The Snake Game

Sowmika Adepu ([adepu.s@northeastern.edu](mailto:adepu.s@northeastern.edu)) Kousalyaa Senthilvel ([senthilvel.k@northeastern.edu](mailto:senthilvel.k@northeastern.edu)),

Sarthak Jasani  ([jasani.sa@northeastern.edu](mailto:jasani.sa@northeastern.edu)),

Information Systems, Northeastern University

***Abstract*: This project aims to develop a classic arcade-style Snake Game using Java programming language. The game involves controlling a snake to eat food while avoiding obstacles and the snake's own body. The project's implementation involves defining the game board, entities, game logic, and user interface components such as keyboard input, graphics, and sound effects. Collision detection algorithms are also used to detect collisions between game entities and handle game over situations. The game features multiple difficulty levels to cater to players of different skill levels. The project also emphasizes the importance of testing and debugging to ensure the game runs smoothly without errors. The final product is a polished and engaging Snake Game that showcases the capabilities of Java for game development. This project can serve as a valuable resource for developers interested in game development or learning Java programming. JavaFX is used to create the game's graphical user interface, with the scene graph used to manage the visual components of the game. The report includes a detailed discussion of the different object-oriented programming concepts used in the implementation, including the use of inheritance and polymorphism. Additionally, the testing process used to ensure that the game was working as expected, including unit testing, integration testing, and user acceptance testing, is discussed. In conclusion, the use of object-oriented programming allowed for a clean and modular implementation of the game's logic, making it easier to maintain and extend the codebase**

***Keywords- (***Dashboard , Start, Score , Settings, Quit , Snake, Apple, OOPS concepts, Inheritance and Polymorphism, Interfaces, Collections and Iterators, Lists and Sets, FileI/O, Animations)

# **I. Problem Description**

To incorporate all of the OOPs concepts with JavaFx that we learned throughout the course, we wanted to create something entertaining and engaging. We thought there was no better approach than to create a game that everyone enjoys. ​In the process of game creation, we want to explore more in terms of how to create and apply animations that would be visually appealing to the user that is engaged in the game. While we believe that it is important to develop applications to build for business it is even more important to build something that is memorable for longer persistence.

# **II. Analysis (Related Work)**

The Snake game has been around since the late 1970s and has been widely popularized in the video game industry. Many iterations of the game have been developed over the years, ranging from the classic arcade version to more modern and graphically advanced versions.

One popular implementation of the game is the Snake game developed using Java and JavaFX. This version of the game utilizes the object-oriented programming paradigm and the graphical user interface library of JavaFX to create a more modern and interactive gaming experience for the player.

In terms of similar projects, there are numerous open-source implementations of the Snake game available on the internet that have been developed using Java and JavaFX. Overall, the Snake game developed using Java and JavaFX has been widely implemented and studied in both industry and academia, making it a popular and well-understood game development project.



Figure 1. In-game screenshot of the classic Snake-Game.

# **III. System Design**

We incorporate all the OOP ideas with JavaFx that we learned throughout the course to design the application. Users play with our JavaFX application and the application interacts with some text files which are used to do data persistence.

Text

Description automatically generated

Figure 2. System architecture

In the Snake- Game application, we have four pages which are menu pages and game pages:

Graphical user interface

Description automatically generated

Figure 3. Draft UI designs of menu and game pages

When a user starts the game, user goes to the menu page firstly. In this page there are some buttons including START, SCORES, SETTINGS and QUIT. START button is used to start the game which means going to the game page. SETTINGS is used to set up Manual size of the game grid, select or unselect the powerups in apples & helps to select the difficulty level, SCORES is used to view the highest scores of the players and EXIT button is used to close the application. When user click EXIT button, we save the user’s scores to a specific file and then close the application. Other buttons are used to present some specific information like enter name, scores and so on.

Diagram

Description automatically generated

Figure 4. Workflow of Snake-Game

# **IV. Implementation**

To have a brief overview, we have defined the following classes. The front stage part and the game stage part will be discussed separately as followed.

1. main method

It extends the "Application" class and overrides its "start" method. The "start" method is called when the JavaFX application is started, and it initializes the main menu of the game by creating a new instance of the "ViewManager" class and calling its "initializeMainMenu" method.

