Stephen F. Austin State University

Team 1- Snake Game User Guide

Students: Adan Morones
Janeta Grigoras
Mark Turner
Sreelakshmi Atholi

Fall 2024

1. Introduction

The Snake Game App is a modern rendition of the classic arcade game, designed to bring simple gameplay to players of all ages. This app combines a sleek, user-friendly interface, making it an engaging choice for casual gaming, improving reflexes, and helping in logical thinking.

In this game, the snake can be controlled and moved around the grid, growing longer each time it eats a fruit. The objective of the game is to collect as many fruits as possible and avoid colliding with the edges of the grid or the snake's own body. With every fruit consumed, the challenge increases as the snake grows and it becomes trickier.

The Snake game app is a blend of dynamic animations, key controls, and randomized elements that keep every playthrough fresh and exciting. This game journey will empower the timeless test of skill and strategy.

---

2. Program Overview

The Snake Game is implemented using five classes and one interface:

- **App**: Entry point of the game.
- **View**: Manages the graphical interface.
- **Model**: Handles game logic, including the snake, fruits, and walls.
- **Controller**: Links user actions to game behaviors.
- **SnakePart**: Represents individual parts of the snake.
- **Interface Fruits**: A blueprint for all types of fruits.
  - **TastyFruit**: Fruits that make the snake grow.
  - **ToxicFruit**: Fruits that reduce the snake's size or end the game.

1. *App.java*

Initializes the game by creating instances of the view, model, and controller.

Method Summary

Modifier and Type	Method and Description
void	start(Stage primaryStage) The method is called when the JavaFX application starts.
void	main(String[] args) Main method that launches the JavaFX application.

2. *View.java*

View uses JavaFx to create a window with a grid layout representing the game environment. The *View* class manages the display of game elements like snake, fruits, and the score.

Field Summary

Modifier and Type	Field and Description
private ArrayList<Rectangle>	tiles Represents the tiles in the game grid.
private Stage	stage Primary window for displaying the game.
public GridPane	tileGrid Holds the tiles for the game.
private HBox	upperPane Holds the score label.
private double	sceneWidth Width of the scene in the game window.
private double	sceneHeight Height of the scene in the game window.
private int	score The score of the player.
private int	rows The number of rows in the game grid.
private int	columns The number of columns in the game grid.
private SnakePart	snake Represents the head of the snake.
private ArrayList<SnakePart>	sBody List of SnakePart objects representing the body of the snake.
Button	quitButton When clicked, it exits the game.
private ArrayList<Fruit>	listOfFruit List of Fruits that will be generated for the game.

Constructor Summary

Modifier	Constructor and Description
public	View(Stage stage) Initializes the game view, sets up the game grid, and creates the initial snake and fruit.

Method Summary

Modifier and Type	Method and Description
private void	setGrid() Sets up the grid layout with tiles.
private void	displaySetUp() Sets up the games components, such as the snake, fruits, score label, and quit button.
public void	show() Displays the game.
public void	setLabel(int score) Updates the score level on the top of the screen.
public void	removeFruit(Fruit fruit) Removes the eaten fruit from the screen, as well as from the list of fruits
public void	addFruit() Adds the fruits in the list to the grid.
public void	updatePos(double centerX, double centerY) Updates the position of the snake’s body.
public ArrayList<Rectangle>	getTails() Returns a list of the tails on the grid.
public ArrayList<SnakePart>	getsBody() Returns a list of the body parts of the snake.
public double	getTilesColumns() Returns the number of columns in the tileGrid.
public double	getTilesRows() Returns the number of rows in the tileGrid.
public double	getSceneWidth()

	Returns the width of the scene.
public double	getSceneHeigth() Returns the height of the scene.
public SnakePart	getSnake() Returns the head of the snake.
public HBox	getBottomPane() Returns the bottom pane of the screen.
public ArrayList<Fruit>	getListOfFruit() Returns the list of the fruits in the game.
public int	getScore() Returns the current user's score.

