SNAKE GAME

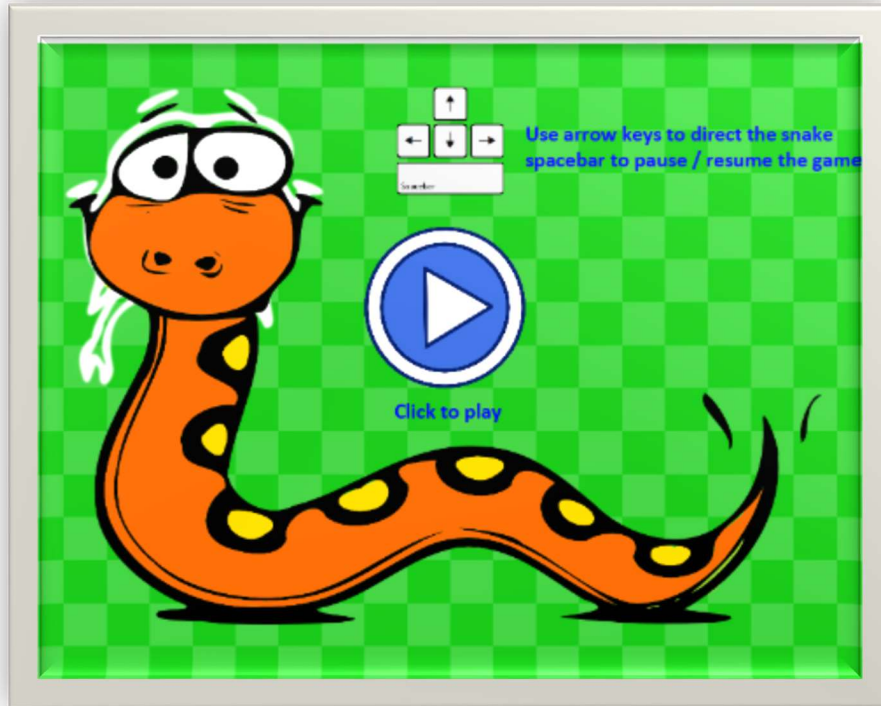**FINAL PROJECT**

COURSE: COIS 2240H
Professor: Alaadin Addas
Institution: Trent University

GROUP MEMBERS:
Fraser Raney
Shivam Parekh
Svyatoslav Loboda
Jiatao Wen

Date Published: April 6th, 2018

TABLE OF CONTENTS

SOFTWARE OVERVIEW

Usage

The player scores points when the snake eats an apple, but the snake grows longer, which makes it more challenging. Enjoy!

Functionality

An introductory scene is displayed with the gameplay instructions, the scene switches to the game with a mouse click.

The player chooses the direction of the snake using the arrow keys on the keyboard. The player may pause the current game using the spacebar.

There are 5 difficulty levels and the length that the snake increases per point increase with each level. When the game ends, the high scores are displayed and the game returns to the introductory scene. If the player has beat a high score they are prompted to enter their initials.

There are sound effects. Scores are stored in a database.

Importance

The development of our version of the game became a fundamental experience for us. The project continued to develop from its inception; we omitted features that were in the original project plan such as a faster game speed and menu bar. We reasoned through these decisions and made other modifications to our original idea, such as, increasing the number of difficulty levels from 3 to 5. We
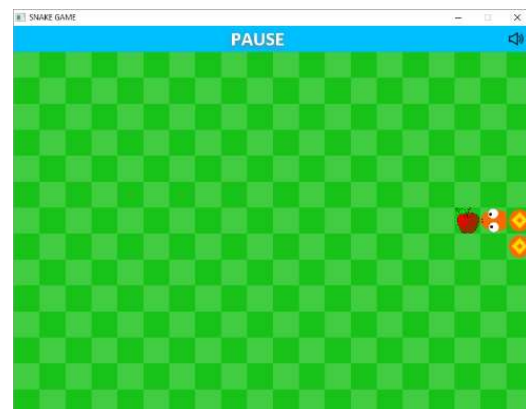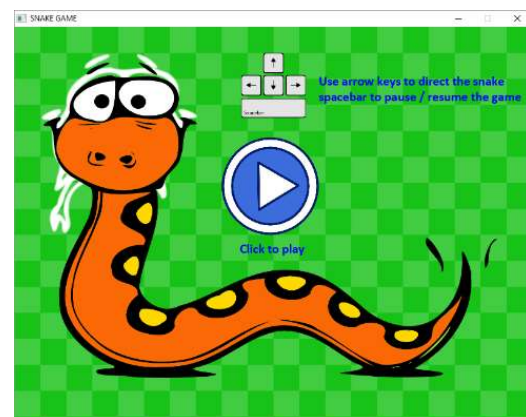
learned that project development is an organic experience. Even our roles in the process changed from the original plan. The importance of the project is not the software or its documentation, but what we all gained in its development.

SOFTWARE DESCRIPTION TO INDUSTRY TARGETED

At least one person in 63% of US households plays video games regularly, and 47% are between 18 and 47. Video games are not exclusively enjoyed by males, in fact, 41% of gamers are women (Frank, 2016).

