

# Project Documentation for Java

February 10, 2023

**PROJECT: SNAKE AI**

**PROFESSOR: Prof. Logofatu**

## Table of Contents

1. Team Work
2. Introduction
3. Problem Description
4. Related Work
5. Proposed Approaches
6. Implementation Details
7. Experimental Results and Statistical Tests
8. Conclusions and Future Work

### Declaration of Authorship

We hereby certify that the project report we are submitting is entirely our own original team work, except where otherwise indicated. We are aware of the University's regulations concerning plagiarism, including those regulations concerning disciplinary actions that may result from plagiarism. Any use of the works of any other author, in any form, is properly acknowledged at their point of use.

Parth Gajera, 1380680

Devansh Kumar, 1380664

Ekta Balhotra, 1379109

Nancy Balar, 1398122

# 1 Team Work

## Team Members

1. Parth Gajera (1380680): Sorting out the main algorithm for the game
2. Devansh Kumar (1380664): Designing the Graphical User Interface
3. Ekta Balhotra (1379109): Database Integration and the final project documentation
4. Nancy Balar (1398122): Database Integration and final project documentation

## Work Structure and Brain Storming

In the first week of the project, after we were assigned our topics we discussed the basic structure of the game and how we are going to proceed with developing it. We divided our tasks into different departments for instance Algorithm Sorting, Designing of the Graphical User Interface, and Database Integration, and one or two team members were assigned to it depending upon the difficulty level.

We had our weekly Team Review Meetings every week on Thursdays either in person or over Discord to check our progress in the project and weekly tasks were assigned to each one of the team members and reviewed every week so that the workload can be balanced. All of us were constantly brainstorming for the ideas of how we can develop this game in the best way possible regardless of the department we were assigned to and all these ideas and approaches were discussed in a very healthy way in the weekly Team Review Meetings.

In case of problems, that we faced along the way, all the team members discussed them and came up with collective solutions which helped us overcome the obstacles in the most optimal way.

In the first week of our brainstorming, we decided to approach the problem in the most basic way, which was by thinking about what exactly is the basic idea behind the game. The basic idea behind the game was to find the shortest distance between the apple and the snake. In the previous semesters, we have learned a lot of Algorithms that can be used to find the shortest distance between the given two points some of which have also been discussed in the later sections. So each one of us started implementing these Algorithms in order to find the shortest distance in the most optimal way and the team member who was successful in finding the most optimal way took the task further by involving Artificial Intelligence in it. So after the first week of brainstorming on the right Algorithm, we finalized A\* as our main approach for solving this game.

In the second week, we continued helping each other in sorting out the Algorithm further while implementing Artificial Intelligence to it which was one of the biggest challenges faced by us in the whole project. In the third week, one of our teammates started working on the Graphical User Interface of the game, because after the Algorithm it is the second most important thing while designing this project because, in case of designing a game from the scratch, it is very important that it looks interesting enough for the user to play it again and again.

In the fourth week, we were done with sorting out the Algorithm for single-player and multiplayer but implementing Artificial Intelligence was still a work in progress. Alongside, the Graphical User Interface was also designed and finalized for both the single-player and the multiplayer mode so considering the second week of the project we were quite in a good position and highly motivated to proceed further.

Since the concepts of how to make a Graphical User Interface were completely new for us as beginners so we had to also learn all the concepts from the start in order to implement them in our project. For building the Graphical User Interface, we mainly focused on learning the concepts of Java Swing and AWT which were quite interesting and challenging at the same time.

In the fifth week, we were almost done with sorting out our Algorithm with the implementation of Artificial Intelligence too and now we thought of integrating a Database into it in order to display and update the high scores. Again considering ourselves as beginners in regard to Java as a programming language, we started learning the concepts of integrating a Database into Java which was quite hard for us to get a hang of considering the increasing pressure due to the passing time but we did it eventually and at this point, we were done with almost everything that we needed for our project, the only thing that was left was integrating everything together which was done in the sixth week.

In this week we particularly worked on integrating everything together and also started working on our Java Documentation side by side. Apart from integrating things together, we also focused on the betterment of our Algorithm and Graphical User Interface. We also introduced sound effects to our game in order to make it even more interesting for the user to play.

No doubt the combined effort of our time helped us meet the deadlines on time, but one another thing that actually helped us improve our project and quenched our thirst to learn new things was the feedback talks that were organized for us. Not only did it help us keep a check on time but the valuable feedback from our professor helped us in improving our project a lot by showing us where we were lacking and how we could improve it further and the idea of showing our progress through an impromptu presentation also helped us boost our confidence and prepared us well for our final presentation

## 2 Introduction

### General Project Idea

Snake is one of the most popular games that all of us must have played at some point of time in our childhood, so everyone is quite clear about what the basic idea of the game is, still we would like to explain what the basic idea behind this game is.

It includes a snake that moves around in a grid searching for an apple, and each time it eats an apple, the player scores a point and the body of the snake increases in its length.

The catch here is that the snake should not touch the sides of the grid or into its own body otherwise the person will lose the game. It becomes interesting to play because the length of the body increases as the player progresses in the game.

Our project was not only designing this simple game but also introducing Artificial Intelligence to it, which means the snake will move on its own, finding the shortest way possible to eat the apple

During our brainstorming sessions, we thought of introducing three modes in our game namely, Single Player, Multiplayer, and Artificial Intelligence

In the single-player mode, as the name suggests, a single player will control the snake in order to grab the apple increasing his score. This mode is not implemented with Artificial Intelligence, hence the player himself will control the snake, giving it directions by clicking on the suitable keys which were hard coded.

In the multiplayer mode, two players play against each other, and the one with the highest score in other words the user controlling the snake which has eaten most of the apples wins the game. Since there were two players competing against each other, so two snakes were required in this case

Lastly in the Artificial Intelligence mode, we integrated our Algorithm with Artificial Intelligence, which means there were two snakes on the panel but they were not controlled by the user as in the earlier cases, rather they were moving on their own according to the coded Algorithm and competing against each other in finding the shortest way to the apple

Apart from the different modes, a high score list with a separate panel was also included in the game, which at the end of the game displayed the score of the player playing the game and also updated the high score list when needed

### 3 Problem Description

#### **Finding the right Algorithm**

1. Find the shortest distance (Continuously changing)
2. Considering Obstacles which are moving in every single Frame which being refreshed
3. Array-List of Frame Pixels (2D)
4. Snake body(2D)
5. Apple Coordinate(2D)
6. Boundaries
7. To solve out whether neighbors are free as well as inside our Frame
8. Choose the best possible Neighbor
9. Array-List of Snakes, coordinate of apple are continuously changing
10. Developing own Datatype of neighbors and Snake Manhattan Distance
11. Give direction according to best possible neighbor of snake head
14. Create Objects of all classes and do with OOP Concept with help and class

and related method

The main problem faced while developing the Snake Game was to find the shortest distance possible from snake's head to the apple considering all the obstacles which are there, such as the second snake, boundaries of our Game Panel. So, we started by analyzing different algorithm based on path and cost to reach target node, which includes:

Algorithm	Description	Complexity
1. Dijkstra Algo	This algorithm solves the shortest path from One node to another which have a non-negative edge.	$O(V^2)$ (V - Vertices)
2. Bellman-Ford Algorithm	It works similar to Dijkstra, but Nodes may be negative edge.	$O(EV)$ (V - Vertices) (E - Edge)
3. A* Algorithm	It helps to find a shortest path considering all Obstacles, with help of using Heuristics.	$O(b^d)$ b = branching factor d = number of nodes on the path

As Snake is continuously moving in our Frame, it was quite a challenge to hold the X-Coordinates and Y-Coordinates, so we decided to use the concept of Array-List. Because the inbuilt Datatypes are integer, float, double, etc and we saw that it is not a perfect fit for our Snake Coordinates, so we decided to create our own Snake class and use that class as a DataType of ArrayList.

```
90 ArrayList<Snakebody> snakes = new ArrayList<>();  
    5 usages  
91 ArrayList<Snakebody> snakes2 = new ArrayList<>();  
92
```

For understanding the Algorithm, we first learned about the basic functionality of A\* Algorithm which includes:

Manhattan Distance: this is the basic shortest path, not considering any Obstacles

To check the cost of that path:  $g(n)$

To check the cost of the remaining path:  $h(n)$

To pick out the minimum  $f(x) = g(n) + h(n)$  and explore that until we get the

targeted node

Last problem was to make a Class of neighbors which holds all attributes of neighbors and that includes X-Coordinates, Y-Coordinates, Parents of that neighbor and Manhattan Distance.

## **Graphical User Interface**

### **Learning Swing**

The Swing API is a collection of expandable GUI components that makes it easier for developers to create front-end/GUI Java applications. Since it has almost all of the controls that are equivalent to AWT controls, it is built on top of the AWT API and functions as a replacement for the AWT API.

Swing offered a wide range of sophisticated controls, including table, slider, color picker, tree, and TabbedPane controls. Depending on the available values, the look and feel of a Swing-based GUI application can be modified in real time.

### **Learning AWT**

Even though Swing has replaced AWT but we still had to use AWT to develop GUI. It is platform independent therefore Components are displayed according to the Operating system we are using.

The `java.awt.*` provides classes for `TextField`, `Label`, `TextArea` but for development of our game, mostly we used Swing, unless we wanted to use `ActionListener`.

### **Learning ActionListener**

Whenever we created a button and it was clicked, it was detected by the class `ActionListener`. Officially the Interface that is imported is `java.awt.event.ActionListener`; It has only 1 method i.e `public void actionPerformed`

### **Learning javax.sound.sampled**

When we start the game, a background music is played, which, was a trouble for us since we did not really have any experience handling Sounds in Java.

After Stack Overflowing a few hours and after reading through Java Oracle website we found out `import javax.sound.sampled.*`; This package provides an interface for the capture, mixing digital audio.

But just learning about `javax.sound` was not enough, because we were not able to access the `Audioclips` just through that. Therefore we had to use `import java.io.File`; Since during the first 4 weeks, we did Kattis and other Exercise Questions, we were able to learn about File management and were able to implement file management without any difficulties.

### **Learning MyKeyAdapter**

In our Game Project, we implemented 3 Modes of Gameplay, `SINGLEPLAYER`, `MULTIPLAYER`, `ARTIFICIAL INTELLIGENT`. In order to implement single-player and multiplayer, we had to give the user the access to keyboard and its keys.

User had to guide the snake to the Apple themselves, with the help of UP, DOWN,

LEFT, RIGHT keys. In order to do that the class MyKeyAdapter which extended KeyAdapter was used with the method public void keyPressed(KeyEvent e). This method is implemented under @Override. In order to understand it better a code snippet is added below.

```
public class MyKeyAdapter extends KeyAdapter{
    @Override
    public void keyPressed(KeyEvent e){

        switch(e.getKeyCode()){

            case KeyEvent.VK_LEFT:
                if(direction != 'R') direction = 'L';
                break;
            case KeyEvent.VK_RIGHT:
                if(direction != 'L') direction = 'R';
                break;
            case KeyEvent.VK_UP:
                if(direction != 'D') direction = 'U';
                break;
            case KeyEvent.VK_DOWN:
                if(direction != 'U') direction = 'D';
                break;

        }

    }
}
```

In terms of multiplayer, the basic logic of working of the both the snakes was same. The only different thing was. there were more switch(e.getKeyCode()) cases.

## Database

### Connecting Database with the Application

As a basic plan we first started to show the scores on the Terminal of the IDE for all the cases like “Single Player”, “Multi Player” and “AI”. After being successfully able to show the score for all the different types of modes of the game, we finally decided to use Database for our Application. For this initially, we created a database by creating a schema in the Workbench platform, which was again not that difficult, but the real challenge was connecting that database with our main Java code. To connect the database with the main Java code, we

had to use the concept of JDBC Driver (From the Java programming language, there is global data access through the Java Database Connectivity (JDBC) API. You can access practically any data source, including relational databases, spreadsheets, and flat files, using the JDBC API). In order to connect to the Database we have to type in the URL (Database URL) which includes IP address and port number. After building the connection the next issue was to know the methods that are used to call the functions of the database because the connection established between the Java code and the Database was of no use to us if we did not know how to call the functions of our Database in our main code. After following all the above steps, a connection was formed with the Database and with the help of SQL query statements, we were able to get the required out on the “Scoreboard” Panel. The next hurdle that we were facing after sorting out the connection issue with the main code was in the case when a single player with the same name was added to the list of “Scoreboard”. When we were trying to add the Player with the same name, the new entry was showing again as a separate row in Database. But as per requirements, we wanted it to update the score instead of creating a new entry. For this issue we tried to figure out a solution for it. We created if – else conditional loop to check whether the Player with the same name already exist in the database or not. If the player already exists in the Database, the query will try to grasp the score of that Player, and it will try to update it with new score. At the end there was an error, which was highlighting the Result Set (The result set is an object that symbolizes a collection of data that is typically returned in response to a query from a data source. The requested data components are held in rows and columns of the result set, which can be moved around using the cursor.). I was unaware about the situation that one needs to close the Resultset and Statement separately although the Connection is closed afterwards. For example, if for some reason you are using a ”primitive” type of database pooling and you call “connection.close()”, the connection will be returned to the pool and the ResultSet/Statement will never be closed and then you will run into many different new problems! So you can’t always count on “connection.close()” to clean up. It is wise to always explicitly close ResultSets, Statements and Connections when you are finished with them as the implementation of close could vary between database drivers.

