Faculty of Engineering, Alexandria University
Computers and Systems Engineering Department
Second year: (CS 221) Programming II
Assignment V - Report

Circus of plates

Made by:

Bishoy Nader Fathy (22)

Amr Mohamed NasrEldine (46)

Marc Magdi Kamel (52)

Michael Raafat Mikhail (55)

I. Design Description

We divided the game into 15 packages as follows:

1- **views:** Contains the layout files and the Entry java class to the game.

2- **controllers:** Contains the JavaFX controllers that connect the views with models

3- **models.graphics:** Contains the graphical parent for all the game objects as plates, players, tracks … etc

4- **models.plates:** contains the plates classes that stores its properties.

5- **models.plates.generator:** contains classes responsible for the creation of plates. It contains the plates dynamic loader, plates pool, plates factory and plates generator classes.

6- **models.plates.movers:** contains classes responsible for moving plates. It contains the track class and the gravity class.

7- **models.plates.collection:** contains plates collection classes and the plates iterator class.

8- **models.player:** contains the model for player with class representing the player's hand.

9- **models.player.input.manger:** contains classes that handle player motion with keyboard and mouse input.

10- **models.dto:** contains the data transfer objects classes across all the game.

11- **game.control:** contains classes responsible for managing the game logic and managing the game thread.

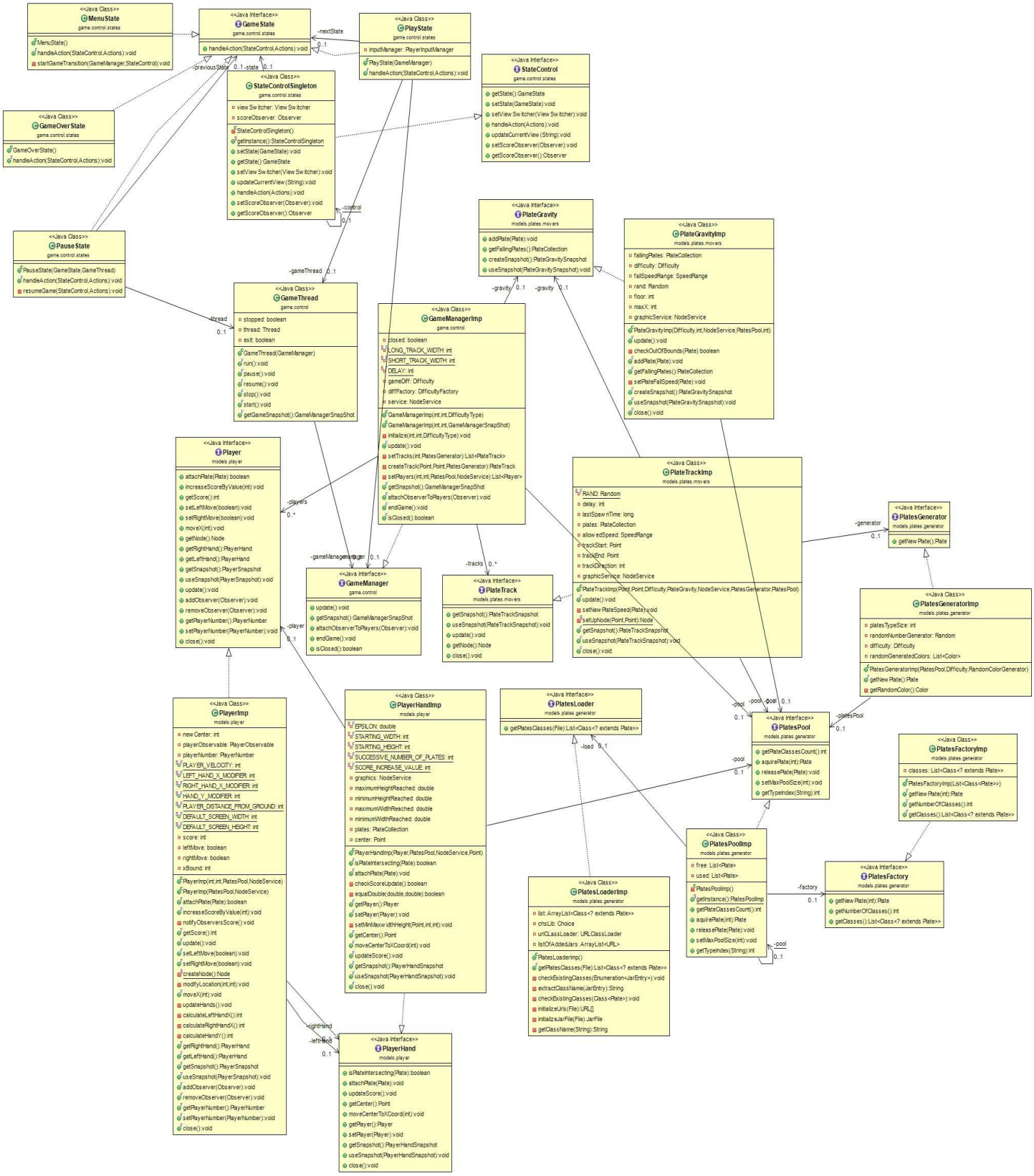
12- **game.control.states:** contains classes responsible for managing the game states either play, pause or menu state. Every state has it has its own class.

13- **game.control.difficulties:** contain classes responsible for managing the game difficulties. Every difficulty has its own class either easy, medium or hard.

14- **file.manger:** contains classes responsible for saving and loading in Json format.

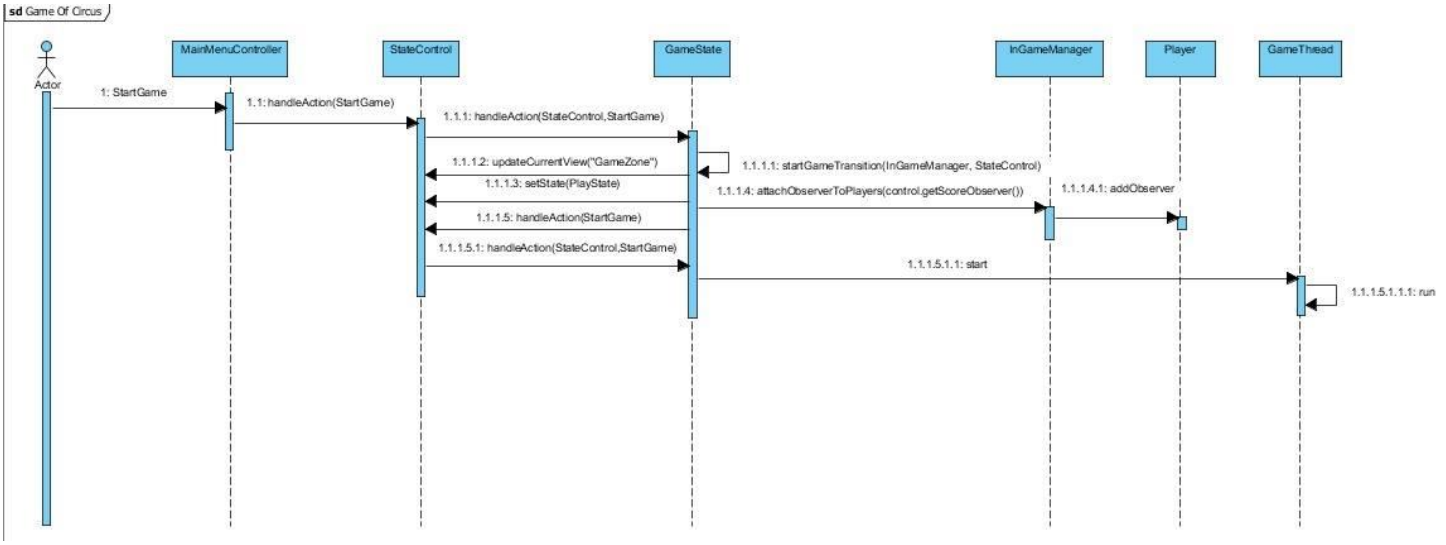
15- **logs:** contains the log classes responsible for logging using log4j. These classes are used in every other class in the application.

