Alexandria University

Faculty of Engineering

Computer and Systems Engineering Dept.
CS-221: Programming-2

Circus of Plates

Ahmed Magdy (12)

Moamen Rafaat (52)

Mohamed Ayman (61)

Mahmoud Hussien (70)

Design Description:

Circus of plates is a GUI game based on JavaFX library with an intensive usage of different design patters to ease the development and extension anytime later.

The main design is based on MVC pattern to separate the data of the game from the controller and the user interface.

Other design patters are used as follows:

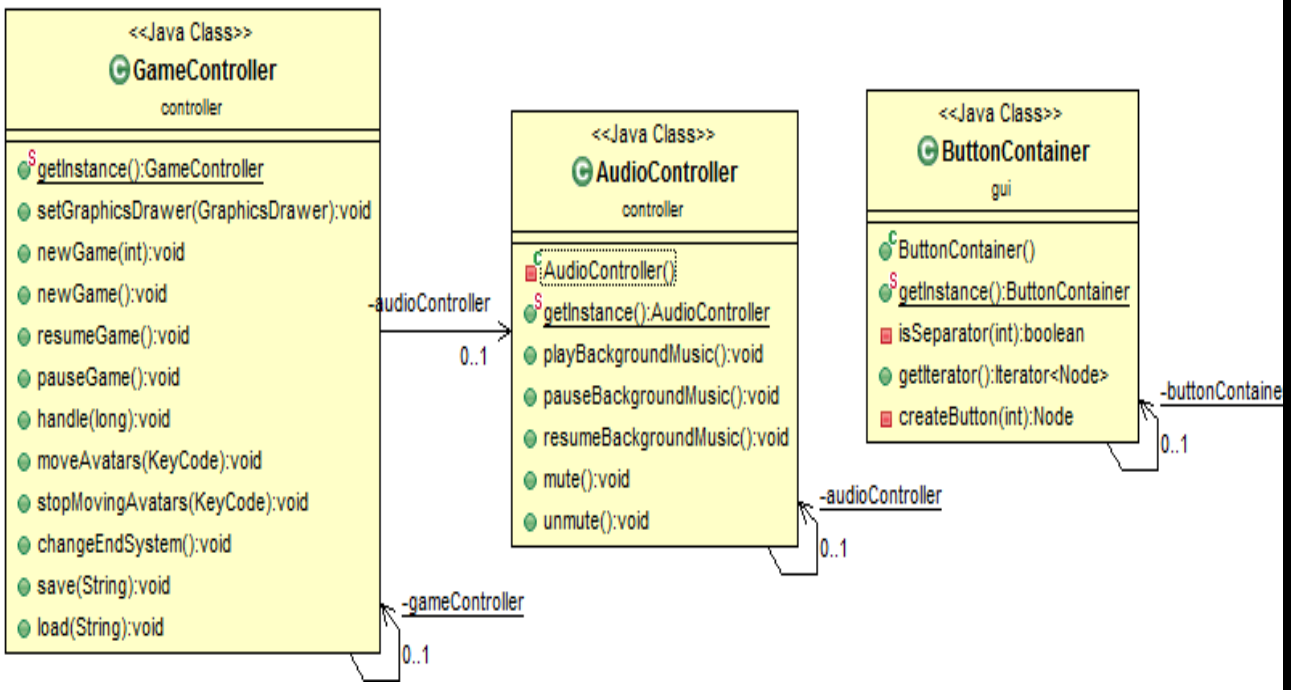
- Singleton is used to represent any system component and controllers that doesn't need more than one object during the runtime.
- Factory pattern is used with the generating the buttons of the main UI and creating new shapes from whatever loaded shapes during the runtime.
- Object Pool is used to limit the usage of resources represented at the falling shapes by setting a maximum limit for the used shapes and re-using the shapes after they vanish instead of creating new shapes.
- Iterator is used to iterate through all the GUI Components to build it at the start of the game.
- Dynamic Linkage is used for dynamically loading the shapes used in the game at run-time.
- Snapshot Design Pattern is used to save the game status
- State is used to differ the behavior of the character’s stacks depending on the current state of the stack.
- Strategy is used to distinguish between the two modes of the game, will be clarified in the user guide section.
- Observer is used to notify the system with any change would lead to changing the current game state (e.g. scoring a point or ending the game).

Log4j is used to provide a complete system logging.