## **Integrating the GUI and the Database**

While integrating the Graphical User Interface with the Database, the major problem that we faced was not knowing where to call the Database class in the main code, due to this we were having trouble, displaying the scores of the players because as a result of this problem, the functions of the database were not being called properly and hence there were a lot of errors encountered while displaying the score

Another hurdle that we faced, while we were integrating this Database with the Graphical User Interface in our code was fetching the data from the database



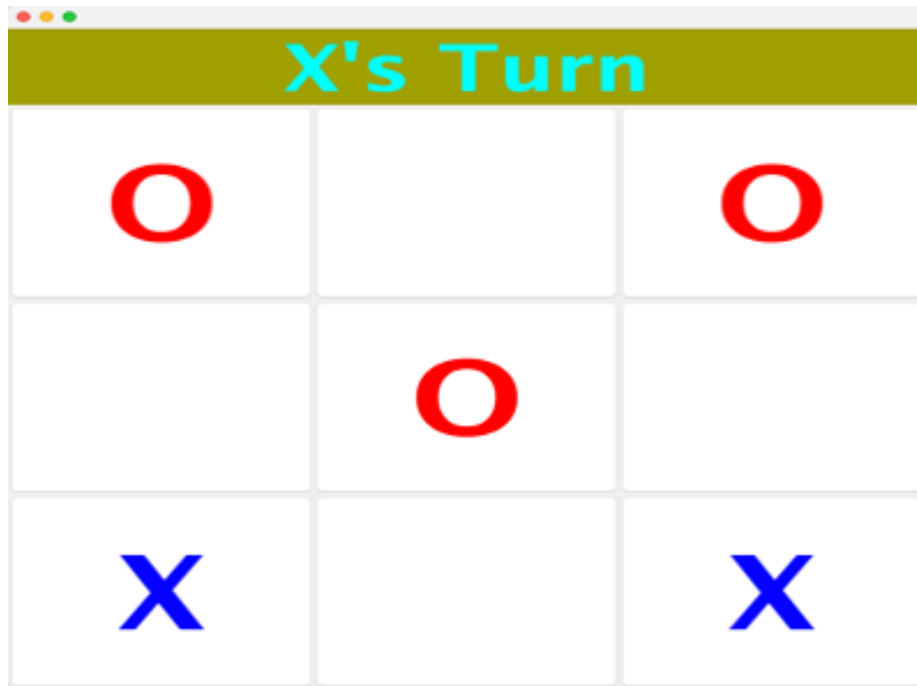
and showing the desired output on our GUI instead of the terminal. For instance, if a player played this game, at the end his high score must be stored as an entry along with his name in our database, and then this entry must be shown as an output on the Graphical User Interface of our Score Board Panel

## 4 Related Work

Since we all were beginners in Java, even after understanding java SWING, AWT, ACTIONLISTENER, we were not good enough to start making a snake game already.

### **TicTacToe**

In order to gain some experience about using swing and making GUI, we started with TICTACTOE game. After gaining some more experience with java Swing we started with snake AI.



We learned about the different types of Layouts, that a Panel has, and how to align the containers that we add on the Panel and how to customise them. A code snippet is shared below, that clears how the containers are added to the Panel. And how they are customized.



```

Java
Copy Caption ...

for(int i = 0; i < 9; i++){
    buttons[i] = new JButton();
    button_annel.add(buttons[i]);
    buttons[i].setFont(new Font("MV Boli", Font.BOLD, 120));
    buttons[i].setFocusable(false);
    buttons[i].addActionListener(this);
}

```

We started with a simple UI which was made just using JFrame, JPanel and some of the other classes from Swing package. Even after developing this application we were not able to develop the snake Game. Not only that, it was still difficult for us, to even create a snake. Creating a snake happened with the help of Graphics class and with the help of JPanel class, which has a public void paintComponent method, which can @Override. With the help of this method, we were able to initiate the drawing process of the Snake, Background, and apple the spawn on the screen at random location.

So as we have already mentioned in the above sections, the basic idea of this game is, that the snake finds the shortest path to reach the apple every time the coordinates of the apple change since they are generated randomly. In order to find the shortest path, we came across three suitable algorithms which we could implement in our game. These Algorithms have been discussed below:

### 1. A completely randomized Algorithm

Since our project involved implementing Artificial Intelligence in the basic Snake game, that meant that not only the coordinates of the apple but the coordinates of the snake's next move should also be generated randomly. For that, we thought of this completely randomized Algorithm in our early stages. In this Algorithm, the snake is given the initial coordinates and the Artificial Intelligence selects the next valid cell randomly, which makes the snake move randomly inside the grid.

The biggest disadvantage here is, the coordinates of the apple (which are also generated randomly) play no role in finding the next coordinates of the snake, and hence it keeps moving until an apple is found and the game usually ends up with a score of 1 or occasionally with a score of 2. Since this method is completely randomized and takes a lot of time till an apple is found, it is one of the least effective Algorithms that can be used while developing the game of Snake AI

### 2.Shortest Path BFS - Breadth First Search

Breadth First Search is a graph transversal algorithm, even though it is not one of the shortest path-finding algorithms, but still can be considered as one of the options while developing the Snake AI game

In this algorithm, the root node is the head of the snake, the root checks for the available neighboring nodes which are nothing else but positions on the grid at the present depth before going deeper. It terminates when the fruit is found. It is obviously better than the first discussed algorithm since the algorithm is finding the coordinates of the apple as it goes deeper, but it still has a major disadvantage that is, the length of the snake, which increases after each eaten apple is not taken into consideration, this means that the algorithm works perfectly well until the increasing length of the snake interrupts the shortest path to the apple.

```

procedure BFS(G, root) is
  let Q be a queue
  label root as explored
  Q.enqueue(root)
  while Q is not empty do
    v := Q.dequeue()
    if v is the goal then
      return v
    for all edges from v to w in G.adjacentEdges(v) do
      if w is not labeled as explored then
        label w as explored
        w.parent := v
        Q.enqueue(w)

```

### 3. Dijkstra Algorithm

A graph search algorithm called Dijkstra's algorithm determines the shortest route between two nodes in a graph. By determining the shortest route between the snake's current location and the apple's location, you may use it to make the snake go in the direction of the apple in a game of Snake. At the beginning of the Project we tried to solve the path according to this algorithm. The Dijkstra algorithm for the Snake game operates as follows:

Create a priority queue, and then add the snake's beginning location to it.

The beginning point should be noted as visited.

While there are still items in the priority queue:

- a. Remove the node from the queue with the lowest priority.
- b. Verify that the apple is the dequeued node. If so, we've discovered the shortest route.
- c. Adding its unvisited neighbors to the queue and marking them as visited if the dequeued node is not the apple. Based on the distance to the current node, update their distances.

Using the data kept in the nodes, the shortest route may be found from the apple to the beginning place. The snake will use this algorithm to move as swiftly as possible toward the apple while avoiding barriers like walls or its own body.

To better comprehend how Dijkstra's algorithm functions for the game of Snake, let's look at an illustration.

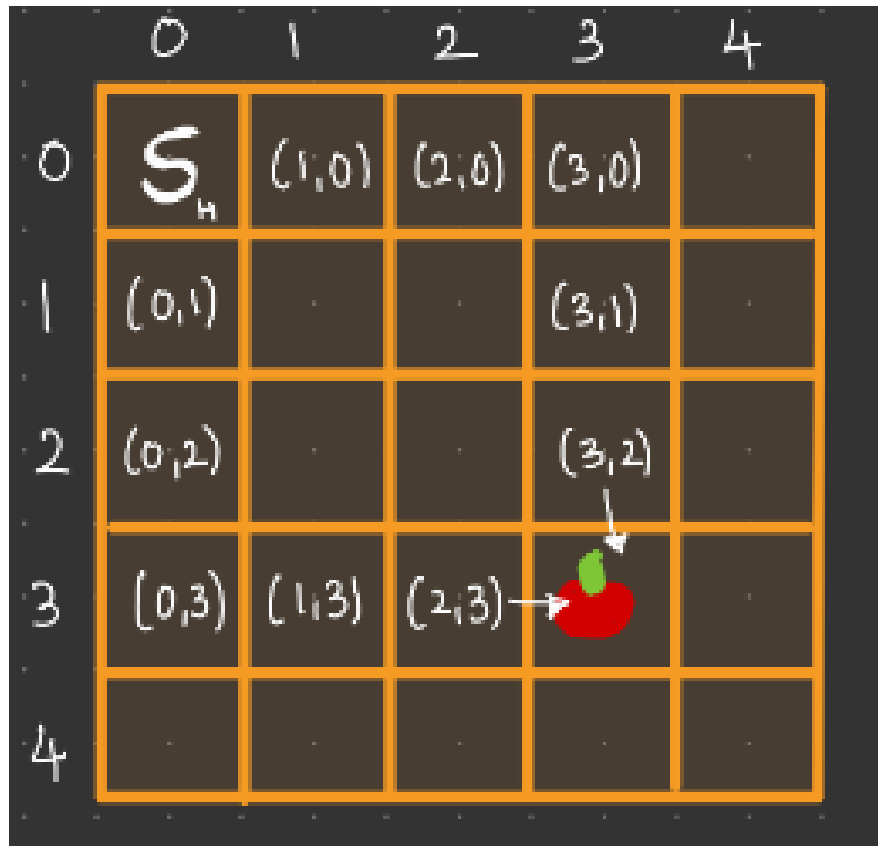
Think about a 5x5 2D grid where the snake starts at location (0, 0) and the apple is at place (3, 3). Each cell in the grid is a node in a graph that represents the grid.

1. Create a priority queue from scratch and add the snake's beginning point (0, 0) to it with 0. The number of steps needed to get to the food is represented by this distance.
2. The beginning point should be noted as visited.
3. While there are still items in the priority queue:
  - a. Remove the node from the queue with the lowest priority (and shortest distance). It is in this instance (0, 0).
  - b. Verify that the apple is the dequeued node (3, 3). If so, we've discovered the shortest route.
  - c. Add its unvisited neighbors to the queue and mark them as visited if the dequeued node is not the apple (3, 3). Depending on the distance to the current node, adjust their distances (0, 0). For instance, (0, 0)'s neighbors are (0, 1), (1, 0), and (1, 1). These neighbors' separations have been changed to 1.
  - d. Continue performing steps 3a to 3c until the apple is located or the line is finished.

Using the data kept in the nodes, the shortest path can be found from the apple (3, 3) to the beginning place (0, 0). The ultimate route would resemble this: (0, 0) -> (0, 1) -> (0, 2) -> (0, 3) -> (1, 3) -> (2, 3) -> (3, 3).

In a game of Snake, this is how Dijkstra's algorithm determines the shortest path. The algorithm makes sure the snake travels as quickly as possible to the apple while avoiding barriers like walls or its own body.

This could be illustrated with the help of the following diagram:



#### 4. A\* Algorithm

A\* is a graph search technique that calculates the shortest path between two nodes in a network while taking into account both the anticipated distance from the starting node and the actual distance to the target node.

Here is how the Snake game's A\* algorithm operates:

Create an open list (priority queue) from scratch and add the snake's beginning location to it. Nodes that have not yet been visited are found on the open list. To determine the starting place's F-value, which is the sum of the predicted distance to the target node and the distance from the starting node (G-value), mark the starting point as visited (H-value). Based on the cost of moving from the starting node to the current node, the G-value calculates the actual distance from the starting node. The predicted distance to the target node, known as the H-value, is determined using a heuristic algorithm. The Manhattan distance, which is the total of the distances between the current node and the target node in both the horizontal and vertical directions, is a typical heuristic function for a Snake game.

While there are items on the open list:

- a. Dequeue the node from the open list with the lowest F-value and designate

it as the current node.

b. Verify that the node in question is the food. If so, we've discovered the shortest route.

c. Adding the node's unvisited neighbors to the open list if the current node is not the food. They then calculate their F-values by multiplying their G-value by their H-value, which is updated dependent on the cost of moving from their present node to the neighboring node.

d. Keep going back to steps 3a through 3c until the meal is located or the open list is finished.

In a game of Snake, this is how the A\* algorithm determines the shortest path. The F-value, which combines a node's distance from the starting node and its predicted distance to the target node, determines which nodes are given the highest priority by the A\* algorithm. This makes it possible for the algorithm to more quickly and effectively determine the shortest route to the foods.

Let's take a simple example of a Snake game with a 3x3 grid, where the snake starts at (1,1) and the food is at (3,3). Here's how the A\* algorithm would work:

Initialize the open list with the starting position of the snake at (1,1) and mark it as visited. The F-value of the starting node is calculated as  $F = G + H$ , where  $G = 0$  (the distance from the starting node) and  $H = \text{Manhattan distance between (1,1) and (3,3)} = 2$ . So,  $F = 0 + 2 = 2$ .

Dequeue the node with the lowest F-value from the open list, which is the starting node (1,1), and mark it as the current node.

Check if the current node is the food. Since it is not, add its unvisited neighbors to the open list and calculate their F-values. For example, the neighbors of (1,1) are (1,2), (2,1), and (2,2). The G-value of (1,2) is 1 (the movement cost from (1,1) to (1,2)), H-value is 1 (the Manhattan distance between (1,2) and (3,3)), and F-value is 2. The G-value of (2,1) is 1, H-value is 2, and F-value is 3. The G-value of (2,2) is 2, H-value is 1, and F-value is 3.

Dequeue the node with the lowest F-value, which is (1,2), and mark it as the current node. Check if the current node is the food. Since it is not, add its unvisited neighbors to the open list and calculate their F-values. For example, the neighbors of (1,2) are (1,3) and (2,2). The G-value of (1,3) is 2, H-value is 0, and F-value is 2. The G-value of (2,2) is 2, H-value is 1, and F-value is 3.

Dequeue the node with the lowest F-value, which is (1,3), and check if the current node is the food. Since it is the food, we have found the shortest path, which is (1,1) -> (1,2) -> (1,3).

Our project is mostly based on A\* star algorithm. We made 3 lists for separating the paths that has been calculated with the help of the algorithm. The detailed description of the code is represented in the Algorithm description section of this documentation.

