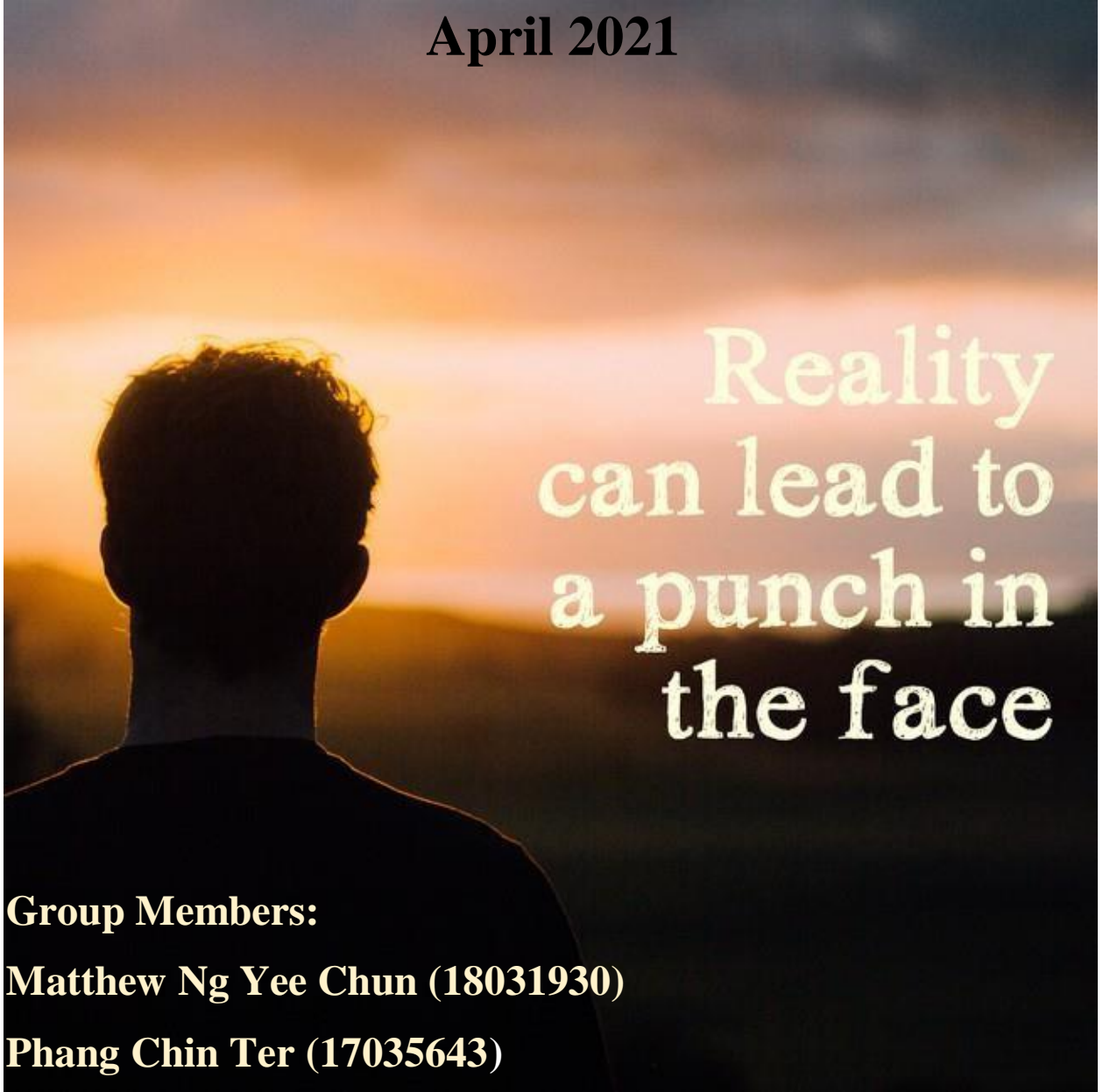


**Assignment 1**  
**CSC3206 Artificial Intelligence**  
**April 2021**



Reality  
can lead to  
a punch in  
the face

**Group Members:**

**Matthew Ng Yee Chun (18031930)**

**Phang Chin Ter (17035643)**

**Woo Wai Hong (16026189)**

**Loh Mui Hin (14085906)**

*Generated by Inspirobot*

## Table of Contents

List of Figures .....	2
List of Tables .....	3
Problem Description.....	4
Problem Formulation .....	4
Search Algorithm Implementation .....	5
Terminologies and Clarification for the algorithm explanation .....	6
Algorithm explanation.....	7
Results .....	8
Discussion .....	23
Recommendation .....	26

## List of Figures

Figure 1.1: Snake's decision making .....	24
Figure 1.2: Snake trapping itself .....	24
Figure 2.1: BFS - before moving .....	25
Figure 2.2: BFS - after moving .....	25
Figure 3.1: GFS - before moving .....	25
Figure 3.2: GFS - after moving .....	25

## List of Tables

Table 1: BFS results with 1 food .....	9
Table 2: GFS results with 1 food .....	12
Table 3: BFS results with 2 food .....	16
Table 4: GFS results with 2 food .....	19
Table 5: Result tabulation .....	23

### Problem Description

In this project, we are tasked with creating two agents to play a snake game. The goal in a snake game is to accumulate as many points by consuming food without having the snake bite itself. Both agents are to implement an uninformed and informed search algorithm respectively. Each agent has 3 ‘challenges’ or milestones to complete.

Therefore, the 3 requirements of each agent are listed as follows:

1. One food at any time; Non-increasing snake length; Snake length of one; reaching at least 15 points
2. One food at any time; Increase snake length with food; Starting snake length of one; reaching at least 10 points
3. Two generated when there is no food available in the maze; increasing snake length with food; Starting snake length of one; reaching at least 10 points


The aim of this project is to be able to develop 2 separate search algorithms that are able to complete all 3 requirements. The aim of the project has been completed where all 3 requirements for each agent. The results of each search algorithm have been documented in the **results** section.

### Problem Formulation

The fundamental components of a search algorithm are the goal, possible actions, the states as well as the solution. Therefore, the goal, the possible actions, and the states for the snake game are the food location, the possible directions that a snake can move at any given time, and the current state consisting of the snake body and food locations respectively. Moreover, the maze acts as the problem space. For the informed search algorithm, the distance between the snake’s head and the food serves as the heuristic function. Additionally, the solution is a series of actions which leads from the initial position of the snake’s head to the goal - the food. Moreover, the agents implementing their respective search algorithms will act as the snake.