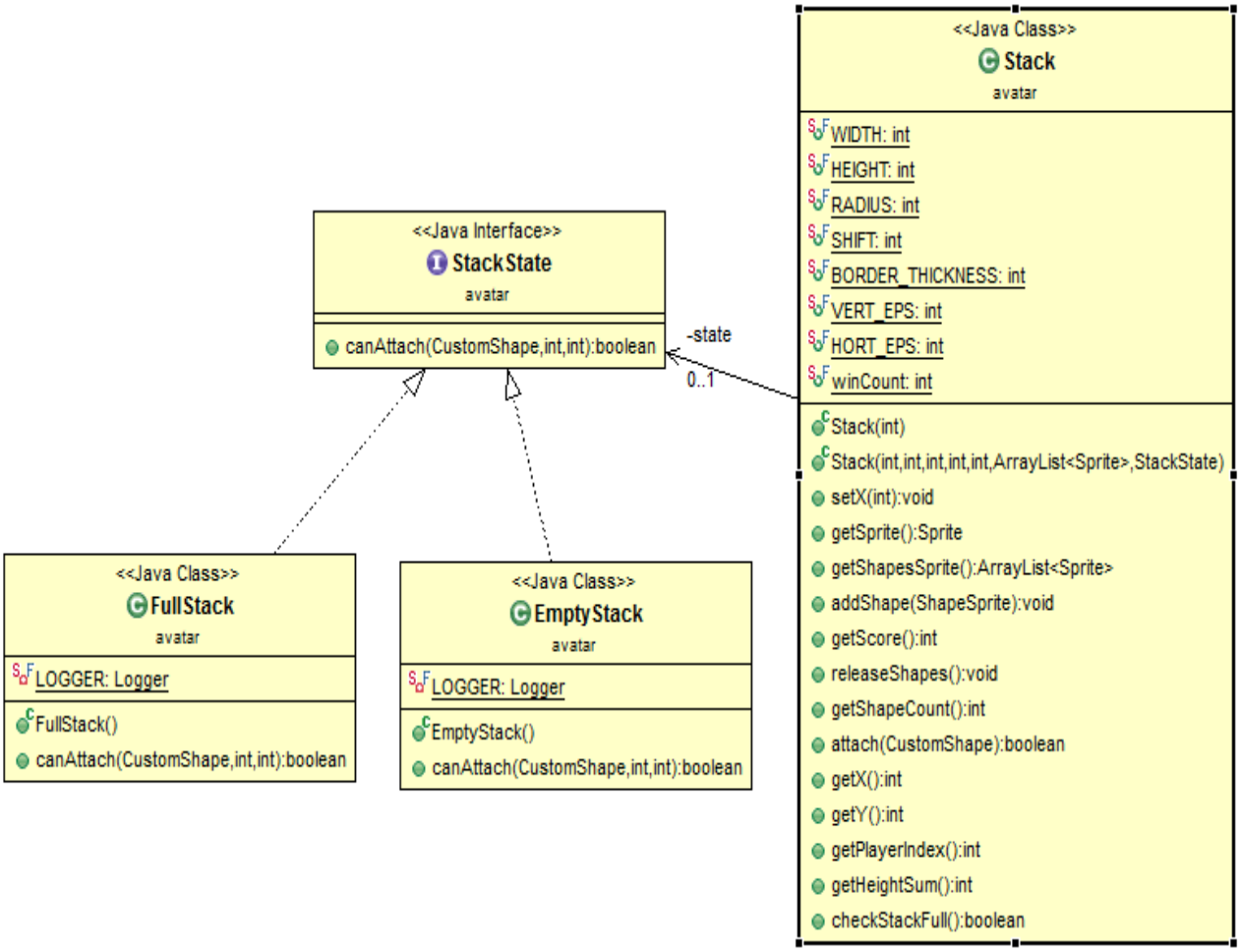
UML Class Diagrams:

- **Singleton Pattern**

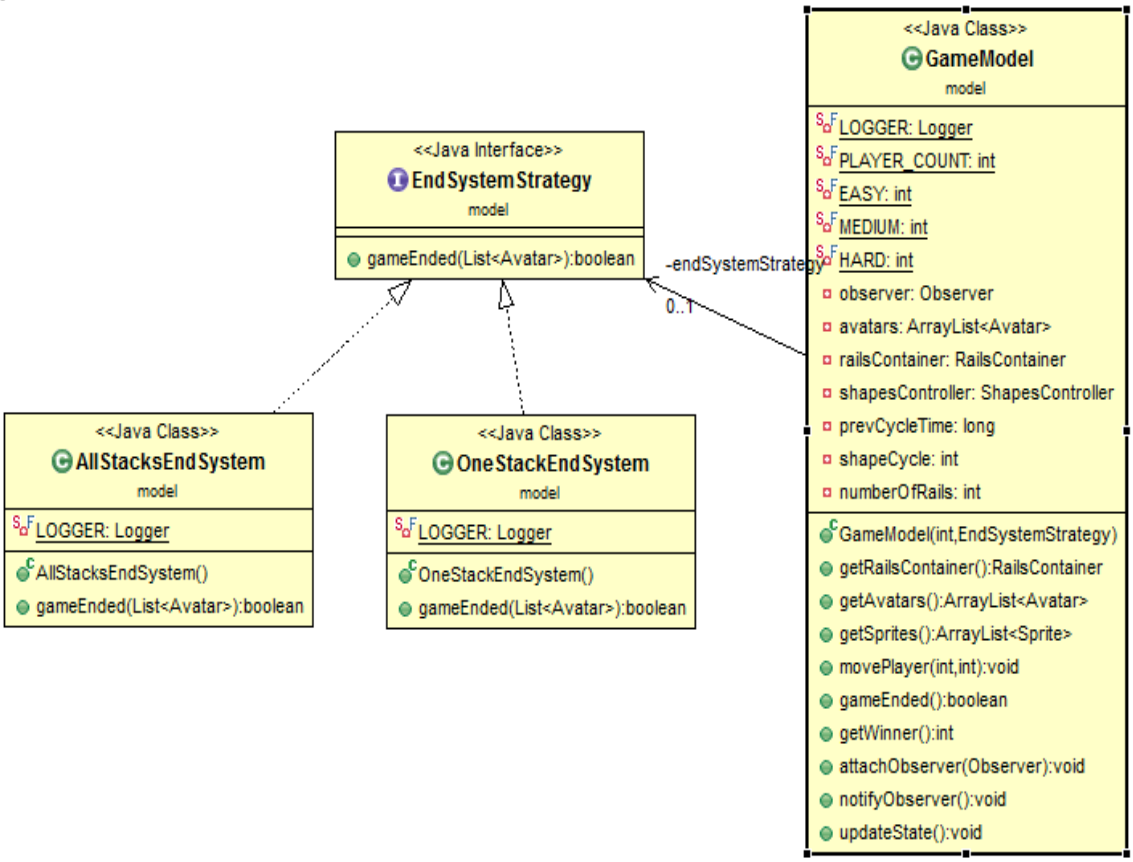
  Singleton is used to create one instance of the variant system and controller classes that doesn't required more than one object during the run-time.



- **State Pattern**
  State pattern is used to vary the behavior of the character's stacks depending on the current state of it.