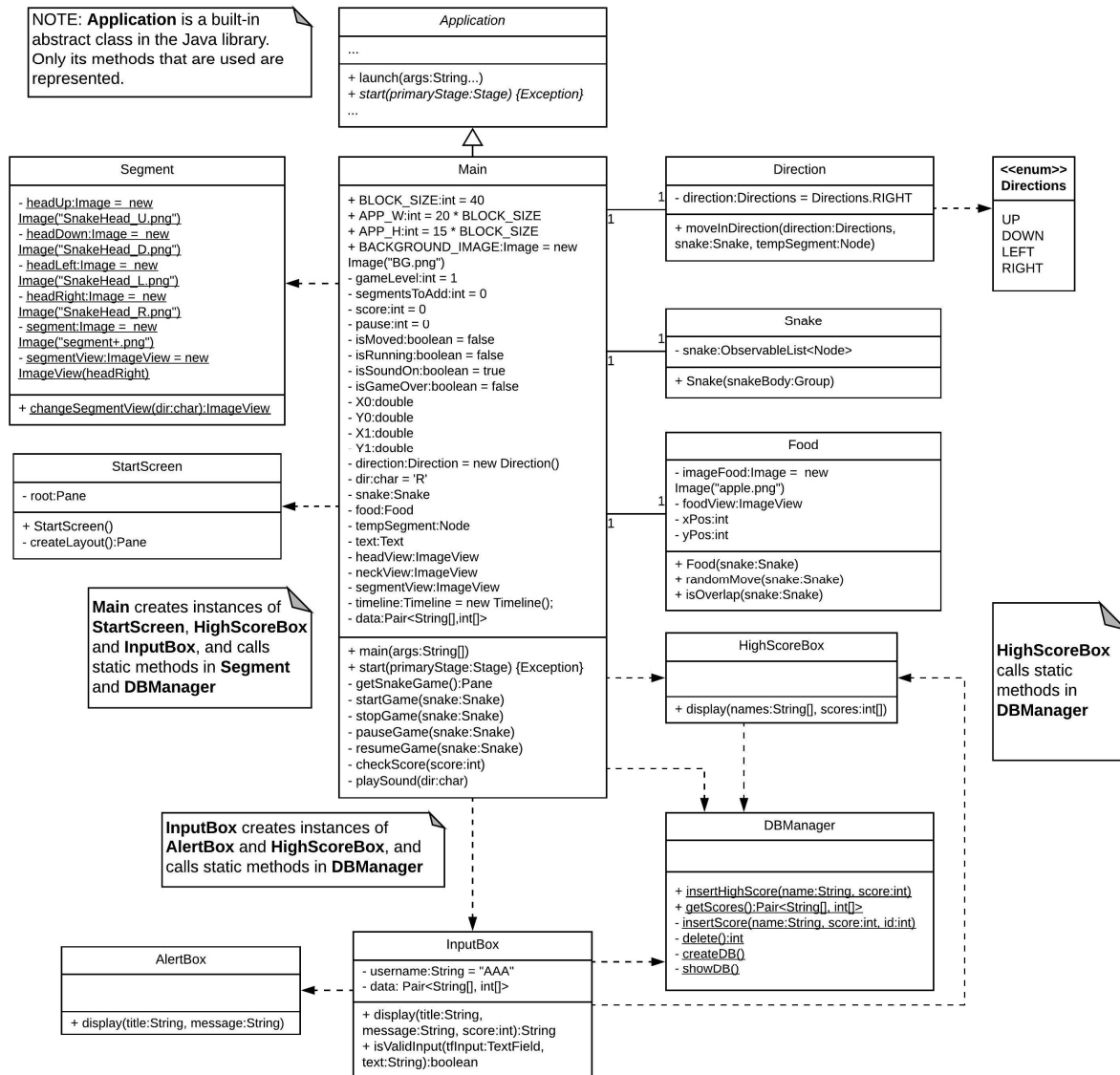The game is intended to be free to play, but hopefully, it will be engaging enough so that a potential player might play long enough to see a couple banner ads, and hopefully click on them.

- People of all ages.

- People who like to play retro video games, such as hipsters and computer science students.

- People who might be looking for a brief distraction, and who don't mind looking a couple ads.

SCOPE OF WORK (SOW)

CLASS DIAGRAM (CLASSES)

CLASS DIAGRAM

NOTE: **Application** is a built-in abstract class in the Java library. Only its methods that are used are represented.

Application

...

+ launch(args:String...)
+ start(primaryStage:Stage) {Exception}
...

Segment

- headUp:Image = new Image("SnakeHead_U.png")
- headDown:Image = new Image("SnakeHead_D.png")
- headLeft:Image = new Image("SnakeHead_L.png")
- headRight:Image = new Image("SnakeHead_R.png")
- segment:Image = new Image("segment+.png")
- segmentView:ImageView = new ImageView(headRight)

+ changeSegmentView(dir:char):ImageView

Main

+ BLOCK_SIZE:int = 40
+ APP_W:int = 20 * BLOCK_SIZE
+ APP_H:int = 15 * BLOCK_SIZE
+ BACKGROUND_IMAGE:Image = new Image("BG.png")
- gameLevel:int = 1
- segmentsToAdd:int = 0
- score:int = 0
- pause:int = 0
- isMoved:boolean = false
- isRunning:boolean = false
- isSoundOn:boolean = true
- isGameOver:boolean = false
- X0:double
- Y0:double
- X1:double
- Y1:double
- direction:Direction = new Direction()
- dir:char = 'R'
- snake:Snake
- food:Food
- tempSegment:Node
- text:Text
- headView:ImageView
- neckView:ImageView
- segmentView:ImageView
- timeline:Timeline = new Timeline();
- data:Pair<String[],int[]>

+ main(args:String[])
+ start(primaryStage:Stage) {Exception}
- getSnakeGame():Pane
- startGame(snake:Snake)
- stopGame(snake:Snake)
- pauseGame(snake:Snake)
- resumeGame(snake:Snake)
- checkScore(score:int)
- playSound(dir:char)