3. ***Model.java***

Field SummaryConstructor Summary

Modifier and Type	Field and Description
View view	view
Int dirX	dirX Stores the direction of the x value of the snake. Either 1 or -1.
Int dirY	dirY Stores the direction of the y value of the snake. Either 1 or -1.
Double centerX	centerX Stores the x coordinate for the snake
Double centerY	centerY Stores the y coordinate for the snake
SnakePart snake	snake Contains a rectangle along with dirX & dirY as well as centerX & centerY
int speed	Speed Contains the speed of the snake.
ArrayList<Rectangle>	Tiles

tiles	Contains a list of the tiles.
ArrayList<SnakePart> sBody	sBody List of SnakePart objects representing the body of the snake.
IntegerProperty score_property	score_property Used to increment the score as Fruit are devoured.
int currentDirection	currentDirection Stores the direction that the snake head is currently moving in. 1 is left, 2 up, 3 is right, 4 is down
int lastStoredDirection	lastStoredDirection Stores the last stored direction. 1 is left, 2 up, 3 is right, 4 is down
ArrayList<Fruit> listOfFruits	listOfFruits Stores fruits.

Constructor Summary

Modifier	Constructor and Description
public	Model(View view) Initializes the game model, sets up the snake movement, fruit spawner, and update the snake direction and size.

Method Summary

Modifier and Type	Method and Description
Public void	updateDirection(int, SnakePart) Updates the currentDirection according the the int passed through the parameter.
Public void	moveSnake() Moves the snake’s coordinates.
Public void	growSnake(int) Grows the snake by one snakePart.
Public void	shrinkSnake(int) Removes the snake by one snakePart.

Public boolean	isGameOver() Check if the conditions are met to end the game. Returns true is the conditions are met, returns false if the game should continue.
public boolean	checkWallCollision() Check if the snake hits the wall. Returns true is the snake hits the wall else it returns false.
public boolean	isGameWon() Returns true if the snake fills up the grid
Public void	changeDirection(int) Changes the direction on the snake.
public int[]	getFrontCoordinate() Returns an int array holding the x coordinate in 0 and the y coordinate in 1
Public boolean	checkBodyCollision() Returns true if the snake moves over itself. False otherwise.
Public int[]	randomXY() Generates a random x and y coordinate to spawn a new fruit. Returns an int array holding the x in 0 and the y in 1
Public void	spawnNewFruit() Spawns a new Fruit on the grid where the snake isn’t
Public void	addGoodFruit() Adds a fruit that increases the score.
public void	goodAndEvil(int) Randomly decides if it will spawn more fruit, and randomly decides if it will spawn a good or bad fruit.
Public void	eatFruit() When the snake intersects the fruit it will disappear and modifies the score based on the type of the fruit.
public int	getDirX() Returns dirX
public int	getDirY() Returns dirY
public void	setDirX(int) Sets dirX = to the int in the parameter
public void	setDirY(int) Sets dirY = to the int in the parameter

public double	getCenterX() Returns centerX.
public double	getCenterY() Returns centerY.
public int	getLastStoredDirection() Returns the lastStoredDirection. 1 = left, 2 = up, 3 = right, and 4 = down.
public IntegerProperty	getScoreProperty() Returns the scoreProperty.

4. ***Controller.java***

The controller is the class that manages interactions between the game model(game logic) and the game view(user interface). The class handles the user inputs, updating the game state, and ensuring the interface reflects these changes in real time.

Modifier and type	Field and Description
Model	model Holds the game logic, snake movement and score
View	view Manages the graphical user interface and visual representation of the game.
Timeline	timeline Animating the game by repeatedly updating the game state.

Constructor summary

Modifier	Constructor and Description
public	Controller(Model model, View view) Initializes the game by linking the Model and View.

Method Summary

Modifier and Type	Method and Description
public void	setupGameControls() Sets up various controls and game features by calling all necessary methods.
public void	setUpKeyControls() Configures the key controls for the game, it listens for arrow key presses to change snake direction
public void	move() Moves the snake and updates the view,checking for game over and game win conditions. Calls the corresponding methods for actions based on the game status.
public void	animate() Starts the animation of the game by initializing a Timeline to repeatedly call move() method. The creates the game loop, where the snake moves and updates continuously.
public void	setupCounter() Sets up a listener to update the score label in the view whenever the score changes in the model.
public void	startGame() Starts the game by playing the animation timeline beginning the game loop.
public void	endGame() Ends the game by stopping the animation timeline and displaying the game over message with the final score.
Public void	winGame() End the game by stopping the animation timeline and displaying a “you won” message with the final score.

