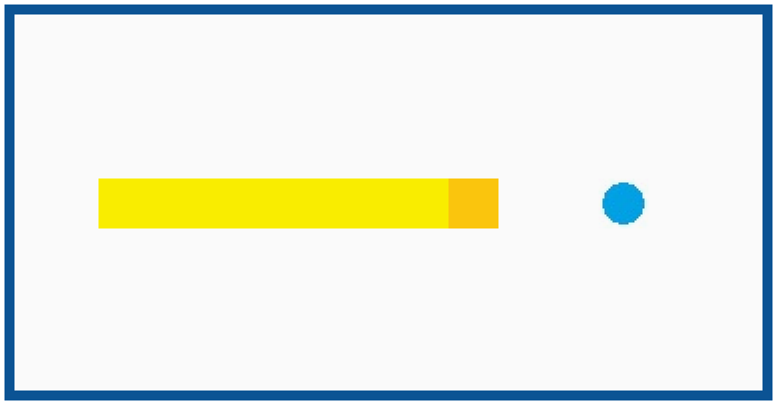


Constricted Constrictor

Design Document

Team 26

Matthew Der

Chirag Nath

Kevin Shi

Ran Guo

Index

- Purpose
    - Functional Requirement
    - Non-Functional Requirement
- Design Outline
    - Components
    - High-Level Overview
- Design Issues
    - Functional Issues
    - Non-Functional Issues
- Design Details
    - Class Diagrams
    - Class Descriptions
    - Sequence Diagrams
    - Database Design

Purpose

Snake games are one of the most classic game designs: since the original Snake's debut in 1997 on the Nokia 6110, many new versions of the game have been designed and that game is still popular and it is still a good way to relax and kill time. However, most of them are very basic and do not have many features, which make them increasingly monotonous the more one plays; players often experience visual fatigue after staring at the predictable movement of the snake for a long time.

The purpose of our project is to develop a new Snake Game application which will provide a lot of new functionalities in addition to the basic gameplay of existing snake games. With our Snake game, players will be able to enjoy a modern take on the design through the inclusion of powerups, multiple levels, and different movement patterns, while still keeping the core concepts and gameplay.

Functional Requirements

1. User account:

   As a player,

   a. I would like to register for an account.
   b. I would like to be able to login with my account.
   c. I would like to be able to change my login credentials.
   d. I would like to have my credentials be encrypted (or stored into a database).

2. Basic Functionalities

   As a player,

   a. I would like to start the game by pressing the start button.
   b. I would like to bring up a ‘how-to-play’ menu/screen when a button is interacted with.
   c. I would like to be able to pause the game and bring up a pause screen by hitting esc.
   d. I would like to be able to resume the game from the pause screen.
   e. I would like to be able to quit and return to the home screen from the pause screen.
   f. I would like to be able to choose between playing again or returning to the home screen when I lose.
   g. I would like to be able to click on a “special thanks” button to see the developers of this game.

h. I would like to be able to exit out of the game by clicking the close button on the right corner of the game interface.
i. I would like to be able to minimize the game to a small window by clicking the minimize button.

3. BGM and Sound Effects

   As a player,

   a. I would like to be able to listen to background music while playing.
   b. I would like for the background music to decrease in volume when the pause screen is up.
   c. I would like to be able to toggle background music.
   d. I would like to have sound effects for gameplay interactions.
   e. I would like to be able to toggle sound effects.

4. UI-Related Functionalities

   As a player,

   a. I would like to be able to see the background color change when game is ongoing/paused

5. Gameplay

   As a player,

   a. I would like to be able to move the snake in the four basic directions (up, down, left, right).
   b. I would like to receive points from eating beans.
   c. I would like to be able to play a 1v1 versus mode with a partner.

   d. I would like to be able to accumulate energy when I play.
   e. I would like to activate a special skill when my energy is full.
   f. I would like to be able to have different obstacles (like walls) to increase the unpredictability of the game.
   g. I would like to be able to have a timed big apple shown up randomly after a bean has been eaten.
   h. I would like to be able to activate a big apple powerup to double my score for a short period of time.
   i. I would like to be able to activate power ups that either increase or decrease the speed of the game.
   j. I would like to be able to activate power ups that affect the tangibility of the snake.