Direction

- direction:Directions = Directions.RIGHT

+ moveInDirection(direction:Directions, snake:Snake, tempSegment:Node)

<<enum>> Directions

UP
DOWN
LEFT
RIGHT

Snake

- snake:ObservableList<Node>

+ Snake(snakeBody:Group)

Food

- imageFood:Image = new Image("apple.png")
- foodView:ImageView
- xPos:int
- yPos:int

+ Food(snake:Snake)
+ randomMove(snake:Snake)
+ isOverlap(snake:Snake)

StartScreen

- root:Pane

+ StartScreen()
- createLayout():Pane

Main creates instances of **StartScreen**, **HighScoreBox** and **InputBox**, and calls static methods in **Segment** and **DBManager**

HighScoreBox

+ display(names:String[], scores:int[])

HighScoreBox calls static methods in **DBManager**

InputBox creates instances of **AlertBox** and **HighScoreBox**, and calls static methods in **DBManager**

DBManager

+ insertHighScore(name:String, score:int)
+ getScores():Pair<String[], int[]>
- insertScore(name:String, score:int, id:int)
- delete():int
- createDB()
- showDB()

AlertBox

+ display(title:String, message:String)

InputBox

- username:String = "AAA"
- data: Pair<String[], int[]>

+ display(title:String, message:String, score:int):String
+ isValidInput(tfInput:TextField, text:String):boolean

Main handles all events and manages the game animation and game state. The Main class stores instances of the Food, Snake and Direction classes that it uses to render and track the positions of sprites. An instance of StartScreen to display the introductory scene in the game and instances of HighScoreBox and InputBox in its checkScore method when the game ends. The InputBox creates an

instance of AlertBox during input validation. Main, InputBox, and HighScoreBox require data from the

scores database and access it through the methods of the DBManager class.
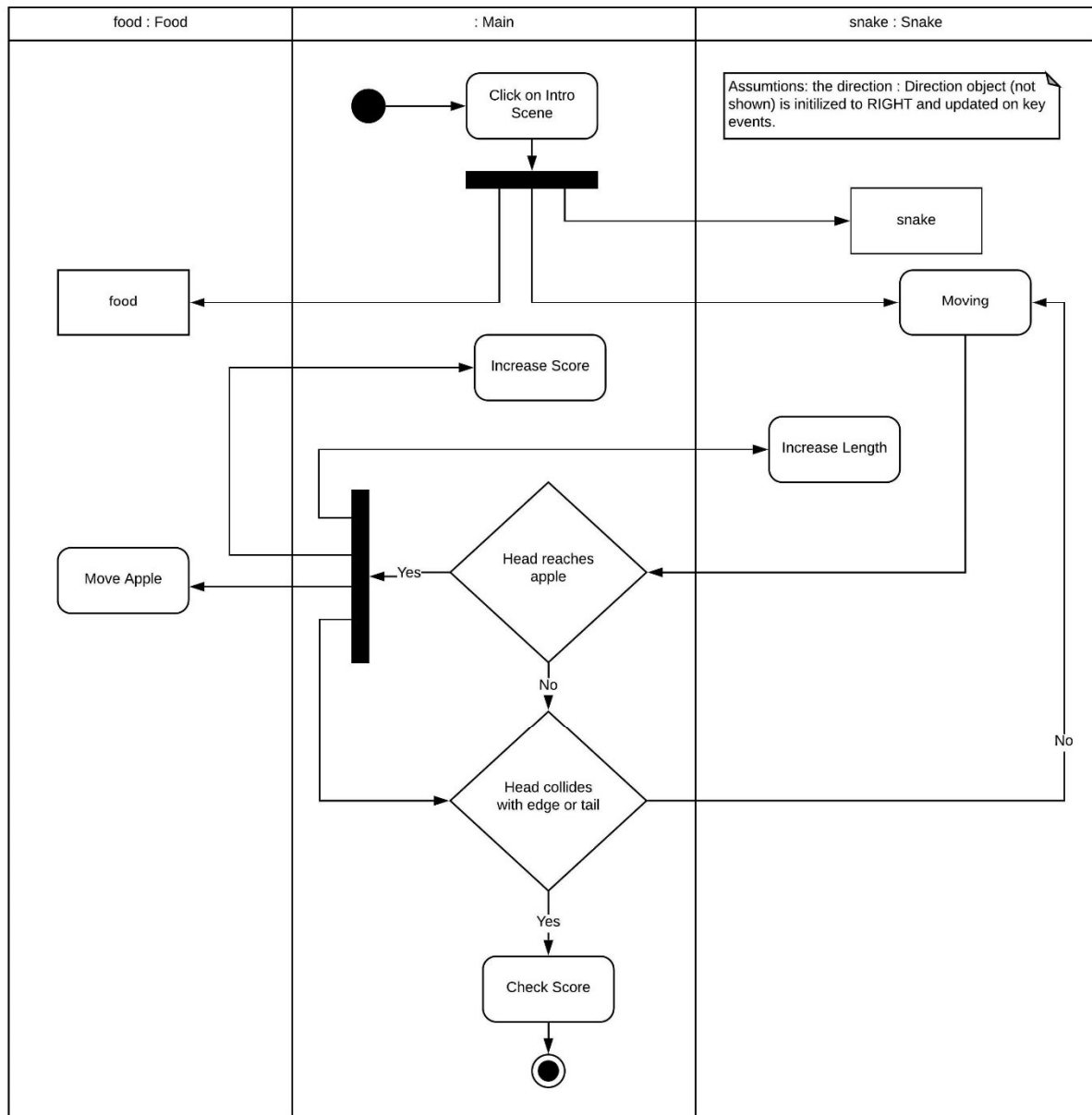
SEQUENCE DIAGRAM