The following rules are established for the search algorithm implementation for the agent:

1. Food cannot spawn inside the snake’s body.
2. At least one food will replenish when there is no food available in the maze.

3. The game ends when the snake dies either by trying to exit the maze or biting itself.
  4. For snakes with *dynamic lengths*, it will grow in length or size for every food it consumes.
  5. For snakes with *dynamic lengths*, it will not go backwards as it will bite itself
  6. If the agent manages to accumulate at least 15 points when attempting requirement 1 or at least 10 points when attempting requirements 2 and 3 as described in the previous section, then the agent is considered to have won in that specific round.
- 

When implementing a search algorithm for playing the snake game, 2 criteria is expected to be fulfilled - generation of the solution and the search tree. Firstly, the solution produced by the search algorithm must reflect the actions taken by the snake in order to reach a food inside the game UI. Secondly, a search tree should be constructed to complement the solution generated for the snake UI. Although the search tree has no direct impact on the algorithm's ability to navigate to a food, it is necessary as it serves as a form of logical error checking.

### Search Algorithm Implementation

According to the requirements and the problem scenario - snake game, we are tasked with developing 2 separate agents utilizing the uniformed and informed search algorithm respectively.

On one hand, an uninformed search algorithm, or blind search, operates in a brute-force manner, traversing the tree in a particular format with no additional information besides information available in the search space. On the other hand, the informed search algorithm, or heuristic search, functions by using estimates based on a common knowledge that is related to the problem externally and not implicitly defined by the problem. Therefore, 2 separate search algorithms were selected for implementation for both categories - Breadth-First Search (BFS) and Greedy Best-First search (GFS). The reason these 2 search algorithms were selected is due to their simplicity.

The implementation of BFS is straightforward by exploring all of the possible directions a snake can move towards as well as the possible directions the previous direction opens up can also be explored. The implementation of the GFS is similar to the BFS with a slight difference. The GFS will be exploring the problem space based on the heuristic function of each direction or nodes. For

this problem scenario, the heuristic function is the estimated straight-line distance between the snake's head and the food.

During the development of the search algorithms, several considerations were made apparent and addressed accordingly. These considerations are worth mentioning and therefore been listed as follows:

1. A snake with dynamic length must account for its body so that the snake will not try to traverse backwards into itself in order to reach a food.
2. The snake should terminate itself if it corners itself. Otherwise, the UI will enter an infinite loop and will be required to be terminated manually.
3. The snake should never try to exit the maze unless it has cornered itself as mentioned in consideration 2.
4. The snake should account for its tail's position when trying to reach for a food. When not addressed, the snake will assume it will never reach the food and explore every other node even though a clear path will be created after the snake has moved its tail away. This addressing this issue is necessary in order to maximize the point accumulation capabilities of the snake.
5. Stalling functionality for the snake. If added, it may greatly maximize the point accumulation capabilities of the snake. However, this was not implemented as there is no clear distinction that the point accumulated by the snake at any given run is solely due to the search algorithm or due to the stalling functionality.

#### Terminologies and clarification for the algorithm explanation

To preface the terms used in the upcoming explanation sections on both algorithms - informed and uninformed, additional clarifications on some of the common terminologies used are listed as follows:

1. Goal test is performed during the node generation phase.
2. Parent refers to the parent node whereas child/children refer to the child/children node(s).
3. Nodes in the explanation refers to the location of the snake's head in the maze
4. Nodes were not used in the algorithm implementation; however, this term is used for universal clarity on the algorithm process

5. State refers to the current location. For instance, the initial state of the agent will be the spawn point of the snake.
6. When a node is added to the frontier and search tree, it is implied that the heuristic function information is also performed and added as well.
7. The algorithm also has other functionalities such as terminating when there are no possible actions left. However, it is left out of the algorithm explanation for simplicity's sake.
8. As for the **informed search algorithm**, it utilizes a heuristic function. In this scenario, the heuristic function is the estimated distance between the snake's head and the food.

#### Algorithm explanation:

The BFS algorithm implementation steps are listed as follows:

1. Add the initial node to the frontier and search tree.
2. Iterate through the nodes in the search tree to locate the frontier node.
3. Mark the frontier node as explored and delete the frontier node. Save the state of the frontier node.
4. Perform goal test. If the state of the frontier node is the goal, the algorithm is terminated and one can proceed to step 8. Otherwise, proceed to step 5.
5. Generate the children of the explored node and add the children into the frontier and the search tree.
6. Update the information of the parents of the newly generated children in the search tree.
7. Repeat steps 2 to 6 until the goal is found or it is impossible to reach the food location in the current problem space.
8. Return both the solution and the search tree.

The GFS algorithm implementation steps are listed as follows:

1. Add the initial node to the frontier and search tree.
2. Iterate through the nodes in the search tree to locate the frontier node.
3. Mark the frontier node as explored and delete the frontier node. Save the state of the frontier node.
4. Perform goal test. If the state of the frontier node is the goal, the algorithm is terminated. Proceed to step 9. Otherwise, proceed to step 5.

5. Generate the children of the explored node and add the children into the frontier and search tree.
6. Sort the frontier with newly added children based on the estimated distance heuristic function so that the shortest distance node will be on the top of the frontier stack.
7. Update the information of the parents of the newly generated children in the search tree.
8. Repeat steps 2 to 7 until the goal is found or it is impossible to reach the food location in the current problem space.
9. Iterate through the explored list in reverse. Use the directions required for the initial state to reach the goal as the solution.
10. Return both the solution and the search tree.