### **Difference between Dijkstra's Algorithm and A\* Algorithm**

Finding the shortest path between two nodes in a network is typically done using either the Dijkstra's method or the A\* algorithm, both of which are graph search techniques. They are different in their strategies and how they rank the

nodes, though.

The traditional shortest-path finding algorithm, Dijkstra's algorithm ranks nodes purely by their distance from the starting node. It is a breadth-first search method that investigates every path that could lead from the starting node and modifies the distances between them as it proceeds.

In contrast, the A\* method outperforms Dijkstra's algorithm. Nodes are ranked according on their proximity to the target node and their anticipated distance from the starting node. A heuristic function, which is a gauge of how close a node is to the target node, is used to determine the estimated distance. A\* is quicker and more effective than Dijkstra's algorithm because it can prioritize nodes that are more likely to go to the target node thanks to this heuristic function.

The primary distinction between Dijkstra's algorithm and A\* algorithm in a Snake game is how they select the route to the food. Dijkstra's algorithm will investigate every route that could lead to the food, but A\* algorithm will give priority to routes that are more likely to do so. As a result, A\* algorithm locates the shortest path to the food quicker and more effectively than Dijkstra's method.

The A\* algorithm is generally preferred due to its speed and efficiency in finding the shortest path based on both the distance from the starting node and the estimated distance to the target node, even though Dijkstra's algorithm and A\* algorithm can both be used to find the shortest path in a Snake game.

## 5 Proposed Approaches

### Algorithms

#### 1. Hard coding the directions of the Snake with respect to the Coordinates of the Apple

While developing the GUI in the initial phase of the Project, we thought of a simple Algorithm, that could make snake move without the help of the player using the keyboard, but since it was a very basic algorithm it could only make the snake move, and if the head of the snake bit its tail, it was not able to detect that, it was a fault and. Game should be over by them.

#### Pseudocode 1

Obviously, this algorithm was not enough to make snake move without making pass through its body, which in scenario would kill it. So we worked for a week on improving this algorithm and we got a pretty good working algorithm, which was able to make snake move towards the apple successfully without getting out with an avg. score of 30 Apples every game.

```

if(applesx > x[0]){
    direction = 'R';
}
if(applesx < x[0]){
    direction = 'L';
}
if(applesy < y[0]){
    direction = 'U';
}
if(applesy > y[0]){
    direction = 'D';
}
else{
}

```

## Pseudocode 2

Obviously, this algorithm was not enough to make snake move without making pass through its body, which in scenario would kill it. So we worked for a week on improving this algorithm and we got a pretty good working algorithm, which was able to make snake move towards the apple successfully without getting out with an avg. score of 30 Apples every game.

```

if(check == 0) {
    if (applesy < y[0]) {
        if (direction != 'D') {
            direction = 'U';
        }
    }
    if (direction == 'U') {
        boolean snakethere = false;
        for (int i = 0; i < bodyParts; i++) {
            if (x[0] == x[i] && y[0] - 25 == y[i] || x[0] == x[i] && y[0] - 50 == y[i] || x[0] == x[i] && y[0] - 75 == y[i] ||
                y[0] == 0 /*|| x[0] == x[i] && y[0] - 100 == y[i]*/ ||
                x[0] == a[i] && y[0] - 25 == s[i] || x[0] == a[i] && y[0] - 50 == y[i] /*|| x[0] == x[i] && y[0] - 75 == y[i]
                */ || y[0] == 0) {
                    snakethere = true;
                }
        }
    }
    if (snakethere) {
        boolean snakethere1 = false;
        for (int i = 0; i < bodyParts; i++) {
            if (x[0] - 25 == x[i] && y[0] == y[i] || x[0] - 50 == x[i] && y[0] == y[i] || x[0]-75 == x[i] && y[0] == y[i] ||
                x[0] == 0 /*|| x[0]-100 == x[i] && y[0] == y[i]*/ ||
                x[0] - 25 == a[i] && y[0] == s[i] || x[0] - 50 == a[i] && y[0] == s[i] /*|| x[0]-75 == x[i] && y[0] ==
                y[i]*/ || x[0] == 0) {
                    snakethere1 = true;
                    break;
                }
        }
    }
    if (snakethere1) {
        direction = 'R';
    } else {
        direction = 'L';
    }
    check = 1;
} else {check = 1;}
}
}

```



The code given above, is an part of the whole Algorithm that we developed in the initial weeks. This part explains, for example the apple has spawned on the co-ordinates that are above the snake. The direction in which the snake is moving at that movement is determined and if it not 'D'. Neighbors of the head of the snake will be checked, in which if the coordinate above the head of the snake is free, then snake will change its direction towards the Apple i.e. 'U'. If the UNIT BLOCK above the snake's head is blocked either with the Snake's other Body parts or by the 2nd snake, then the direction will be either 'L' or 'R'.

In order choose between these two directions, same process is followed again. Same is the case with the part of the code shown below. If the apple has spawned on the co-ordinates that are to the 'LEFT' of the snake. The direction in which the snake is moving at that movement is determined and if it not 'R'. Neighbors of the head of the snake will be checked, in which if the coordinate on the left of the snake's head is free, then snake will change its direction towards the Apple i.e. 'L'. If the UNIT BLOCK to the left of snake's head is blocked either with the Snake's other Body parts or by the 2nd snake, then the direction will be either 'U' or 'D'.

In order choose between these two directions, same process is followed again.

```

if(check == 0) {
    if (applesx < x[0]) {
        if (direction != 'R') {
            direction = 'L';
        }
        if (direction == 'L') {
            boolean snakethere = false;
            for (int i = 0; i < bodyParts; i++) {
                if (x[0] - 25 == x[i] && y[0] == y[i] || x[0] - 50 == x[i] && y[0] == y[i] || x[0] - 75 == x[i] && y[0] == y[i] ||
                    x[0] == 0 /*|| x[0] - 75 == x[i] && y[0] == y[i]*/)
                    || x[0] - 25 == a[i] && y[0] == s[i] || x[0] - 50 == a[i] && y[0] == s[i] /*|| x[0] - 75 == x[i] && y[0] == y[i]*/ ||
                    x[0] == 0) {
                        snakethere = true;
                    }
            }
            if (snakethere) {
                boolean snakethere1 = false;
                for (int i = 0; i < bodyParts; i++) {
                    if (x[0] == x[i] && y[0] + 25 == y[i] || x[0] == x[i] && y[0] + 50 == y[i] || x[0] == x[i] && y[0]+75== y[i] ||
                        y[0] == 575 /*|| x[0] == x[i] && y[0]+100== y[i]*/)
                        || x[0] == a[i] && y[0] + 25 == s[i] || x[0] == a[i] && y[0] + 50 == s[i] /*|| x[0] == x[i] && y[0]+75== y[i]*/
                        || y[0] == 575) {
                            snakethere1 = true;
                            break;
                        }
                }
                if (snakethere1) {
                    direction = 'U';
                } else {
                    direction = 'D';
                }
                check = 1;
            } else {check = 1;}
        }
    }
}

```

This is then repeated for both the cases that are not listed above i.e., if the Apple has spawned to the RIGHT or to the SOUTH of the snakes head



## A\* Algorithm

```
boolean flag = true;
while (flag) {
    int max = 1000;
    int current_x = 0;
    for (int i = 0; i < open.size(); i++) {
        if (open.get(i).getDistance() < max) {
            max = open.get(i).getDistance();
            current_x = i;
        }
    }

    //System.out.println(current_x);

    neighbors2 current = new neighbors2(open.get(current_x).getNeighbors_x(),
    open.get(current_x).getNeighbors_y(), Distance.open.get(current_x).getNeighbors_x(), apple_x_coordinate, open.get(current_x).getNeighbors_y(), apple_y_coordinate, open.get(current_x).getParent_x(), open.get(current_x).getParent_y());
    //System.out.println(open.get
    // (current_x).getParent_x());
    //System.out.println("current_x" + current.getNeighbors_x());
    //System.out.println("current_y" + current.getNeighbors_y());

    for (int i = 0; i < 4; i++) {
        neighbors2 n = new neighbors2(current.getNeighbors_x() + i, current.getNeighbors_y(),
        i, apple_x_coordinate, current.getNeighbors_x() + i, apple_y_coordinate, current.getNeighbors_x(),
        current.getNeighbors_y());
        //System.out.println("n_x" + n.getNeighbors_x());
        //System.out.println("n_y" + n.getNeighbors_y());
        if (n.getNeighbors_x() == apple_x_coordinate && n.getNeighbors_y() == apple_y_coordinate) {
            //n.setParent_x(current.getNeighbors_x());
            //n.setParent_y(current.getNeighbors_y());
            //n.parent_x = current.getNeighbors_x();
            //n.parent_y = current.getNeighbors_y();
            //System.out.println("parent_x");
            //System.out.println("parent_y");
            flag = false;
            continue;
        }
        if (n.isClose_toGoal() && n.isFree() && n.isNeighbor()) {
            //n.distance = Distance(n.getNeighbors_x(), apple_x_coordinate, n.getNeighbors_y(), apple_y_coordinate);
            //n.setParent_x(current.getNeighbors_x());
            //n.setParent_y(current.getNeighbors_y());
            //n.parent_x = current.getNeighbors_x();
            //n.parent_y = current.getNeighbors_y();
            //System.out.println("parent_x");
            //System.out.println("parent_y");
            open.add(n);
        }
    }
    //open.size() - 1;
    open.remove(current_x);
    close.add(current);
}
```

This code is implementing a search algorithm A\*, to find a path from a starting position to an end position (represented by the coordinates of an apple) in a 2D grid. It uses two lists, "open" and "close", to keep track of the nodes that have been visited and the nodes that need to be visited. The algorithm starts with the node in "open" with the lowest distance to the end position and adds its neighbors to "open". If a neighbor is the end position, the loop stops. If not, the current node is added to "close" and removed from "open". The algorithm repeats until the end position is found or there are no more nodes in "open".

## Input/Output Format

In regard to our game, as mentioned earlier, three modes have been introduced in our game, single player, multiplayer, and Artificial Intelligence. For each of the modes, a separate Graphical User Interface has been designed so that the user can distinguish between them.

In the case of single-player mode and multiplayer mode, we have included text fields by using the concepts of Java Swing. These text fields act as input for our user where he enters his name, in the case of multiplayer mode two users are present so there are two text fields for each of the users to enter their name. In the case of Artificial Intelligence, no text fields have been included since it is not controlled by the user, so there is no need to enter their names.

As an output, we have designed a separate Graphical User Interface for displaying our high scores.

In the single-player mode, the maximum number of apples eaten by the snake is shown at the end as the score, the user got while playing the game, as an output.

In the multiplayer mode and the Artificial Intelligence mode, the user (multiplayer mode) or the snake (Artificial Intelligence mode), the maximum number of apples eaten by the snake is shown as an output on the high score panel

This high score panel gets updated each time with a new high score as the user plays the game again and again, due to the integrated database, as hence we have an updated version of the high score panel each time the user plays the game

Not only the updated high score, but the user also gets to see the list of the other users who have played this game till now along with their high scores, because all the names that are taken as input from the users and the high scores that are calculated, as the user reaches the end of the game, are stored in the connected database and then shown as an output

## 6 Implementation Details

### 1. Algorithm

To Implement an Algorithm we started step by step to construct our code and that steps includes:

#### **SnakeBody:**

As for start, main task in Algorithm was creating a Datatype which shows the basic functionality and requirements of Snake

```

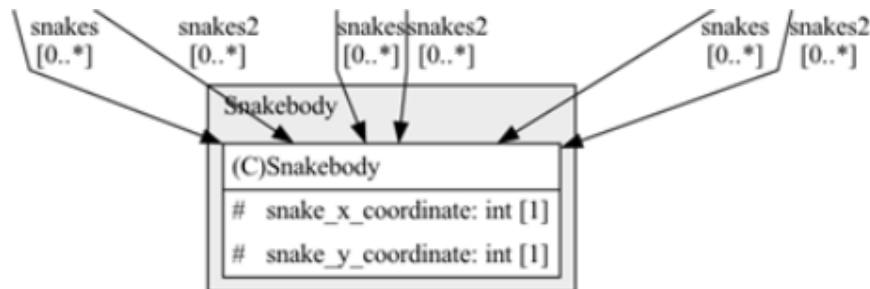
10 usages
class Snakebody{
    2 usages
    int snake_x_coordinate;
    2 usages
    int snake_y_coordinate;

    2 usages
    public Snakebody(int snake_x_coordinate, int snake_y_coordinate){
        this.snake_x_coordinate = snake_x_coordinate;
        this.snake_y_coordinate = snake_y_coordinate;
    }

    5 usages
    public int getSnake_x_coordinate() { return snake_x_coordinate; }
    5 usages
    public int getSnake_y_coordinate() { return snake_y_coordinate; }
}

```

This code defines a class Snakebody that has two instance variables snake\_x\_coordinate and snake\_y\_coordinate that represent the x and y coordinates of a snake body part in a game. The class has a constructor that takes in two integer arguments to initialize these values and two getter methods to access these values.