GAME LOOP SEQUENCE
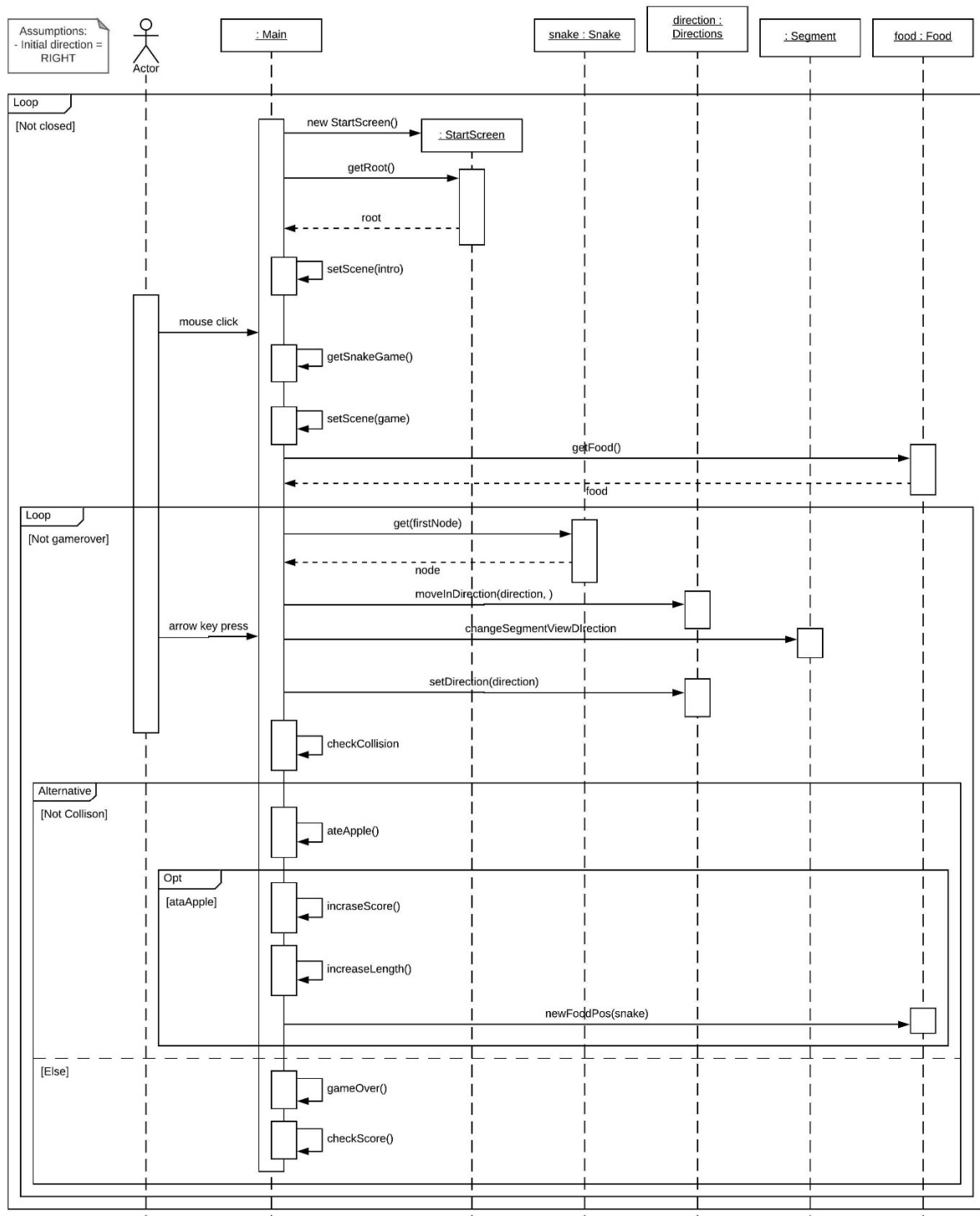
The program is essentially two event driven loops. The fist loop consists of sequence of events once the game is opened and until the user closes the window. The introductory screen is shown, when player clicks the scene it switches to the game Scene, and when the game ends the Scene returns to the introductory screen.

The game is the sequence of events in which the player interacts with the program through the snake object and the arrow keys (also spacebar for pause, not shown). The snake changes position on the screen according to the players selected direction and grows when the snake's head reaches the food object's position. When the snake's head and food collide the food changes position. The game ends if the snake's head collides with an edge of the screen or one of its body members.

ACTIVITY DIAGRAM

PLAY LOGIC ACTIVITY

food : Food	: Main	snake : Snake

Click on Intro Scene

Assumtions: the direction : Direction object (not shown) is initilized to RIGHT and updated on key events.

snake

food

Increase Score

Moving

Increase Length

Move Apple

Yes — Head reaches apple

No

Head collides with edge or tail

No

Yes

Check Score

The program shows the introductory Scene initially and starts the game once the player clicks on the screen. The Main class instantiates the food and snake game objects and the snake moves on the

screen. The player controls the movement (not shown). If the head reaches the food the score and length of the snake is increased. If the head collides with an edge of the window or one of its body members, the game ends and the score is checked against the high scores database (not shown).