5. *SnakePart.java*

The class represents an individual segment of the snake’s body. It holds the position, size, direction, and the representation of each body part.

Field Summary

Modifier and Type	Field and Description
private double	centerX The X-coordinate of the snake’s body part
private double	centerY The Y-coordinate of the snake’s body part
private Rectangle	head A rectangle representing a part of the snake’s body
private boolean	isHead A flag indicating if the current part is the head of the snake
private int	dirX The X direction of the snake part
private int	dirY The Y direction of the snake part

Constructor Summary

Modifier	Constructor and Description
public	SnakePart(double x, double y, double size, Color color, boolean isHead) Constructs a new part of the snake with the specified parameters.

Method Summary

Modifier and Type	Method and Description
public double	getCenterX() Returns the X-coordinate of the snake’s body part
public double	getCenterY() Returns the Y-coordinate of the snake’s body part
public void	setCenterX(double centerX) Sets the X-coordinate of the snake’s body part
public void	setCenterY(double centerY)

	Sets the Y-coordinate of the snake's body part
public Rectangle	getRectangle() Returns the current body part of the snake
public void	setDirX(int dirX) Sets the X direction of the snake's body part
public void	setDirY(int dirY) Sets the Y direction of the snake's body part
public int	getDirX() Return the X direction of the snake's body part
public int	getDirY() Return the Y direction of the snake's body part

6. ***Fruits.java***

The interface defines the common behavior for all fruit types in the game. It ensures that any fruit class implements methods to retrieve the position and the actual fruit.

Method Summary

Modifier and Type	Method and Description
public double	getCenterX() Returns the X coordinate of the fruit
public double	getCenterY() Returns the Y coordinate of the fruit
public Circle	getFruit() Returns the current fruit

7. ***TastyFruit.java***

The class represents the good fruit that the snake can eat. When the snake consumes the fruit, the score increases. The *TastyFruit* class implements the *Fruit* interface.

Field Summary

Modifier and Type	Field and Description
private double	centerX The X coordinate of the fruit
private double	centerY The Y coordinate of the fruit
private Circle	fruit The Circle object representing the fruit

Constructor Summary

Modifier	Constructor and Description
public	TastyFruit(double centerX, double centerY, double radius) Constructs a new fruit with the specified center and radius, and color red.

Method Summary

Modifier and Type	Method and Description
public double	getCenterX() Returns the X coordinate of the fruit
public double	getCenterY() Returns the Y coordinate of the fruit
public Circle	getFruit() Returns the current fruit

8. *ToxicFruit.java*

The class represents the bad fruit that the snake can eat. When the snake consumes the fruit, the score decreases and the snake shrinks. The *ToxicFruit* class implements the *Fruit* interface.

Field Summary

Modifier and Type	Field and Description
private double	centerX The X coordinate of the fruit
private double	centerY The Y coordinate of the fruit
private Circle	fruit The Circle object representing the fruit

Constructor Summary

Modifier	Constructor and Description
public	ToxicFruit(double centerX, double centerY, double radius) Constructs a new fruit with the specified center and radius, and color red.

Method Summary

Modifier and Type	Method and Description
public double	getCenterX() Returns the X coordinate of the fruit
public double	getCenterY() Returns the Y coordinate of the fruit
public Circle	getFruit() Returns the current fruit

3. Features and Capabilities

- **Dynamic Gameplay**: Random fruits spawn with effects based on type (tasty & toxic).
- **Snake Growth**: The Snake grows larger when eating tasty fruits.
- **Snake Decay:** The Snake gets smaller when eating toxic fruits.