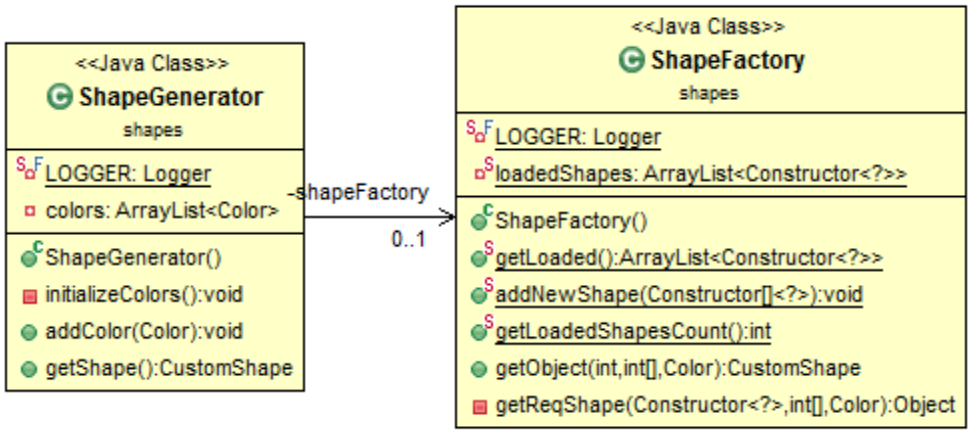


- **Strategy Pattern**
  Strategy pattern is used to vary the behavior of the game in the two game modes, One-Stack-Full and All-Stack-Full.



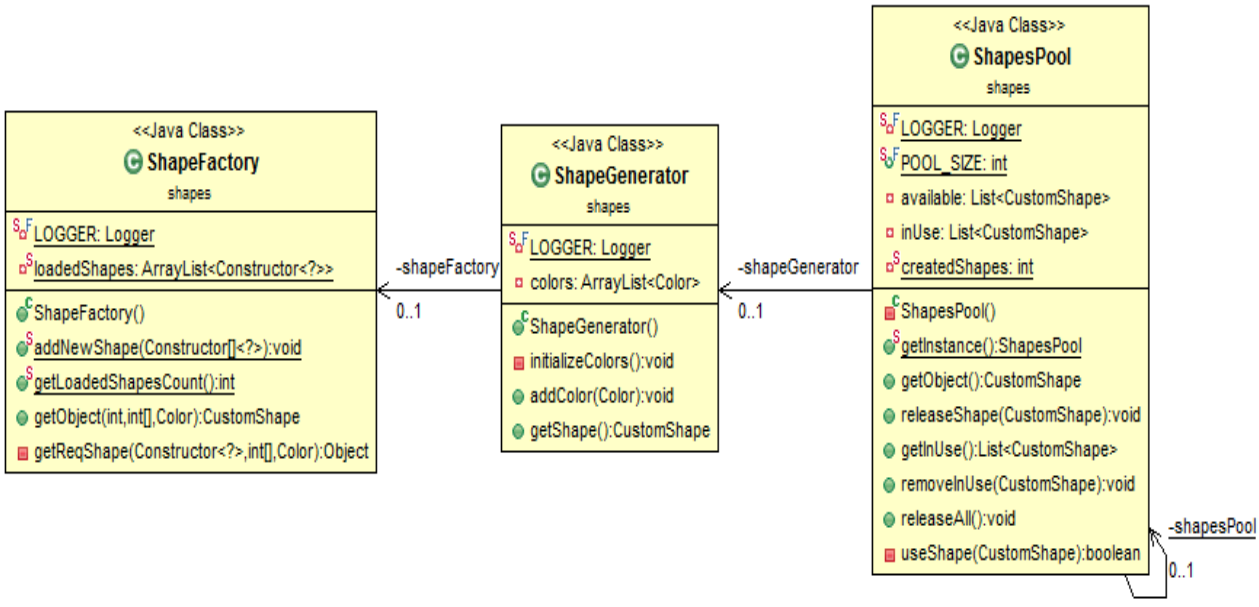
- **Factory Pattern**

  Factory is used to generate the dynamically loaded falling shapes during the run-time.