II. Class Diagram

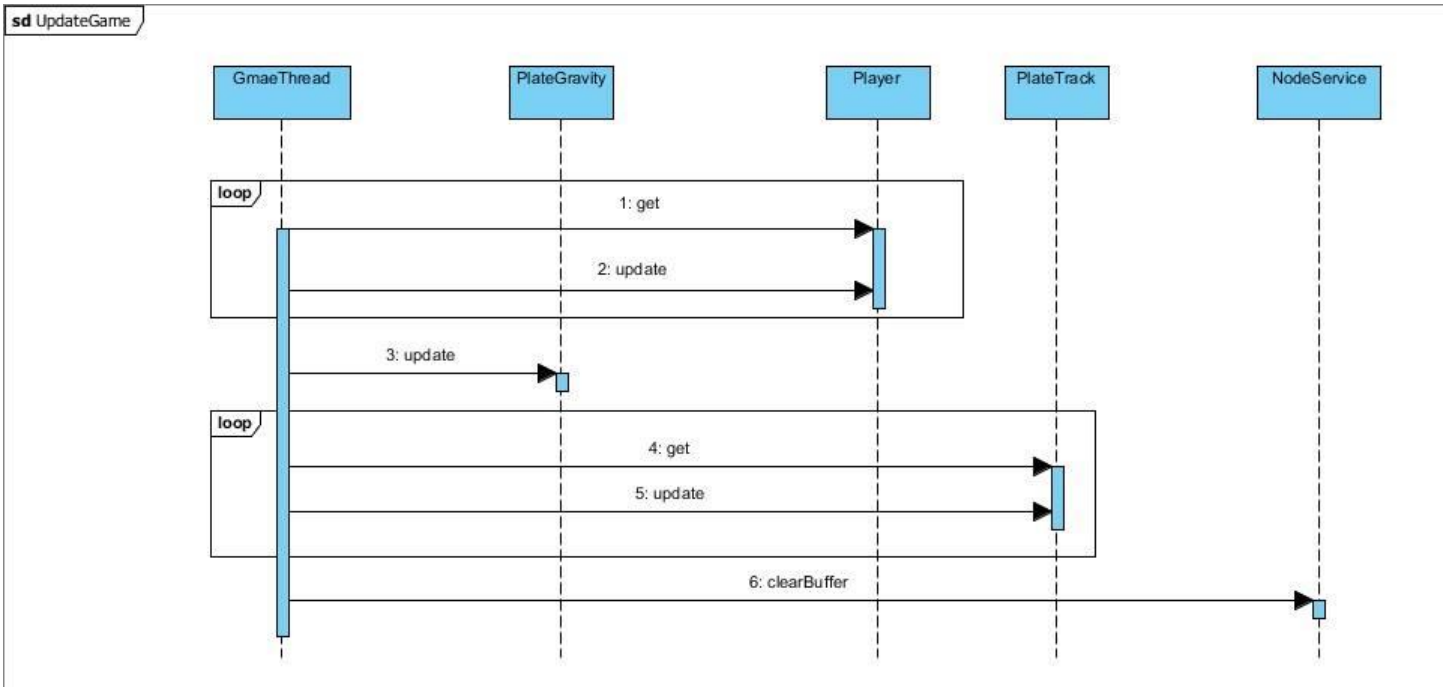


III. Sequence Diagram

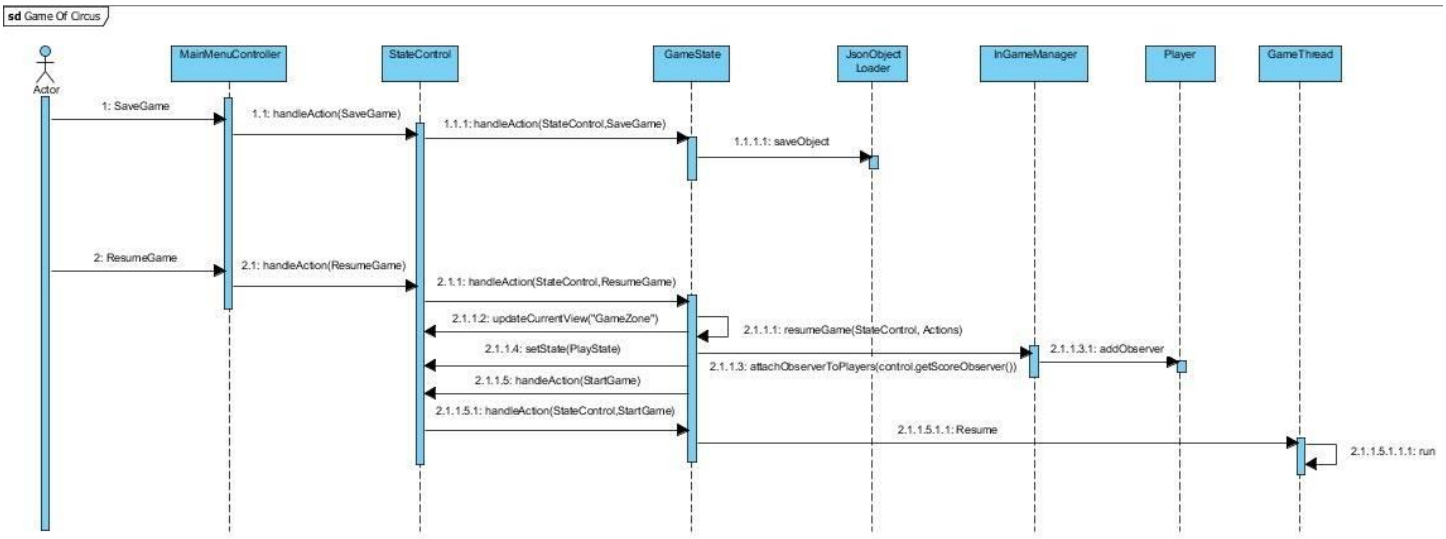
Start Game



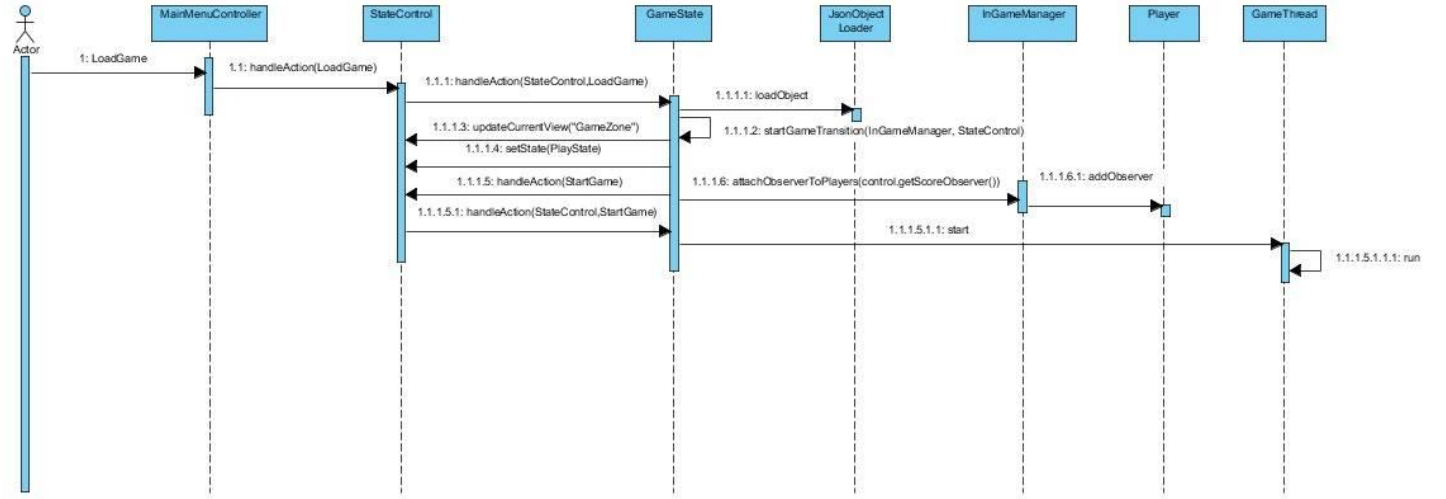
Game Update



Save Game

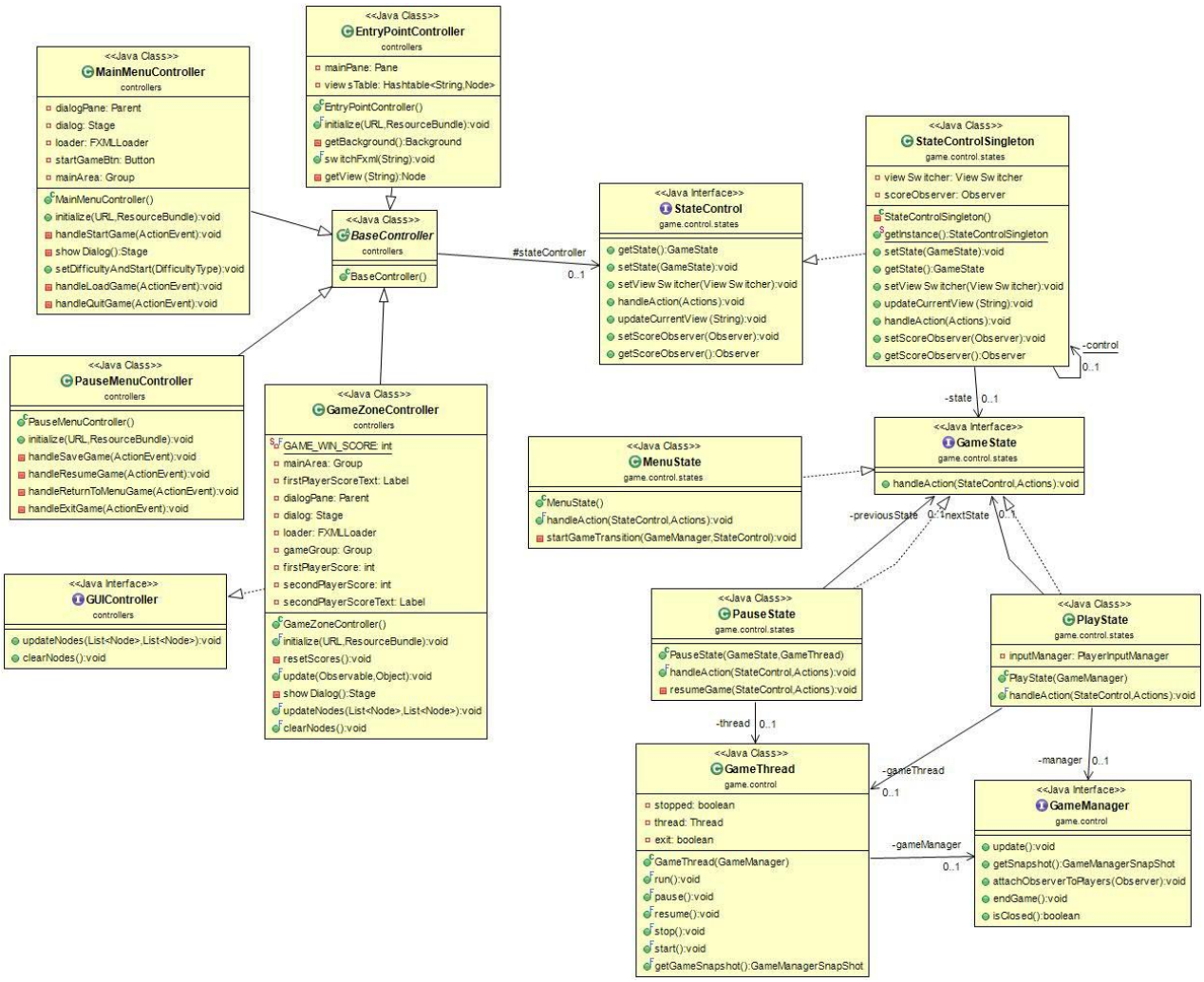


Load Game



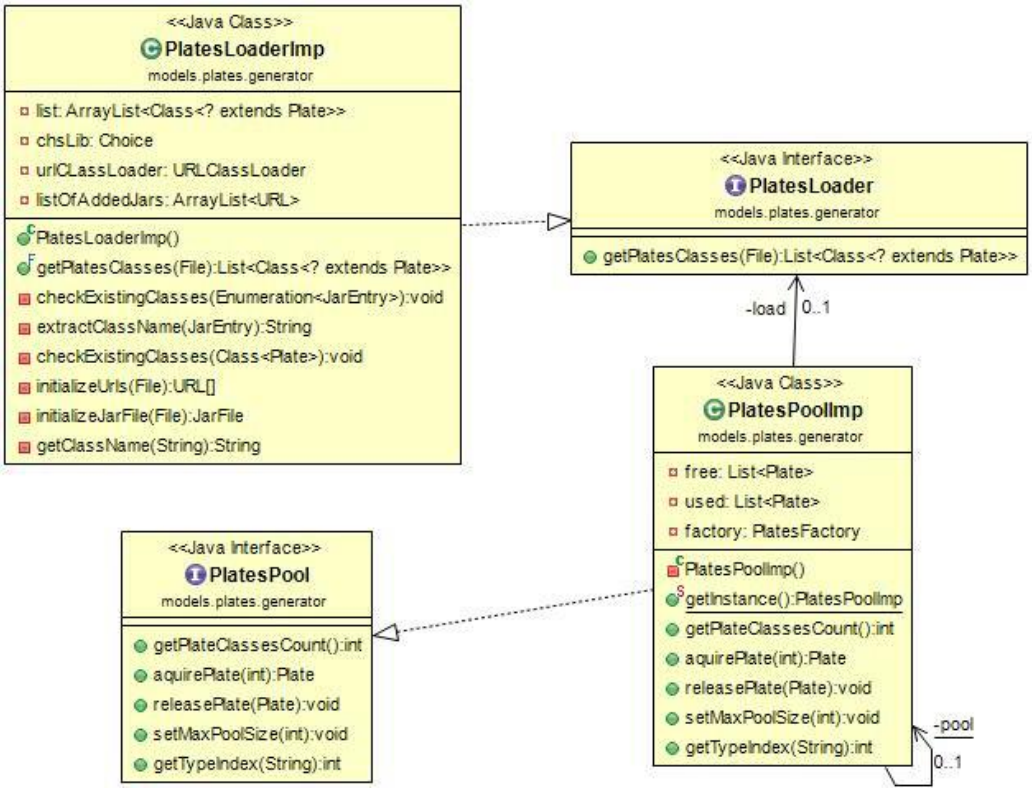
IV. <u>Design Patterns Used</u>

1. ***MVC:*** We used MVC for separation of concerns in the game. We used JavaFX.
   - The views are .fxml files for the GUI.
   - The controllers are java classes where each class is responsible for a specific .fxml file. The JavaFX uses annotation for connecting UI with java code.
   - The models is the main core of the game including classes for representing plates, players, tracks … etc and also classes for management of the game logic.