6. Automatic PathFinding

   As a player,

   a. I would like to get power-up after I eat a special food. Pathfinding will be triggered and the AI/Pathfinding algorithms will take over and it will play itself for a while
   b. I would like to enter the AI mode to see how the snake looks for the bean on its own.

7. Score Rating

   As a player,

   a. I would like to save the 10 highest scores.

   b. I would like to save the 10 longest surviving times.

8. Customization/Settings

   As a player,

   a. I would like to be able to change the difficulty level of the game.
   b. I would like to change the map for the game.
   c. I would like to be able to customize the look of the snake.
   d. I would like to be able to customize the keys to control the snake movement to keys of my preference.

Non-Functional Requirements

1. UI/Appearance

   As a developer,

   a. I would like to make the user interface pleasing, understandable and easy to navigate.

   b. I would like to use Java Swing and some other Java graphic libraries to build the UI.

2. Security

   As a developer,

   a. I would like to use a database instead of a file to store the user information.

   b. I would like to choose MySQL as the database.

   c. I would like to use JDBC and Druid datasource to access the database.

3. Gameplay

   As a developer,

   a. I would like to ensure the quality of the gameplay.

   b. I would like to make sure that the gameplay should be both smooth and responsive.

4. Usability

   As a user,

   a. I would like this application to be user-friendly and easy to use.

   b. I would like this application to not crash very often.

5. Design/Architecture

   As a developer,

   a. I would like this project to have a good architecture.
   b. I would like this project to have a MVC architecture.
   c. I would like this project to be well-designed and well-structured so that it can be maintained and improved easily.

Design Outline

Components

Our project is an advanced Snake game, and our implementation will be based on a typical MVC design with a new Database component. Player inputs will be sent to the controller which will update the model and then push those changes back to the view.

1. Model
    a. The model is where all of the background information will be stored.
    b. The model will keep track of locations on the map such as the snake's head and body, the location of the next fruit, and any power-ups or obstacles.
    c. The model will also keep track of values such as the snake's current direction, the current score, active power ups, menu status, etc.
    d. The model will contain some entities like the Snake class and the User class used to store user information.

2. View
    a. The view is where the player will interact with the game.
    b. The view will show the desired menu and UI after the player does a certain interaction.

   c. The view will update the frame according to the information in the model layer

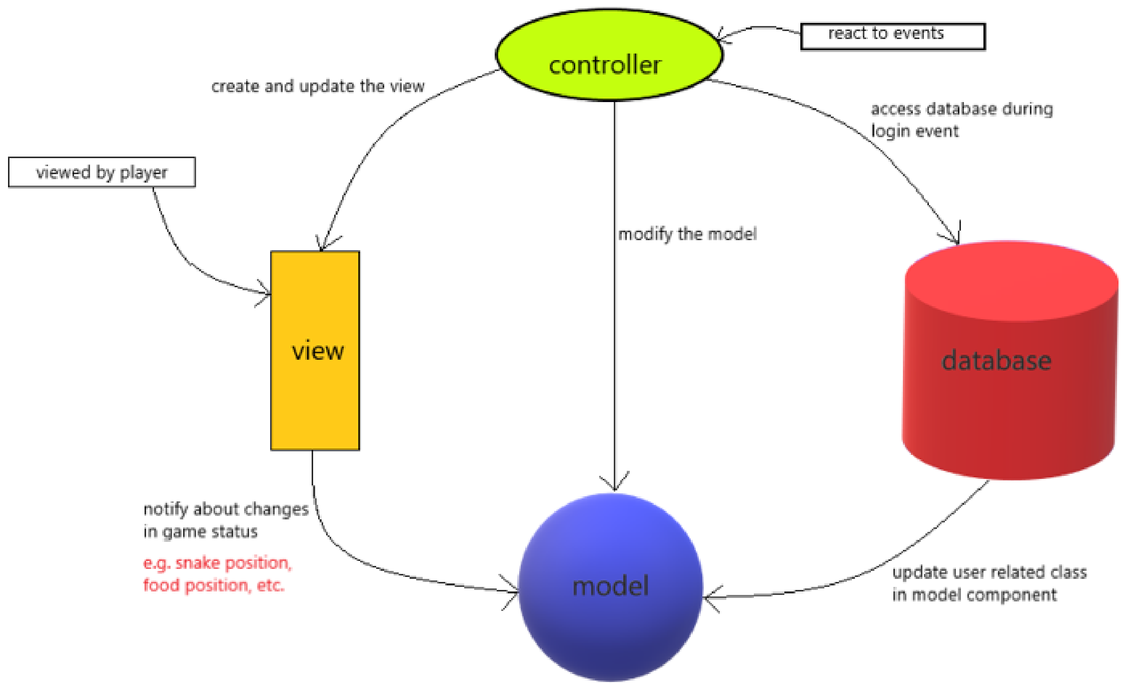
3. Controller
   a. The controller layer will contain actions associated with different buttons/interactions from the view.
   b. It will contain objects that control and handle user's interaction with the view and model.
   c. It will contain the object used to access the database layer. It will be similar to the service layer in a 3-tier architecture of a web app that accesses the database access object to access user information from the database.