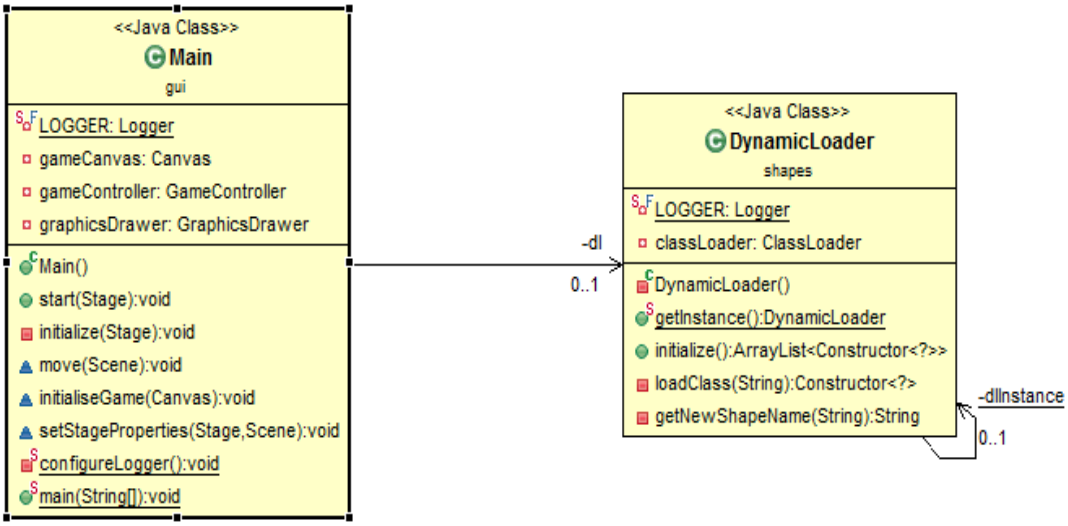


- **Object Pool Pattern**

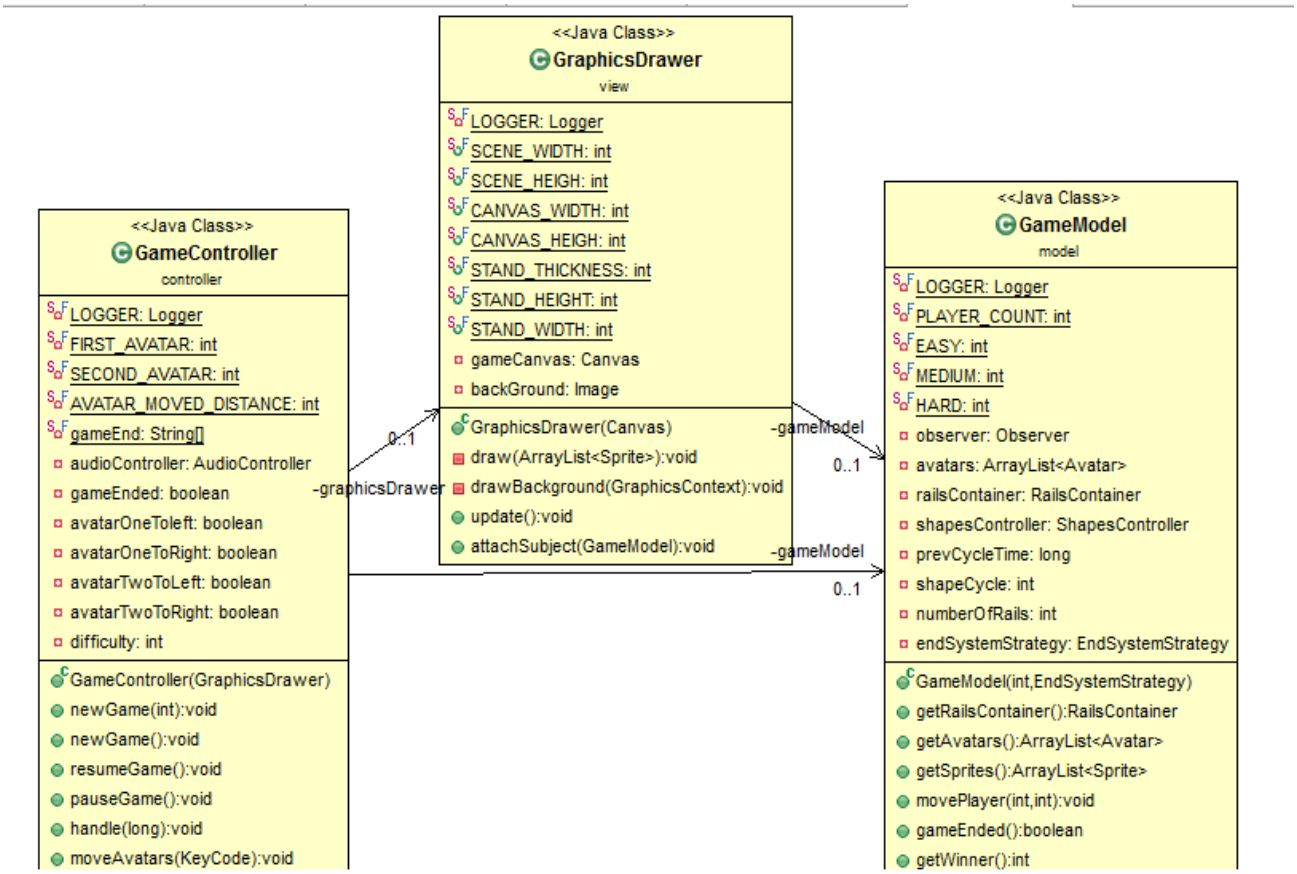
  Object pool is used to manage the creation and usage of the shapes to limit the usage of JavaFX objects since they're heavy objects.



- **Dynamic Linkage Pattern**
- Dynamic linkage is used to dynamically load the falling shapes to be used in the game as long as they're compatible with the defined interface.