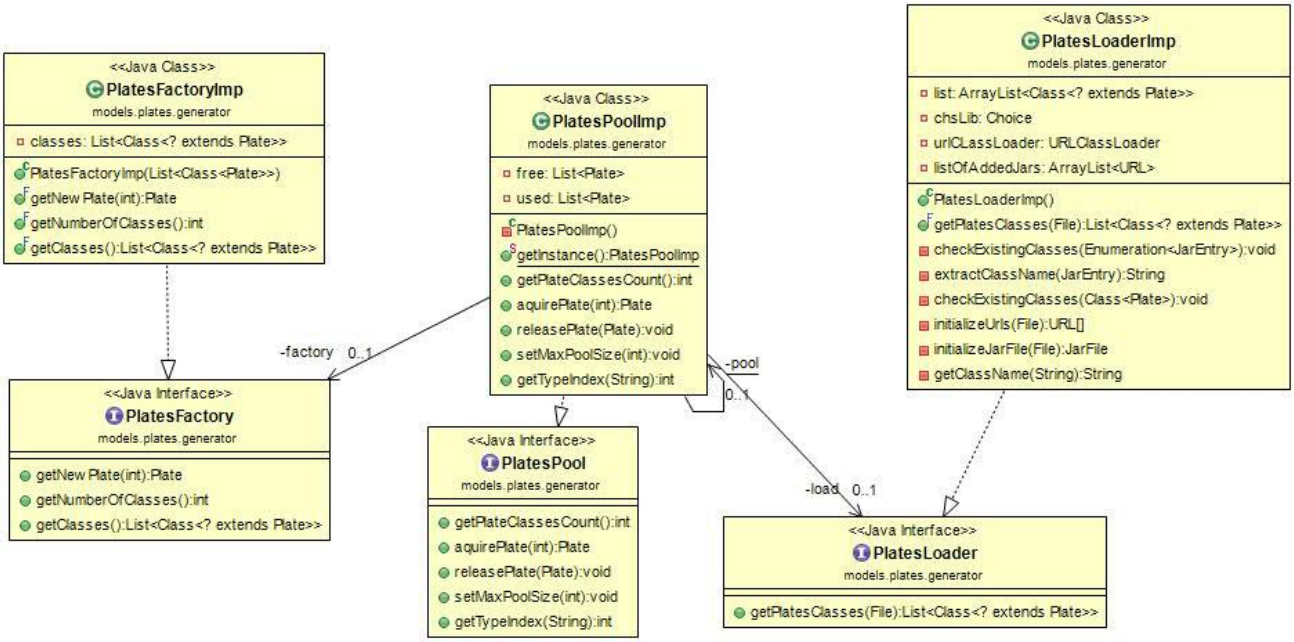
2. Dynamic Linkage

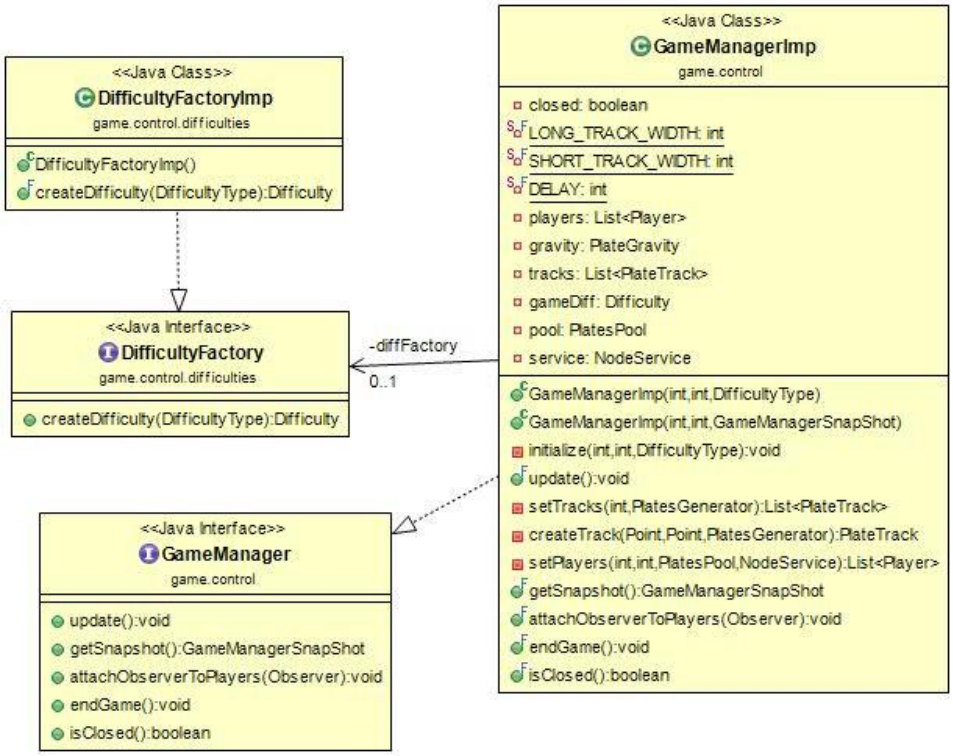
We used Dynamic Linkage for loading all plate types, The game loads any class that extends an abstract class called Plate. The user can add plates by loading their classes from an external .jar file.



3. *Factory*

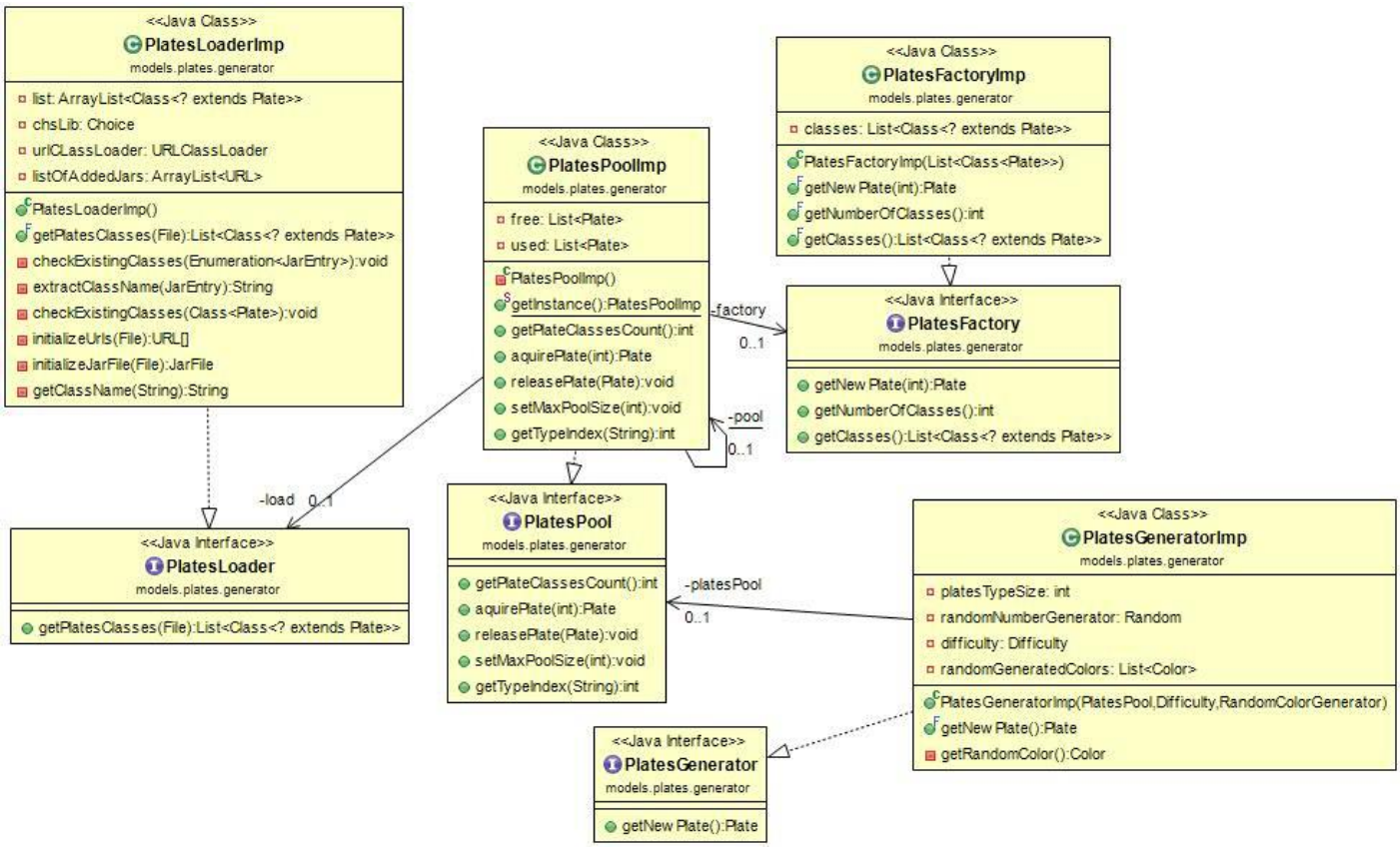
We used the Factory pattern for creating plates from the loaded classes. The pool loads the plates classes dynamically and register these classes to the factory then requests a new plate from the factory and sends the index of the requested plate class and the factory returns an instance from that class to the pool.