### NeighborsClass:

As an A\* Algorithm, neighbor plays a critical role to find out the shortest path from Snake head to Apple Coordinates.

```

class neighbors2{
    int neighbors_x;
    int neighbors_y;
    int distance;
    int total_free_neighbors = 0;
    int parent_x, parent_y;

    public neighbors2(int neighbors_x, int neighbors_y) {
        this.neighbors_x = neighbors_x;
        this.neighbors_y = neighbors_y;
    }

    public neighbors2(int neighbors_x, int neighbors_y, int distance, int
parent_x, int parent_y){
        this.neighbors_x = neighbors_x;
        this.neighbors_y = neighbors_y;
        this.distance = distance;
        this.parent_x = parent_x;
        this.parent_y = parent_y;
    }

    public void setParent_x(int parent_x) {
        this.parent_x = parent_x;
    }

    public void setParent_y(int parent_y) {
        this.parent_y = parent_y;
    }

    public int getNeighbors_x(){return neighbors_x;}
    public int getNeighbors_y(){return neighbors_y;}
    public int getDistance(){return distance;}
    public int getTotal_free_neighbors(){return total_free_neighbors;}

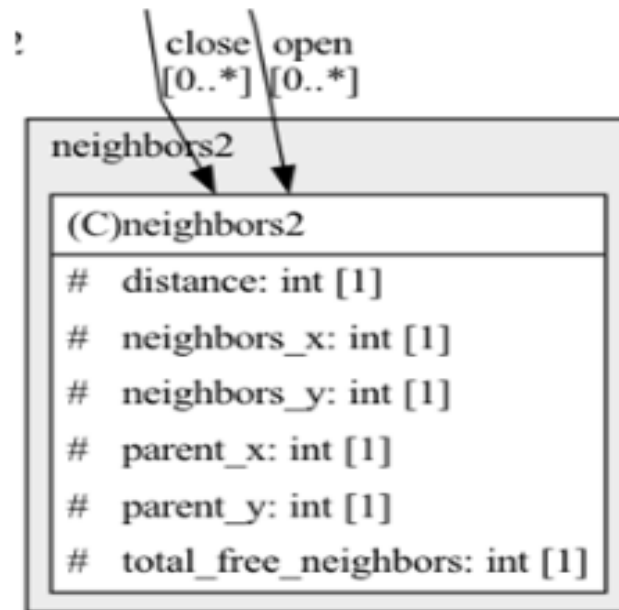
    public int getParent_x(){return parent_x;}
    public int getParent_y(){return parent_y;}
}

```

The neighbors2 class is a simple class that contains two variables, neighbors\_x and neighbors\_y, that represent the x and y coordinates of a neighbor. It also has an variable distance to store the distance between the current node and the neighbor node, as well as total.free.neighbors, which is the number of free neighbors.

Additionally, the class has two variables, parent\_x and parent\_y, to store the x and y coordinates of the parent node. The class has two constructors. The first constructor takes in two parameters, neighbors\_x and neighbors\_y, and sets the variables with the same name to these values.

The second constructor takes in five parameters, neighbors\_x, neighbors\_y, distance, parent\_x, and parent\_y, and sets the variables with the same name to these values. The class also has a set of getter methods to retrieve the values of each variable.



**ArrayList:** for open and close List:

OpenList:

Here comes all neighbors which still needs to be process

CloseList:

Once neighbor which is in OpenList gets processed, then it will add in CloseList

```

ArrayList<neighbors2> open = new ArrayList<neighbors2>();
ArrayList<neighbors2> close = new ArrayList<neighbors2>();

```

**Manhattan Distance:**

It's a shortest distance from one point to another.

Manhattan Distance =  $|x_1 - x_2| + |y_1 - y_2|$

```

public int Distance(int x1, int x2, int y1, int y2) {
    return (abs(x1 - x2) + abs(y1 - y2))/25;
}

```

To check out weather neighbor is free or not:

```

public boolean is_free(neighbors2 n) {
    int a = 0, a2 = 0;
    for (int i = 0; i < snakes.size(); i++) {
        if (n.getNeighbors_x() == snakes.get(i).getSnake_x_coordinate() &&
            n.getNeighbors_y() == snakes.get(i).getSnake_y_coordinate()) {
            a = a + 1;
        }
    }

    for (int i = 0; i < snakes2.size(); i++) {
        if (n.getNeighbors_x() == snakes2.get(i).getSnake_x_coordinate() &&
            n.getNeighbors_y() == snakes2.get(i).getSnake_y_coordinate()) {
            a2 = a2 + 1;
        }
    }

    if (a == 0 && a2 == 0) return true;
    else return false;
}

```

This is a method `is_free` that takes a `neighbors2` object `n` as its argument. The method checks if the location represented by the `neighbors2` object is free i.e. not occupied by a snake. To do this, the method checks if the x and y coordinate of the `neighbors2` object is not present in two separate lists of `Snakebody` objects called `snakes` and `snakes2`.

The method increments the value of `a` by 1 if the x and y coordinate of the `neighbors2` object is found in the list `snakes`. Similarly, the method increments the value of `a2` by 1 if the x and y coordinate of the `neighbors2` object is found in the list `snakes2`.

Finally, the method returns `true` if both `a` and `a2` are equal to 0, meaning the location is not occupied by any snake, and returns `false` otherwise, meaning the location is occupied.

**To check out weather neighbor is in Frame or not:**

```

public boolean is_neighbor(neighbors2 n) {

    if (n.getNeighbors_x() >= 0 && n.getNeighbors_y() >= 0 &&
        n.getNeighbors_x() <= 600 && n.getNeighbors_y() <= 600) {
        return true;
    } else return false;

}

```

This is a method that checks if a given `neighbors2` object is a valid neighbor or not. The method checks if the `neighbors_x` and `neighbors_y` values of the object are within the range of 0 and 600. If they are within this range, the method returns `true`, indicating that it is a valid neighbor. If they are not within this range, the method returns `false`, indicating that it is not a valid neighbor.

**One of the most important functions: To check weather that neighbor**



is in closeList or not:

```
public boolean is_not_in_close_list(neighbors2 n) {
    int b = 0;
    for (int i = 0; i < close.size(); i++) {
        //System.out.println(n.getNeighbors_x());
        //System.out.println(close.get(i).getNeighbors_x());
        //System.out.println(n.getNeighbors_y());
        //System.out.println(close.get(i).getNeighbors_y());
        if ((n.getNeighbors_x() == close.get(i).getNeighbors_x()) &&
            (n.getNeighbors_y() == close.get(i).getNeighbors_y())) {
            b += 1;
        }
    }
    if (b != 0) return false;
    else return true;
}
```

This code is checking if a given neighbors2 object n is not in the list close. The close list is used to store the coordinates of neighbors that have already been processed. The function returns true if n is not in close, and false otherwise. The function loops over all elements in the close list, and for each element it checks if the neighbors\_x and neighbors\_y values of n are equal to the corresponding values of the current element in the loop. If they are equal, it means that n is already in close, so the function returns false. If the loop finishes without finding a matching element, the function returns true.

**Last Function: is next neighbor is apple or not:**

```
int[] a1 = {25, -25, 0, 0};
int[] a2 = {0, 0, 25, -25};
```

```
public boolean next_neighbor_is_apple(neighbors2 n){
    int c = 0;
    for(int i = 0 ; i< 4 ; i++){
        neighbors2 na = new
neighbors2(n.getNeighbors_x()+a1[i],n.getNeighbors_y()+a2[i]);
        if(na.getNeighbors_x() == apple_x_coordinate && na.getNeighbors_y()
== apple_y_coordinate){
            c+=1;
        }
    }
    if (c != 0) return true;
    else return false;
}
```

The method next\_neighbor\_is\_apple takes in an instance of neighbors2 class as an argument and checks if the next position of the neighbor is an apple. It creates 4 instances of the neighbors2 class na by adding the values of a1[i] and a2[i] to the neighbors\_x and neighbors\_y of the argument n respectively. Then, it checks if the neighbors\_x and neighbors\_y match the apple\_x\_coordinate and apple\_y\_coordinate. If they match, c is incremented by 1.

If c is not equal to 0, the method returns true as the next position of n is an apple. Otherwise, the method returns false.

## Final Implementation

```
A_Star_Algorithm ai2 = new A_Star_Algorithm(snakes2, snakes,
apple_x, apple_y, snakes2.get(0).getSnake_x_coordinate(), snakes2.get(0).getSnake_y_coordinate(), applesEaten2);
```

```
public A_Star_Algorithm(ArrayList snakes, ArrayList snakes2, int
apple_x_coordinate, int apple_y_coordinate, int snake_x, int snake_y, int
number_of_apple) {
    this.snakes = snakes;
    this.snakes2 = snakes2;
    this.apple_x_coordinate = apple_x_coordinate;
    this.apple_y_coordinate = apple_y_coordinate;
    this.snake_x = snake_x;
    this.snake_y = snake_y;
    this.number_of_apple = number_of_apple;
}
```

This is a constructor for the A\_Star\_Algorithm class. It initializes the member variables with the values passed as parameters. The member variables being initialized are:

Snakes : ArrayList of first snake

snakes2 : ArrayList of Second snake

apple\_x\_coordinate

apple\_y\_coordinate

snake\_x : Head of Snake(X-Coordinate)

snake\_y : Head of Snake (Y-Coordinate)

number\_of\_apple

```
neighbors2 start = new neighbors2(snake_x, snake_y, Distance(snake_x,
apple_x_coordinate, snake_y, apple_y_coordinate), snake_x, snake_y);
if(next_neighbor_is_apple(start)){
    dir_x = apple_x_coordinate;
    dir_y = apple_y_coordinate;
    //System.out.println(dir_x);
    //System.out.println(dir_y);
    return direction_call(dir_x, dir_y);
}
```

If next neighbor is apple, then algorithm will simply return the coordinates of Apple and accordingly direction will be chosen.

If that's not the Case, then algorithm will continue until it will not find the coordinate of Apple or until it will not reach to Apple.

```

{
    // for(int i = 0 ; i < 4; i++){
    //     neighbors na = new neighbors(start.getNeighbors_x()+ a1[i],
    start.getNeighbors_y()+a2[i] );
    //     if(na.getNeighbors_x() == apple_x_coordinate &&
    na.getNeighbors_y() == apple_y_coordinate){
    //         }
    //     }
    // }
    //start.distance = Distance(snake_x, apple_x_coordinate, snake_y,
    apple_y_coordinate);
    //start.setParent_x(snake_x);
    //start.setParent_y(snake_y);
    //start.parent_x = snake_x;
    //start.parent_y = snake_y;

    open.add(start);
}

```

If next neighbor is not an Apple, then Head of a Snake will add to OpenList and visit all neighbor of SnakeHead.

```

int max = 1000;
int current_i = 0;
for (int i = 0; i < open.size(); i++) {
    if (open.get(i).getDistance() < max) {
        max = open.get(i).getDistance();
        current_i = i;
    }
}

```

This code is part of a loop that is finding the node with the smallest distance value in a list of nodes called "open". The loop iterates through all elements in the "open" list. For each iteration, it checks if the distance value of the current node is smaller than the current maximum value stored in the "max" variable. If it is, then the "max" value is updated to the current distance value and the index of the node is stored in "current\_i". After the loop, "current\_i" will contain the index of the node with the smallest distance in the "open" list.

```

for (int i = 0; i < 4; i++) {
    neighbors2 n = new neighbors2(current.getNeighbors_x() + a1[i],
    current.getNeighbors_y() + a2[i], Distance(current.getNeighbors_x() +
    a1[i], apple_x_coordinate, current.getNeighbors_y() +
    a2[i], apple_y_coordinate), current.getNeighbors_x(), current.getNeighbors_y()
    );
    //System.out.println("n_x"+n.getNeighbors_x());
    //System.out.println("n_y"+ n.getNeighbors_y());
    if (n.getNeighbors_x() == apple_x_coordinate && n.getNeighbors_y() ==
    apple_y_coordinate) {
        //n.setParent_x(current.getNeighbors_x());
        //n.setParent_y(current.getNeighbors_y());
        //n.parent_x = current.getNeighbors_x();
        //n.parent_y = current.getNeighbors_y();
        //System.out.println(n.parent_x);
        //System.out.println(n.parent_y);
        //close.add(n);
        flag = false;
        continue;
    }
    if(is_not_in_close_list(n) && is_free(n) && is_neighbor(n)) {
        //n.distance = Distance(n.getNeighbors_x(), apple_x_coordinate,
        n.getNeighbors_y(), apple_y_coordinate);
        //n.setParent_x(current.getNeighbors_x());
        //n.setParent_y(current.getNeighbors_y());
        //n.parent_x = current.getNeighbors_x();
        //n.parent_y = current.getNeighbors_y();
        // System.out.println(n.parent_x);
        // System.out.println(n.parent_y);
        open.add(n);
    }
}
}

```

For Loop will iterate for 4 times: UP, DOWN, LEFT, RIGHT. And create an Object of class 'neighbors2' n and set a parent of that Object to a current neighbors: 'current.getNeighbors\_x()' and 'current.getNeighbors\_y()'. If n is a coordinate of apple than flag value will became false and continue to next while loop, where while loop got terminated. If not so, then n will check if the node is not in the close list, is a free, and is a neighbor, and if all conditions are met, it adds the n to the open list.

```
if(open.size() > 0){
    open.remove(current_i);
}
close.add(current);
```

This code snippet is part of the A\* algorithm. The open list contains all the cells that are not yet visited and the close list contains all the cells that have already been visited. Here, the code removes the current neighbor (current\_i) from the open list. Then, the current neighbor is added to the close list, which means that the current neighbor has now been visited.

```
for(int i = close.size()-1; i >= 0 ; i--){
    if(snake_x == close.get(i).getParent_x() && snake_y ==
close.get(i).getParent_y()){
        dir_x = close.get(i).getNeighbors_x();
        dir_y = close.get(i).getNeighbors_y();
        break;
    }
}
//System.out.println(dir_x);
//System.out.println(dir_y);
return direction_call(dir_x,dir_y);
}
```

This code is checking if the current position of the snake is present in the close list. If it is, the code retrieves the dir\_x and dir\_y values for the position from the close list and calls the direction\_call function with these values. The close list is used to keep track of the positions that the snake has already visited, while the open list is used to keep track of the positions that are yet to be visited. The dir\_x and dir\_y values represent the direction that the snake should take to reach its next position. The code uses a loop to iterate over the close list in reverse order (from the end to the start). For each position in the close list, the code checks if the current position of the snake matches the position stored in the list. If a match is found, the code retrieves the dir\_x and dir\_y values for that position and calls the direction\_call function with these values. The loop is exited once a match is found.

```

public char direction_call(int x, int y) {
    if (x == snake_x + 25 && y == snake_y) {
        return 'R';
    } else if (x == snake_x - 25 && y == snake_y) {
        return 'L';
    } else if (x == snake_x && y == snake_y + 25) {
        return 'D';
    } else return 'U';
}

```

This code is a function called "direction\_call". Given two input parameters x and y, it returns a character that represents a direction based on the relative positions of the current snake's position and the target position. If x is equal to snake\_x + 25 and y is equal to snake\_y, it returns the character 'R' which means the snake should move right. If x is equal to snake\_x - 25 and y is equal to snake\_y, it returns the character 'L' which means the snake should move left. If x is equal to snake\_x and y is equal to snake\_y + 25, it returns the character 'D' which means the snake should move down. If none of the above conditions are met, it returns the character 'U' which means the snake should move up.