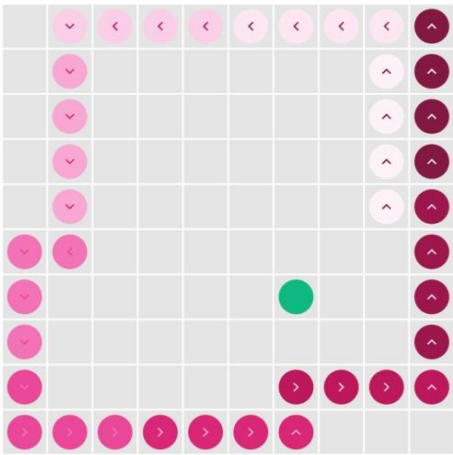
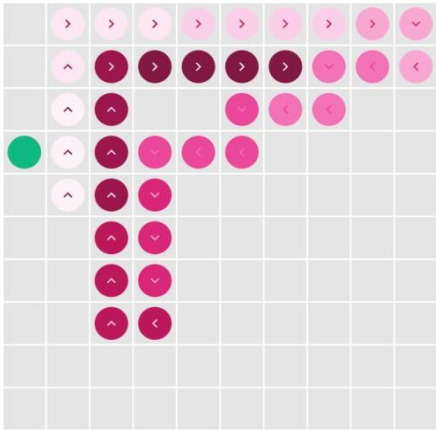
## **Results**

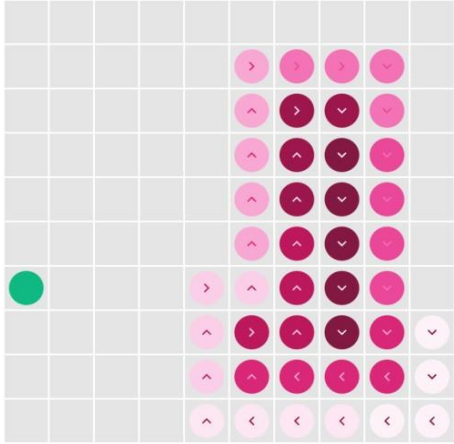
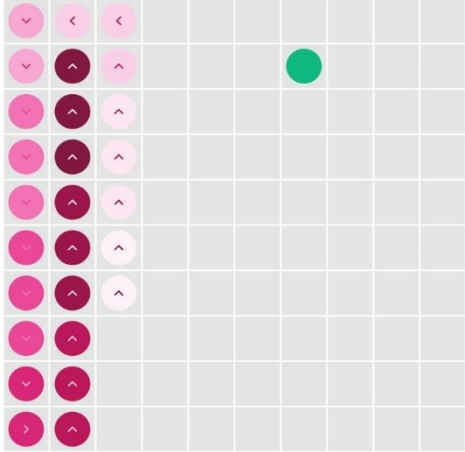
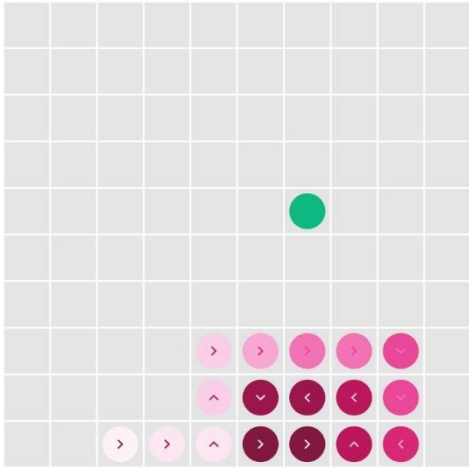
Results for requirement 1 were not recorded due to 2 reasons. Firstly, it is impossible for the game to actually end if the snake does not grow in length. Secondly, the only data that is procurable from this requirement - time taken for each point - is irrelevant and not considered in this project. Therefore, only the results of each agent completing requirements 2 & 3 are recorded. The average accumulated points for each test category was recorded across 10 test runs with a maze size of 10 by 10.

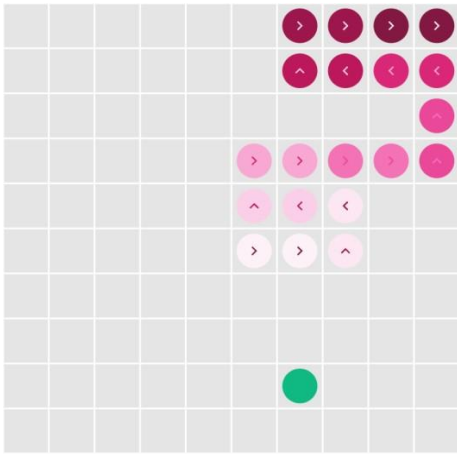
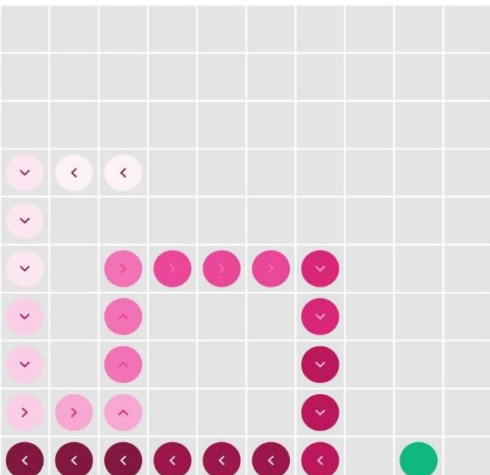
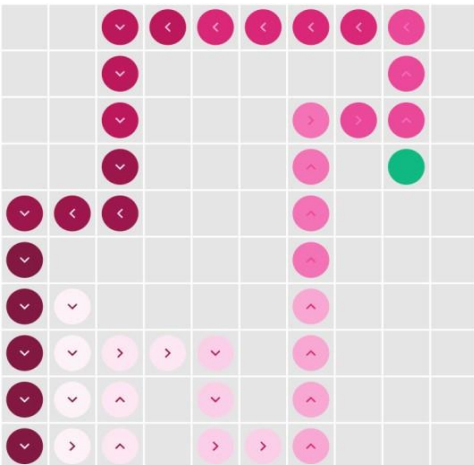


## Requirement 2

Agent 1- BFS

Run number	Results	
	State	Accumulated points
1		39
2		36

3		42
4		26
5		16

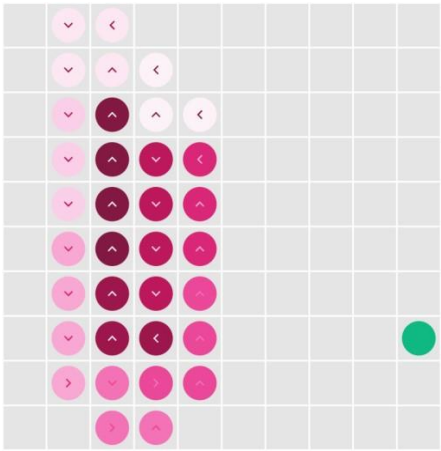
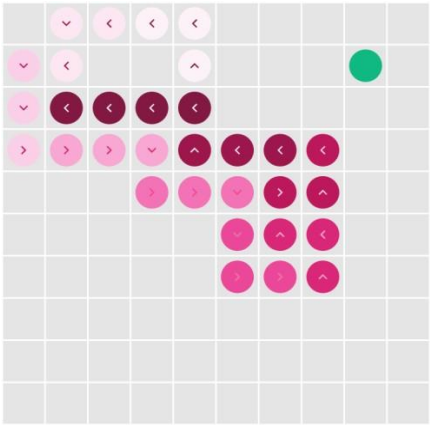
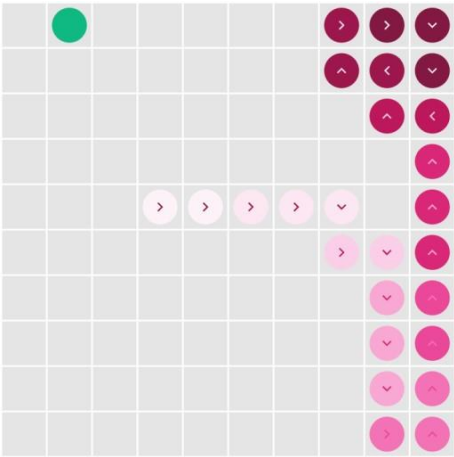
6		19
7		26
8		40