4. *Object Pool*

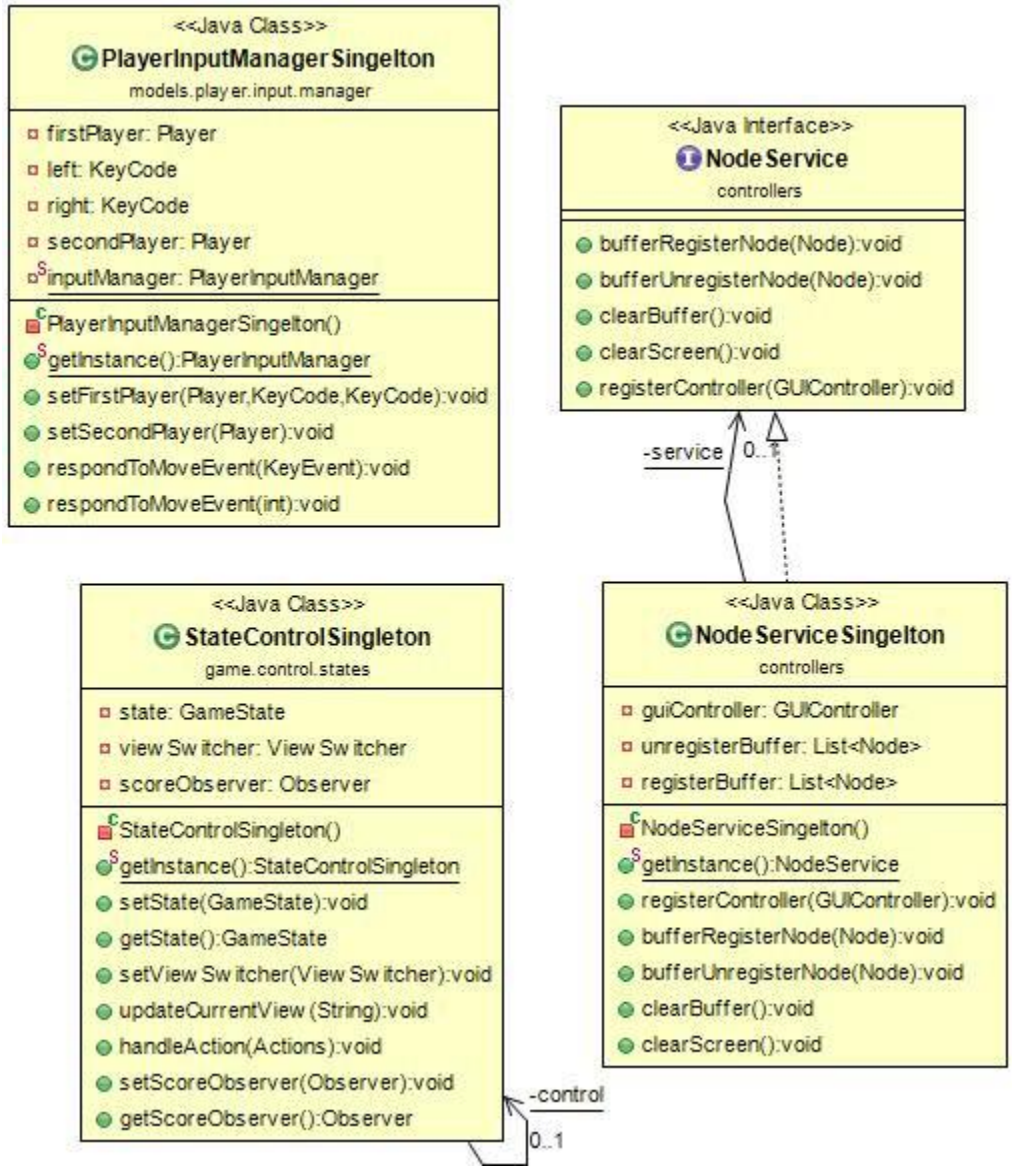
We used the Object pool pattern during the game for managing and reusing plates that are created by the factory. This ensures a specific number of plates. After a plate reaches the bottom of the screen the plate is added again to the pool to be shown again on the screen. Also after a player catches three plates of the same color they are added to the pool to be reused again.



5. *Singleton*

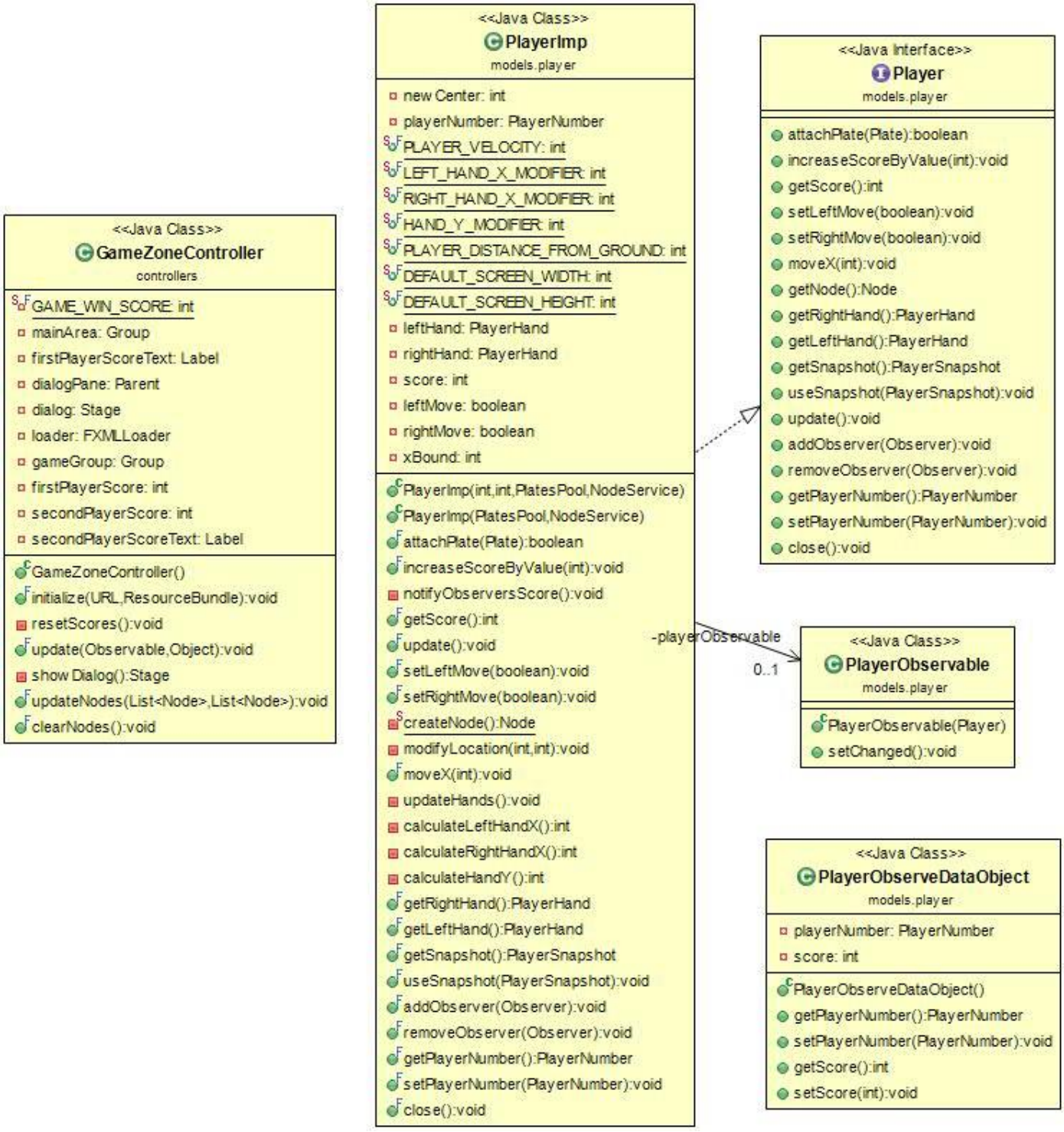
We used Singleton pattern for the classes that we want to ensure only one instance is available. We used it in the following classes:

- PlayerInputManager class: responsible for management the mouse and keyboard input controls for the two players.
- StateControl class: responsible for controlling the state of the game either menu, play, pause, close. It also switch the game views according to the current state.
- NodeService class: responsible for adding and removing nodes from the screen.



6. *Observer*

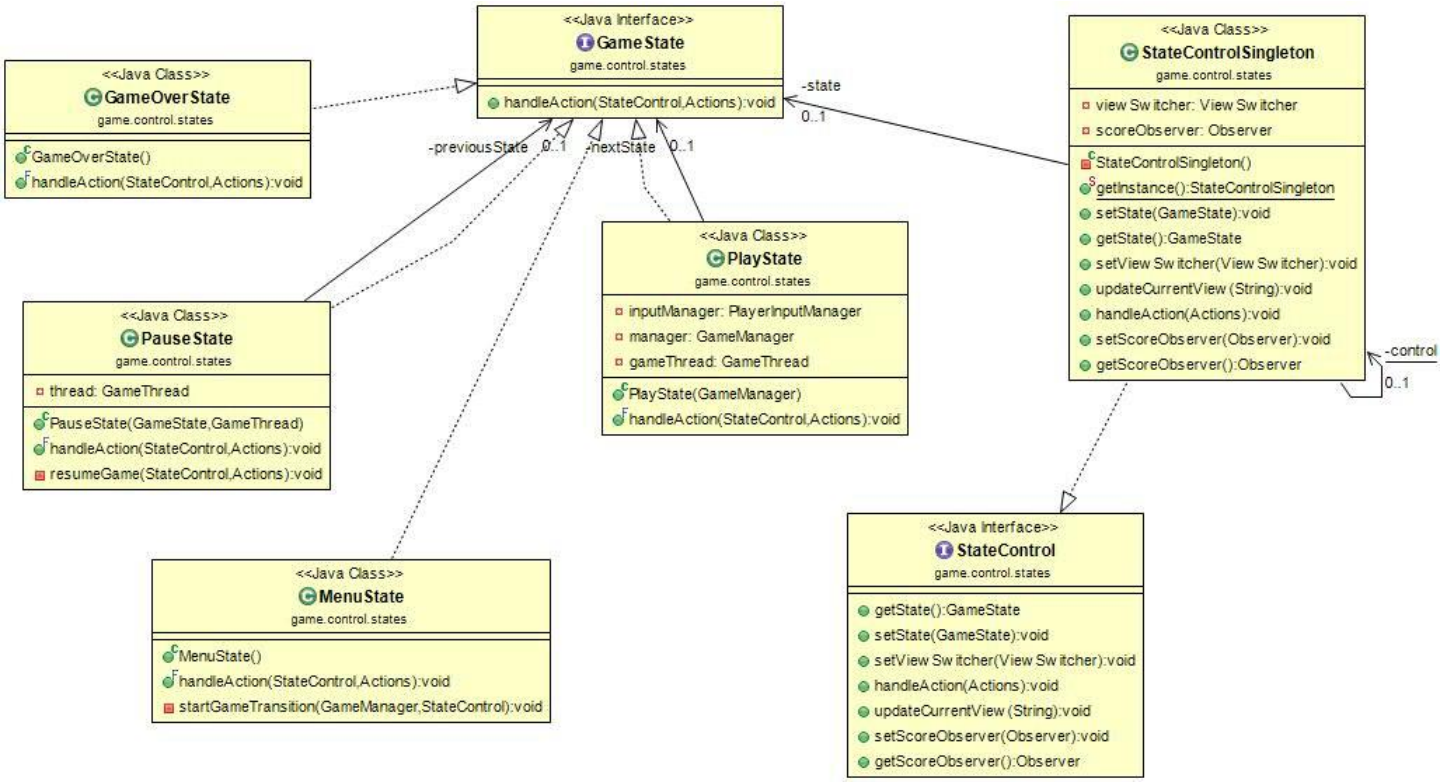
We used observer pattern to notify the UI when the score of a player is changed, we used delegated observer where the player represents the subject and the Game controller represents the Observer that is notified when score is increased or the game is ended.