SCORES ACTIVITY
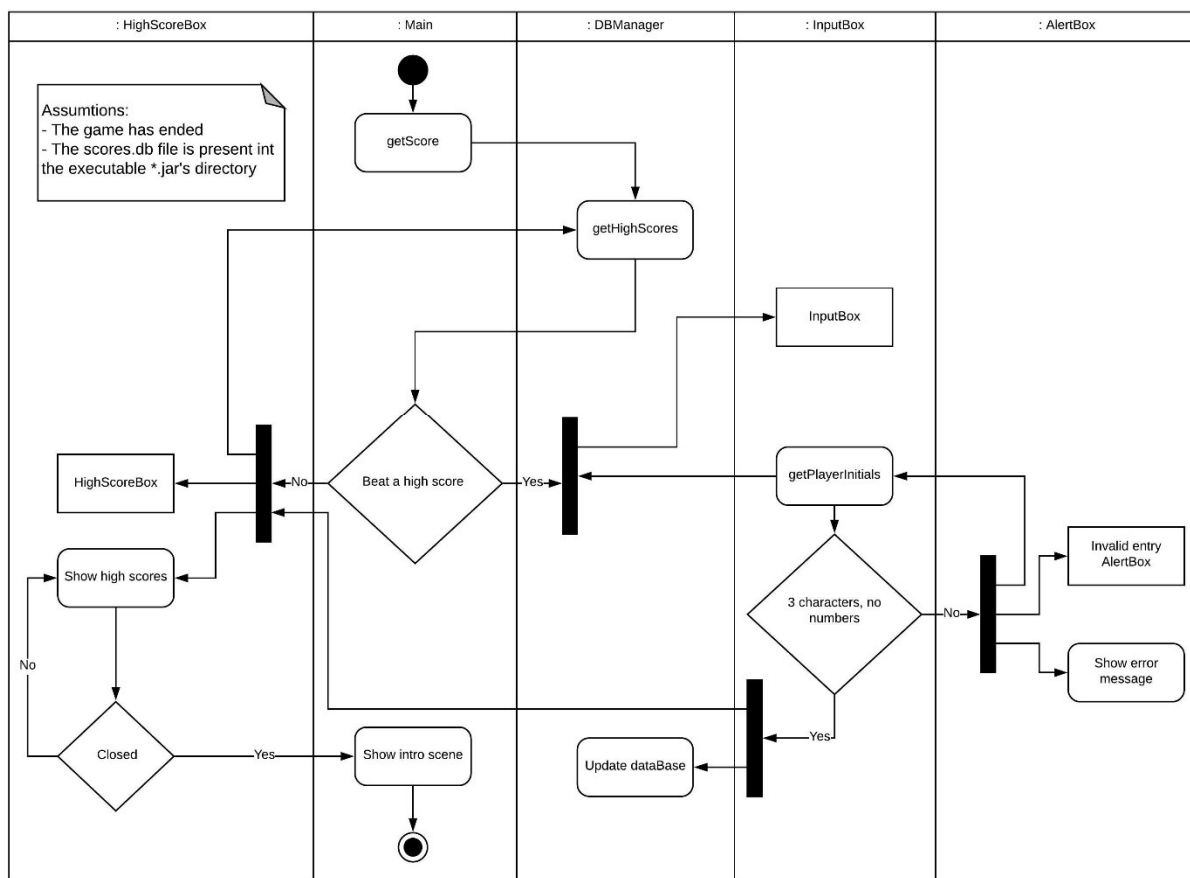
When the game ends, the high scores are retrieved from the database file using the DBManager class and the player's score is checked against the high scores. If the player has not beat a high score, the high scores are displayed in an instance of the HighScoreBox class. If the player beat a high score an instance of InputBox is created to get the player's initials. If the player enters characters other than letters or less than 3 characters an error message is displayed in an instance of the AlertBox class. Once the player has successfully entered their name the database is updated using the DBManager class and an instance of HighScoreBox is create and the new high scores are shown. When the HighScoreBox is closed, the introductory scene is shown.

STATECHART DIAGRAM

GAME CONTROL
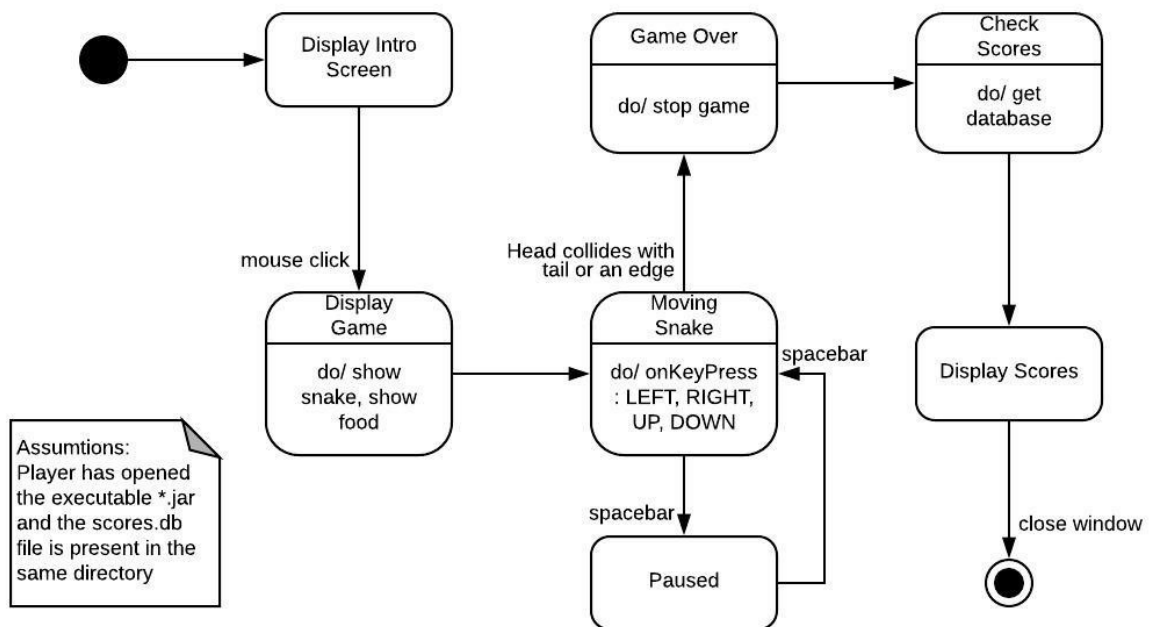
When the game window opens the introductory scene is displayed. When the player clicks on the scene, the scene switches to the game scene. The food and snake objects are shown, and the snake is moving. When the player presses spacebar the game is paused, if the game is already paused, the game is resumed. The game ends if the snake collides with its tail or an edge of the window. The player's score is check and the high scores are shown.