4. Database
   a. This layer will be used to directly access the database.
   b. It will contain a data access object class which uses JDBC and Druid datasource to fetch information from the database.

High-Level Overview

Our project will be based on a Model-View-Controller Model. Since it will support a multi user system, there will also be another component which is the database part.

Diagram of interaction between components:


controller
react to events
create and update the view
access database during login event
viewed by player
modify the model
view
database
notify about changes in game status
e.g. snake position, food position, etc.
model
update user related class in model component

Design Issues

Functional Issues:

1. How should we store user information to implement the multi user system?
    - Option 1: In text files.
    - Option 2: In a database.

   Choice: Option 2

   Justification: While implementing a user account system, it is important to decide where to store the users' information. Storing user information in text files will be very insecure that all users can see other users' information. Using a database will be a good choice. By doing that, all the user information, database accesses and authentication steps will be invisible to all users. This will be a secure way to implement that function.

2. What Information do we need when the user creates an account?
    - Option 1: Username and password only.
    - Option 2: Username, password and email address/phone number.

   Choice: Option 2

   Justification: For the safety of all user information, user information will be stored in the database. In addition to username and password which are necessary for identifying a specific user, we will also ask for the user's email so that we can send a confirmation email to confirm that the owner is not a bot and the user is the owner of that email address.

3. How should we store each user's rating of the highest scores they get?
    - Option 1: A text file containing high scores of all users and read the text into memory.
    - Option 2: Stored in another table in the database.

    Choice: Option 2

    Justification: Since the user information is stored in a table in the database. It would be appropriate to store the scores ratings for each user in another table in the database using foreign key to indicate which user they belong to.

4. How should we implement the automatic pathfinding of the snake?
    - Option 1: Use an algorithm based on the BFS search to find the shortest path to the next food generated.
    - Option 2: Use a more advanced algorithm or many algorithms combined to find the appropriate path to the next food with some AI features implemented.
    - Option 3: Use an AI-based trained model to perform the pathfinding task.

    Choice: Option 2

Justification: If the pathfinding system is implemented using the BFS algorithm, error will always occur at some point since that algorithm will always find the shortest path while a real player will not go straight to the food. However, that problem could be solved if we improve the algorithm or implement that functionality using an AI-based trained model to perform the task. Since starting to learn machine learning from zero would be a little bit hard, we're gonna implement it with advanced algorithms while including some features of AI.

5. How should we implement the 1 vs. 1 multiplayer system?
    - Option 1: Use concurrency to create two game panels that will run in parallel so that each player can play on their panel.
    - Option 2: Just create two panels and let the two panels update together in one thread.

Choice: Option 2

Justification: Instead of using concurrency, we think that this can be done by just using one thread and updating the screen at exactly the same rate. It will also listen to events from both panels to update the frame.