7. State

We used the State pattern for managing game states. It's responsible of managing the transfer between the different views according to the current game state. We have the following states for the game:

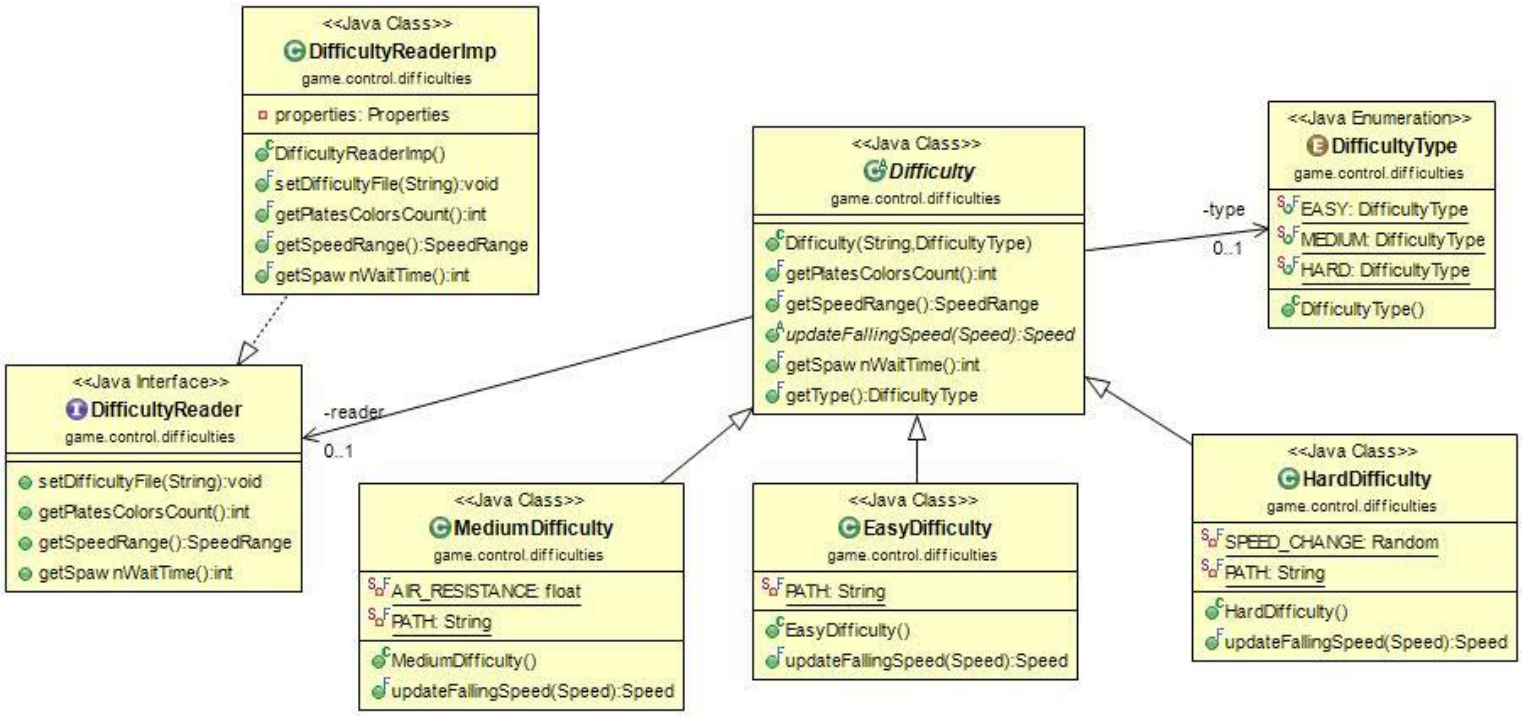
- Menu State: initial state at the start of the game.
- Play State: where the game starts and players starts playing.
- Pause State: where the game is paused and pause menu appears.
- Game Over State: when the game is over and one player is a winner.



8. Strategy

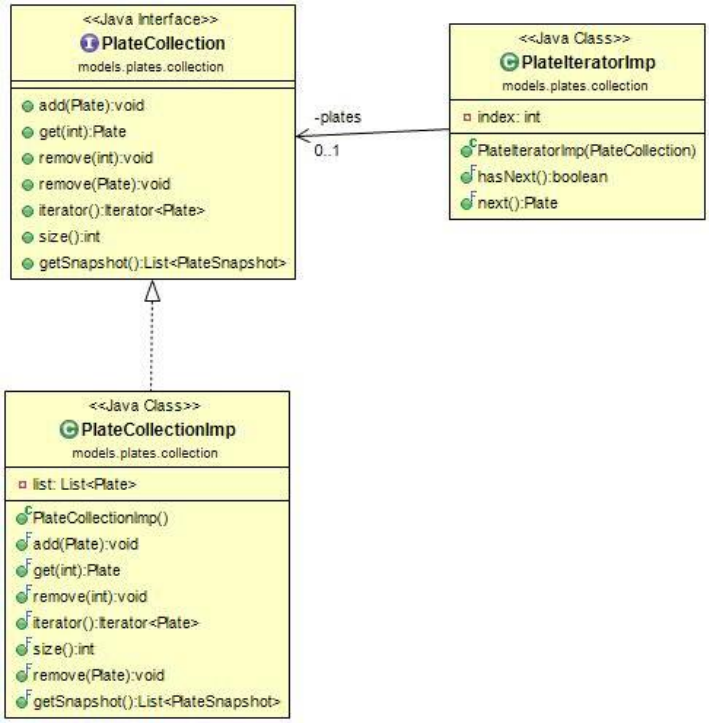
We used the Strategy pattern to select the strategy of moving the plates according to the selected difficulty.

- In easy difficulty the plates move in a straight line.
- In medium difficulty the plates move in a projectile motion.
- In the hard difficulty the plates move in a sinusoidal wave.



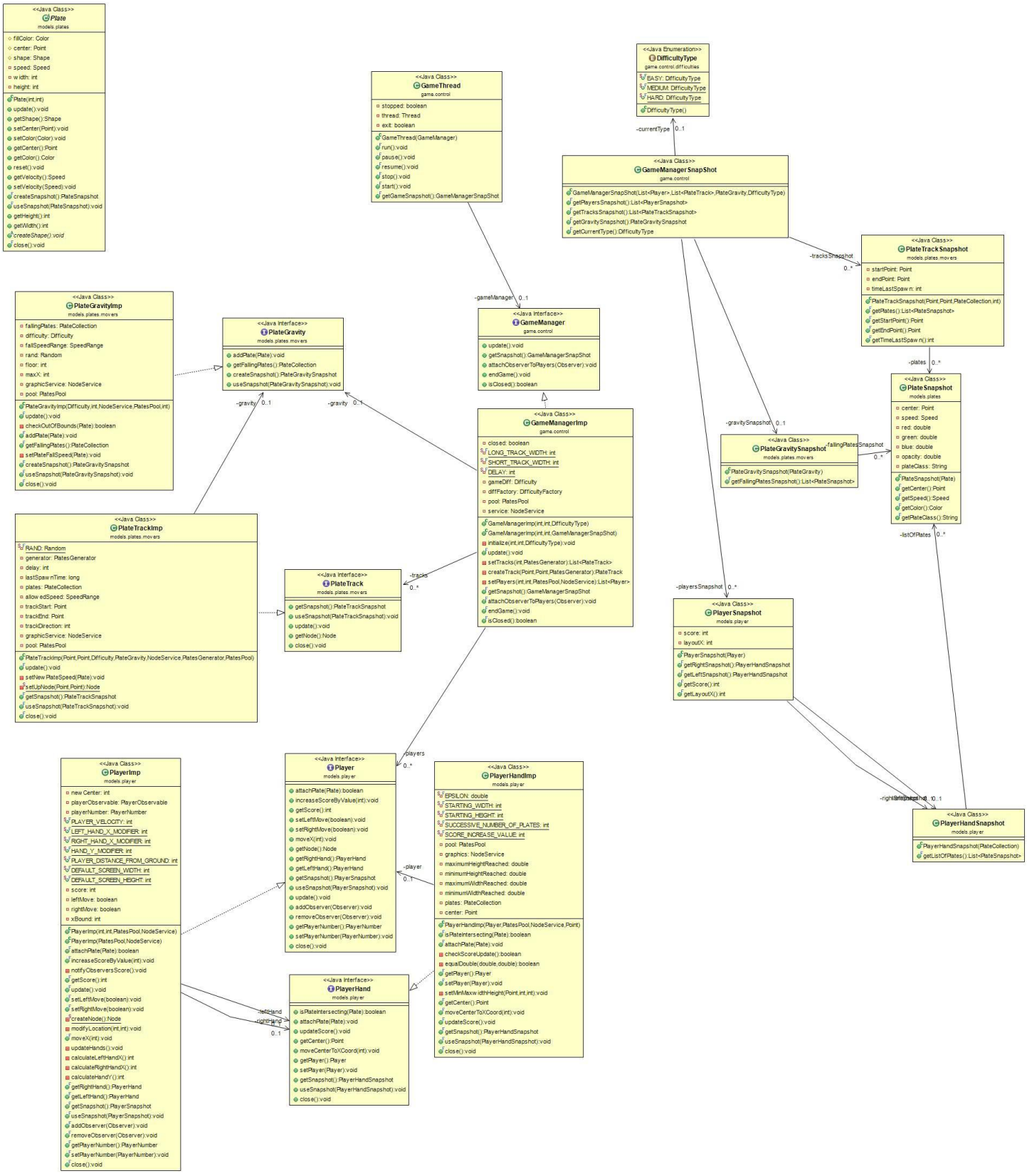
9. Iterator

We implemented the iterator design pattern in the plates collection. Every player has two plates collections. We used the iterator to loop over the collections for score calculations and to make the plates move with the movement of player.