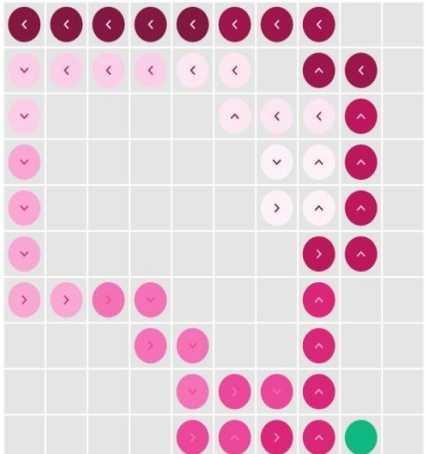
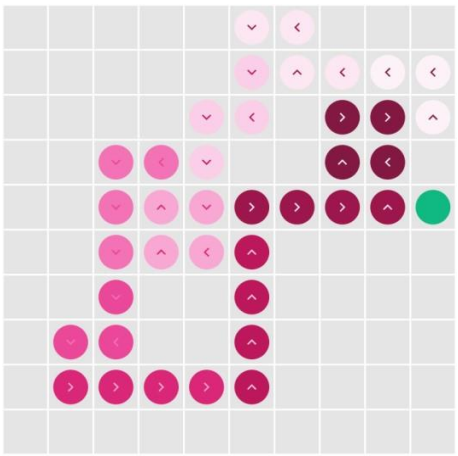
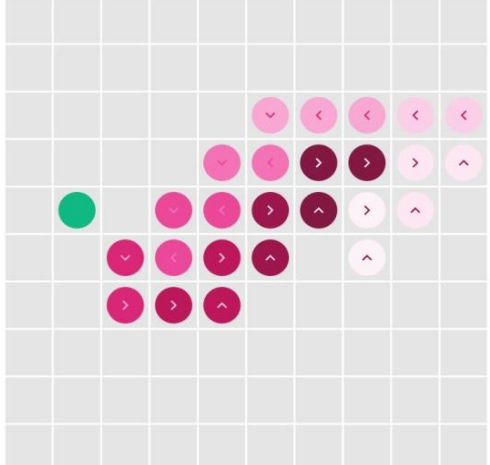
9		33
10		25

Table 1: BFS results with 1 food

Agent 2 - GFS

Run number	Results	
	State	Accumulated points
1		22

2		34
3		30
4		25

5		47
6		37
7		24

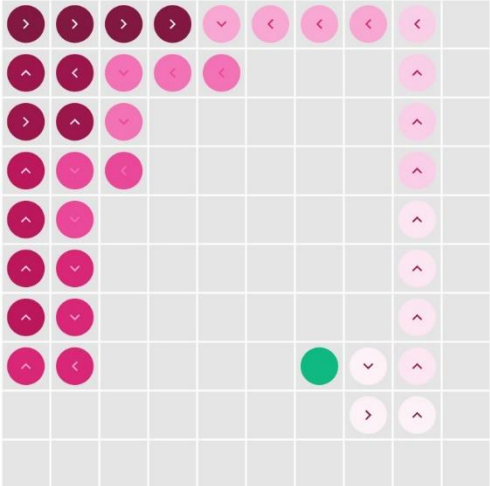
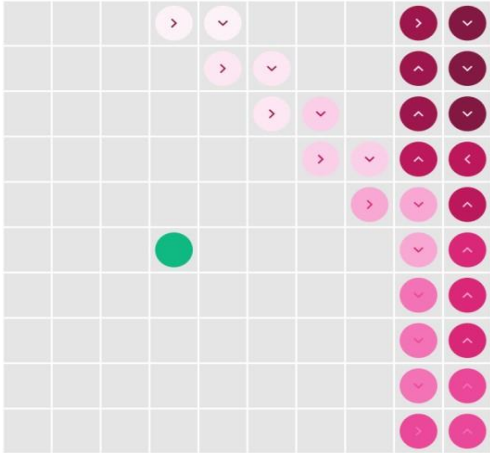
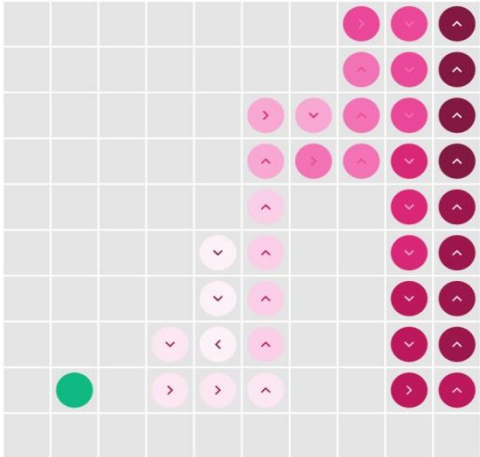
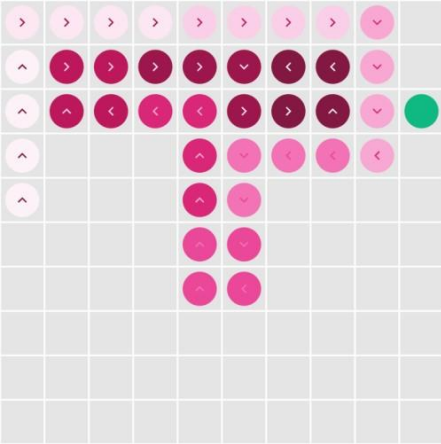
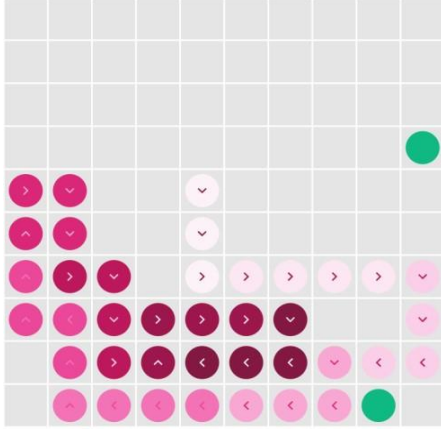
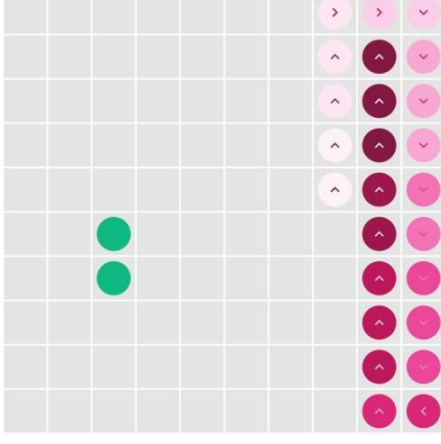
8		37
9		28
10		36