It also defines a static "viewManager" object of type "ViewManager", which is used to manage the different views of the game, there is a static method called "getViewManager" that returns the "viewManager" object. This method is used by other classes to access the "ViewManager" instance and switch between different views of the game.

2. Game stage

2.1 Initialization: This class handles the user interactions with the main menu of the game. The class contains four methods: handleStart, handleSetting, handleScore, and handleQuit.The handleStart method is called when the user clicks the start button in the main menu. It first retrieves the powerUp value and grid size from the user input fields in the viewManager object. Then, it sets the difficulty level based on the user's selection. It initializes the game stage, launches the game by creating objects of Game, Obstacle, Snake, and Apple classes, and creates a GameController object. Finally, it sets the gameScene and closes the primary stage. The handleSetting method is called when the user clicks the setting button in the main menu. It hides the mainSubScene and settingSubScene and shows the scoreSubScene. The handleScoremethod is called when the user clicks the score button in the main menu. It hides the mainSubScene and scoreSubScene and shows the settingSubScene. The handleQuit method is called when the user clicks the quit button in the main menu. It terminates the application by calling the System.exit method. The launchSnake method initializes the objects required for the game, such as Game, Obstacle, Snake, and Apple. It also creates a GameView object and a GameController object to manage the game's visual display and user inputs. It sets up the gameScene and adds all the required components to the gamePane. The setGridSize method validates the user input values for grid size, and if they are valid, it sets the X\_VALUE and Y\_VALUE accordingly. The setDifficulty method sets the difficulty level based on the user's selection.

In summary, the MainMenuController class is responsible for managing user interactions with the main menu of the Snake game. It sets up the game environment by initializing objects and setting up the game view. It also handles the transition between the different sub-scenes in the main menu., as mentioned in part 1.

2.2 Start button: The handleStart method is used to handle the "Start" button action in the main menu of the Snake game. This method creates a new instance of the Game class with the specified width, height, power up mode, and difficulty level. It then creates new instances of the Obstacle, Snake, and Apple classes using the Game object. Next, it creates a new GameView object and a new GameController object. The GameController object takes the Snake, Apple, Game, GameView, Obstacle, and gameStage objects as parameters. Then, it creates a new StackPane object to hold the GameView elements and sets the scene to the gameScene object. It sets the gameScene object's OnKeyPressed event handler to the handle method of the GameController object. Finally, it adds the GameView elements to the StackPane object and sets the padding and alignment of the elements. This method is called when the "Start" button in the main menu is clicked. It initializes the game and launches it by setting the game scene to the game stage, closing the main menu stage, and showing the game stage. [7].

2.3 Settings button: The handleSetting method is part of the MainMenuController class, which controls the main menu screen of a simple snake game. This method is responsible for handling the user's click on the "Settings" button, which triggers the transition to the settings screen. When the user clicks the "Settings" button, the handleSetting method is called. This method accesses the viewManager object, which is an instance of the ViewManager class. The viewManager object is responsible for managing the different sub-scenes (i.e., sub-windows) of the game.

The handleSettingmethod first calls the translateSubSceneOut method on the main sub-scene and the setting sub-scene of the game. This method is responsible for animating the sub-scenes out of view, giving the impression that they are sliding off the screen.

After the main sub-scene and setting sub-scene have been animated out of view, the translateSubSceneIn method is called on the score sub-scene. This method animates the score sub-scene into view, giving the impression that it is sliding into the screen.

In summary, the handleSetting method is responsible for handling the user's click on the "Settings" button and triggering the transition to the settings screen. It accomplishes this by animating the main sub-scene and setting sub-scene out of view and animating the score sub-scene into view.[8].

2.4 Quit button: The handleQuit method is a simple method that is called when the user clicks the quit button in the main menu. The method simply calls the System.exit(0) method, which terminates the currently running Java Virtual Machine (JVM). This effectively quits the game and closes the application. This method is a simple method that is called when the user clicks the quit button to exit the game.

2.5 Score button: The handleScore method is responsible for handling the event triggered when the user clicks on the "Score" button in the game's main menu. This method is called when the handleScore button is clicked in the GUI. The handleScore method is relatively straightforward, as it simply triggers a few actions that enable the user to view the game's high scores.