### Final Algorithm

```

public char find_neighbours2() {
    neighbors2 start = new neighbors2(snake_x, snake_y, Distance(snake_x,
apple_x_coordinate, snake_y, apple_y_coordinate), snake_x, snake_y);
    if(next_neighbor_is_apple(start)){
        dir_x = apple_x_coordinate;
        dir_y = apple_y_coordinate;
        //System.out.println(dir_x);
        //System.out.println(dir_y);
        return direction_call(dir_x, dir_y);
    }else{
        // for(int i = 0 ; i< 4; i++){
        //     neighbors na = new neighbors(start.getNeighbors_x()+ a1[i],
start.getNeighbors_y()+a2[i] );
        //     if(na.getNeighbors_x() == apple_x_coordinate &&
na.getNeighbors_y() == apple_y_coordinate){
        //         //
        //         }
        //     }
        // }
        //start.distance = Distance(snake_x, apple_x_coordinate, snake_y,
apple_y_coordinate);
        //start.setParent_x(snake_x);
        //start.setParent_y(snake_y);
        //start.parent_x = snake_x;
        //start.parent_y = snake_y;
        open.add(start);
    }
}

```

```

        boolean flag = true;
        while (flag) {

            int max = 1000;
            int current_i = 0;
            for (int i = 0; i < open.size(); i++) {
                if (open.get(i).getDistance() < max) {
                    max = open.get(i).getDistance();
                    current_i = i;
                }
            }

            //System.out.println(current_i);

            neighbors2 current = new
neighbors2(open.get(current_i).getNeighbors_x(),
open.get(current_i).getNeighbors_y(), Distance(open.get(current_i).getNeighb
ors_x(), apple_x_coordinate, open.get(current_i).getNeighbors_y(), apple_y_coo
rdinate), open.get(current_i).getParent_x(), open.get(current_i).getParent_y(
));

            //System.out.println(open.get
            // (current_i).getParent_x());
            //System.out.println("current_x"+ current.getNeighbors_x());
            //System.out.println("current_y"+ current.getNeighbors_y());

```

```

        for (int i = 0; i < 4; i++) {
            neighbors2 n = new neighbors2(current.getNeighbors_x() +
a1[i], current.getNeighbors_y() + a2[i], Distance(current.getNeighbors_x() +
a1[i], apple_x_coordinate, current.getNeighbors_y() +
a2[i], apple_y_coordinate), current.getNeighbors_x(), current.getNeighbors_y(
));

            //System.out.println("n_x"+n.getNeighbors_x());
            //System.out.println("n_y"+ n.getNeighbors_y());
            if (n.getNeighbors_x() == apple_x_coordinate &&
n.getNeighbors_y() == apple_y_coordinate) {
                //n.setParent_x(current.getNeighbors_x());
                //n.setParent_y(current.getNeighbors_y());
                ///n.parent_x = current.getNeighbors_x();
                ///n.parent_y = current.getNeighbors_y();
                //System.out.println(n.parent_x);
                //System.out.println(n.parent_y);
                //close.add(n);
                flag = false;
                continue;
            }
        }

```

```

        if(is_not_in_close_list(n) && is_free(n) && is_neighbor(n))
        {
            //n.distance = Distance(n.getNeighbors_x(),
apple_x_coordinate, n.getNeighbors_y(), apple_y_coordinate);
            //n.setParent_x(current.getNeighbors_x());
            //n.setParent_y(current.getNeighbors_y());
            //n.parent_x = current.getNeighbors_x();
            //n.parent_y = current.getNeighbors_y();
            // System.out.println(n.parent_x);
            // System.out.println(n.parent_y);
            open.add(n);

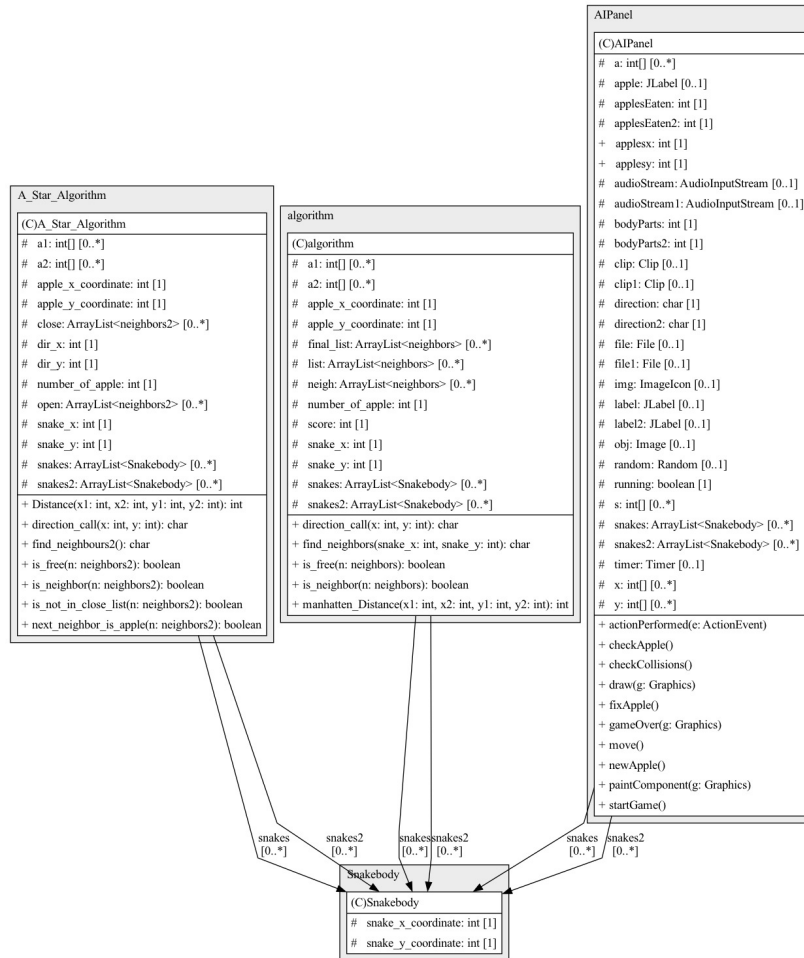
        }
    }
    if(open.size() > 0){
        open.remove(current_i);
    }
    close.add(current);
}

```

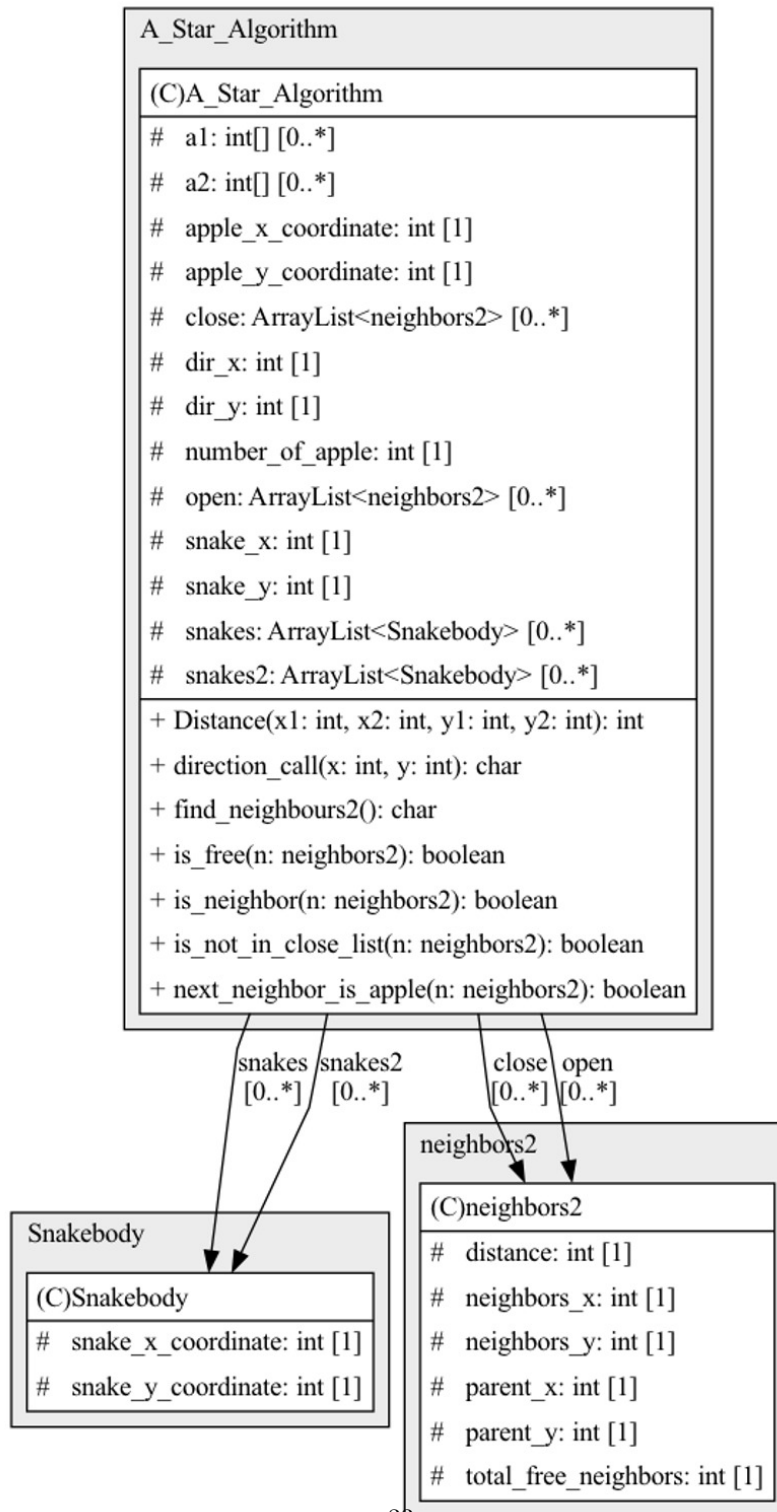
```

//for(int i = 0 ; i< close.size(); i++){
//    System.out.println("x = " + close.get(i).getNeighbors_x());
//    System.out.println("y = "+ close.get(i).getNeighbors_y());
//    System.out.println("Distance "+close.get(i).getDistance());
//    System.out.println("Parent of x "
close.get(i).getParent_x());
//    System.out.println("Parent of y
+close.get(i).getParent_y());
//}
    for(int i = close.size()-1; i >= 0 ; i--){
        if(snake_x == close.get(i).getParent_x() && snake_y ==
close.get(i).getParent_y()){
            dir_x = close.get(i).getNeighbors_x();
            dir_y = close.get(i).getNeighbors_y();
            break;
        }
    }
    //System.out.println(dir_x);
    //System.out.println(dir_y);
    return direction_call(dir_x,dir_y);
}

```







## 2. GUI

### MenuBar

Before User wants to play the game. They had to select, which mode they wanted to play. We have included 3 different modes to play this game. Single player

- Single Player
- Multiplayer
- AI

In order to make that happen a class MenuBar was created which extends JPanel and implements ActionListener. which used classes like JPanel JFrame ImageIcon and JButton to create the user interface that we created. A code snippet and the UI is shared below.

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;
import javax.sound.sampled.*;
import javax.swing.*;
import javax.swing.border.Border;

public class MenuBar extends JPanel implements ActionListener {

    JPanel panel = new JPanel();
    JFrame frame = new JFrame();
    public static JButton b1,b2,b3,b4;
    JLabel label = new JLabel();
    ImageIcon img = new ImageIcon("Snake/snake.png");

    ImageIcon background = new ImageIcon("bg.png");

    JLabel bg_l = new JLabel();

    File file = new File("Snake/bg.wav");
    AudioInputStream audioStream = AudioSystem.getAudioInputStream(file);
    Clip clip = AudioSystem.getClip();
```

In order to give more immersive experience to the user, we also used classes like Clip AudioInputStream to add background game music for the game.



For user to choose which he mode he wanted to play, he has to click one on the 3 above listed buttons. If those buttons were just implemented without @Overriden actionPerformed method, it would not have been possible to start the gameplay.

A code snippet is shared below, to explain how and which buttons respond and which way.

