EECS 2030 Final Project

Snake Game

Haider Al-Tahan  
213788708

Jianxiong Wang

Ammiel Cruz

**Project Description:**

This project is a custom swing based snake game. The objective of the game is for the player to control a snake that enjoys eating food. There are three types of food in this game, Golden Apple, Apple, and Poisoned Apple. The Golden Apple increases the score of the player by 50, the Apple increases the size of the snake and score of the player by 1-3. The Poisoned Apple on the other hand, will end the game. If the player hits any of the four boundaries, the game is over. Additionally, if the size of the snake reaches the maximum (the size of the game screen) then it would be considered a win and the player would be moved to the next level where the snake is faster and the game generally becomes more challenging. However, the player would gain double the score!

The game store and sync the score and name of the player upon losing the game (when the game over screen is shown) to a NoSQL Database. Data is synced across all clients in realtime, and remains available when the application goes offline. This was made possible by using a google service called Firebase Realtime Database. The Firebase Realtime Database is a cloud-hosted database. Data is stored as JSON and synchronized in realtime to every connected client. Additionally, all of your clients share one Realtime Database instance and automatically receive updates with the newest data.

This assignment applies multiple advanced topics in object-oriented programming. Starting with the most general such as Javadoc, Encapsulation / Information Hiding, Mutable and Immutable Classes, Functional Programming, and Polymorphism. As group, the most challenging feature was Networking and Sockets.

**Features and implementations:**

Encapsulation/Information Hiding

Information Hiding and Encapsulation are complementary concepts. Information Hiding is the practice of separating how to use a class from the details of its implementation, ie. A mechanism for restricting access to some of the object’s components. On the other hand, Encapsulation means that the data and methods of a class are combined into a single unit, which hides the implementation details (JAVA provides encapsulation using class). In this project, Encapsulation was used by facilitating a bundle of data with the methods operating on data by using Classes. Additionally, Information Hiding was used for restricting access to some of the object’s components by using the private modifier. For example, the Tile class that bundle data responsible for coordination and holds private x and y variables such that it restricts access.

Overloading/Constructors

Constructors in a class are invoked to initialize the object. Overloading is the ability to create multiple methods/constructors of the same name with different implementations. In this project, overloading and constructors were used both at the same time to create multiple constructors for different uses. For example, the Tile uses these concepts to make a constructor and a copy constructor, as such:

*/\*\*  
 \* Constructor  
 \*  
 \** ***@param*** *x x coordinate  
 \** ***@param*** *y y coordinate  
 \*/*public Tile(int x, int y) {  
 this.x = x;  
 this.y = y;  
}  
  
 */\*\*  
 \* Copy Constructor  
 \*  
 \** ***@param*** *t another Tile  
 \*/*public Tile(Tile t) {  
 this.x = t.x;  
 this.y = t.y;  
 }

Static Methods and Variables

Static methods and variables belongs to the class and not to the object (instance) and can access only static data, it cannot access non-static data (instance variables). In this project, the static modifier is used mainly in the Constants class. Therefore, there is no need to initiate the Constants class to use the variables.

Mutable and Immutable Classes

Inner Classes

Interfaces / Abstract Classes

Inheritance

Polymorphism

Generics

Swing / GUI / Event-Driven Programming

Array and ArrayList / Collections

Exceptions and File I/O

Functional Programming

Multithreading

**Junit:**

**Simple UML Diagrams:**