When this method is called, it first retrieves the ViewManager instance associated with the game, which manages the GUI components of the game. The method then calls three methods from the ViewManager instance: translateSubSceneOut on the mainSubScene and settingSubScene, and translateSubSceneIn on the scoreSubScene. These methods animate the different sub-scenes that are displayed on the GUI by sliding them in and out of view. By calling these methods, the handleScore method effectively hides the main menu and settings scenes and displays the high scores scene.

3. Front stage

3.1 constructor

This is a Java program that controls a simple snake game using the JavaFX library. It includes a MainMenuController that handles user interactions with a game menu. The class includes methods for starting the game, handling game settings and high scores, and quitting the game. The handleStart method initializes the game with the selected game settings and launches it in a new game stage [9]. The launchSnake method creates the game objects and initializes the game view with the canvas and labels. The setGridSize method validates and sets the game grid size based on user input. The setDifficulty method sets the game difficulty level based on user selection. Overall, the MainMenuController class is the main part of the game's user interface, managing user interactions and starting the game with the correct settings.

3.2 handleStart:

This method is called when the user clicks on the "Start" button on the main menu. It sets the game parameters such as power-ups, grid size, and difficulty based on the user's input from the view. It then initializes a new game and creates all the necessary objects such as the snake, apple, and obstacle. It also creates a new game view and game controller objects and sets the scene for the game. Finally, it hides the main menu stage and shows the game stage for the user to start playing.

Methods called here will be discussed as followed:

3.2.1 setGridSize:

This method takes the user's input for the grid size of the game and checks if it is valid (between 5 and 100). If it is valid, it sets the X and Y values for the game grid.

3.2.2 setDifficulty:

This method checks the user's input for the difficulty level of the game (easy, medium, hard) and sets the difficulty level accordingly (E, N, H).

3.2.3 launchSnake:

This method creates a new game object with the grid size, power-up, and difficulty level set earlier. It then creates the snake, apple, and obstacle objects for the game. It creates a new game view and game controller object and sets the scene for the game. Finally, it adds all the necessary elements to the game pane and aligns them accordingly.

3.2.3.1 GameView.makeScene:

This method takes in the game scene, snake, and apple objects, and creates a new game view with the necessary elements such as the grid canvas, game over label, game won label, high score label, and score label.

3.2.3.2 GameController.handle:

This method is a key event listener that listens for the user's input on the keyboard and handles the movement of the snake accordingly.

3.2.4 handleQuit:

This method is called when the user clicks on the "Quit" button on the main menu. It exits the program with a status code of 0.

3.2.5 handleSetting:

This method is called when the user clicks on the "Settings" button on the main menu. It hides the main menu and settings subscene, and shows the score subscene for the user to view high scores.

3.2.6 handleScore:

This method is called when the user clicks on the "High Scores" button on the main menu. It hides the main menu and score subscene, and shows the settings subscene for the user to adjust game settings.

# **V. Evaluation**

The explanation and sample run screenshots:

When the player starts the game, the game home screen welcomes the user with its intuitive and modern UI as shown below

Every element of this UI is meticulous crafted using Javafx Scenes and subscenes for the smoother transitions

Graphical user interface, text, application, chat or text message

Description automatically generated

Figure 6. Game Home Screen

To guide the player, we have added description of our game in the dashboard and there is set of different points mentioned about to apples to ensure the player is exited to play.  
  
The description of the five different color of apples shows the game flexibility and advancement of the application

Before starting the game, player can customize the settings as per the players requirement in "SETTINGS" tab where the player can customize the game settings according to their preferences. This customization can be done by selecting from the available options for difficulty, grid size, and bonus apples.

The difficulty level can be set by the player depending on their skill level or preference. There may be multiple difficulty levels to choose from, ranging from easy to hard, or beginner to expert.

Grid size refers to the size of the game board or playing area. This can also be customized by the player according to their preference. Larger grid sizes can provide a more challenging game experience, while smaller grid sizes can be easier to manage.

Bonus apples are additional items that can be collected by the player during the game, and they can add extra points or benefits to the player's score. The player can choose whether to include bonus apples in the game or not, by checking or unchecking the corresponding checkbox in the settings.