SNAKE MOVEMENT STATE
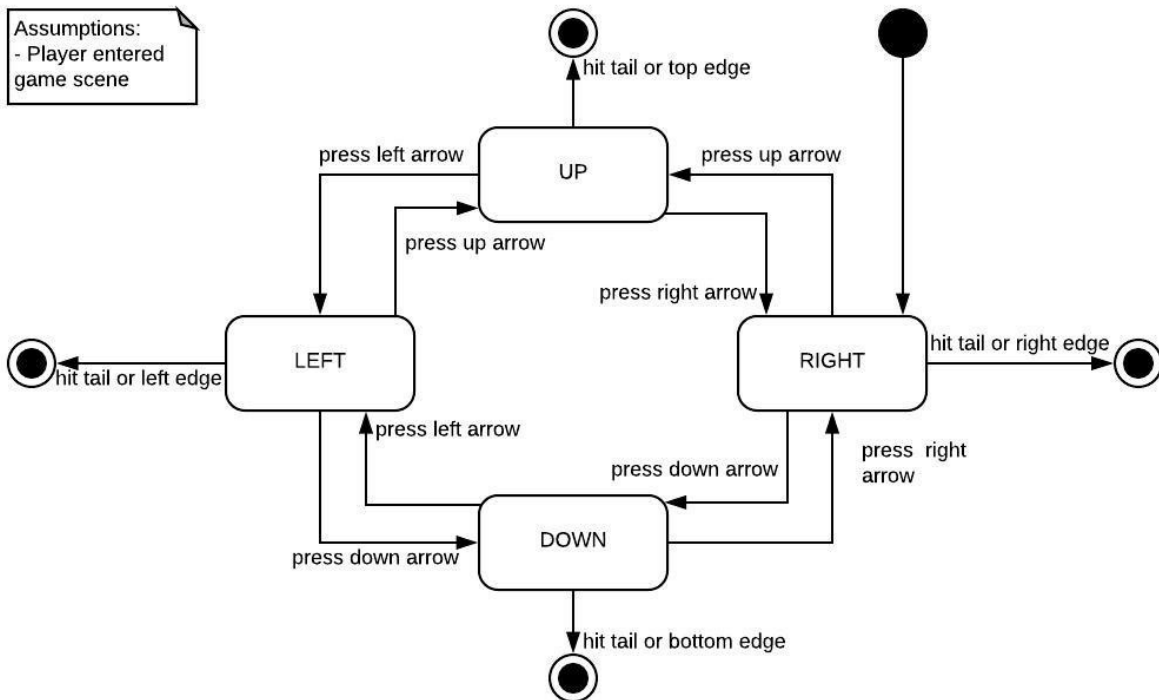
Since the game ends when the snake collides with one of its body members, the player's choice of direction is limited to two choices at any point in the game. As an analogy, if we consider that the snake is always moving forward from the snake's perspective, it can only choose to go left or right. That is the snake cannot move backwards. The analogy does not correspond to the arrow keys that player uses to control the game, it is only an explanation of the control logic.

PROJECT RETROSPECTIVE

What was the easiest to implement?

The current version of the Snake game is based primarily on Svyatoslav's efforts. We had made a graphical version of the game earlier, however, shortly after its development, we decided to go with Svyatoslav's idea as the input handling was substantially better, that is, more-user friendly. The easiest part of the earlier game to develop was when Fraser once had his head around the Canvas object, which was the Food class. It is a very simple sprite that finds a random position and stores an image. As far as the final version, the simplest class that we contributed was the "AlertBox", which is used for the input validation.

What was the most difficult?

Building the "DBManager" class was challenging. However, we were able to find a helpful tutorial and apply what was necessary to implement and manage the high score database for the game. The earlier version of the game had unresolved control issues where the user could "turn too quickly" by pressing three of the arrows rapidly causing the snake to fold back on itself in one game position. Also, the snake movement was slow to respond, that is, the player would have to choose the next direction slightly before the snake entered a game position.

What can be done differently?