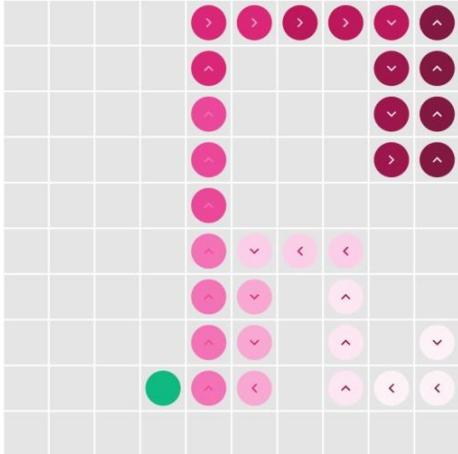
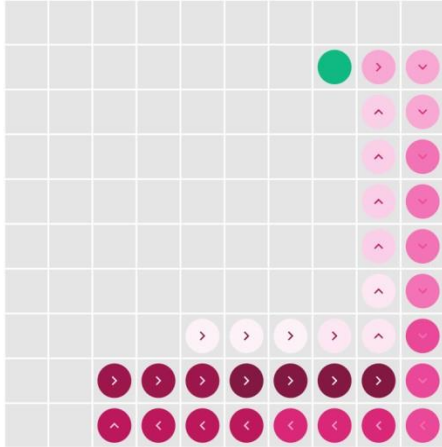
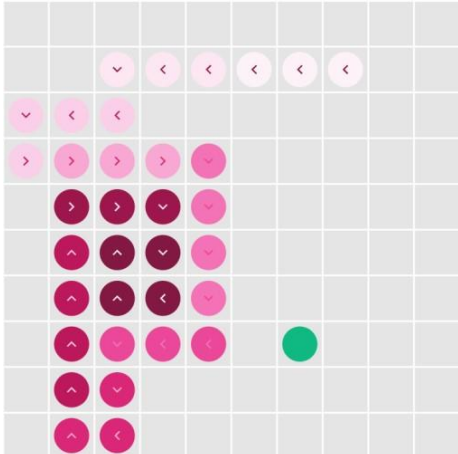
Table 2: GFS results with 1 food

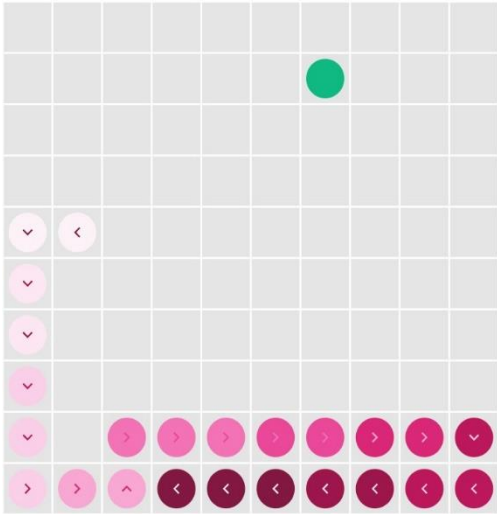
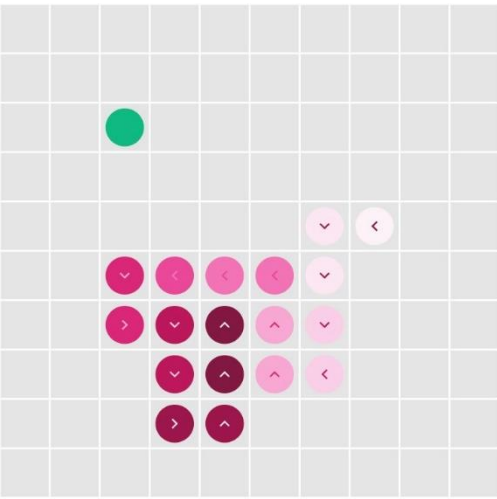
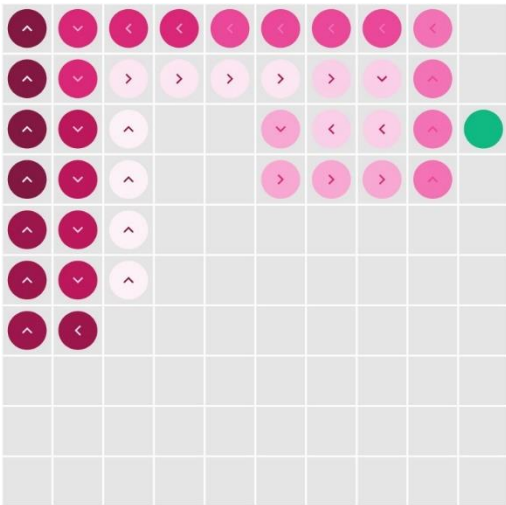
### Requirement 3