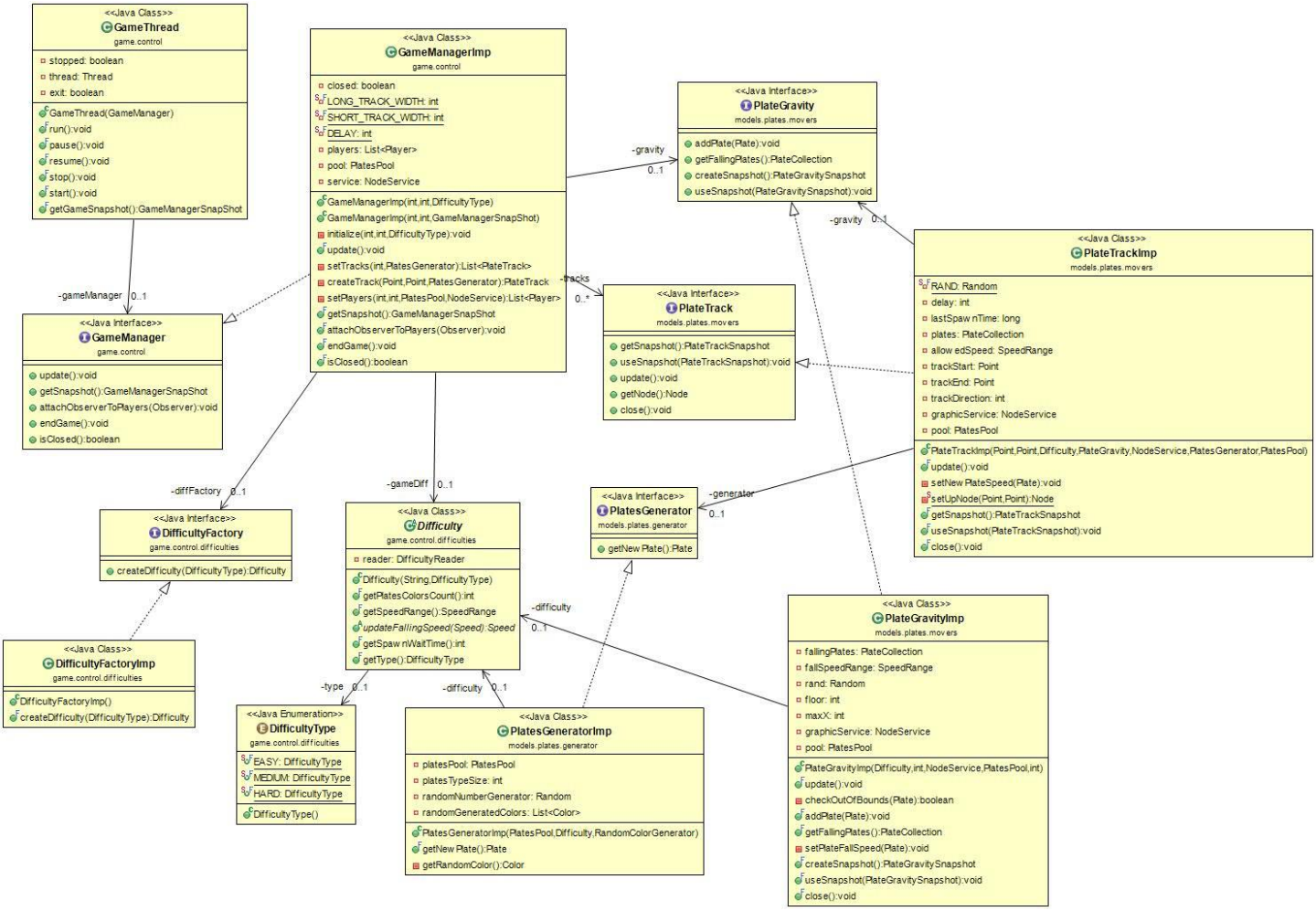
10.Snapshot

We used the snapshot pattern for saving and loading the game. Every savable object [Plates, Player, Tracks, Gravity, etc …..] has a corresponding snapshot.



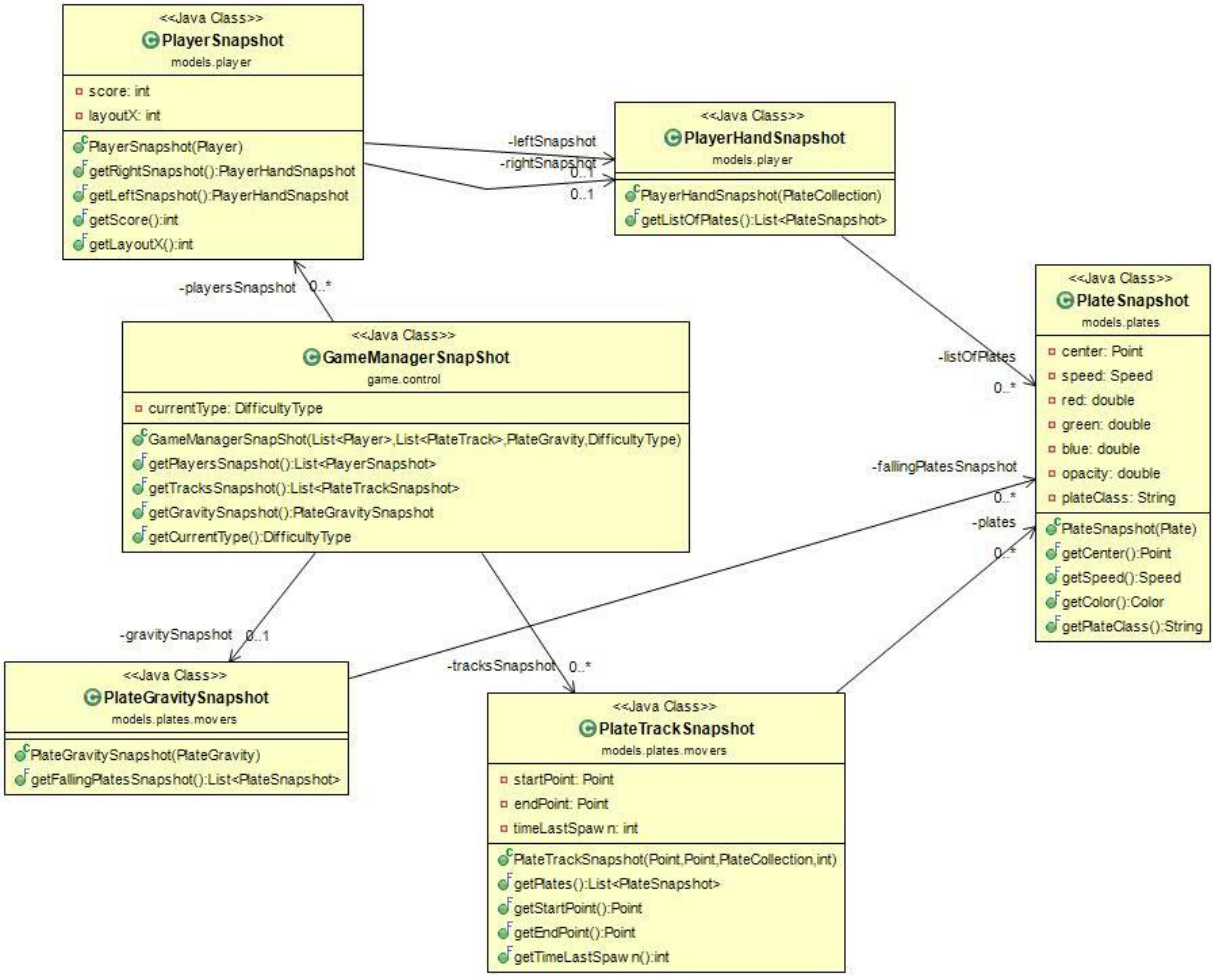
11.Façade

We used the Façade to the make the game manager an abstraction for all the interior classes . All the interior classes are not seen outside the game manager.