Once the player has selected their preferred settings, these choices will be recorded in the game through checkboxes. This means that the player's chosen settings will be displayed on the game screen during play, allowing them to easily see the chosen difficulty level, grid size, and whether bonus apples are included. This can also help the player keep track of their chosen settings for future games.

Graphical user interface, text, website

Description automatically generated

Figure 7. Settings Tab

.  
Playing the main game is always fun, so, let get into the main part! Once the player clicks on the "START" tab, the game screen will appear. The game screen will display the chosen difficulty level, grid size, and bonus apples, as well as the player's score, remaining time, and number of lives.

The objective of the game is to guide the snake to eat as many apples as possible without colliding with the walls or its own tail. As the snake eats the apples, it grows longer, making it more difficult to navigate through the grid.

The player can control the snake's movements using the arrow keys on their keyboard. The game will continue until the player has run out of live.

Chart, scatter chart, bubble chart

Description automatically generated

Figure 8. Snake-Game screen

when the snake catches an "apple," the player's score increases. However, the game has a unique feature where the color of the apple changes after the snake eats a certain number of high apples. For instance, after receiving 5 points, the red apple will turn blue. After accumulating 15 points, the apple changes to yellow. Once the player reaches the yellow colored apple, the game will also increase in speed, making it more challenging.

As the player continues to progress in the game, the colors of the apple will change. After getting an additional five points, the apple will turn multicolor. Once the player reaches a total of 25 points, the apple will change to brown. The color-changing feature adds an extra level of excitement and challenge to the game, as players need to focus on the changing colors and adjust their strategies accordingly.

Moreover, as the game progresses and the player's score increases, the speed of the snake also increases. This further adds to the game's difficulty, as the player needs to maintain control of the snake while it moves faster. Therefore, players need to be quick and agile to keep up with the pace of the game and avoid running into obstacles.

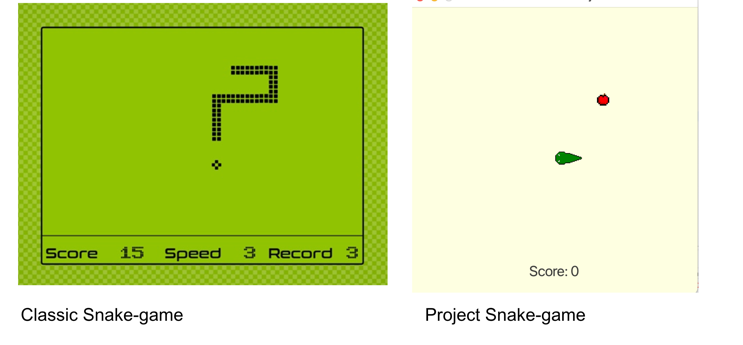
Text

Description automatically generated

Figure 9. Highest Scores of the game with different players

When the game ends the user can always check his scores in the **Scores** tab as tested above.

**Comparison between our solution and other people’s work:** The Snake Game is an ageless classic, carrying the legacy of its own. There is no denying that classic is fun to play even in today’s Gen-Z era. We are not comparing our work to the classic; we want to take the concept of Snake-Game to the new and fun horizon.



`

Figure 10. Comparison between the native Snake game to our proposed solution

**User Testing and Reviews**: This game undergone multiple revisions and tests by real users. Three people were randomly picked at the Snell Library, Northeastern University and were given questionnaire, based on test result few drastic changes were done to the game-mechanics. After few repetitions, another batch of people were picked for the testing, and these are the real reviews we have collected.

*This is some real fun I had after long time, this game looks legit and invokes the memories that I’ve experienced in my childhood. I used play snake game in old nokia phones. Thanks’ for asking to review this.*

*-Ankur, Abhinav (MSIS).*

*Snake game is a crazy idea to pull off. I love this game, it’s awesome. I never thought we can make games like these for final projects. I’m a Bio engineering grad, if I’ve known that I can make game’s and learn, I’ve could a picked CS or IS as my major. Too bad, it’s already late.*

*-Prem Varma (MS Bio Engineering)*

# **VII. Discussion (Reflection)**

The results and the data or outcome of our project; are provided with some insightful discussions in this section.