Agent 1- BFS

Run number	Results	
	State	Accumulated points
1		39
2		38
3		24



4		31
5		33
6		33

<p>7</p>	 <p>A 10x10 grid. A green circle is at row 5, column 7. A cluster of pink circles with arrows is at the bottom left, spanning rows 6-8 and columns 1-10. The arrows point in various directions: up, down, left, right, and diagonally.</p>	<p>23</p>
<p>8</p>	 <p>A 10x10 grid. A green circle is at row 3, column 2. A cluster of pink circles with arrows is at the bottom right, spanning rows 4-7 and columns 3-7. The arrows point in various directions: up, down, left, right, and diagonally.</p>	<p>17</p>
<p>9</p>	 <p>A 10x10 grid. A green circle is at row 7, column 10. A large cluster of pink circles with arrows is at the top left, spanning rows 1-7 and columns 1-10. The arrows point in various directions: up, down, left, right, and diagonally.</p>	<p>39</p>

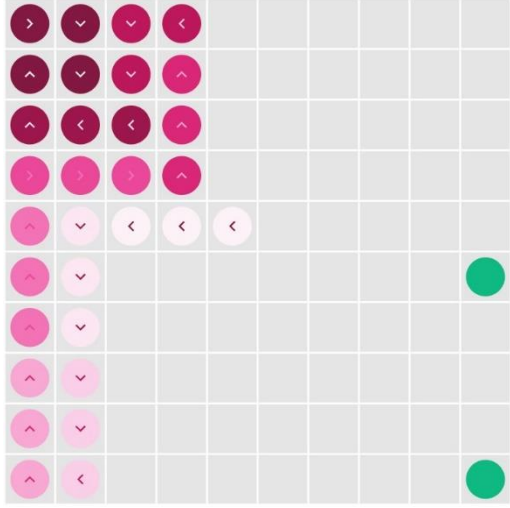
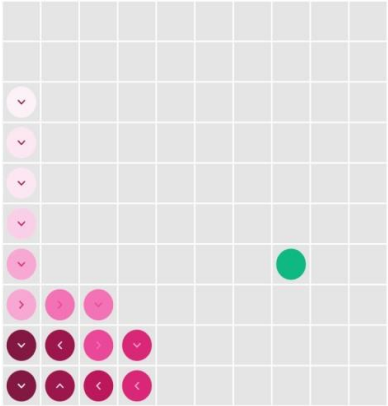
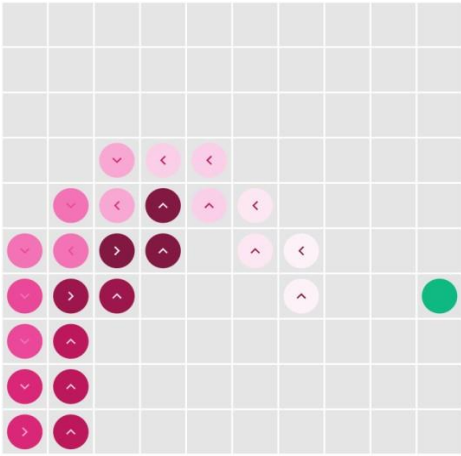
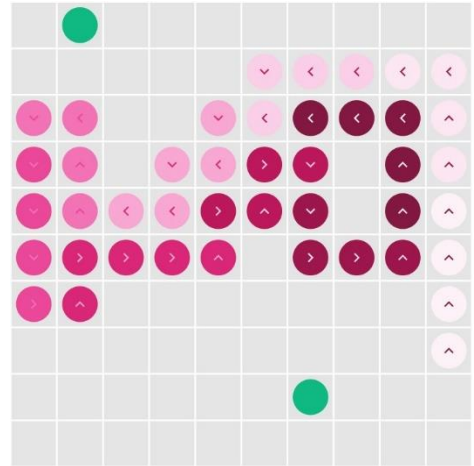
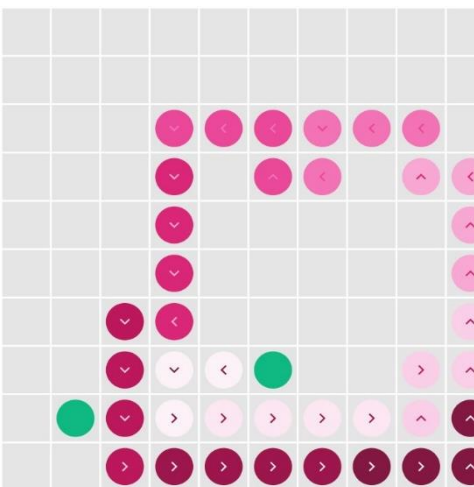
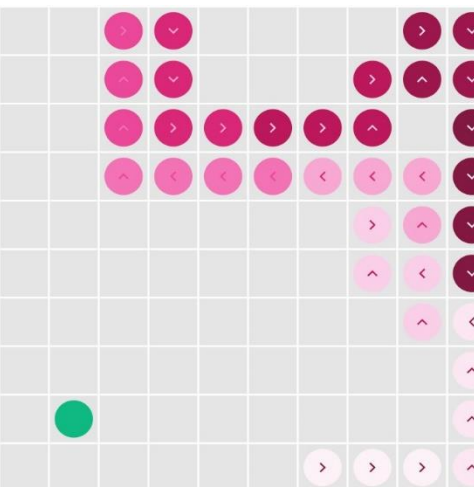
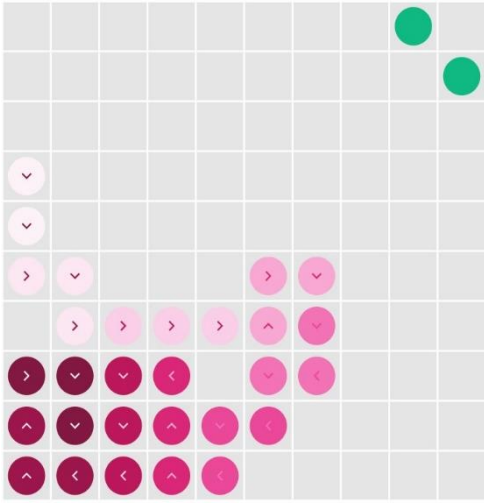
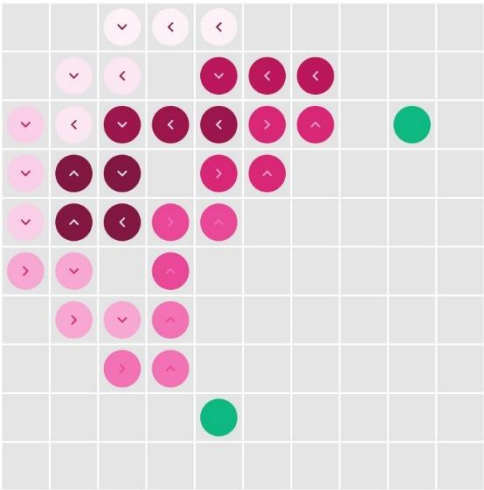
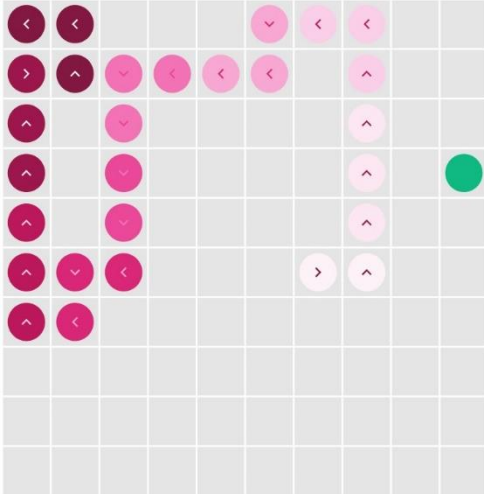
10		30
----	---	----

