Group Name : McFries

Members : Liew Hsien Loong 18030221
Steven Soo Yon Zhang 16039166
Ng Wei Xiang 18033167
Teh Jiing Ren 18027565

We were given a snake game and the assignment was to generate the AI algorithm to automate the snake in achieving the objective of eating the food using uninformed and informed searches. The main problem that we are trying to solve is to let the snake decides its path to the location of the food. Thus, the exact problem that the algorithm is trying to solve is the process of deciding which path is better accounting into the cost so that the shortest path is achieved. The snake has 4 directions to move, North(N), South(S), East(E) and West(W). By moving from the current snake's location to the food location, a series of direction will be generated to obtain the solution and moving the snake. Thus, the algorithm solves the thinking process of the snake through a series of direction and generating a search tree every step to view the logic and thinking process made. We were also given 3 different challenges for the snake game which were : one food at a time with non increasing snake length, one food at a time with increasing snake length, and two food when there is no food on the map with increasing snake length. Achieving any of these challenges will be sufficient but we are encourage to attempt and achieve all of them.

For implementation, we decided to implement breadth first search for uninformed search and A-star first search for informed search. The 2 main difference between these 2 algorithms is accounting of cost of path taken whereby the informed search will try to achieve the path with lowest cost. Other than that, there is not much difference between the 2 which is why we have same parameters and functions implemented by the 2 of them to save time and effort since these parameters and functions performs the same thing. That includes functions like getLocation(), getChildren, getSolution(), getParentId(), and getSearchTree().

Breadth-first search is an uninformed search where a simple strategy in which the root node is expanded first, then all the successors of the root nodes are expanded next, then their successors, and so on. To be more space efficient when storing states, we used a class to declare each node in the search tree to be an object that stores Id, state, expansion sequence, children, actions taken, removed node, and parent Id. It also has two functions called addChildren for the expanded node and addAction for the direction action. In bfs class, we use it to store the snake as initial state, goal state state, space number of nodes, number of expansions and actions. We use the function getChildren to add expansions sequence, to find the children of nodes. Function getSolution is to recreate the solution path. Function getSearchTree is to create a search tree.

In function algorithm(), we used three empty arrays to store the frontier, explored states, and removed node. Frontier variable will store an array of node state that have not visited. We will continually expand the first node in the frontier until we generate a node with a goal state. Thus, we need to have a loop to repeatedly expand the first node in the frontier until the goal node is generated. In the loop, we first expand the first node in the frontier to obtain the children and move the expanded node from frontier to the explored array. We will check if the children generated should be added to the frontier or removed. If children is expanded previously in the explore array or if it is generated previously but not expanded yet, then it should be removed. Else it should be added to the frontier. Before a child is added to the frontier, it will have a goal test. We will loop through to find the entire solution path. If it has the goal state, the goal node is used to trace back it's parent which will be in the explored array. Then the predecessor to the parent of the goal node can be traced similarly. This process is repeated until the initial state with no parent to obtain the solution and the searches tree.

A* is one of the most favoured informed search used where it takes into account of the cost in reaching the destination. A* search performs by summing up the movement cost and estimated movement cost of the origin and destination. Although A* search takes a longer search time, but the result is more accurate and complete which is why we chose it rather than greedy breadth first search. The heuristic function named calHeuristic() was used to calculate by subtracting the distance of food location and snake location and transform it to an absolute value. We used Manhattan distance to calculate the heuristic instead of Euclidean distance because Manhattan distance is always a better estimate of the remaining path length and never overestimates. Using a more accurate heuristic reduces the number of nodes expanded. Due to the nature of informed search we prioritized the shortest path, we had a function named prioritiseGoalStates() to complete the feature of prioritizing the state that is to the goal node using the heuristic function for calculation.

In function algorithm(), we used three empty arrays to store the frontier, explored states, and removed node. Initial Path Distance will be 0 since it's the start node. We take the first goal node as it is the closest based on estimated cost. We will continually expand the first node in the frontier until we generate a node with a goal state. Thus, we need to have a loop to repeatedly expanding the first node in the frontier until the goal node is generated. In the loop, we first expand the first node in the frontier to obtain the children and move the expanded node from frontier to the explored array. Then we will store the expanded node from the frontier as temporary parent. Delete the expanded node from the frontier. We will not check if the children generated should be added to the frontier or remove because a star search does not care which node has been explored before. We will perform calculations, g(n) temporary parent + distance between children and temporary parent, h(n) distance from goal state to child state, f(n) combined heuristic g(n) + h(n). Copy the child to frontier, sort the frontier by the combined heuristic f(n). We will loop through to find the entire solution path. If it has the goal state, the goal node is used to trace back it’s parent which will be in the explored array. Then the children to the parent of the goal node can be traced similarly. This process is repeated until the initial state with no parent to obtain the solution and the searches tree.