Time was the biggest challenge in the project. We started early, but the development of the game took longer than expected. In retrospect, setting more realistic goals at the inception of the project would have been sensible; however, we are all pleased with the resulting game.
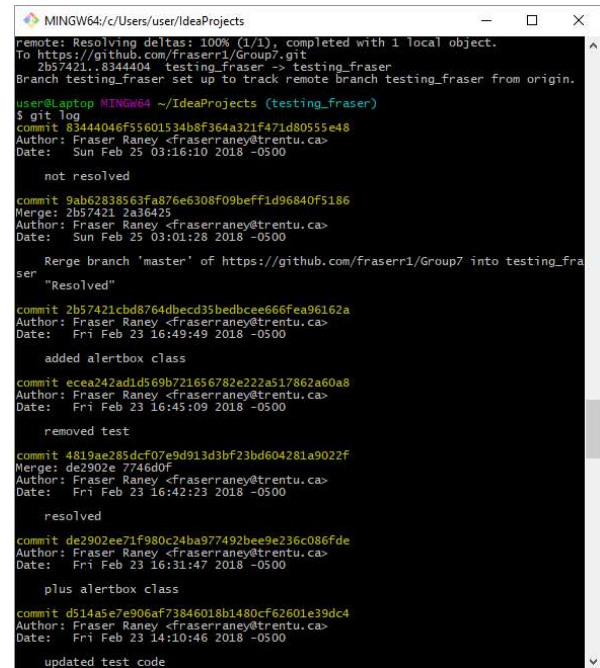
What have we learned?

JavaFX and JDBC represented huge learning curves, but the knowledge that we gained from the project is invaluable. Using GitHub to coordinate the project development was again something that was completely new to us and the experience of developing the game as part of a team was a great experience.

APPENDIX

SOFTWARE DEVELOPMENT JOURNAL

Feb 24, 2018 - Issues with GitHub:

- Workspace was in .gitignore to switch to master so I could create a new remote outside of the group 7 repo for assignment 2 (COIS 2240)

- Several commits were lost when I restored the test project to a working commit, including ConfirmBox and AlertBox classes

- In the future make sure to commit the entire directory to ensure that no files such as workspace.xml are ignored

Feb 25, 2018

- Added a bordered layout to the test project ☺ first glimpse of the GUI? Hopefully.
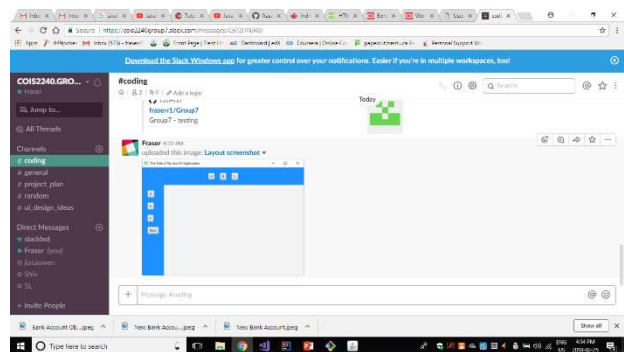
Feb 27 - Mar 1, 2018

- Created an input window class (part of the final project)

Mar 3, 2018

- Svyat begins working on a graphical Snake game

"Hey, Gents. I'm going to upload a working code for a snake game to GitHub tonight. It is very primitive and needs to be rewritten to meet our needs, but I believe it is a good foundation. It is a code by Almas Baimagambetov - https://www.youtube.com/watch?v=nK6I1uVlunc. This project is quite challenging since some of us basically must learn Java and JavaFX from the scratch, but it's fun. I am trying to write my own code and my snake moves much smoother but I'm using personal timeline for each snake segment and since this function is asynchronous it is extremely difficult to control it without multithreading, and I don't know how to do it. It's like learning a new natural language and you know what you want to say but you don't have enough vocabulary to do it. Anyways, look at the code and let's try to improve it."

Mar 3, 2018

- After looking at the reference Svyat mentioned, I begin looking into implementing the graphical Snake game in an alternative way since the logic of moving the tail to the head seemed counter-intuitive for sprites.

Mar 11, 2018

- I made a breakthrough, using PowerPoint to create transparent backgrounds for my game images and implementing them using JavaFX's Canvas object.

Mar 13, 2018

- I have been looking into animating the snake, I think I have an algorithm to keep track of the head separately from the segments, so far, I just have a moving snake head and a randomly appearing apple that disappears when they meet.
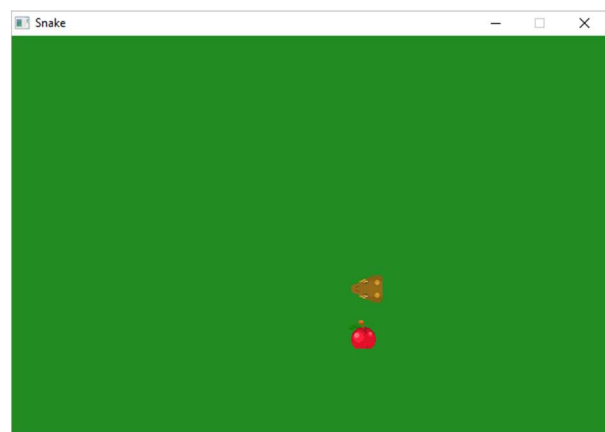
March 15, 2018

- Basic Snake game.

Mar 18, 2018

- Snake game separated into logical classes.