```

@Override
public void actionPerformed(ActionEvent e) {

    if(e.getSource() == b1){
        clip.stop();
        clip.close();
        frame.dispose();
        startPanelMultiPlayer multiplayer = new startPanelMultiPlayer();
    }

    if(e.getSource() == b3){
        clip.stop();
        clip.close();
        frame.dispose();
        try {
            AI ai = new AI();
        } catch (UnsupportedAudioFileException ex) {
            throw new RuntimeException(ex);
        } catch (LineUnavailableException ex) {
            throw new RuntimeException(ex);
        } catch (IOException ex) {
            throw new RuntimeException(ex);
        }
    }

    if(e.getSource() == b2){
        frame.dispose();
        clip.stop();
        clip.close();
        startPanelSingle sps = new startPanelSingle();
    }

    if(e.getSource() == b4){
        clip.stop();
        clip.close();
        frame.dispose();
        HighScore highscore = new HighScore();
    }
}

```

### Game mode

Single player, multi player and artificial intelligent mode of the game work on the same GUI concept where startPanelSingle extends JFrame and implements ActionListener.

This class is used to get user Input, before user starts playing game, they are asked to enter their name/names, so that their scores can be updated in the Databases. Code snippet and GUI are explained and displayed below.



The UIs displayed above, take names from the user and then store them with the at the end reached score in the Database.

This is a Java code for a graphical user interface class "startPanelSingle" which extends JFrame and implements ActionListener. The class creates a JFrame window with a JLabel, a JTextField, a JComboBox, a JButton and a JPanel. The window has a size of 600x200, title "Snake MultiPlayer", is not resizable, centered and set to close on exit. The JPanel uses null layout and has a preferred size of 500x200.

The JButton has an action listener which disposes the current window and opens a new window "SinglePlayer" when clicked. The JLabel displays the text "name : ", the JTextField allows the user to enter a name and the JComboBox displays a list of color choices. The components are positioned on the JPanel using setBounds method.

This is better understood with the snippet listed below.

```

class startPanelSingle extends JFrame implements ActionListener{
    JLabel name = new JLabel("name : ");
    public static JTextField tf = new JTextField();
    public static String[] choices = { "blue", "BLACK", "cyan", "orange", "lightgray", "white", "yellow"};
    public static JComboBox<String> cb = new JComboBox<String>(choices);
    JButton next = new JButton("GAME!!!");
    JPanel newPanel = new JPanel();
    startPanelSingle(){
        this.setSize(600,200);
        this.setTitle("Snake MultiPlayer");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setResizable(false);
        this.pack();
        //
        this.setVisible(true);
        this.setLocationRelativeTo(null);
        newPanel.setLayout(null);
        newPanel.setPreferredSize(new Dimension(500,200));
        next.addActionListener(this);
        newPanel.add(name);
        newPanel.add(tf);
        newPanel.add(cb);
        newPanel.add(next);

        name.setBounds(210, 40, 100,40);
        tf.setBounds(260, 45, 150,30);
        //
        cb.setBounds(350, 40, 150,40);
        next.setBounds(245, 90, 100,40);

        this.add(newPanel);
    }

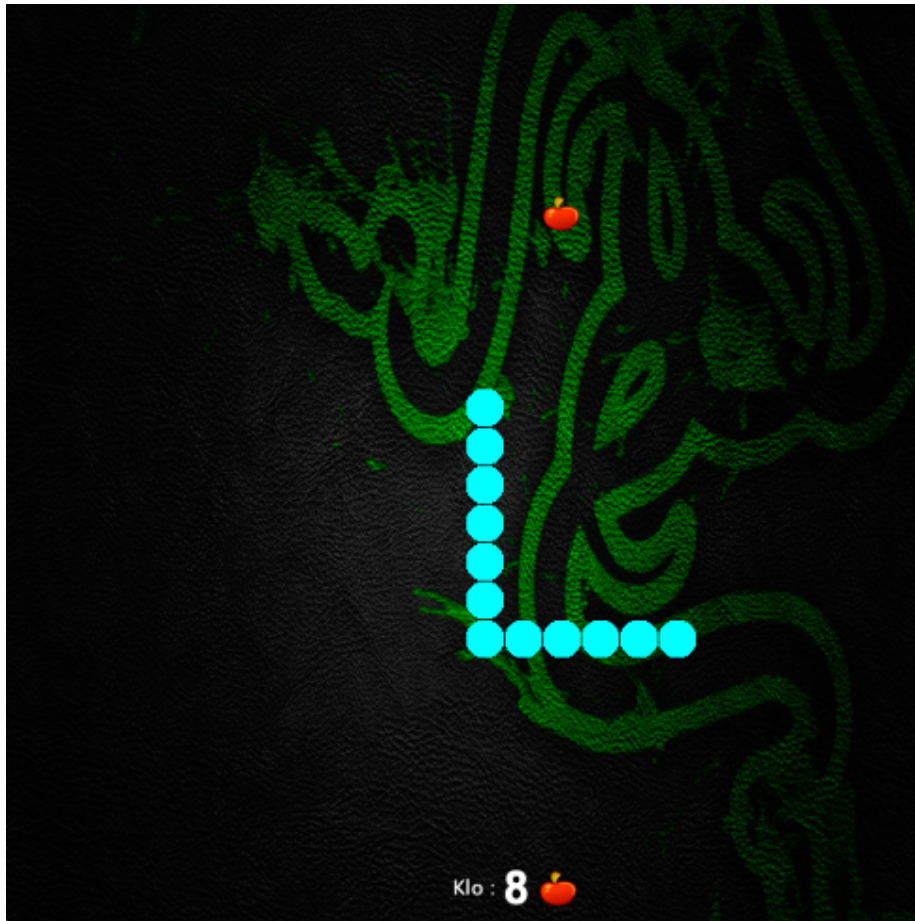
    @Override
    public void actionPerformed(ActionEvent e) {
        if(e.getSource() == next){
            this.dispose();
            try {
                SinglePlayer singlePlayer = new SinglePlayer();
            } catch (UnsupportedAudioFileException ex) {
                throw new RuntimeException(ex);
            } catch (LineUnavailableException ex) {
                throw new RuntimeException(ex);
            } catch (IOException ex) {
                throw new RuntimeException(ex);
            }
        }
    }
}

```

After taking the name/names of the user/users, the actual gameplay starts, where user guide the snake to the apple's location.

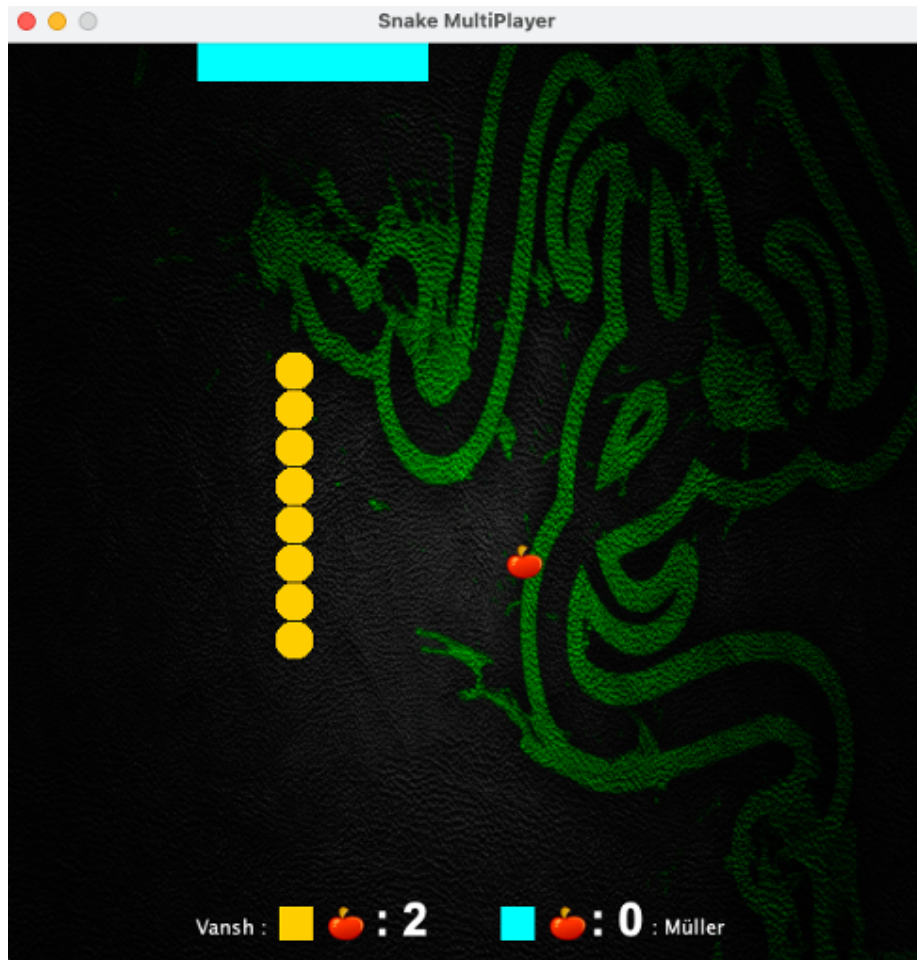
2 players play in multiplayer mode and both of them guide their snakes to the location of the apple, the snake that ate the apple first, is elongated and the score of that snake is incremented.

This is the snippet of the Single player game, where user himself guides the snake to the destination, with the help of 'UP', 'DOWN', 'RIGHT', 'LEFT' arrow keys.



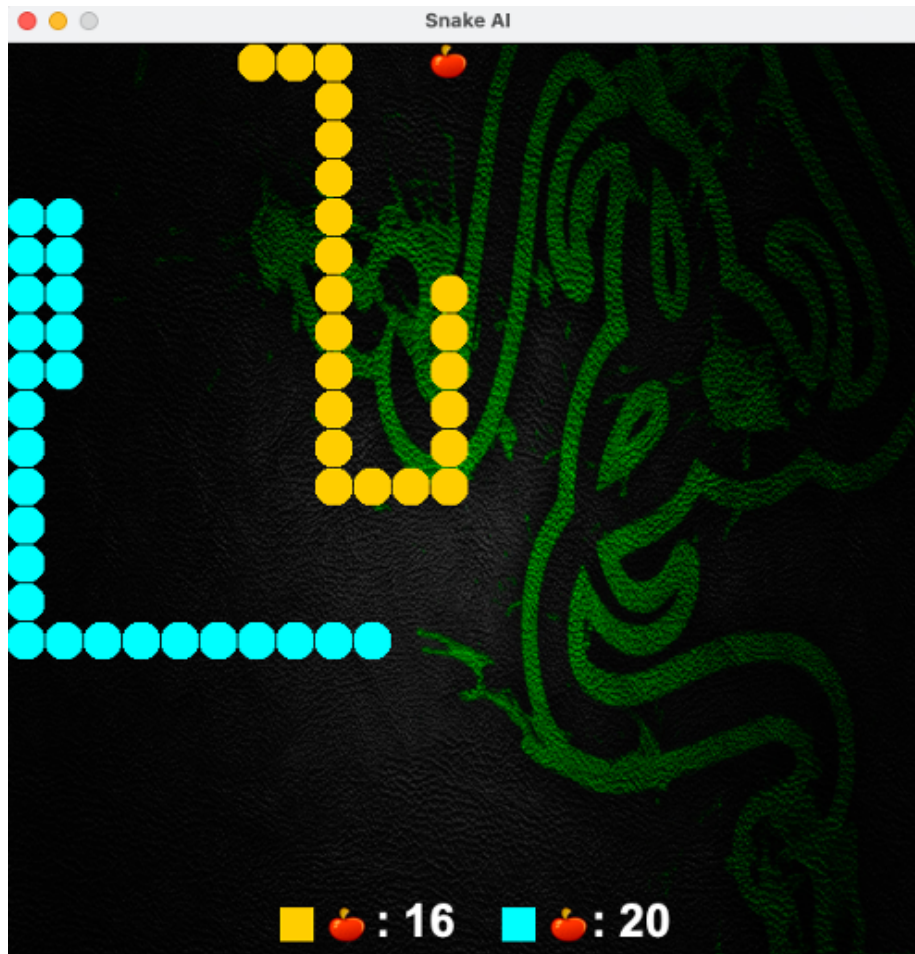
The UI given below is the is snippet of the Multiplayer mode, where 2 snakes are running simultaneously and 2 players control the snakes where Snake 1 is controlled by 'UP', 'DOWN', 'RIGHT', 'LEFT' arrow keys and Snake 2 is controlled by 'W', 'S', 'D', 'A' arrow keys.

After users have completely played the game, and at least one of them has gotten out, then their scores are fetched and stored with their names in the Database.



In terms of the AI Gameplay, snakes(2) move automatically according to the algorithms that we developed, 'Hard coded' 'A\*'. A user interface of that gameplay is listed down below.





Painting all this on the JPanel was not at easy task, at a little bit too much to understand and figure out for us beginners. But here also the Java documentation on the Oracle website and the Stack Overflow helped us a lot. In order to explain how these things were painted on the JPanel code snippets from the different parts of the codes are displayed below.

```
static final int SCREEN_WIDTH = 600;
static final int SCREEN_HEIGHT = 600;
static final int UNIT_SIZE = 25;
static final int GAME_UNITS = (SCREEN_WIDTH * SCREEN_HEIGHT) / UNIT_SIZE;
public int applesx ;
public int applesy ;
char direction = 'R';
boolean running = false;
Timer timer;
Random random;
```

This Java code defines several static and instance variables for a game. The static constants are:

- screenWidth: the width of the game screen in pixels
- screenHeight: the height of the game screen in pixels
- unitSize: the size of each unit in the game in pixels
- gameUnits: the total number of units in the game, calculated by dividing the screen area by the unit size
- delay: the time delay in milliseconds between updates in the game loop.

The instance variables are:

- applesx: the x coordinate of the apple
- applesy: the y coordinate of the apple
- direction: the direction of the player's movement, initialized to 'R' meaning right
- running: a boolean flag to check if the game is running, initialized to false
- timer: an object of the Timer class
- random: an object of the Random class.
- These variables are likely used to control the player's movement, generate the position of the apple, and control the game loop.

```
File file = new File("Snake/hiss.wav");
AudioInputStream audioStream = AudioSystem.getAudioInputStream(file);
Clip clip = AudioSystem.getClip();
//eat audio
File file1 = new File("Snake/eat2.wav");
AudioInputStream audioStream1 = AudioSystem.getAudioInputStream(file1);
Clip clip1 = AudioSystem.getClip();
```

This Java code loads two audio files and sets them up for Background music/playback using the Java Sound API. The two audio files are:

- "Snake/hiss.wav"
- "Snake/eat2.wav"

For each audio file, the following steps are performed:

- A File object is created using the file path
- An AudioInputStream is created using the File object and the getAudioInputStream() method of the AudioSystem class.
- A Clip object is created using the getClip() method of the AudioSystem class.

These Clip objects are likely used to play sound effects in the game, such as a hiss sound and an eating sound.