In the end, our algorithms successfully achieved the aim and solved all of the problems and all 3 of the challenges mentioned earlier in the report. However, our A* search algorithm is seemingly obvious, taking a longer time in estimating the path. Nonetheless in the end, our algorithms worked flawlessly and had the ability to score high points in the game. The search tree was also generated alongside each step made by the machine that perfectly showed its thinking process. Figure 1 achieved a score of 32, Figure 2 shows a score of 24 and figure 3 shows a score of 18 with longer search times, which all of them were much higher than the benchmark of at least 10 points. We were very happy with the performance of the algorithms however there were definitely more that we can do to bring the algorithm to another level. For example, avoiding the collision with its body and searching for the optimal path instead of the shortest ones. However, those are not needed to be implemented in this assignment which we foresee us improving in the future. Nonetheless, the result was definitely meaningful because we not only achieved the requirements and benchmarks, we also went further investigating differences between models while benefiting from thinking from a machine’s thought perspective and process.

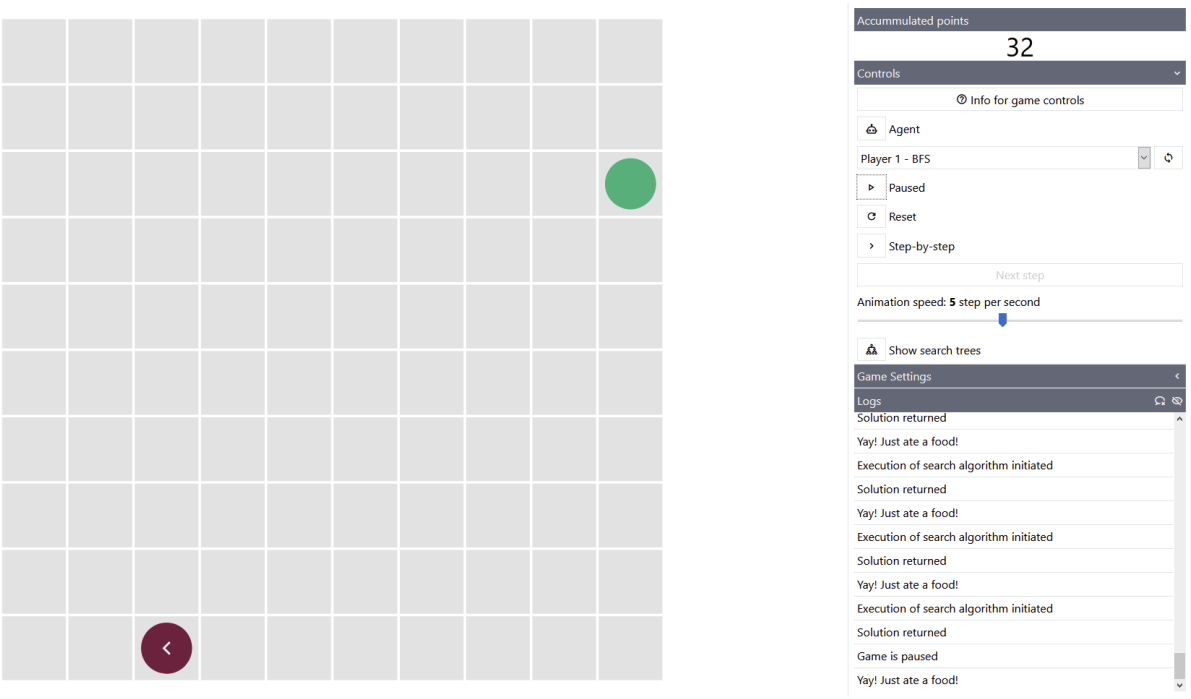

Figure 1 - Accumulated points for challenge A using breadth first search

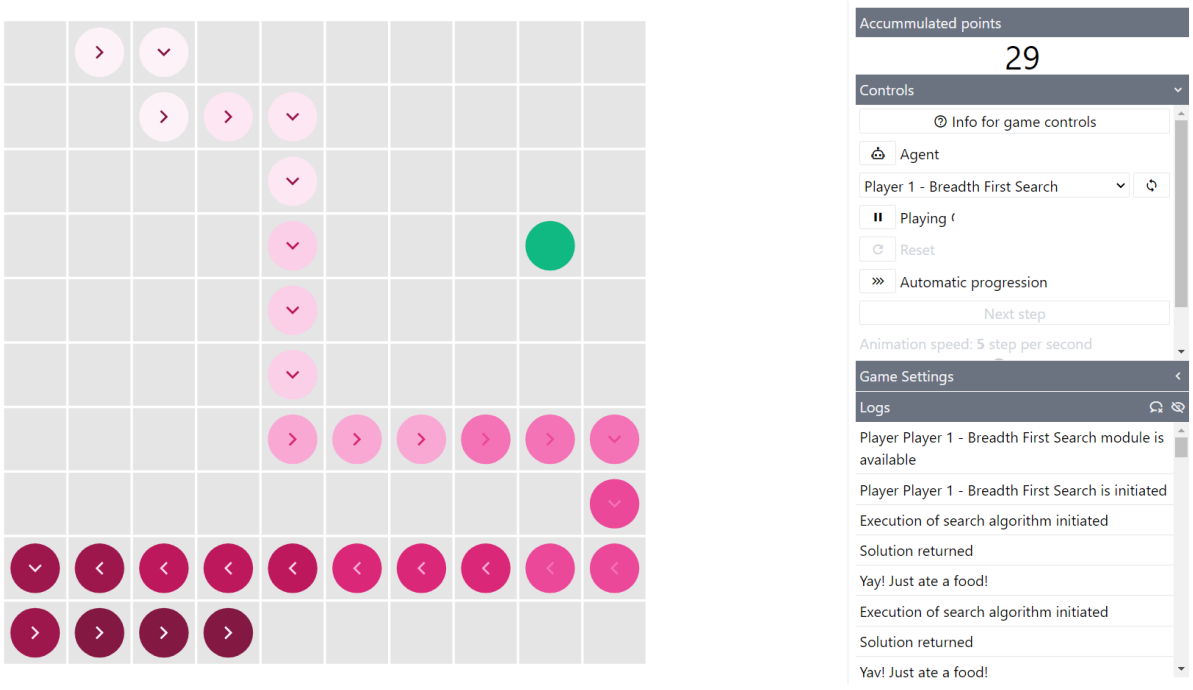