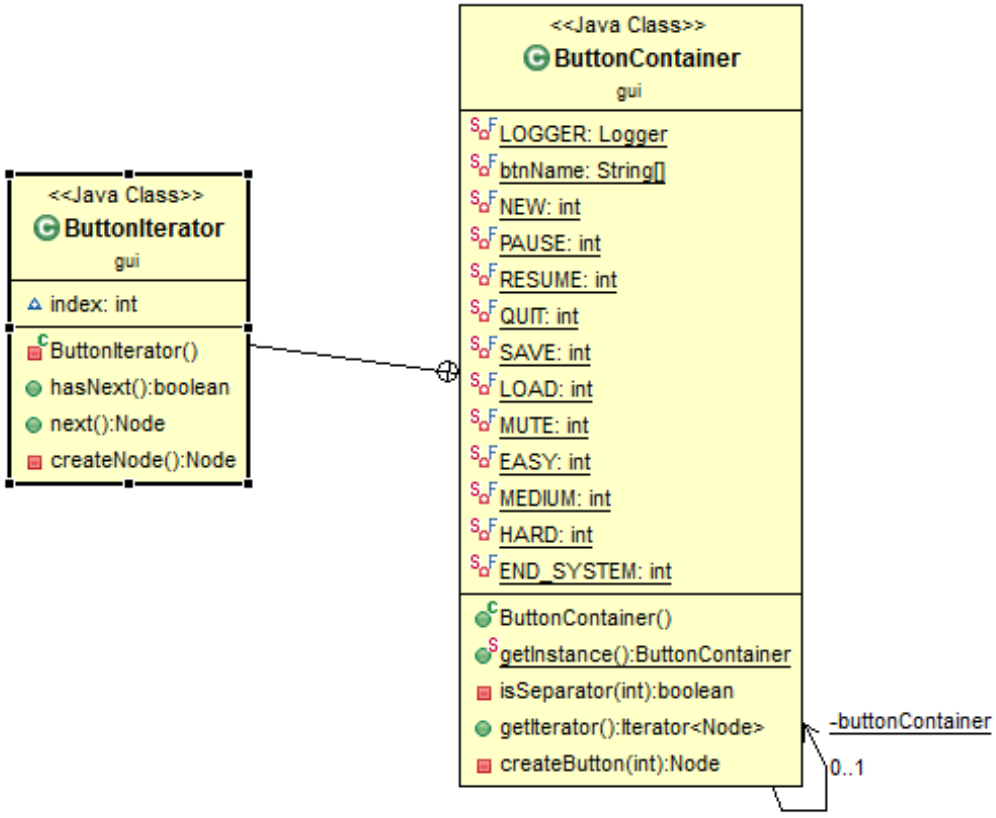


- **Observer Pattern**
  Observer pattern is used to keep the GUI updated with the last results and game stats by notifying the GUI components whenever a change happens in the data to be displayed.

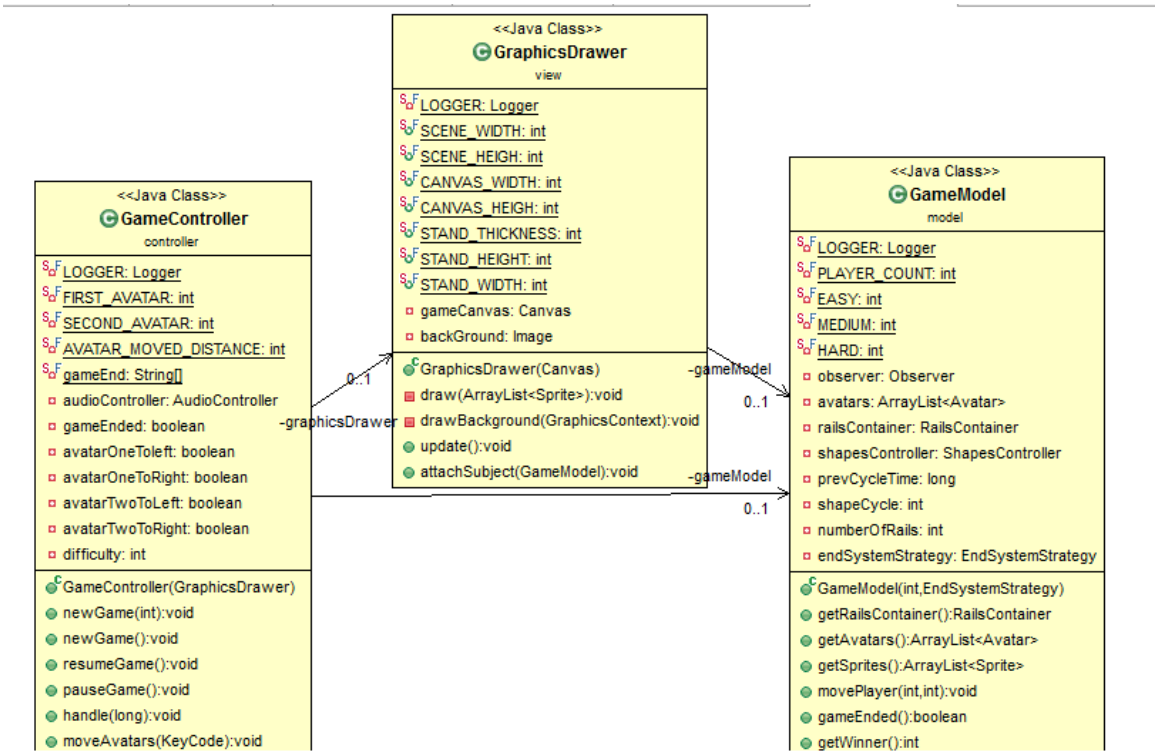


- **Iterator Design Pattern**

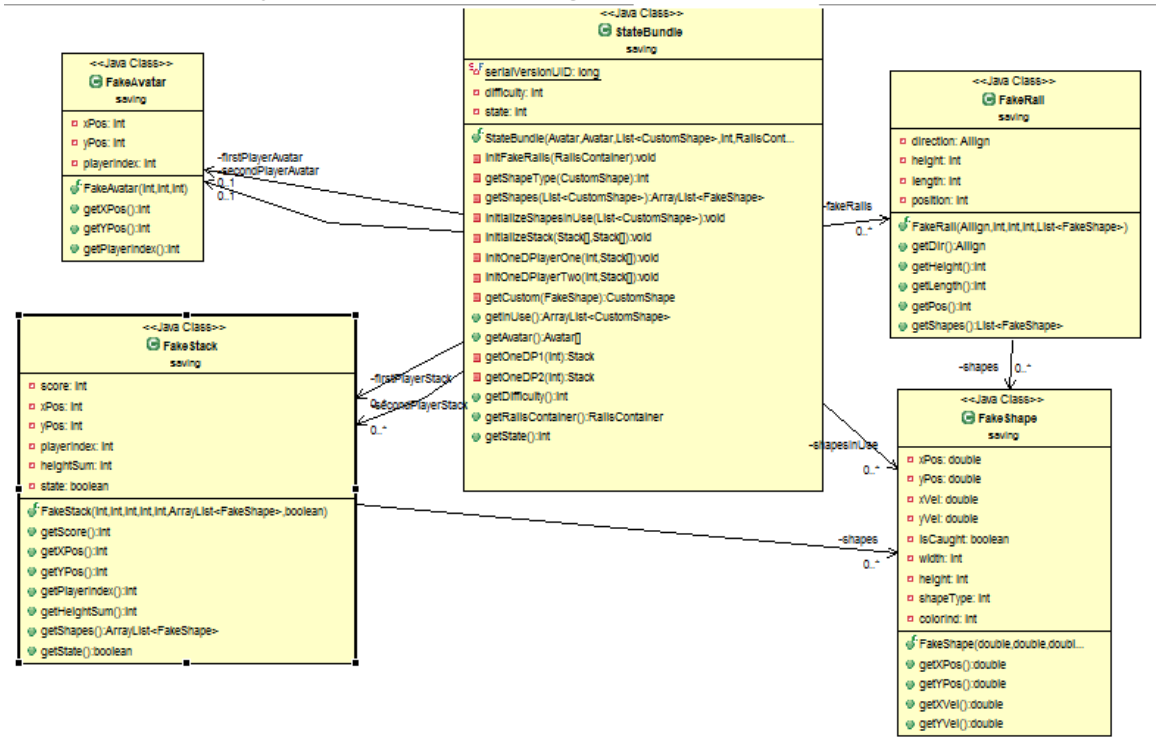
  Iterator is used to iterate through all the buttons used in the GUI to set the user interface.