Non-Functional Issues:

1. What language is appropriate for implementing our application?
    - Option 1: C/C++
    - Option 2: C# with Unity as the game engine
    - Option 3: Java
    - Option 4: Python

    Choice: Option 3

    Justification: We finally decided that we should use java to implement our project. Although python may be more suitable since pygame will be a very good package for implementing games and python is also good at implementing machine learning algorithms, we still want to use Java since that is the language most of us are familiar with and we also had some experiences making user interfaces with Java Swing library.

2. What type of database is appropriate for our user data?
    - Option 1: Relational databases such as Oracle or MySQL
    - Option 2: NoSQL databases such as Redis

    Choice: Option 1

    Justification: A relational database will be suitable for our project and our user information will be stored in tables in that database. We can also use JDBC to easily interact with MySQL. We chose MySQL since it is appropriate for lightweight projects and the amount of data will not be large.

3. What type of software design architecture is good for us to develop the game?

    Option 1: Model-View-Controller (MVC) architecture

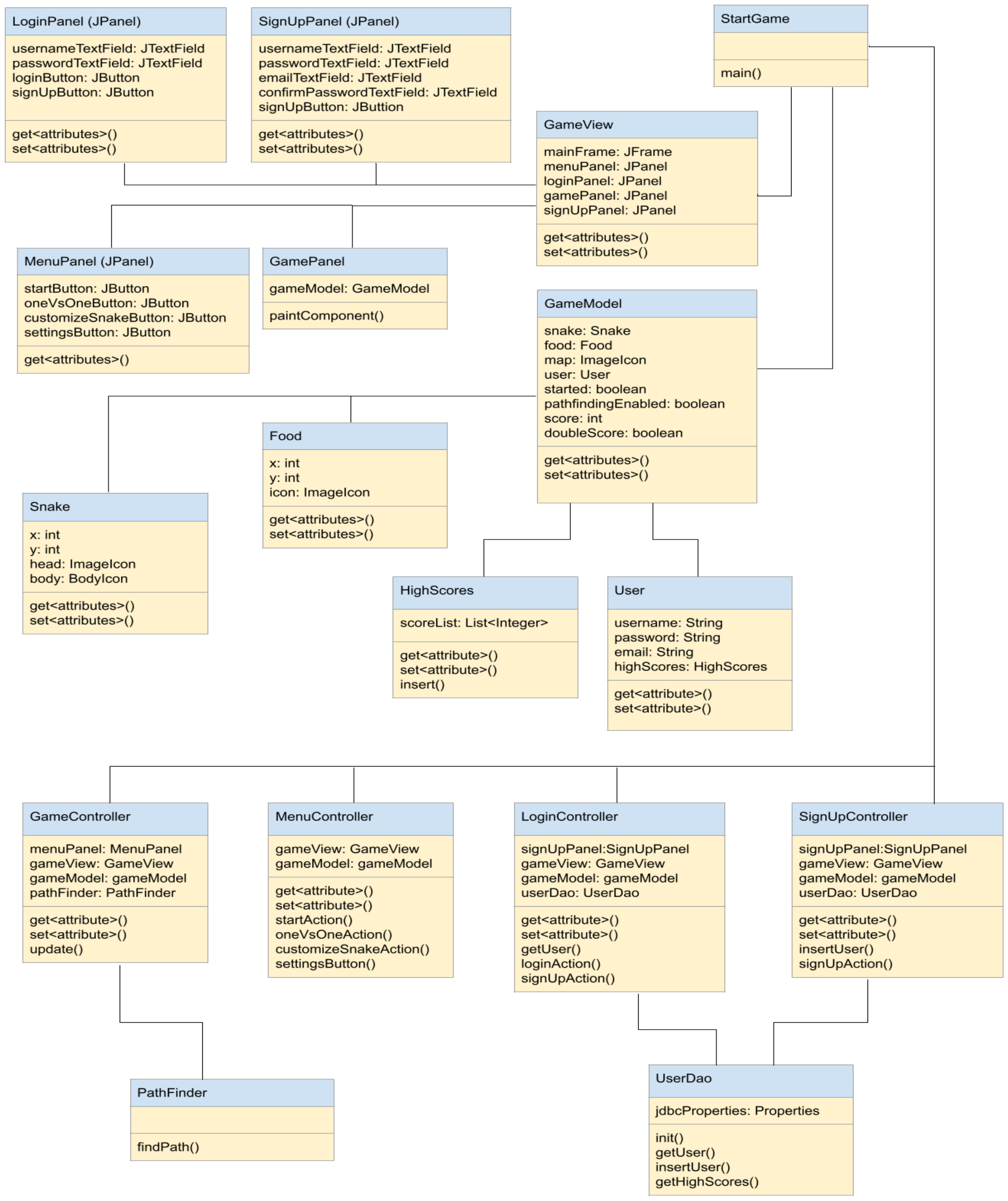
    Option 2: Client-Server architecture