Figure 2 – Accumulated points for challenge B using breadth first search

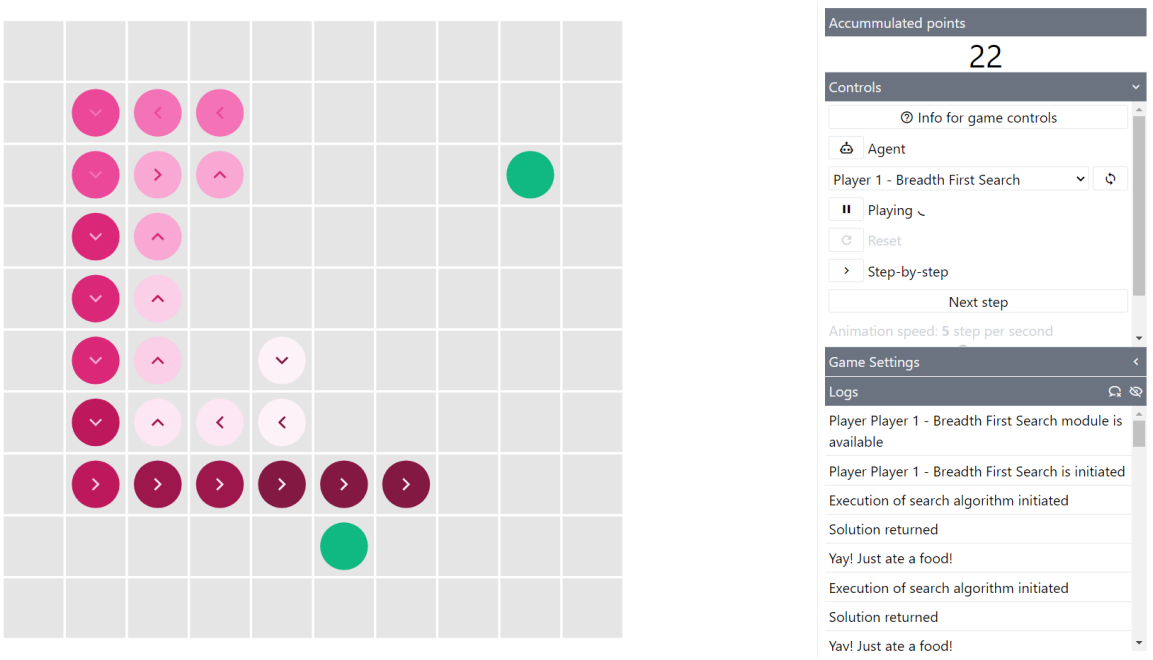

Figure 3 – Accumulated points for challenge C using breadth first search

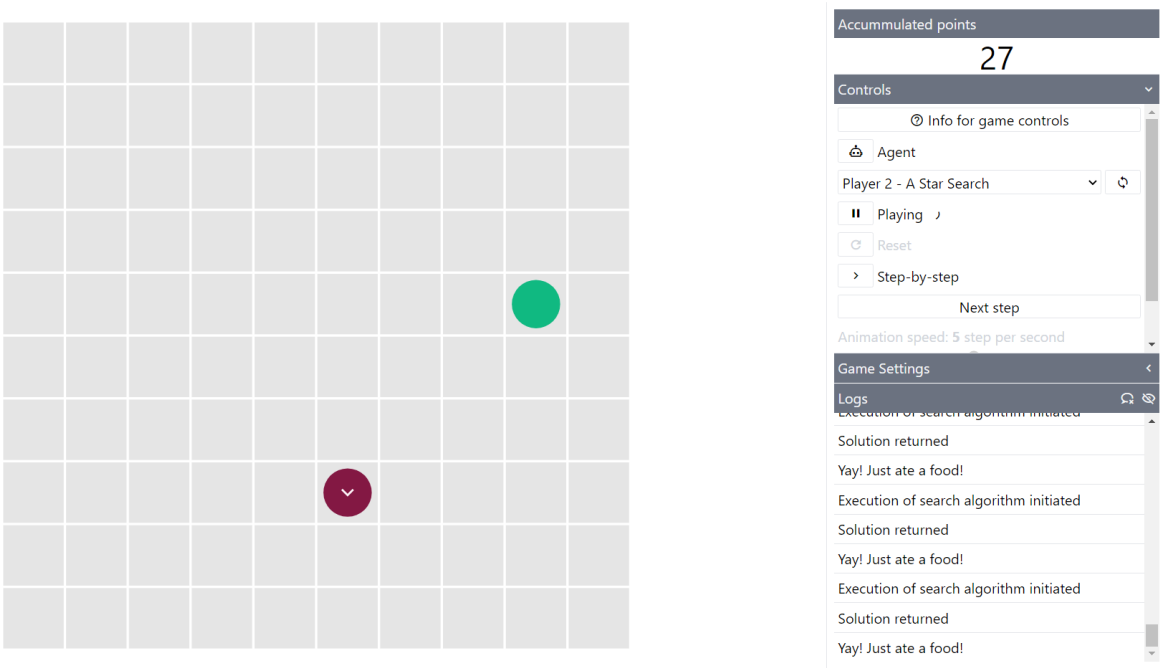

Figure 4 – Accumulated points for challenge A using using A search*

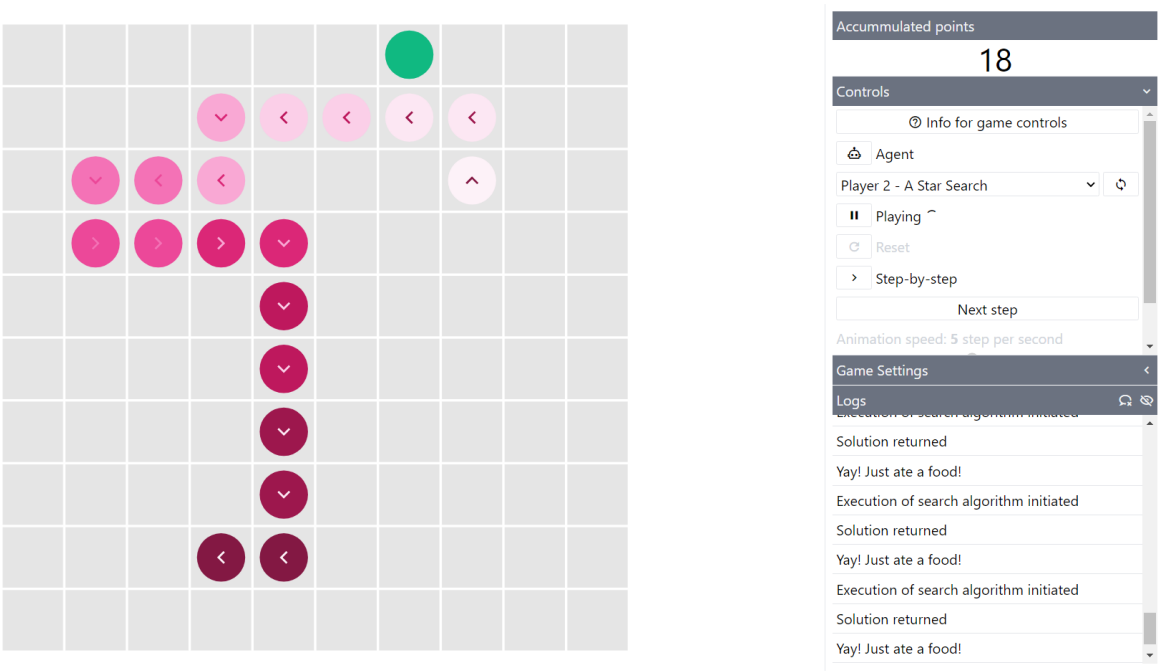

Figure 5 – Accumulated points for challenge B using using A search*

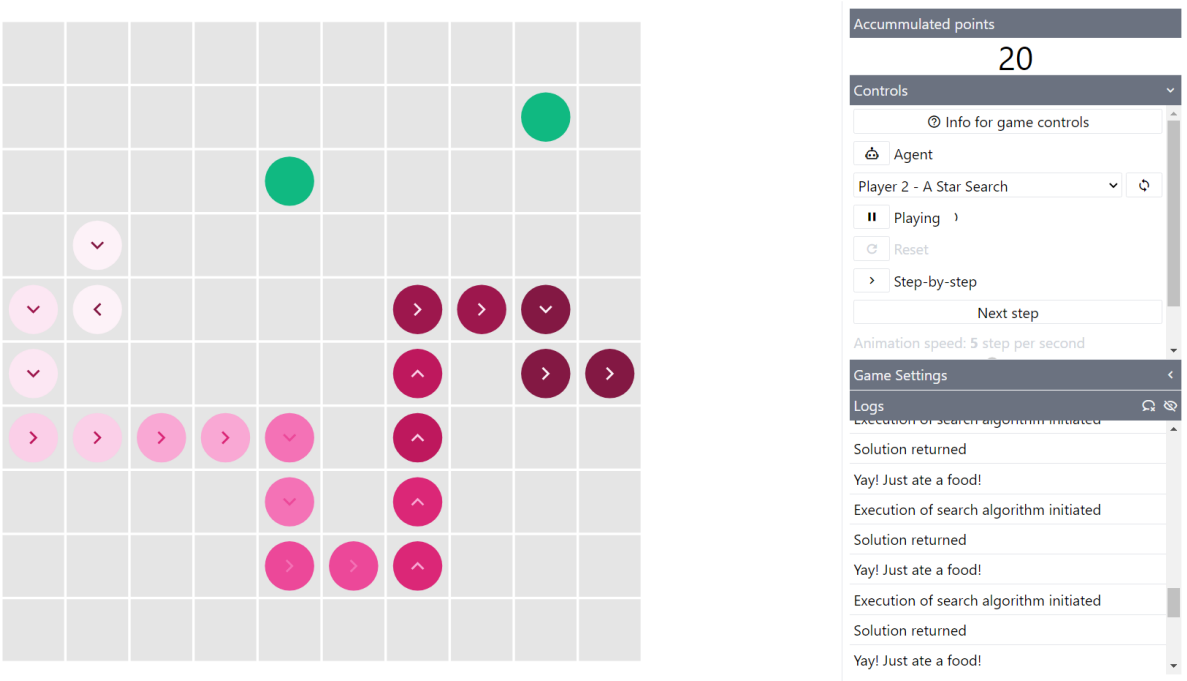

Figure 6 - Accumulated points for challenge C using A search*