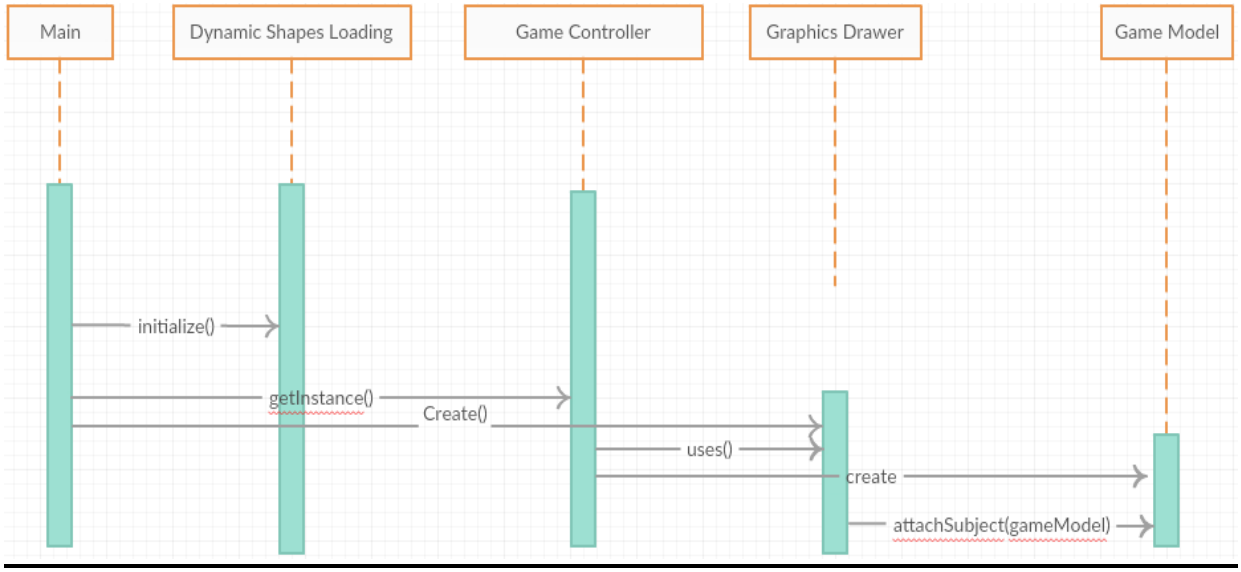
- **MVC design pattern**
  MVC is used to separate the three main components, model which holds the data of the game, view which holds the UI and controller which controls the game logic.