- **Interactive Controls**: Use arrow keys to navigate the snake up, down, left, and right. The direction inputed must not be the opposite direction that the snake is currently heading. Ex: if the snake is heading up, you cannot go down.
- **Score tracking**: The score updates instantly as the snake eats a fruit. The score increases when eating a tasty fruit, but decreases when eating a toxic fruit.
- **Surviving within bounds:** As you traverse and devour fruit you must stay on the grid without going out of bounds if you try to leave the grid, it's GAME OVER. As you devour more fruit the snake's size will also increase, restricting the number of squares on the grid that you can occupy. If you get in your own way and end up hitting yourself, it's GAME OVER. This game's message is best exemplified by a single quote: "mo money mo problems."(the fruit being money)

---

4. Program Limitations

- Currently, the snake's body is represented by body parts scaled 36 times the number of grid tiles to ensure they are visible on the screen. When a new body part is added, we skip over score * 36 parts to position it correctly. While this ensures visibility, it results in inefficient memory usage, as we do not use each of the parts initialized. This approach also lacks flexibility, making it harder to adjust for different screen sizes or gameplay settings without compromising performance or memory efficiency.

---

5. How to run/play the Snake Game

Important! Make sure you have installed JavaFx!

1. Start the Game: The game begins with the snake having a length of 1 and a score of 0.
2. Movement: Use the arrow keys (up, down, left, right) to control the snake's direction.
3. Eating a Tasty Fruit: When the snake eats a tasty fruit, represented as a red dot, it will grow longer by one segment and the score will increase by 1 point.
4. Eating a Toxic Fruit: If the snake eats a toxic fruit, represented by a grey dot, its length will decrease by one segment and the score will drop by 1 point. If your score reaches 0, the game is over.
5. Game Over: The game ends if the snake collides with itself, a wall, or eats a toxic fruit when its score is critical (e.g., 0).
6. To continue playing, avoid collisions and strategically eat tasty fruits to grow the snake and increase the score while avoiding toxic fruits.
7. To win the game your snake must fill up all the screen.

---

6. Example Screens

Main Screen:

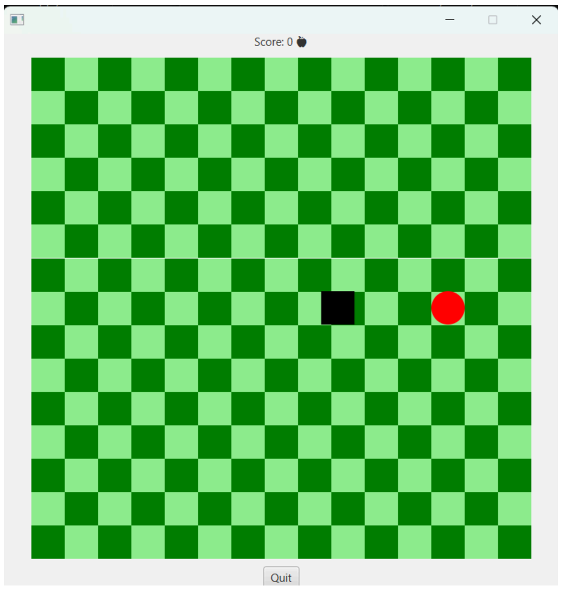

Figure 1. Starting screen of the game

Game Won Screen:

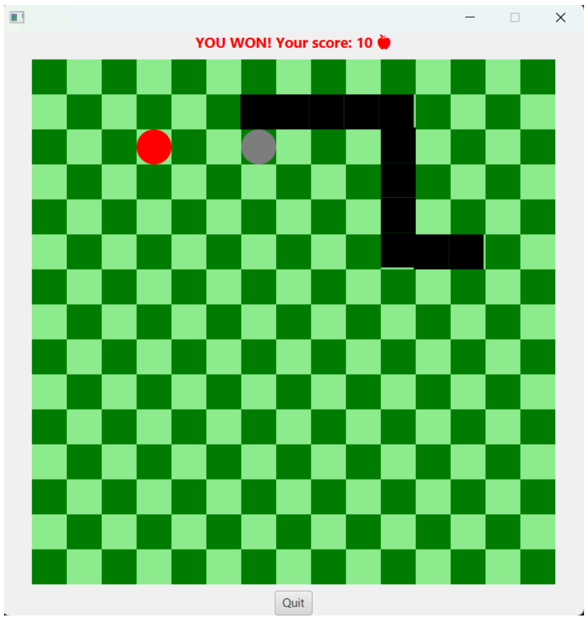
YOU WON! Your score: 10

Figure 2. Winning screen

Note: the logic of the winGame() was changed so we could showcase how winning the game would look like.