SNAKE GAME

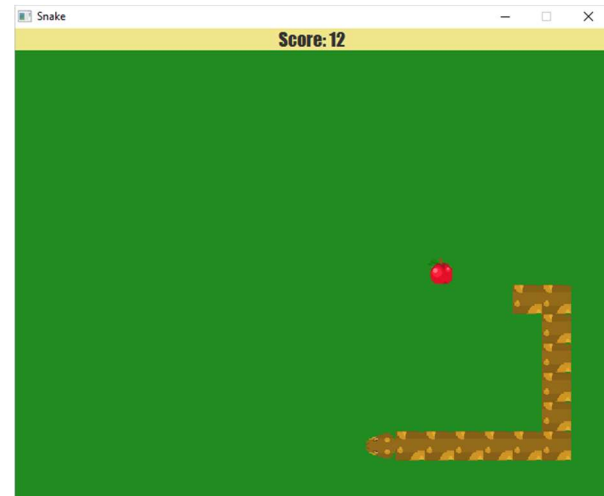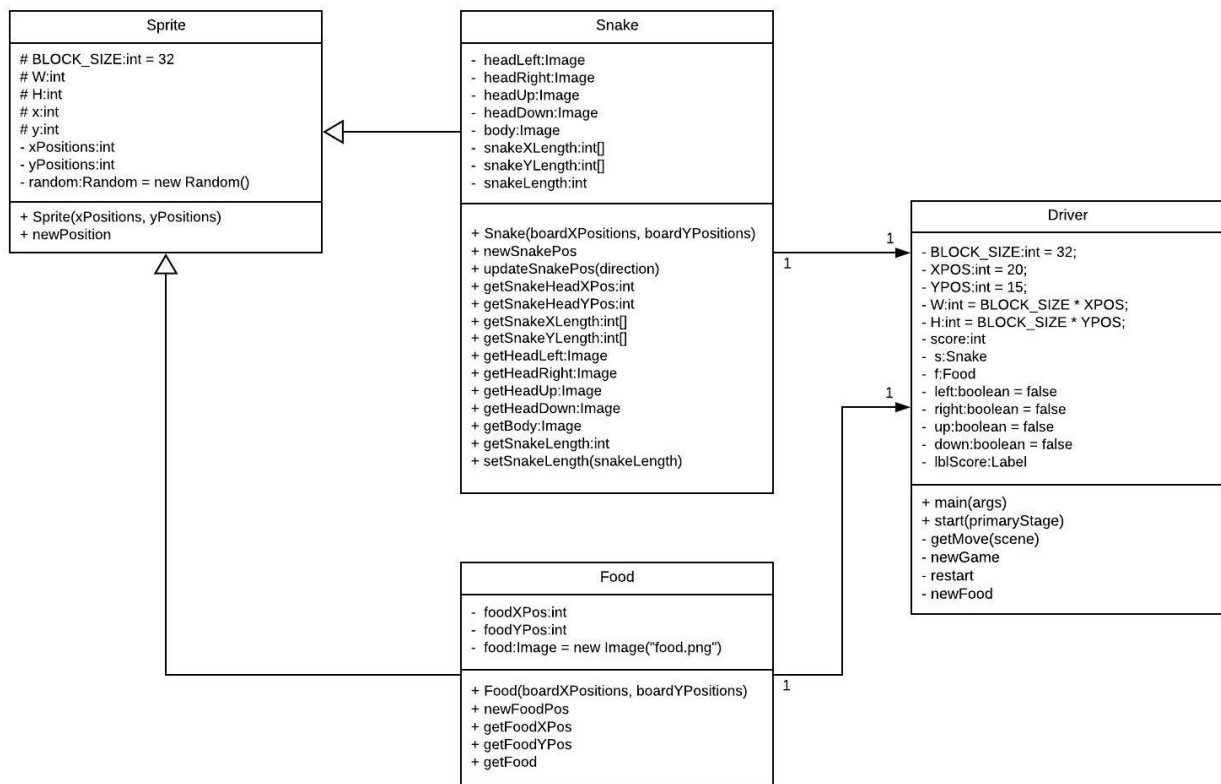| Sprite |
| --- |
| # BLOCK_SIZE:int = 32 |
| # W:int |
| # H:int |
| # x:int |
| # y:int |
| - xPositions:int |
| - yPositions:int |
| - random:Random = new Random() |
| |
| + Sprite(xPositions, yPositions) |
| + newPosition |

Snake
- headLeft:Image
- headRight:Image
- headUp:Image
- headDown:Image
- body:Image
- snakeXLength:int[]
- snakeYLength:int[]
- snakeLength:int
+ Snake(boardXPositions, boardYPositions)
+ newSnakePos
+ updateSnakePos(direction)
+ getSnakeHeadXpos:int
+ getSnakeHeadYPos:int
+ getSnakeXLength:int[]
+ getSnakeYLength:int[]
+ getHeadLeft:Image
+ getHeadRight:Image
+ getHeadUp:Image
+ getHeadDown:Image
+ getBody:Image
+ getSnakeLength:int
+ setSnakeLength(snakeLength)

Driver
- BLOCK_SIZE:int = 32;
- XPOS:int = 20;
- YPOS:int = 15;
- W:int = BLOCK_SIZE * XPOS;
- H:int = BLOCK_SIZE * YPOS;
- score:int
- s:Snake
- f:Food
- left:boolean = false
- right:boolean = false
- up:boolean = false
- down:boolean = false
- lblScore:Label
+ main(args)
+ start(primaryStage)
- getMove(scene)
- newGame
- restart
- newFood

Food
- foodXPos:int
- foodYPos:int
- food:Image = new Image("food.png")
+ Food(boardXPositions, boardYPositions)
+ newFoodPos
+ getFoodXPos
+ getFoodYPos
+ getFood

1
1
1
1

Mar 19, 2018

- Jiatao catches a couple small bugs that are easily resolved.

Mar 20, 2018

- Svyat completes the basic Snake game that will eventually be the final project.
"I added different classes, background image and sounds effects that play on click. Everything is very

primitive, but this is my learning process."

After discussing on the phone, the differences between the two versions Svyat and I decide to go ahead

with his code since the control input handling is much better.

Mar 24, 2018

- First version of the DBManager class and its related input classes for tracking and displaying the high scores.

Mar 29, 2018

- I separated the start screen into its own class and added comments to the code.

Mar 29, 2018

- Shivam starts working on the report.

April 4, 2018

- Shivam makes minor changes to the readme file.
- Shivam adds final touches and completes the final report.