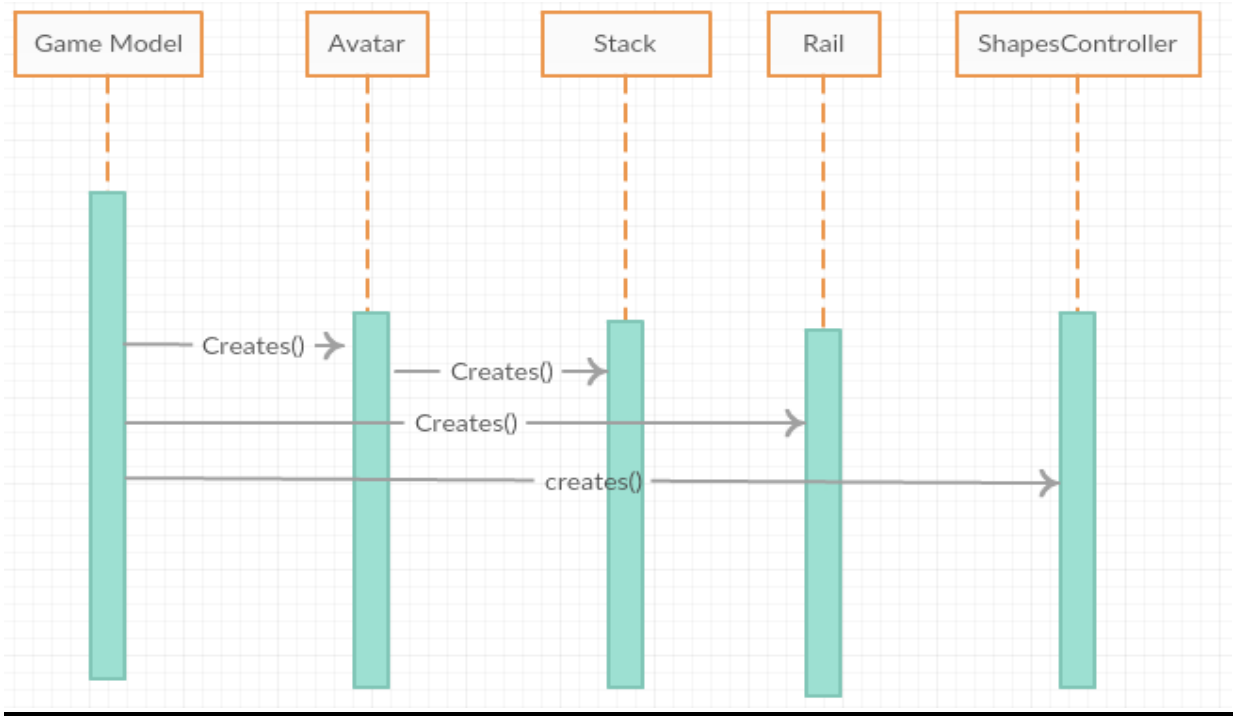


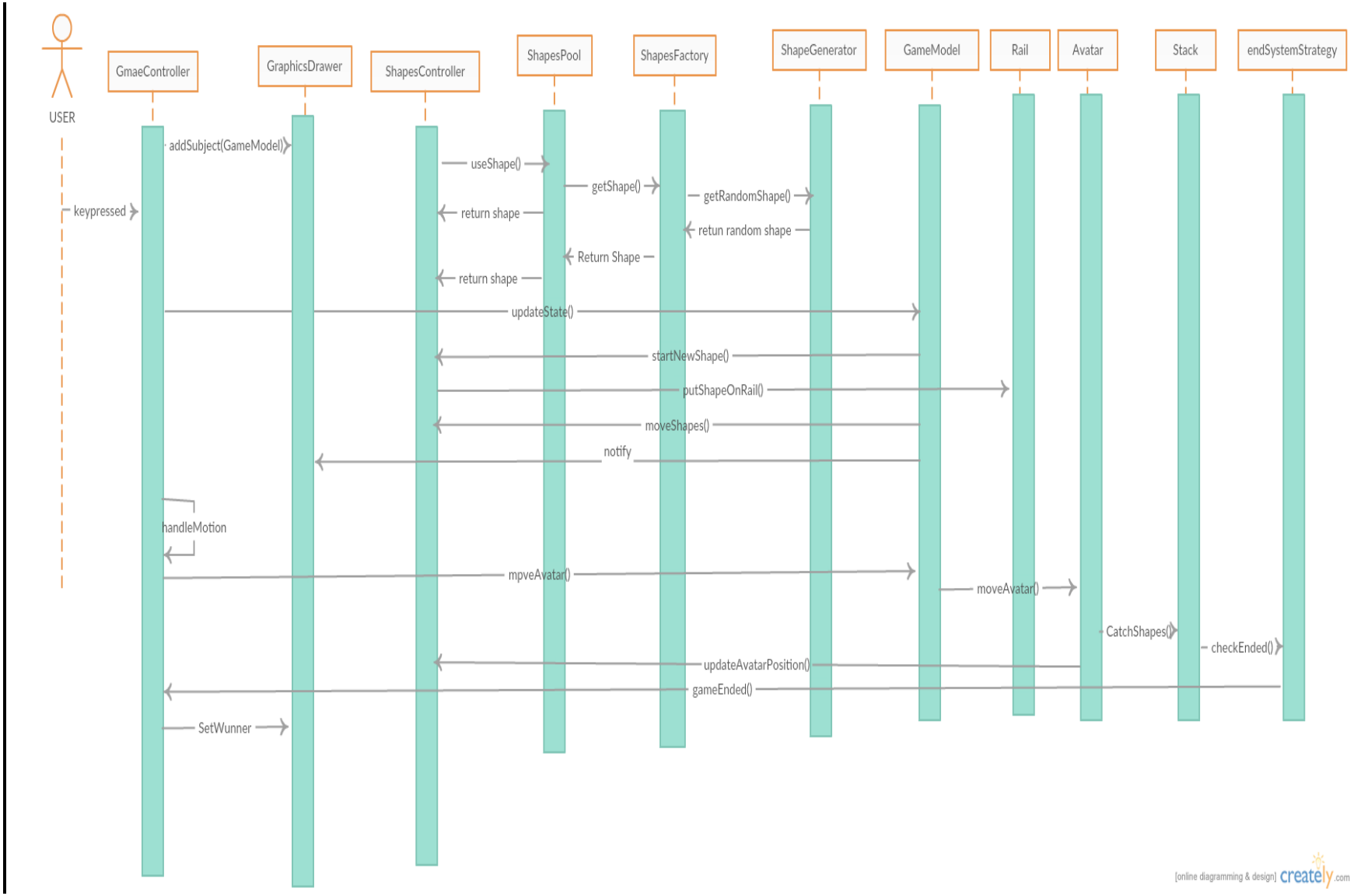
- **Snapshot Design Pattern**
  Snapshot takes a moment (the moment of clicking the save button) and returns an object to be saved using JSON.



Sequence Diagram:






USER
GmaeController
GraphicsDrawer
ShapesController
ShapesPool
ShapesFactory
ShapeGenerator
GameModel
Rail
Avatar
Stack
endSystemStrategy
addSubject(GameModel)
keypressed
useShape()
getShape()
getRandomShape()
return shape
retun random shape
Return Shape
return shape
updateState()
startNewShape()
putShapeOnRail()
moveShapes()
notify
handleMotion
mpveAvatar()
moveAvatar()
CatchShapes()
checkEnded()
updateAvatarPosition()
gameEnded()
SetWunner
[online diagramming & design] creately.com

User Guide:

The game has a friendly user interface that starts the game directly with two characters, Naruto and Hinata.

Player one, modelled as Hinata, is controlled through Left and Right arrows while player two, modelled as Naruto, is controlled through ‘A’ and ‘D’ keys.

The game has two modes (End Systems):

- One-Stack-Full: the game ends when any of the two players reaches the maximum height of the stack, in lined with the score label.
- All-Stack-Full: the game ends when all the 4 stacks of both players reach the maximum height of the stack, in lined with the score label.
- The default mode is All-Stack-Full and it requires starting a new game after changing the mode so that the change would take effect.

