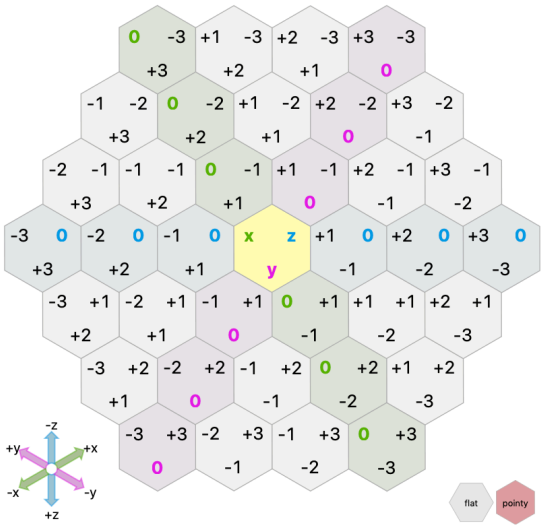
Teun Petersen and Hong Ngoc D
COMP300024 Project A Report

There have been many aspects of the project that we have taken into consideration when analyzing the game. First we had to understand the state in which the player is in, including the player's possibilities, means of finding the path to the destination, and decision making. When it comes to possibilities, we had to understand how a hexagonal board operates and that was when we found this website: https://www.redblobgames.com/grids/hexagons/. It illustrates the tricks and various perspective on the hexagonal grids. The 9th picture (or rather interactive image) shows the board in regards to three axes: x, y, and z. There, we found an interesting pattern: all destinations (specifically the tiles from which the player can exit) for the player (regardless of the color) had in common one thing – one of the axes were equal to 3. For example, if the player's color was red, then the destinations' tiles had an x value equal to 3. That has significantly impacted the way we defined our goals (reaching those tiles), how we find paths, and this became the core of our heuristic. Here is a copy of the website's visualization of three axes looks like:



When it comes to the choice of algorithm, we certainly had in mind the fact that we have a time constraint. Hence, we had to pick an algorithm that was stingy in moves and efficient in thinking. In the end, we have settled down on A* search (A star search). We have noticed that the presence of heuristics makes the player more parsimonious and the search

itself also does not expand on paths that are already expensive, which meant little time to perform the search overall. Our A* search is also based on a data structure that nicely stores all our data and orders it according to its value – the priority queue. This search is also complete, the time needed to run it is acceptable, and it is optimal time wise for our project.

As for the heuristics, we originally wanted to simply implement Manhattan distance, but then realized that the formula does not work on a hexagonal grid. Through tedious yet brief trial and error, we have found a pattern when calculating the shortest path needed to get to the destination, regardless if the tiles are taken or not. In the first paragraph we have mentioned that the destination has one of the axes equal to three. Basing on the color of the player, one can find the distance by subtracting 3 from the defining axis (in red player's case it is the x axis, in blue player's it's the y axis, and in the green player's case it's z axis; all in accordance to the picture in the referenced link). The absolute value is the total cost of moves (no jumps included) that it takes to get to destinations from current the current tile. Therefore the A* search logic is straightforward: if the next move pulls the player closer to the destination, the heuristic cost is lower, lowering the total cost of that move (which also includes the cost of previous moves) and making that path more attractive.

We have also found that A* search takes a long while to find a path when there are four players. Since the search performs very well when there are three or less players, we have decided to perform A* search twice – picking the first players and move them first, before doing the same with the other two players. Though this search is not the shortest path, it allows the search to perform fast and within the given time constraint.

The resulting search tree from A* search has a branching factor of up to 12 because it considers all adjacent tiles (total of 6), and if the tile is taken (regardless if it's another player or an obstacle) then it also looks at the tile ahead for a possibility of jumping. The presence of "directions" ensures that the tile on which the player can jump is a valid move – any tile that is not aligned with the immediately adjacent to the player tiles' direction is not a valid tile to jump on. A* search expands on the most promising path (the least costly one) and changes its priority in the queue. Unfortunately, all paths taken so far have to be kept in the memory, which grows exponentially.