   Choice: Option 1

   Justification: Client-Server model is good for developing web apps, but for our project MVC model will definitely be better because we want it to be highly scalable. We wouldn't like the game to decrease in its performance when there are changes in our application and system processing demands. (Essentially the case when there is an increased number of online gamers) Also by utilizing MVC, different components in our game will have a better interconnectivity.

Design Details

Class Design Diagram



Description of class and the relationship between classes:

Main Class:

StartGame:

- The entry of the whole application. It serves as the start point for the game.
- It will initialize each part of the application.
- It will show the frame in the view objects that is initialized.

View:

GameView:

- This is the basic class for the View component of the application.
- It contains all the JPanels that we use in this project.

LoginPanel:

- This class is the panel used when the application asks the user to login.
- This class contains necessary text fields and buttons for a user to login.
- It will contain a “sign up” button for the user to create an account.

SignUpPanel:

- This class is the panel used when the user wants to create a new account.
- This class contains necessary text fields and buttons for a user to sign up for an account.

MenuPanel:

- This class will be the main menu the player will see after they log in.

- This class contains necessary buttons for implementation of required functionalities.

GamePanel:

- This class will be used as the panel that is shown to the player after the game starts.
- This class will have a paintComponent() method that will be called in every timer event.
- This class will contain the GameModel object to paint the snake and the food.

Model:

GameModel:

- This is the basic class for the model component of the application.
- This class contains necessary information for the controller and view to update accordingly to the current state of the game.
- It contains a snake and food object.
- It contains the current user information.
- It contains many boolean variables indicating the game state.

Snake:

- This class describes the current state of the snake.
- It uses an array to store the x and y coordinates of each piece of the snake.

- It contains two ImageIcon objects which are the images for the snake's head and body.

Food:

- This class describes the current state of the food.
- It contains the x and y coordinates of the food generated.

User:

- This class will contain information about the current user that is using this application.
- It will contain the user's username, password and email address.
- It will contain a HighScores object which stores the user's rating of highest scores he has got.

HighScores:

- This class will contain a sorted list of scores showing the high scores a user has got.
- It has an add() method which will add the new score into the list and sort the list again.

Controller:

LoginController:

- This is the controller class designed for the login panel.
- This class will add action listeners to the buttons and textfields in the login panel.

- This class will contain a UserDao object to access the database to check if the user exists.

SignUpController:

- This is the controller class designed for the sign up panel.
- This class will add action listeners to the buttons and textfields in the sign up panel.
- This class will contain a UserDao object to access the database to insert the new account that is created.

MenuController:

- This is the controller class designed for the menu panel.
- This class will add action listeners to the buttons and textfields in the sign up panel.

GameController:

- This is the controller class designed for the game panel.
- This is the core controller class of this application since it is for the game part.
- This class will set up the timer for a specific rate.
- This class will set a keyListener for the panel.
- It defines how the model will be updated each time a timer event occurs according to the game model object.
- It contains a pathfinder object to implement the pathfinding functionality.

PathFinder:

- It defines the method we will use to find the appropriate path from the snake to the new generated food.