Table 3: BFS results with 2 food

Agent 2 - GFS

Run number	Results	
	State	Accumulated points
1		15
2		23

3		42
4		38
5		37

<p>6</p>		<p>28</p>
<p>7</p>		<p>32</p>
<p>8</p>		<p>27</p>

<p>9</p>		<p>39</p>
<p>10</p>		<p>21</p>

Table 4: GFS results with 2 food

## Discussion

The following table documents the findings for the 10 runs for each category. Additionally, the average of each category as well as its average standard deviation has been calculated and recorded.

Run Number	Requirement			
	2		3	
	Agent 1	Agent 2	Agent 1	Agent 2
1	39	22	39	15
2	36	34	38	23
3	42	30	24	42
4	26	25	31	38
5	16	47	33	37
6	19	37	33	28
7	26	24	23	32
8	40	37	17	27
9	33	28	39	39
10	25	36	30	21
<b>Average</b>	<b>30.2</b>	<b>32</b>	<b>30.7</b>	<b>30.2</b>
<b>Std. dev.</b>	<b>8.62322</b>	<b>7.266361</b>	<b>7.02922</b>	<b>8.42378</b>

Table 5: Result tabulation

Based on the table above, the average for all 4 categories is 30.775, which approximates to 31. Additionally, the volatility of the algorithm has been accounted for through the measurement of the average standard deviation of test runs across each category. Theoretically, if the standard deviation discrepancy is extreme, it will imply that the snake either has gotten very lucky - scoring upwards of 40 points - or it has gotten very unlucky - scoring downwards of 10 points - on the majority of its runs. Therefore, the ideal scenario is where the standard deviation across each category is very similar to each other to validate the results obtained to be fairly consistent. According to Table x, the average standard deviation of all 4 categories is approximately 7.8 with the maximum and minimum standard deviation to be at 7.02 and 8.42 respectively.

There are several possible theories as to why the algorithms score so similarly to each other in terms of accumulated points. Firstly, it is very likely that although both algorithms have noticeable

differences in computational efficiency when testing for data collection, both algorithms function very similarly. The only noticeable difference between both algorithms is the way it plots the path for the snake to travel - one being horizontally or vertically and the other being primarily diagonally as shown in Figures 2.1 & 2.2 and Figures 3.1 & 3.2. Therefore, it is unsurprising that both algorithms will achieve very similar scores. Secondly, perhaps there is indeed a bigger discrepancy in terms of accumulated points for both algorithms. However, due to the design of the algorithms having similar pitfalls as shown in Figure 1.1 and 1.2, the algorithm by default would end similarly to each other. In other words, if the algorithm would enable the snake to live longer, perhaps there could be a difference in scoring.

The figures below depict an observation about the pitfalls for both BFS and GFS algorithms. As shown in Figure 1.1, the snake mindlessly goes for the food based on the black arrows. The red region indicates the tail of the snake. Despite being able to navigate around the maze while accounting for its tail, the snake cannot account for the possible entrapment that may soon follow, marked by the blue line. Predictably, Figure 1.2 shows the snake trapping itself with no ways of escape. This occurs primarily due to the snake only paying attention to the distance between its head and the food and disregarding any other information in the environment - such as considering a path of escape for each food. Due to this phenomenon, the snake most definitely will get itself stuck when the food is spawned in such a way where there is only 1 way of entry and no way for escape - corners for example.



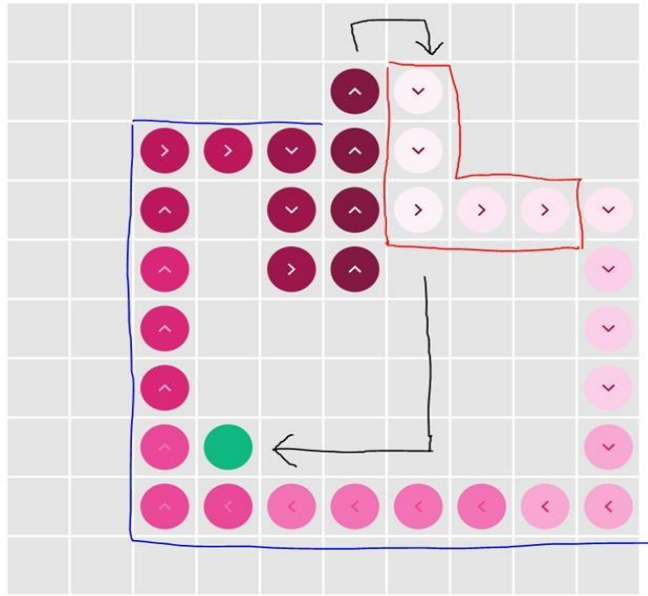


Figure 1.1: Snake's decision-making

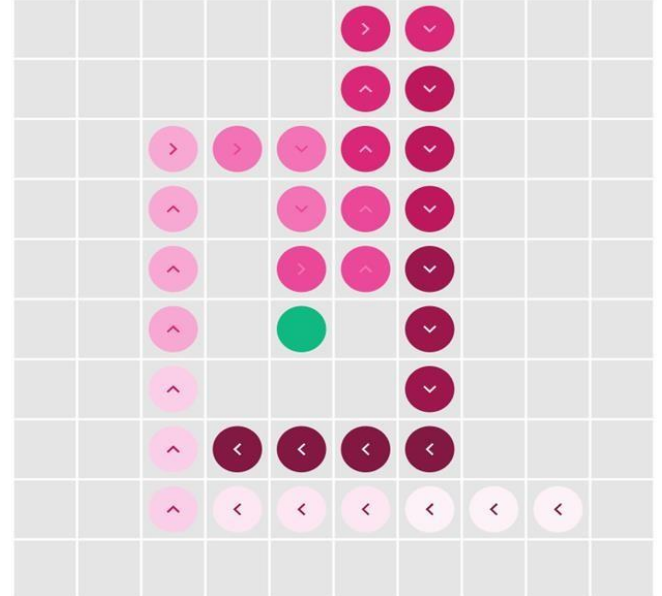


Figure 1.2: Snake trapping itself

The below figures shown are to showcase the observations made on the movement behaviour of the respective search algorithms.