**The Idea:** The objective of the project “Snake Game” is to learn and demonstrate the concepts of Object-Oriented design in an exuberant way. It is started way before the idea of Snake game or before the idea of a game. Collaboration is always the key for as after multiple brainstorming session and confirming that all the possible concepts and beyond are covered using JavaFX with fun. This gave birth to “The Snake-Game”.

**The Challenges:** Building the in Game-Mechanics is quite a challenge. As it not only requires the mastery on the OOD concepts, it also demanded more and beyond from the all the team members. After spending countless. After setting some elbow grease, most of the game mechanics was tweaked and improved including pace of the apples, snake, movements etc. in the process we learned the process of controlling the **Frames-Per-Second (FPS)** in the game.

One of the main hurdles this project encountered is the intuitiveness of the User Interface. As it is the facade of the application, it should be as modern as possible without sacrificing the fluidity of the game or animations. It took countless hours of research, which ended up in learning of uncharted approaches of JavaFX called subscenes.

**The Outcome:**

Apart from learning and exploring the concepts like

* Class Definition​ [1]
* Inheritance and Polymorphism​ [2]
* Interfaces[3]
* Collections and Iterators​ [4]
* Lists​ [5]
* Sets​ [5]

This project made us explore the uncharted concepts of JavaFX like **javafx.animation.\*** [10] class along with mastering the very concept of **Animationtimer** and its manipulation techniques.

Mathematical equations were used to determine the predict the and calculate the co-ordinates of the in-game elements.

The project cannot see it’s full potential unless the team members collaborate synchronously by orchestrating the plans perfectly together. **Collboartion** is a key skill have handy not only for this project, in real-time job scenarios too

# **VIII. Conclusions and Future Work**

In this project, we have come up with a game that is everyone’s favorite i.e., Snake Game using JavaFx and all the concepts that we learned from the Object-Oriented Design Course. We mainly concentrated on changing the usual snake game to give it a new look and feel so as to make it more interesting and at the same time storing the player’s score in a local file once a game is completed.

The following are the problems we found but not yet explored in the project. They are:

* We have built this game such that only one player can play the game at a time.
* We are using files to store user scores as a result of which the scores are not reflected immediately so we have to run the game once again to view a player’s score.
* Once a player loses one game, we aren’t allowing the player to continue to play, they has to start over.
* Player score cannot be seen until they reach highest score.

We as a team feel that we can still improve the game and take it to a level higher. The following are the improvements that we would like to do to this game in future:

* We would like to make it a multi-player game in which more than one player can play at a time.
* We also would like to let the user pick their favorite snake color before they start playing the game.
* As mentioned earlier, we would like to integrate the game with a database to store the information generated from the game which includes storing scores, results from tournaments and rewards won by players. This would make the user score details readily available in the game though they don’t score high.
* We also would like to improve in terms of adding animations and sound effects to the game a little more.

# **IX. Job Assignment**

* Sowmika Adepu - worked on contributed to the user interface (UI) portion of the project. has worked on how the buttons that are visible on the UI accomplish tasks like starting the game and showing score information, and many more, in short, all the operations that are performed by the buttons on the UI were done.
* Sarthak Jasani - worked on the method that controls the game, worked on the implementation of JavaFX, worked on the subscene, animation of the subsccene, worked on the apple's position, points and appearance, worked on the constructors, classes, superclasses and defining objects.
* Kousalyaa Senthilvel -Worked on game dynamics, collision between the snake and apple, generating apple at random intervals at random position, difficulty level, reading player names and updating the highest score in the file, setting width and height of the game panel, calculating the score, generation of obstacles, worked on constructors.

##### **X. References**

1. https://www.geeksforgeeks.org/classes-objects-java/.
2. https://www.codecademy.com/learn/learn-java/modules/learn-java-inheritance-and-polymorphism/cheatsheet.
3. https://www.geeksforgeeks.org/interfaces-in-java/.
4. https://www.geeksforgeeks.org/iterator-vs-collection-in-java/
5. https://www.geeksforgeeks.org/difference-between-list-and-set-in-java/
6. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Button.html
7. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/SubScene.html
8. https://docs.oracle.com/javase/8/javafx/api/javafx/scene/control/Label.html
9. https://docs.oracle.com/javase/8/javafx/api/javafx/stage/Stage.html
10. https://docs.oracle.com/javase/8/javafx/api/javafx/animation/Animation.html