Game Over Screen:

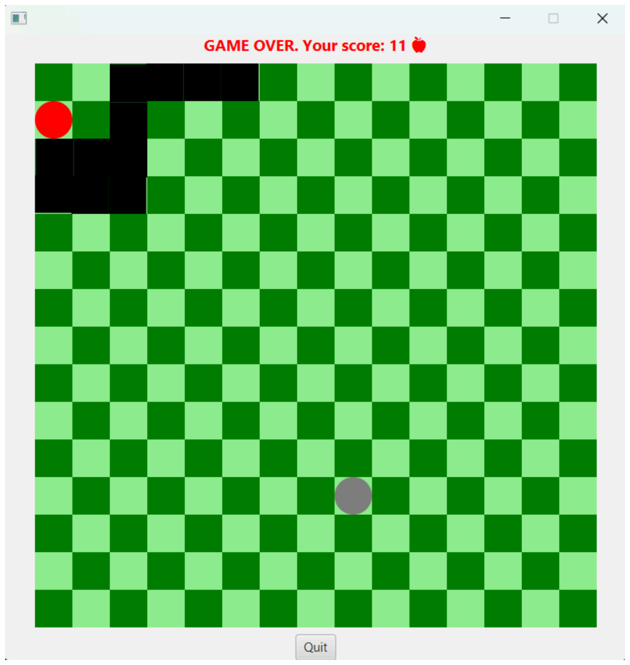

Figure 3. Loosing the game by colliding with itself

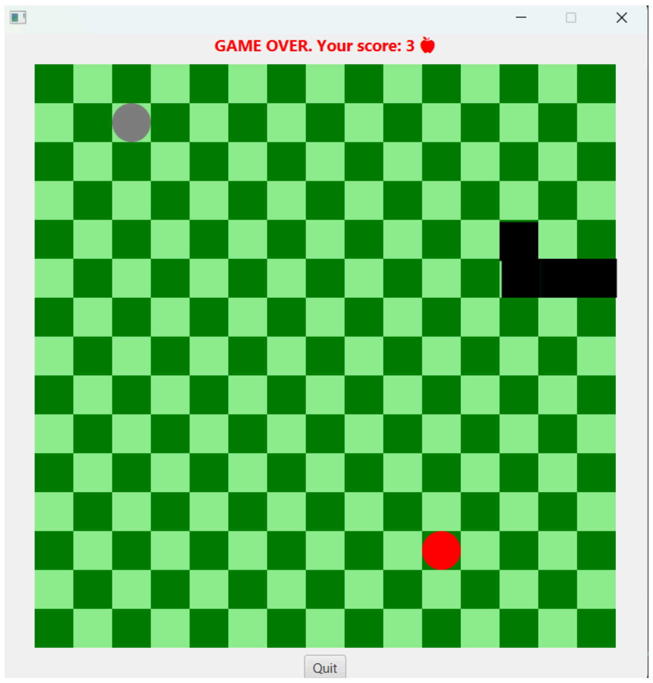

Figure 4. Loosing the game by colliding with itself

7. Conclusion

The Snake Game is a classic arcade game that challenges the player to control a snake throughout the game guiding through the obstacles. Throughout this project we have explored how to implement the classic game using JavaFX, applying the Model-Controller-View design pattern to create a well structured game. The architecture ensures a basic separation between the game logic, user interface, and control flow, which makes it easier to manage and update the game in the future.

Overall, the project not only demonstrates core game development principles but also provides a foundation for more complex games in the future. This game is a fun and engaging project, which is better in simplicity and depth, making it a great exercise for learning programming, game development, user interface design.