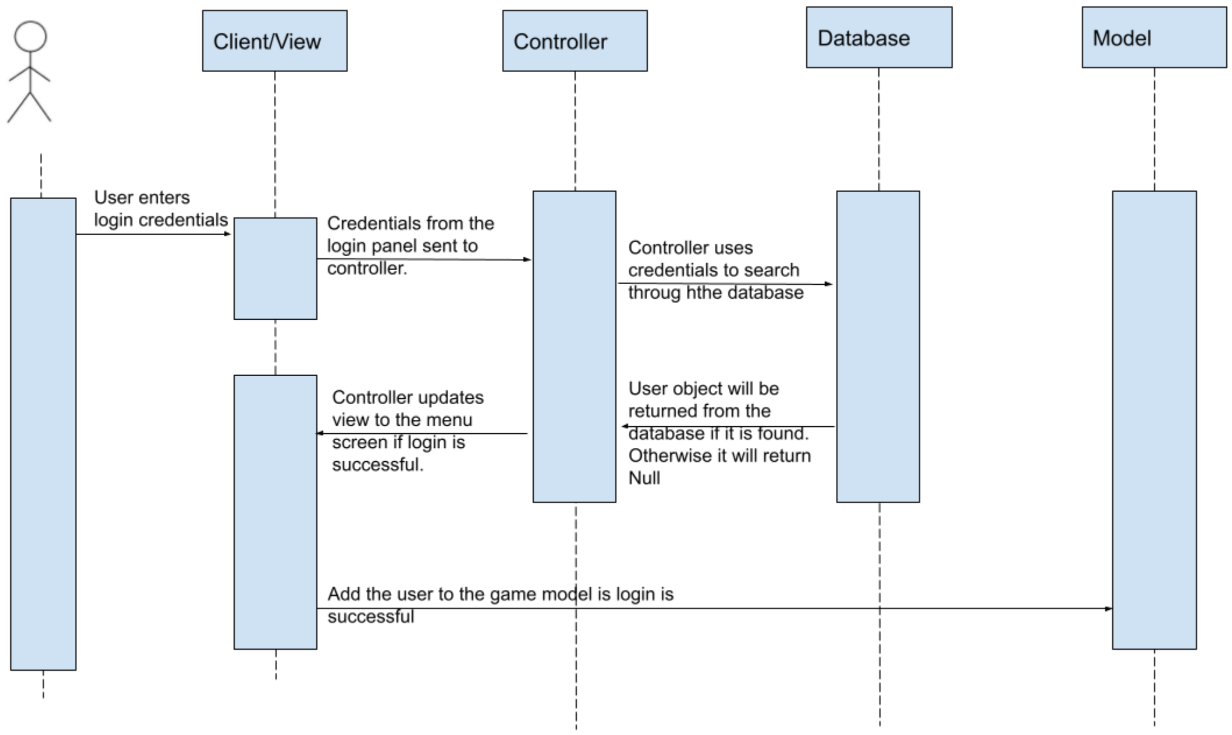
Database:

UserDao:

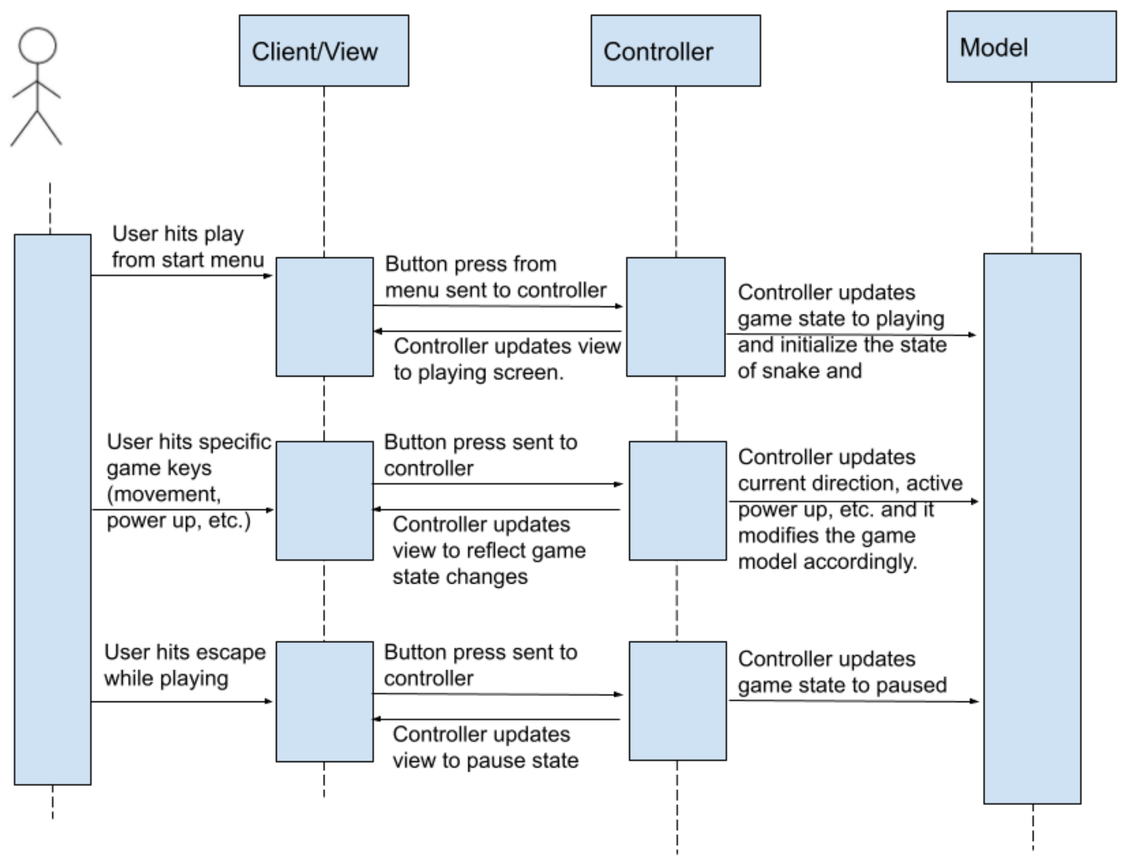
- This is a Data Access Object (DAO) we use to access our database of user information.
- It defines a method called getUser() to access the database and search for the input username and password. It will finally return the user object.
- It defines a method called insertUser() to access the database and add the user information for the account that is created by the player.
- It defines a method called getHighScores() to get the rating of high scores from the database.

Sequence Diagrams

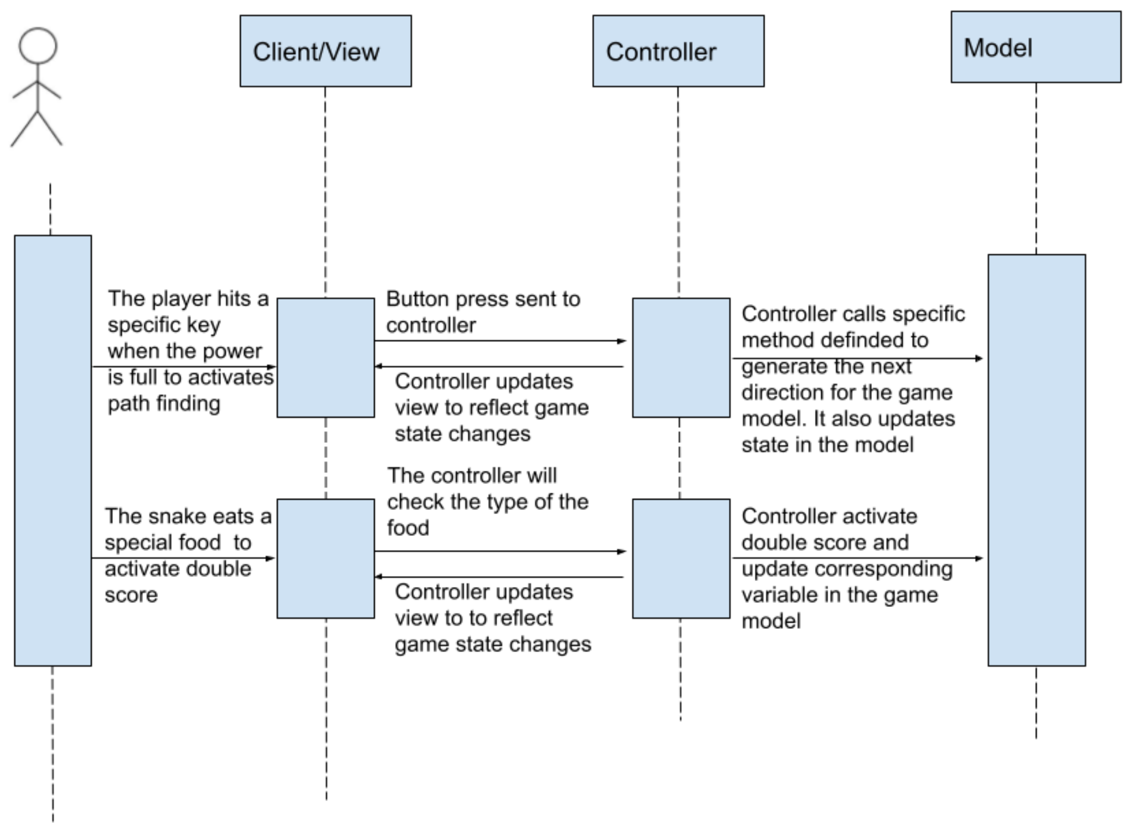
Sequence Diagram of Login:


Client/View
Controller
Database
Model
User enters login credentials
Credentials from the login panel sent to controller.
Controller uses credentials to search throug hthe database
User object will be returned from the database if it is found. Otherwise it will return Null
Controller updates view to the menu screen if login is successful.
Add the user to the game model is login is successful

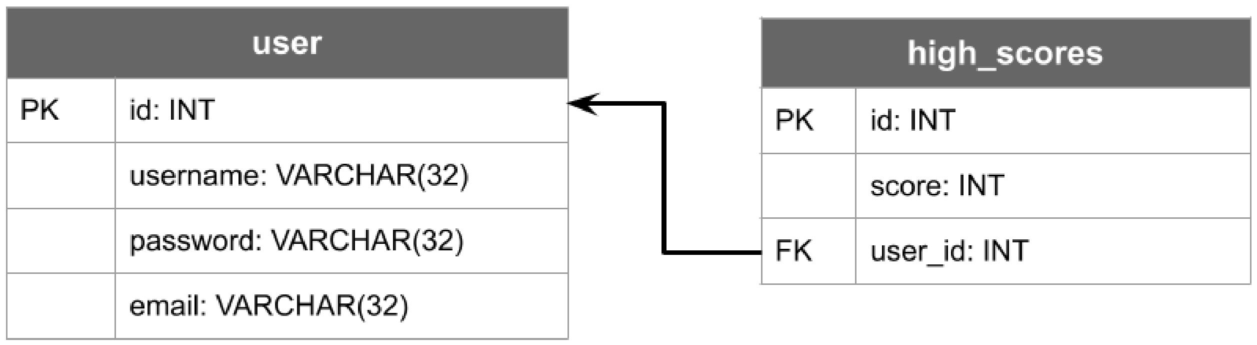
Sequence Diagram of Play:


Client/View
Controller
Model
User hits play from start menu
Button press from menu sent to controller
Controller updates view to playing screen.
Controller updates game state to playing and initialize the state of snake and
User hits specific game keys (movement, power up, etc.)
Button press sent to controller
Controller updates view to reflect game state changes
Controller updates current direction, active power up, etc. and it modifies the game model accordingly.
User hits escape while playing
Button press sent to controller
Controller updates view to pause state
Controller updates game state to paused

Sequence Diagram of Power Ups:


Client/View
Controller
Model
The player hits a specific key when the power is full to activates path finding
Button press sent to controller
Controller updates view to reflect game state changes
Controller calls specific method definded to generate the next direction for the game model. It also updates state in the model
The controller will check the type of the food
The snake eats a special food to activate double score
Controller updates view to to reflect game state changes
Controller activate double score and update corresponding variable in the game model

Database Design


user
PK
id: INT
username: VARCHAR(32)
password: VARCHAR(32)
email: VARCHAR(32)
high_scores
PK
id: INT
score: INT
FK
user_id: INT