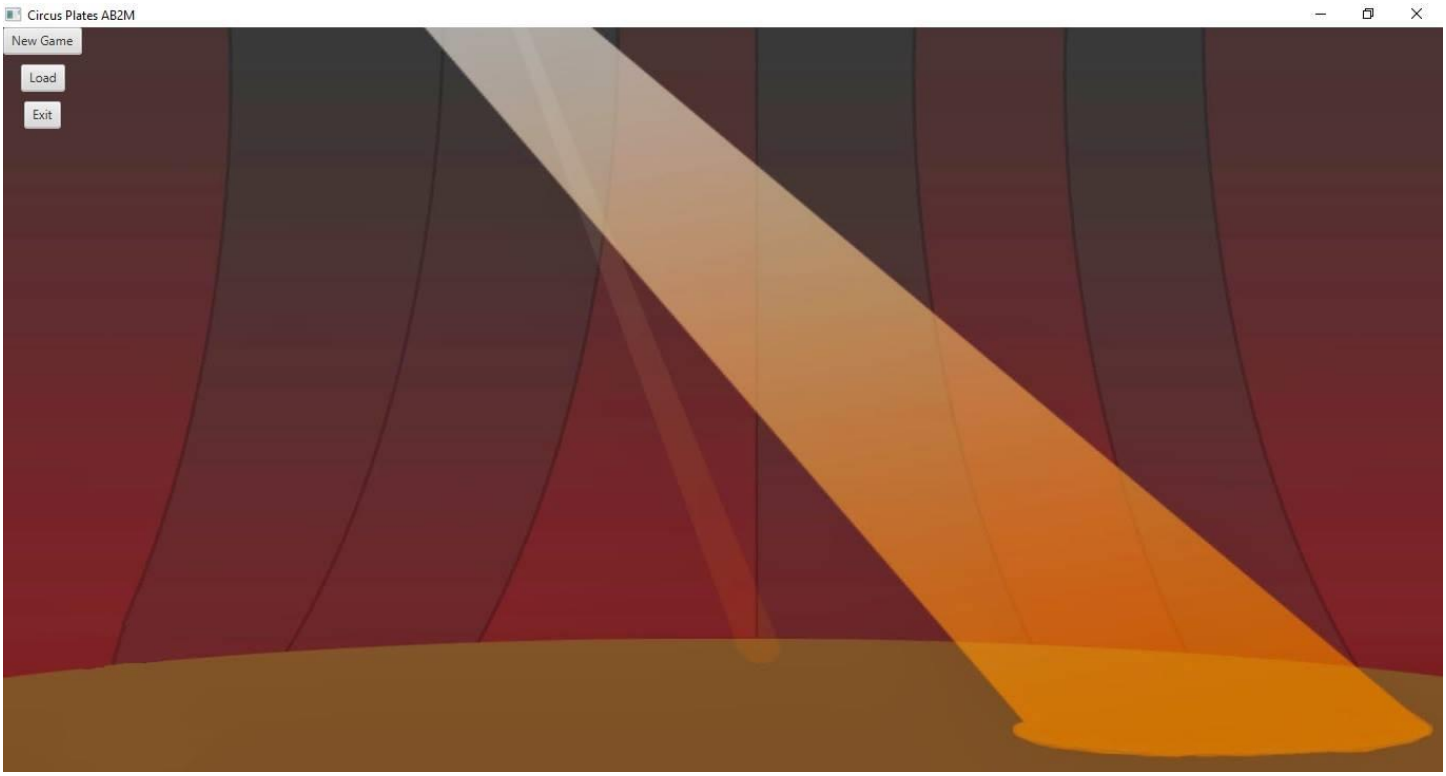


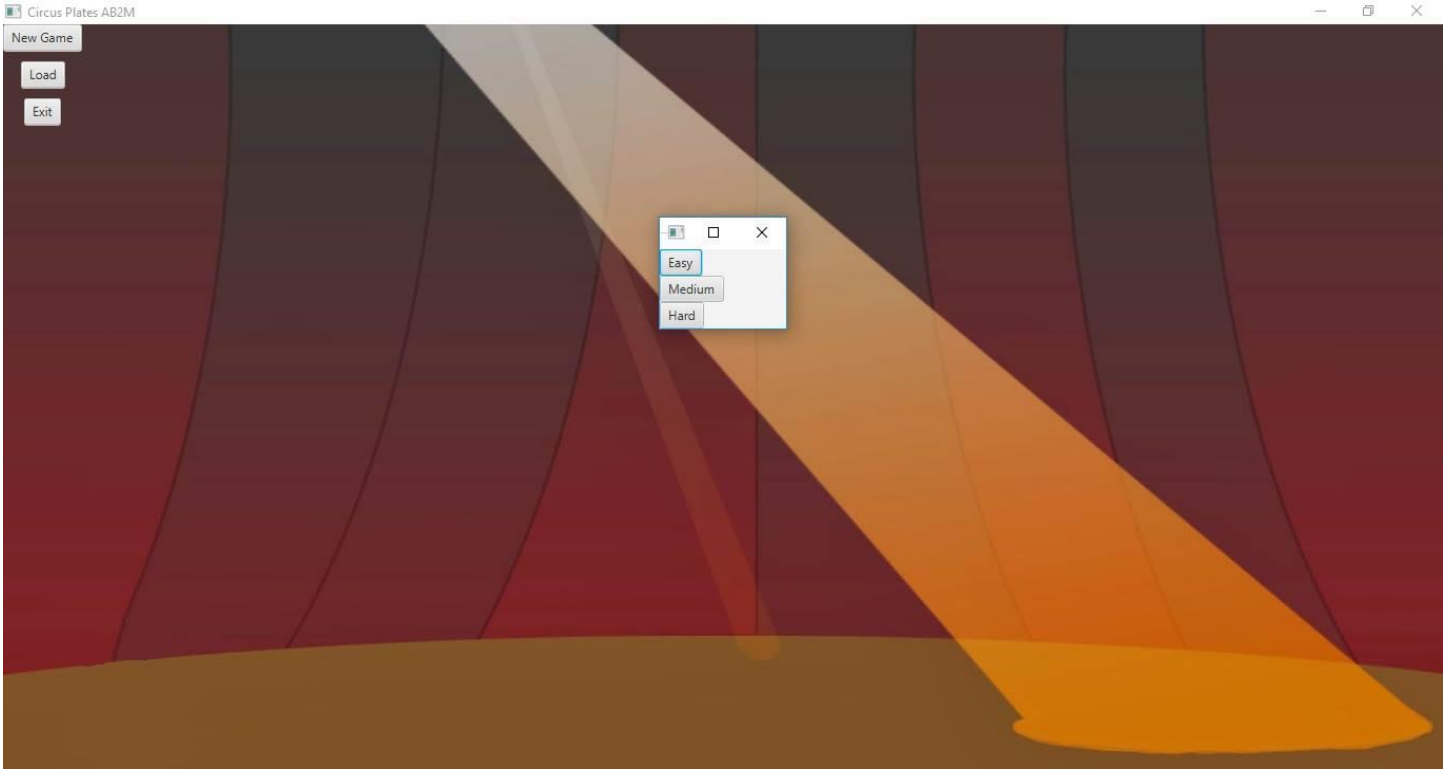
12.Immutable

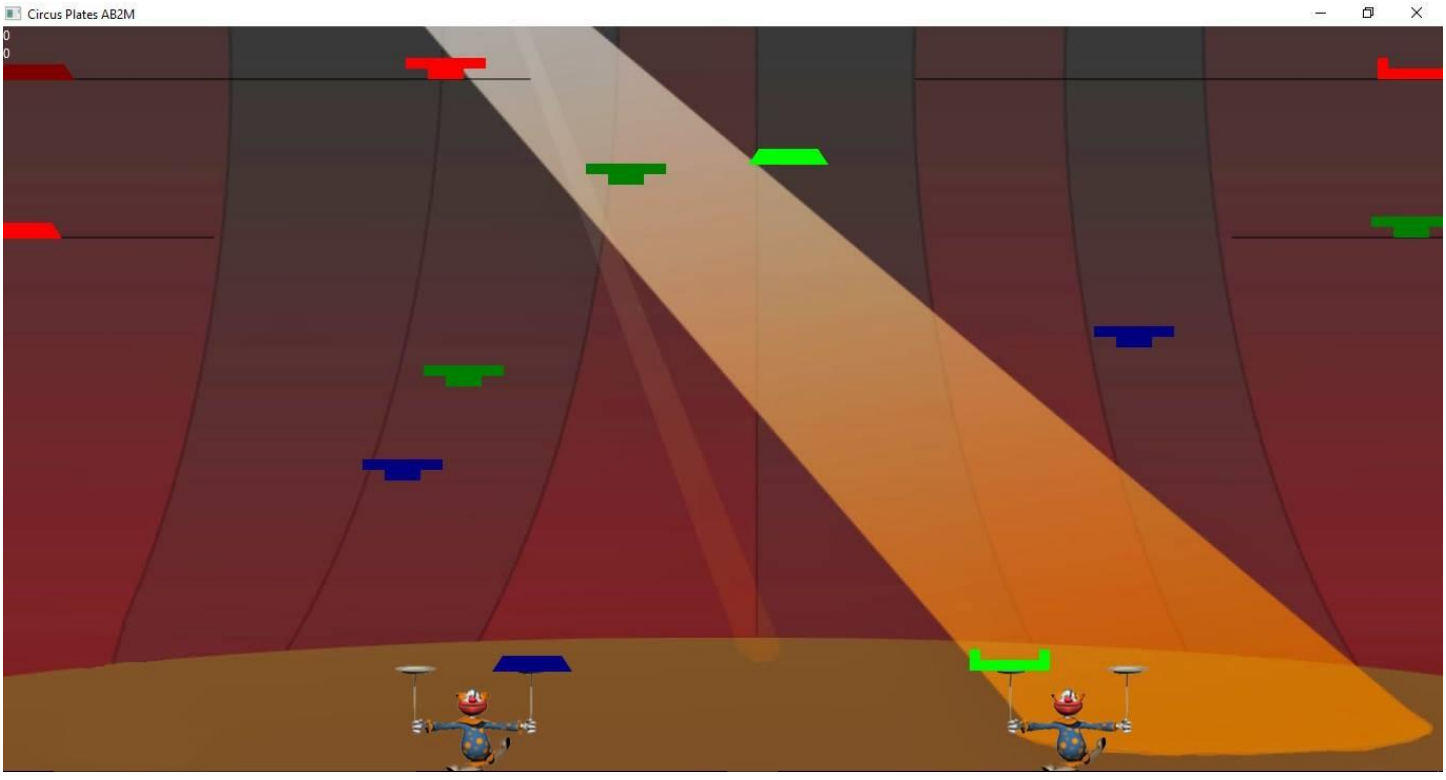
All snapshots are immutable, once constructed can no longer be modified.