```
public void startGame(){
    newApple();
    running = true;

    timer = new Timer(DELAY, this);
    timer.start();
}
public void paintComponent(Graphics g){
    super.paintComponent(g);
    draw(g);
}
```

After running the constructor, the 2 predefined methods startGame() and paintComponent(Graphics g) are called.

The startGame() method creates a new apple, sets running to true, creates a new Timer with a delay of DELAY milliseconds and sets this as the action listener, and starts the timer.

The paintComponent(Graphics g) method is a method of the JComponent class and it is used to paint or draw the components. In this code, it first calls the super.paintComponent(g) method to make sure that the parent class's painting functionality is properly executed. Then it calls the draw(g) method to draw the game components.

In draw method then, the whole background, Snakes, Apple and the scores that are displayed below are painted.

```

public void draw(Graphics g){

    if(!running) gameOver(g);

    clip.start();

    //drawing background
    g.drawImage(obj, 0, 0, 600, 600,this);

    painting apple as icon
    this.add(apple);
    apple.setBounds(applesx, applesy,25,25);

    //drawing snake
    for(int i = 0; i < bodyParts; i++){
        g.setColor(Color.cyan);
        g.fillOval(x[i], y[i], UNIT_SIZE, UNIT_SIZE);
    }

    //name
    name1.setText(startPanelSingle.tf.getText() + " :");

    this.setFont(new Font("Ink", Font.BOLD, 80));
    this.add(name1);
    name1.setForeground(Color.white);
    name1.setBackground(Color.black);
    name1.setHorizontalAlignment(SwingConstants.RIGHT);
    name1.setBounds(200, 534, 120,80);

    //Score
    g.setColor(Color.WHITE);
    g.setFont(new Font("Arial", Font.BOLD, 30));
    g.drawString("" + applesEaten, 325, 585);

    score apple icon
    if(applesEaten > 9){
        this.add(label);
        label.setBounds(360,560,30,30);
    }
    else{
        this.add(label);
        label.setBounds(345,560,30,30);
    }
}

```

The game first checks if the game is running or not using the "running" variable. If the game is not running, it will call the "gameOver" method.

It draws the background image that can be seen in the images displayed above. Drawing of the Snake, is done through the for loop, where i iterates through the body parts of the snake and draw it on the screen with the color of your choice, here we have chosen CYAN.

After that, the score each player, players or even the Ai bots have scored are drawn on the screen and then the score apple icon is drawn based on the number

of apples eaten.

Up until here was the difficult part of drawing graphics on the panel in terms of GUI.

```
public void newApple(){
    applesx = random.nextInt((int)(SCREEN_WIDTH/UNIT_SIZE)) * UNIT_SIZE;
    applesy = random.nextInt((int)(SCREEN_HEIGHT/UNIT_SIZE)) * UNIT_SIZE;

    for(int i = 0; i < bodyParts; i++){
        if (applesx == x[i] && applesy == y[i]){
            fixApple();
        }
    }

    for(int i = 0; i < 25; i++){
        if(applesx == UNIT_SIZE*i && applesy == UNIT_SIZE*22 || applesx == UNIT_SIZE*i && applesy == UNIT_SIZE*23){
            fixApple();
        }
    }
}
```

newApple() method is the method, where the coordinates applesx and applesy are decided with the help of random operator and if the coordinates of the apple were falling on the body of snake, a new method named fixApple() would be called, which further would call the newApple() method again, to change the spawning coordinates of the Apple.

```
public void checkApple(){
    if((x[0] == applesx) && y[0] == applesy){
        clip1.start();
        applesEaten++;
        bodyParts++;
        newApple();
    }
    else {
        clip1.stop();
    }
}
```

In this part of the code, checkApple() method checks if the apple that was spawned was eaten or not, and it was done when, the head of the snake was able to reach the coordinates of the Apple without hitting the other snake or tail of itself. If that was the case, then newApple() method was called again, to spawn the Apple at new coordinates and a sound effect will be played, so that the user feels like the snake has actually eaten something.

```

public void checkCollisions(){
    check if head collides with the body
    for(int i = bodyParts; i > 0; i--){
        if((x[0] == x[i]) && (y[0] == y[i])){
            running = false;
        }
    }

    if (x[0] < 0)           {running = false;}
    if (x[0] > SCREEN_WIDTH) {running = false;}
    if (y[0] < 0)           {running = false;}
    if (y[0] > 525)         {running = false;}

    if(!running) {
        Database();
        timer.stop();
        clip.stop();
        clip.close();
        clip1.close();
    }
}

public void gameOver(Graphics g){
    clip2.start();
    this.add(gameoverimage);
    gameoverimage.setBounds(0,0,600,500);
}

```

In this part of the game, checkCollisions() method would check if there were any collisions that occurred while playing, if yes, the running would be negated and the game would stop and scores and names of the users will updated in the Database.

gameOver() method is responsible for showing the game over screen at the end and play the game over music.

### 3. Database

**Connection to the database:**

```

public void Database(){
    String mysql_db_url = "jdbc:mysql://127.0.0.1:3306/jdbc_snake_ai";
    String mysql_db_user = "root";
    String mysql_pass = "Nenu#1506";

    Connection connection = null;
    try {
        connection = DriverManager.getConnection(mysql_db_url, mysql_db_user, mysql_pass);
        Statement statement = connection.createStatement();
    }
}

```

The program uses JDBC (Java Database Connectivity) to connect to the database.

The connection details are stored in the following variables:

mysql\_db\_url: the URL of the database, in this case it's a local database with IP address 127.0.0.1 and port 3306, and the database name is jdbc\_snake\_ai.

mysql\_db\_user: the username to access the database.

mysql\_pass: the password to access the database.

### Reading Input:

A `BufferedReader` object is created to read input from the user. The program asks for the user's name and stores it in the "name" variable.

### Checking if the name exists in the database:

```
// Check if the name already exists in the database
PreparedStatement checkStmt = connection.prepareStatement("SELECT COUNT(*) FROM scoreboard WHERE name = ?");
checkStmt.setString(1, name);
ResultSet checkResult = checkStmt.executeQuery();
checkResult.next();
int count = checkResult.getInt(1);

// Insert or update the data in the table
if (count == 0) {
    // The name does not already exist, so insert a new row
    PreparedStatement insertStmt = connection.prepareStatement("INSERT INTO scoreboard (name, score) VALUES (?, ?)");
    insertStmt.setString(1, name);
    insertStmt.setInt(2, applesEaten);
    insertStmt.executeUpdate();
    System.out.println("Inserted new user: " + name + " with score: " + applesEaten);
} else {
    // The name already exists, so update the existing row
    PreparedStatement updateStmt = connection.prepareStatement("UPDATE scoreboard SET score = ? WHERE name = ?");
    updateStmt.setInt(1, applesEaten);
    updateStmt.setString(2, name);
    updateStmt.executeUpdate();
    System.out.println("Updated user: " + name + " with new score: " + applesEaten);
}
```

The program uses a `PreparedStatement` object to check if the name entered by the user already exists in the database. If the name exists, the program will update the score for that name, otherwise, it will insert a new row in the database with the name and score.

### Printing all the data in the table:

```
PreparedStatement selectStmt = connection.prepareStatement("SELECT * FROM scoreboard ORDER BY score DESC");
ResultSet result = selectStmt.executeQuery();

while (result.next()) {
    System.out.println("Name: " + result.getString("name") + ", Score: " + result.getInt("score"));
}
```

### Finding the maximum score:

```
PreparedStatement maxStmt = connection.prepareStatement("SELECT MAX(score) as max_score FROM scoreboard");
ResultSet maxResult = maxStmt.executeQuery();
maxResult.next();
int maxScore = maxResult.getInt("max_score");

// Find the user with the maximum score
PreparedStatement selectS = connection.prepareStatement("SELECT name FROM scoreboard WHERE score = ?");
selectS.setInt(1, maxScore);
ResultSet res = selectS.executeQuery();
res.next();
String maxName = res.getString("name");
```

The program uses a `PreparedStatement` object to find the maximum score in the table and the name of the user with the highest score.

### Closing the connection:

```
connection.close();
```

The program closes the connection to the database after performing all the operations.

## Multiplayer:

```
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Player 1 NAME:");
String player1 = br.readLine();

System.out.println("PLAYER 2 NAME:");
String player2 = br.readLine();

PreparedStatement insertStmt = connection.prepareStatement("INSERT INTO scoreboard (Player1, Player2, score) VALUES (?, ?, ?)");
insertStmt.setString(1, player1);
insertStmt.setString(2, player2);
insertStmt.setInt(3, applesEaten);
insertStmt.executeUpdate();

PreparedStatement selectStmt = conn.prepareStatement("SELECT Player1, Player2, score FROM scoreboard WHERE Player1 = ? OR Player2 = ?");
selectStmt.setString(1, player1);
selectStmt.setString(2, player2);
ResultSet result = selectStmt.executeQuery();
```

If the mode of the game is “Multi Player”, the Game Panel will ask to enter two names. The respective scores will be saved in the Database according to the Player name.

### Compare the score between 2 Player:

```
int score1 = applesEatenP1;
int score2 = applesEatenP2;
// Compare the scores and display the result
if (score1 > score2) {
    System.out.println(player1 + " has the highest score of " + score1);
} else if (score2 > score1) {
    System.out.println(player2 + " has the highest score of " + score2);
} else {
    System.out.println(player1 + " and " + player2 + " have the same score of " + score1);
}
connection.close();
```

As shown in the above code, in case of “Multiplayer”, it will compare the score of both the Players and display the highest score on the top.

### Used Libraries:

We need to add the “java.sql” Package to work with our Database and make a connection with the SQL Workbench. The java.sql package contains API for the following:

Making a connection with a database via the DriverManager facility

Sending SQL statements to a database

Retrieving and updating the results of a query

Standard mappings for SQL types to classes and interfaces in the Java programming language

Custom mapping an SQL user-defined type (UDT) to a class in the Java programming language

Metadata

Exceptions

We would like to briefly describe the concepts we have used for this Database.

Connection interface — provides tools for managing connections and their characteristics as well as techniques for producing statements

Statement — used to send basic SQL statements

PreparedStatement — used to send prepared statements or basic SQL statements (derived from Statement)



ResultSet interface — The outcome of a database query is represented via the Java JDBC ResultSet interface. The paragraph describing inquiries demonstrates how a query’s return value is represented as a java.sql. ResultSet.

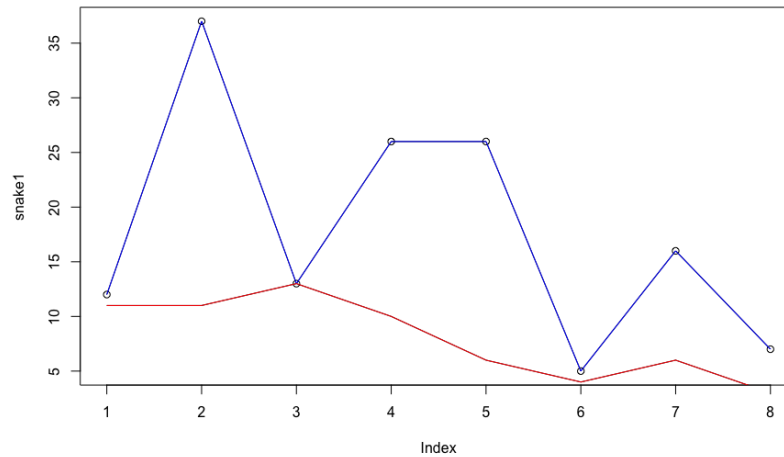
executeUpdate() — Executes the specified SQL statement, which could be an INSERT, UPDATE, or DELETE command, as well as a SQL command that produces no output, like a SQL DDL command.

SQLException — When there is an issue accessing the data, most methods raise SQLException; however, certain methods may do so for other reasons.




IOException — the default class for exceptions thrown while utilizing streams, files, and directories to access data

RuntimeException — when a user invokes a method in the wrong way. A method, for instance, may check to see if one of its inputs is inadvertently null. The method may throw an unchecked exception called a NullPointerException if one of the arguments is null.

## 7 Experimental Results and Statistical Tests



Above shown are the statistical results of the comparison between the hard-coded Algorithm and the A\* Algorithm, and as per the results, it came out that every time the A\* Algorithm performed 60 to 70 times better in comparison to the hard-coded Algorithm.

	 snake1 	snake2 
<b>1</b>	12	11
<b>2</b>	37	11
<b>3</b>	13	13
<b>4</b>	26	10
<b>5</b>	26	6
<b>6</b>	5	4
<b>7</b>	16	6
<b>8</b>	7	3

## 8 Conclusions and Future Work

### 1. How was the teamwork

Being a team of four individuals, who haven't had any previous experience in working with the programming language Java, designing a full-fledged game with the implementation of Artificial Intelligence but quite a challenge. At first, each one of us was quite nervous about whether we will be able to deliver our best or not and do justice to this project and the expectations of our professor, but then as every beginner does, we too started from the start.

The way this project-oriented module was designed, helped us a lot in brushing up our concepts right from the basics and then improving them to the much higher advanced levels.

In the first part of the project, we were given weekly assignments on Kattis, which involved problems that were not only interesting and tricky to solve but also helped us practice our basics, which is quite important if you start learning a new programming language.

We built up our basics by solving these problems on weekly basis as our assign-

ments and eventually learned a lot of new concepts in Java. It started off with basic statement printing and went up to learning the Object Oriented Programming Concepts in Java.

Since all of us were beginners, there were a lot of healthy discussions throughout while working on this project. Everybody's opinion was taken into consideration and whenever there was difficulty in understanding any concept of Java, all the other team members took an initiative to help each other in trying to understand the concept.

Every time a new concept was introduced or was required by the project, we all sat together discussed it thoroughly, watched videos over the internet for easier implementations, referred books and shared all the information among each other so that all were benefited and learnt something new out of it which ideally the goal of doing this project

Weekly meetups were organized over Discord or personally at the University, however it suited the majority of the team members, weekly tasks were assigned to each one of us and these tasks were then discussed in these weekly meetups and the team was quick and responsive whenever there was a problem

Apart from the weekly meetups, we were also quite active in our class, and always questioned whenever there were some doubts present. We took full benefit of the Feedback talks organised by our Professor. We took that chance as an opportunity to get our progress checked, and also boost our confidence by presenting our progress in front of the rest of the students.

Each one of us had their parts prepared and presented accordingly and by this everyone got an equal opportunity to share the stage and present and nobody felt left out

Punctuality and the motivation to deliver our best also helped our team to give their best and complete the project on time while managing all the other modules

Even though each one of us was working on a separate part of the project, namely Algorithm Sorting, Graphical User Interface, and Database, still everyone was always available in times of need which created a kind of the positive balance between the team members, and because of this balance and good understanding among each other we were able to balance the workload of this project with the other modules and by doing this, the rest of our modules were not compromised and hence everything fall back into the place

In a nutshell, the team was well balanced and got along well with each other and due to all these qualities we progressed together, learning new concepts on the way, experimenting with different examples, and hence completing our project on time.

## **2.What did you learn**

### **Learning the basics of Java**

This project as a whole was a complete challenge for us, as we had zero experience in Java as a programming language and were completely new to it. Initially we did our Kattis assignments very diligently in the first part of the

project, the questions we from all the levels, from basic to hard which was good for our learning, we started off with some basic stuff like getting familiar with the syntax since it was a bit hard for us remember in comparison to the other programming languages that we learned till now

We started practicing a few simple problems, through that we practiced the basic syntax of a java program, how to take the input from the user and how to print the output on the terminal, as discussed it was a bit hard to remember the syntax at first but as we practiced those Kattis problems on weekly basis, we got a hang of it and became comfortable with it

After getting comfortable with the syntax and the input output format in Java, we learned the concepts of Object Oriented Programming too. We learned how the basic concepts of Object Oriented Programming like Inheritance, Abstraction, Encapsulation and Polymorphism are implemented in Java. What interesting was the implementation of the concept of Multiple Inheritance in Java, we all know that the concept of Multiple Inheritance is not directly supported in Java because of the ambiguity that is caused by the compiler, so in order to implement Multiple Inheritance in Java, we use the concept of Interfaces

Like this there were many concepts that made Java a very interesting language for us to learn

### **Implementing various Algorithms in Java**

As we have mentioned earlier, we thought about a lot of Algorithms that are used to find the short distance between the two points, in order to head start working on this project during our Brainstorming sessions. We had learned about these Algorithms in the previous semesters, some of these Algorithms include BFS, Dijkstra, A\* etc. We have also briefly discussed about these Algorithms above in this documentation and we have also included a pseudo code While starting this project, the main task was to sort out the Algorithm for the game, so obviously we came up with a lot of Algorithms like these that we have learned before and tried to implement them in Java by coding them, which was again a new learning for all of us, since we had always learned and implemented these Algorithms in C and C++ in our previous semesters, but with Java being a different language than what we have learned till now, it was quite interesting and fun, learning how to implement these Algorithms in Java

Each one of us tried to implement one of these Algorithm on their own in our initial stages while Brainstorming which was followed by a discussion and also sometimes included watching youtube videos and referring books, and by doing all of these we learned a lot about Java as a programming Language, learned about how easy it is to code in Java and how supportive it is in case of Application development, which is why it is counted among one of the most popular and used languages in case of Application Development

### **Implementing the concepts of Java Swing while designing the Graphical User Interface**

Java is considered one of the most versatile language, it has long been the de-facto programming language for creating Web apps, Android apps, and software

development Due to availability of packages like Graphics, AWT etc, it is one of the most popular languages that are used now a days in developing number of softwares and applications

While learning we did not just learn the basic syntax and input output stuff, rather also got introduced to the Object Oriented Programming part of Java, which included numerous interesting concepts like String Builder, implementation of Multiple Inheritance by using Interfaces etc by solving the problems

As we were assigned this project, the first thing that came to our mind was that its a game that is popular among the kids a lot and kids find it interesting when the game they are playing has a lot of interesting graphics involved and also cool sounds in the background. So we were sure that we had to work a lot on the Graphics of the game to make it interesting for the user to play For the graphics part, we learned the concepts of Java Swing and AWT Java The Swing Framework is a part of the Java Foundation Classes that is used to create manage windowed applications. It is built using the AWT (Abstract Window Toolkit) API and is entirely written in Java.

There are many types of classes included in the javax.swing package that provide classes of Java Swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser, etc which we learned about at first while learning the concepts and then implemented the same while designing the Graphical User Interface for our game. We created various buttons, Menus and Jframes by using these concepts. In Java Swing, you can create a Frame either by using the concept of Interfaces or by implementing Inheritance. And hence all the frames that were needed while making the game, were made by using these concepts.

### **Implementing the concepts of Graphics class in Java while designing the Graphical User Interface**

After learning how to create the panels and frames for the game, the next step was to learn how to draw the objects that are required like the snake and the apple, and how to beautify the background grid to make it look interesting

In order to accomplish this, we learned the concepts of the Graphics class included in the AWT package in Java. It is an abstract class that is part of the java.awt package and extends the Object class from the java.lang package. It acts as a superclass for all graphics contexts and allows for the easy realization of various application components on a variety of devices or numerous real images.

Each and every object in the Graphics class is a complete set of all the methods required to implement the fundamental operations of an applet, and as a result, its state includes details about the drawing component, current clip, current color, XOR alternation color, font, and origin translations.

Drawing various visualized components on the screen, which is thought of as a drawing board made up of an infinite number of pixels, is done in the graphics class.

We learned about some of the most important methods that were used to create the objects required by us like the snake, apple etc. These methods were namely

`draw()`, which was used to draw the various shapes or `fill()`, which was used to fill the colour in the objects

### **Connecting Database in Java**

As we have mentioned earlier, during our brainstorming sessions, we decided to include a high score panel in our game, so that the user could see a list of high scores which included the scores of other players who had played the game before, along with his own score. For this we needed a Database, because the name of the user along with his score needed to be stored somewhere and then updated as he played again and again.

In order to implement a Database in our project, we had to first understand what exactly a Database is, we have learned the basics of Databases and SQL alongside this project, in our other module of Databases this semester. A database is a collection of information that has been set up to be easily updated and managed. Computer databases are typically used to aggregate and store information such as sales transactions, customer information, financial data, and product information in data records or files.

Databases can be used to store, manage, and access any kind of data. They compile information on people, places, or things. In order to be seen and examined, it is gathered in one place. Databases are simply a well-organized collection of data.

We had also learned to create a Database schema in Workbench in our Databases module, so to start off with our Databases part of our project, we first created a Workbench schema for our Database with the necessary attributes namely the name of the user which was taken as an input and the score

After creating the Database schema, the next step was to learn how to connect this Database with the Java Code and in order to accomplish we had to learn the concept of JDBC Driver Class, which is actually responsible for building a connection between the Database and the Java Code, and implement it in our code.

The utility classes that are used to complete a task are the driver classes. In order to connect a Java application to a database, JDBC uses driver classes. Driver classes are vendor-specific i. e. Both the Oracle database and the MySQL database offer separate driver classes.

Therefore, we must use the driver class provided by MySQL to connect a Java application to a MySQL database, and we must do the same for other databases. We can go to the official website and download JARs to get the driver class. Later, when connecting the Java application with the database, we can use these JARs. For instance, the Oracle database uses the `OracleDriver` class, and MySQL uses the `Driver` class.

After building the connection, the next that we wanted to know was how to access the functions of the database in our Java code, for that we came across the concept of `PreparedStatement`, which is responsible for this, and hence started brushing up our concepts for the same

A `PreparedStatement` is a pre-compiled SQL statement. It is a subinterface of `Statement`. `Statement` objects lack some useful extra features that are present

in prepared Statement objects. The PreparedStatement object offers a feature to execute a parameterized query rather than hard coding queries.

In a nutshell, by using the concepts of JDBC Driver Class and the PreparedStatement, we learned how to integrate a Database in Java, which not only enhanced the quality of our project but also deepened our knowledge of Databases further

Apart from the technical stuff that we learned while doing this project, we can't ignore the fact that it also taught us how to work in a team, and what all qualities does one need in order to bring everyone together and work in a professional and positive atmosphere. Everything, figuring out the project from the start to solving the problems at step together, we learned what exactly a positive team spirit is and how important it is in order to develop a software or an application

### **3. Ideas for the future development of your application, new algorithms**

#### **Genetic Algorithm**

Another well-liked method for solving this kind of issue is genetic algorithms. This strategy is based on natural selection and biological evolution. A machine learning model (which could, but need not, be a neural network) converts perceptual inputs into action outputs. The snake's proximity to obstacles in each of the four major directions could be used as an input (up, down, left, right). The result would be an action, such as a left or right turn. In the analogy of natural selection, each instance of a model represents an organism, and the parameters represent the genes of the organism.

Starting off, a number of random models are initialized in an environment and let loose, such as neural networks with random weights. The top individuals from each generation are chosen by a fitness function after all the snakes (i.e., models) have died. The Snake fitness function would only choose snakes with the highest scores in this case. The best individuals are then used to breed a new generation, with the addition of random mutations (e.g. randomly tweaked network weights). Some of these mutations will be harmful, some won't change anything, and some will help. The pressure of evolution will choose ever-better models over time.

**Pros:** Simple to understand concept. After training, predicting the next move is quick.

**Cons:** Due to the random nature of mutations, convergence can be slow. The model's performance depends on the inputs it has access to. If the inputs only indicate whether there are obstacles immediately surrounding the snake, the snake won't be able to see the "big picture" and will be more likely to become entangled in its own tail.

---

```

Algorithm:  GA( $n, \chi, \mu$ )
// Initialise generation 0:
 $k := 0$ ;
 $P_k :=$  a population of  $n$  randomly-generated individuals;
// Evaluate  $P_k$ :
Compute  $fitness(i)$  for each  $i \in P_k$ ;
do
{ // Create generation  $k + 1$ :
  // 1. Copy:
  Select  $(1 - \chi) \times n$  members of  $P_k$  and insert into  $P_{k+1}$ ;
  // 2. Crossover:
  Select  $\chi \times n$  members of  $P_k$ ; pair them up; produce offspring; insert the offspring into  $P_{k+1}$ ;
  // 3. Mutate:
  Select  $\mu \times n$  members of  $P_{k+1}$ ; invert a randomly-selected bit in each;
  // Evaluate  $P_{k+1}$ :
  Compute  $fitness(i)$  for each  $i \in P_{k+1}$ ;
  // Increment:
   $k := k + 1$ ;
}
while fitness of fittest individual in  $P_k$  is not high enough;
return the fittest individual from  $P_k$ ;

```

---

## Reinforcement Learning

A rapidly developing and fascinating area of AI is reinforcement learning. Reinforcement learning, at its most basic, entails an agent, a setting, a set of actions the agent may take, and a reward function that rewards the agent for good actions or punishes the agent for bad actions. The agent adjusts its parameters as it explores the environment to maximize its own expected reward. The agent in Snake's case is undoubtedly the snake. The NxN board serves as the setting (with many possible states of this environment depending on where the food and the snake are located). Turning left, right, or continuing straight are all options.

Deep Reinforcement Learning (DRL) combines deep neural networks and the aforementioned principles of RL. Recently, DRL has been used to create super-human chess and go systems, teach computers to play Atari games using only the pixels on the screen as input, and manage robots.

## Deep Q-Learning

DRL has a unique subset called Deep Q-Learning. Although initially difficult to understand, the logic behind it is very elegant. The "Q function," which takes the current environment state as input and outputs a vector containing expected rewards for each potential action, is taught to the neural network. After that, the agent can choose the course of action that maximizes the Q function. The game then changes the environment based on this action and assigns a reward (for example, +10 for eating an apple, -10 for hitting a wall). A neural network with randomly initialized inputs simply approximates the Q function at the start of training.

This is where the Bellman Equation comes in. The neural network is directed in the right direction using this equation to approximate Q. The Bellman Equa-



tion's output increases as the network does. Importantly, the definition of the  $Q$  function contains recursion.

The cool thing about watching Snake AI use Deep Q-Learning to train itself is that you can watch its live exploration and exploitation process. The snake may pass away after 5 moves in some games. Although this may initially seem disappointing, keep in mind that there is a cost involved, and the network will update itself to prevent similar actions in the future. In some games, the snake survives for a very long time, grows a very long tail, and receives a lot of rewards. To help the snake play better in the future, the actions are either positively or negatively reinforced.

**Pros:** It's a really stylish and cool idea. Aside from setting up the environment and reward system, RL can be used for a wide variety of other tasks. In my experience, it converges more quickly than genetic algorithms due to its ability to utilize gradient descent rather than random mutation.

**Cons:** Initially a little difficult to understand. Similar to the genetic algorithm, the performance of the model depends on the inputs that are made available to the network; more inputs entail more model parameters, which lengthens training time.

```
Initialize  $Q(s, a)$  arbitrarily
Repeat (for each episode):
  Initialize  $s$ 
  Repeat (for each step of episode):
    Choose  $a$  from  $s$  using policy derived from  $Q$ 
      (e.g.,  $\epsilon$ -greedy)
    Take action  $a$ , observe  $r, s'$ 
     $Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$ 
     $s \leftarrow s'$ 
  until  $s$  is terminal
```