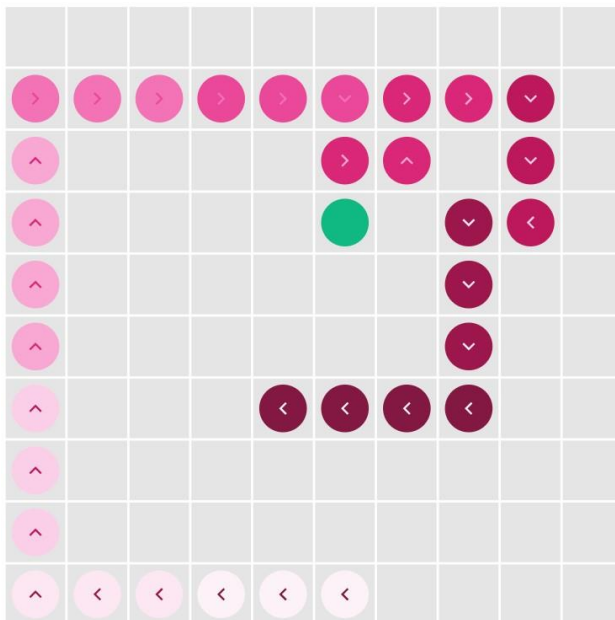


Figure 2.1: BFS - before moving

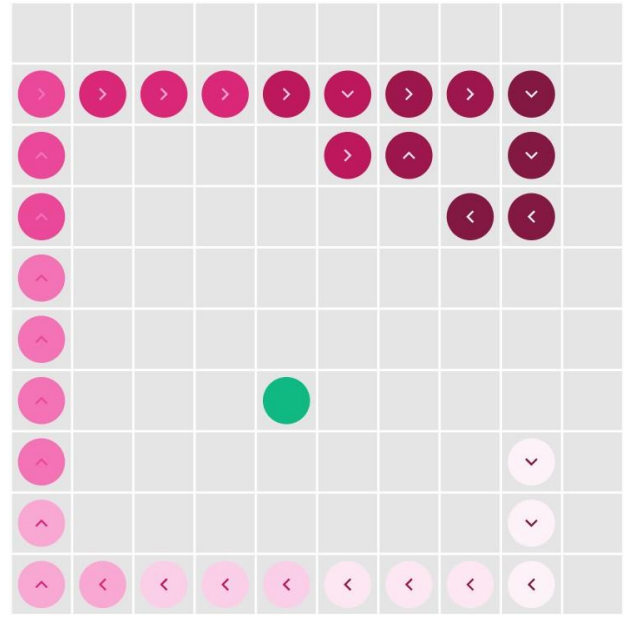


Figure 2.2: BFS - after moving

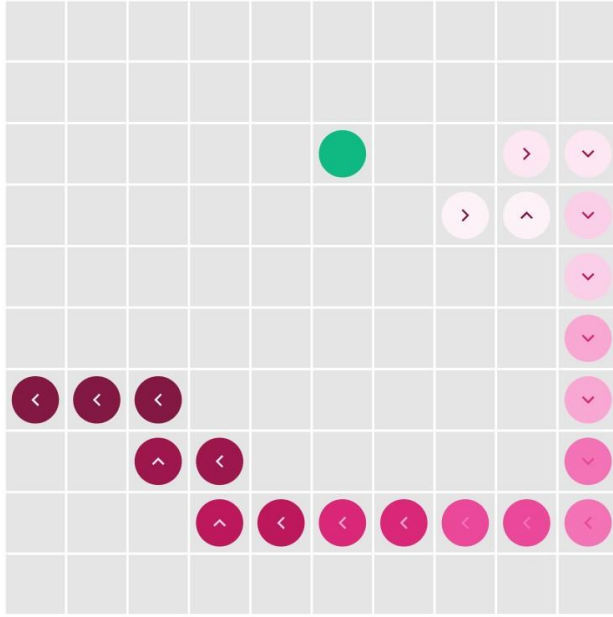


Figure 3.1: GFS - before moving

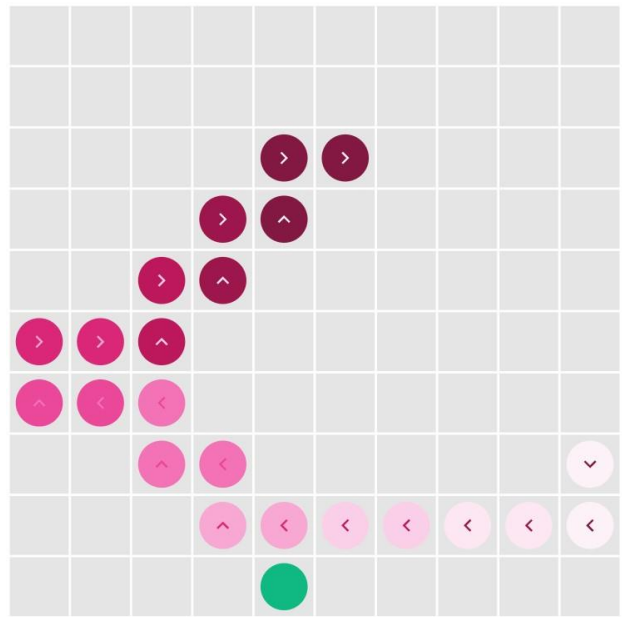


Figure 3.2: GFS - after moving

### Recommendations

Further improvements can be made to both search algorithms to maximize point accumulation each run. One of which is to include a stalling function. The idea of this function is to make the snake follow its tail and/or maximize its survivability by travelling all possible states if it were to trap itself. Secondly, a detection measure should be included where the snake can calculate whether there is both a way to the food and an escape after eating the food. With this detection measure, scenarios where the snake bites itself due to beelining into a corner - a very prone scenario - can be heavily mitigated.