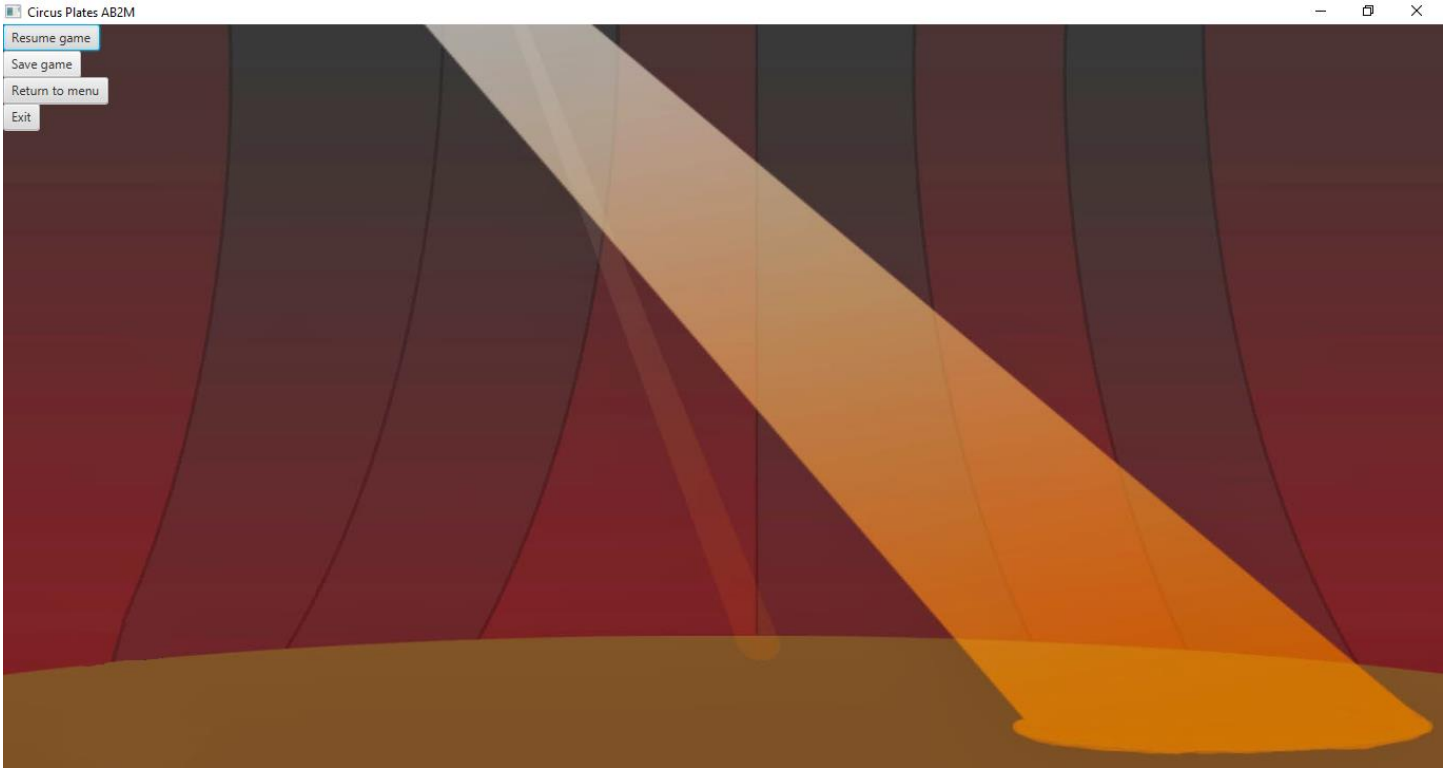


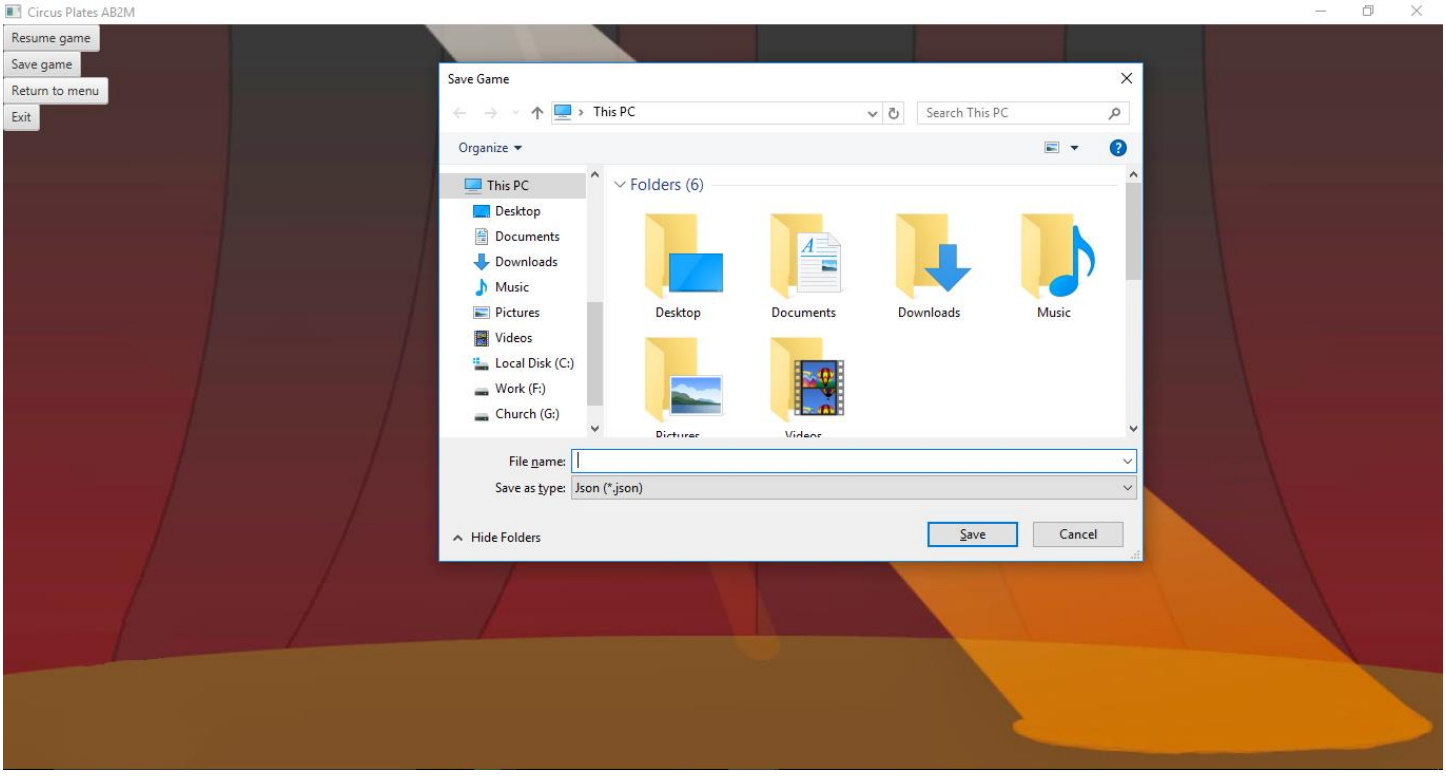
V. GUI Snapshots

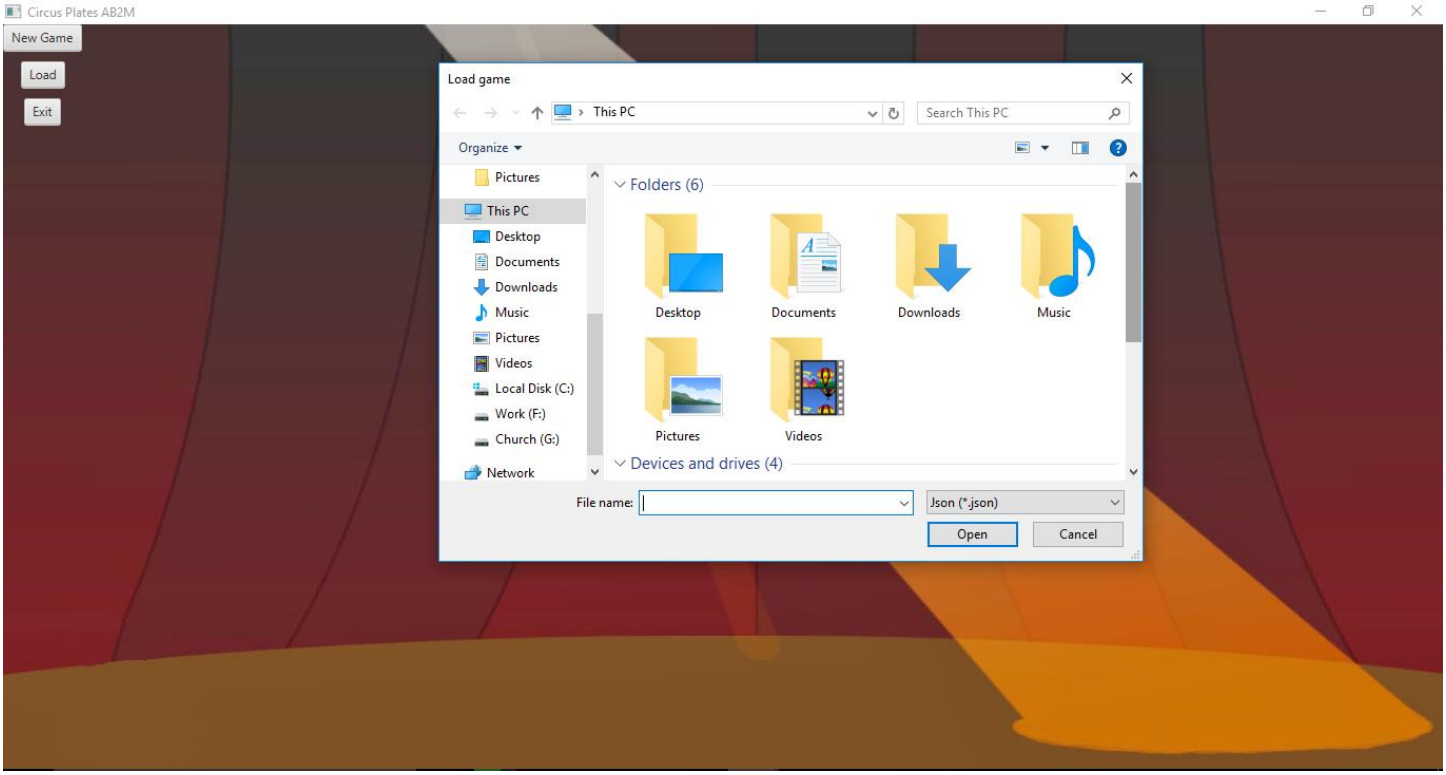


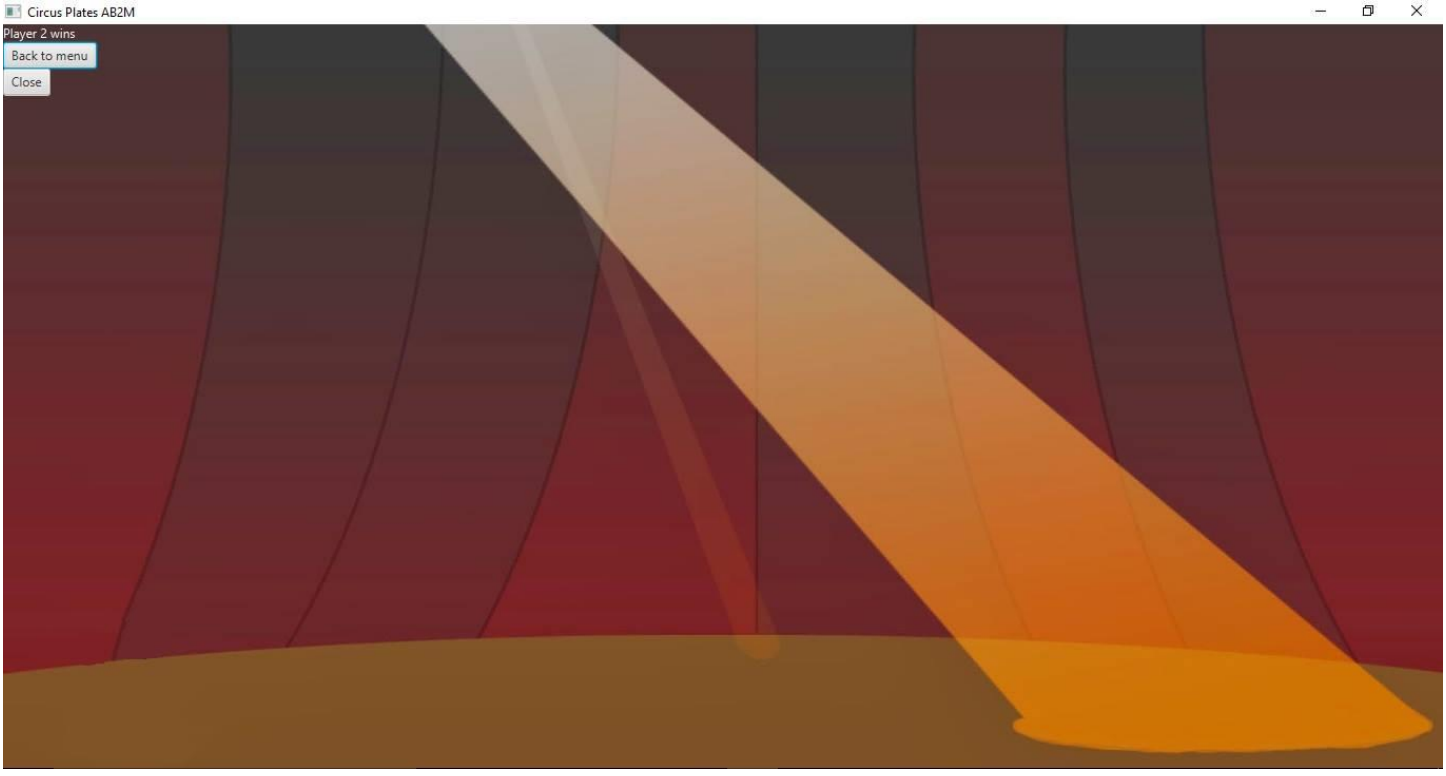

Circus Plates AB2M
New Game
Load
Exit
Easy
Medium
Hard


Circus Plates AB2M
0
0


Circus Plates AB2M
Resume game
Save game
Return to menu
Exit


Circus Plates AB2M
Resume game
Save game
Return to menu
Exit
Save Game
This PC
Search This PC
Organize
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Work (F:)
Church (G:)
Folders (6)
Desktop
Documents
Downloads
Music
File name:
Save as type:
Json (*.json)
Hide Folders
Save
Cancel


Circus Plates AB2M
New Game
Load
Exit
Load game
This PC
Search This PC
Organize
Pictures
This PC
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Work (F:)
Church (G:)
Network
Folders (6)
Desktop
Documents
Downloads
Music
Pictures
Videos
Devices and drives (4)
File name:
Json (*.json)
Open
Cancel


Circus Plates AB2M
Player 2 wins
Back to menu
Close

VI. User Guide

At the beginning of the game the main menu appears which contains three options

- New Game: promote to the user to choose one of three difficulties (Easy – Medium – Hard) and then start a new game.
- Load Game: promote to the user a load dialog to choose a json file load the game form.
- Exit: quit the user out of the game.

Once the user is in the game, one player moves with keyboard arrows and the other one move with the mouse.

Any time in the game you can pause by clicking the ESCAPE key. The pause menu appears which allow the user to

- Resume the game.
- Save the current game.
- Exit.

Players try to catch the falling plates, if they manage to collect three consecutive plates of the same color, then they are vanished and their score increases.

The user gets a point when he collects three consecutive shapes from the same color (even if they are different shapes).

If one player's score reaches 10 then he wins, also if a player catches 30 plates in any hand then he loses.

VII. Design Decisions

Our game is divided into three main services.

- First one is for generating new plates which is connected to the pool to get a new random plate and sets its color with a random color. So no other services know about the factory which creates new plates or the pool that uses a created one.
- The second service is the one responsible for moving the plates which is divided into two sub services the track which control the plate movement horizontally and the gravity that is responsible for falling the plates.
- The last service is the one responsible for updating the views, make the transitions between the states.