The game also provides 3 difficulties:

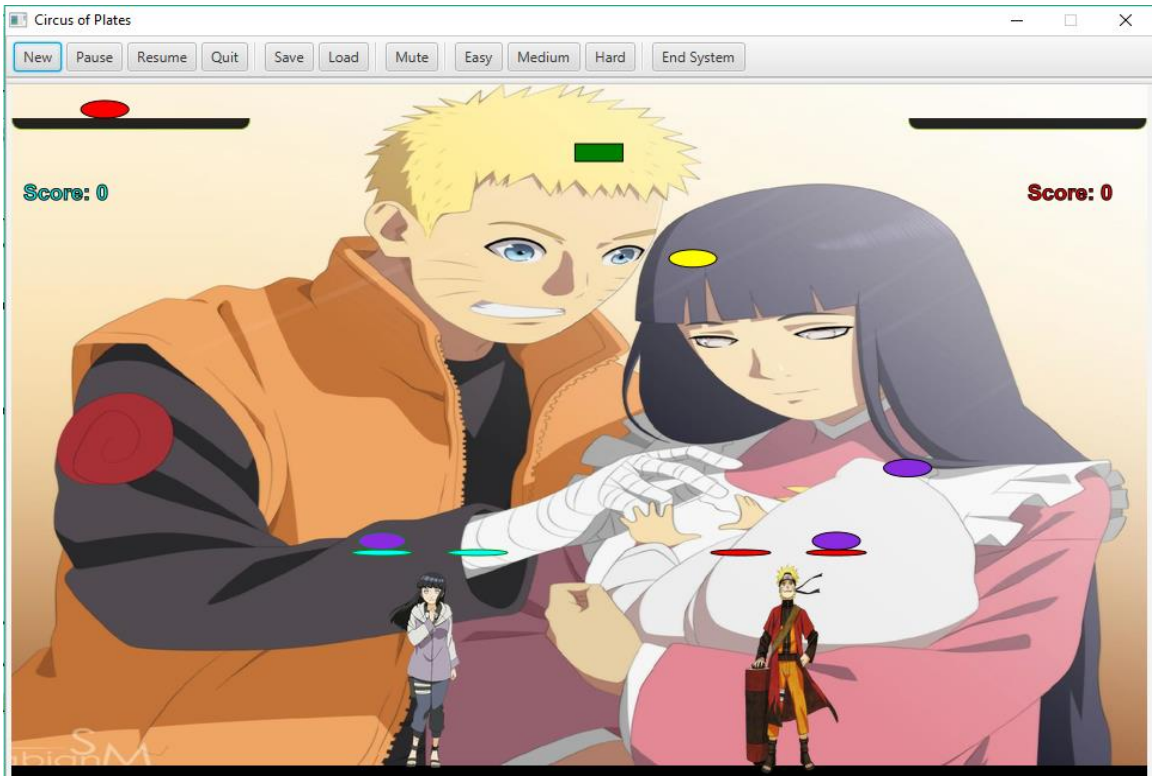
- Easy: There’re two rails and low falling shapes frequency.
- Medium: the falling shapes frequency increases.
- Hard: Two more rails are added and the falling shapes frequency is increased.

After reaching the end of the game the winner is determined as the one who has higher score, else the game ends in tie.

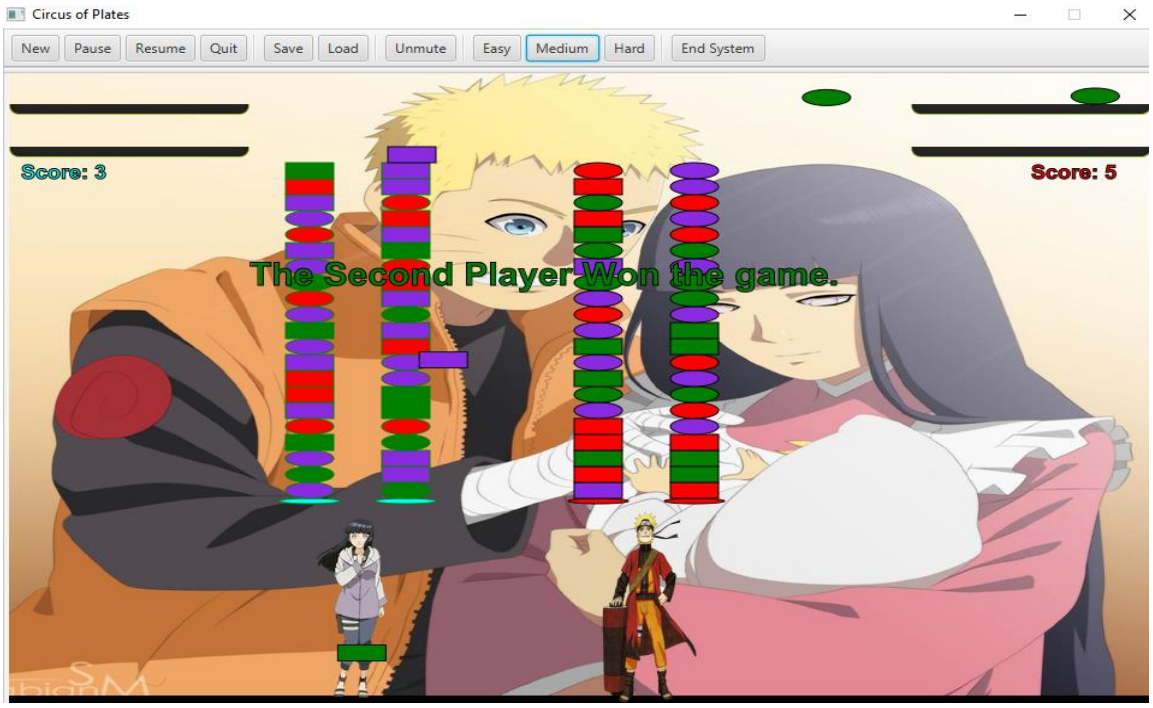
The game support saving and loading.

Snapshots:

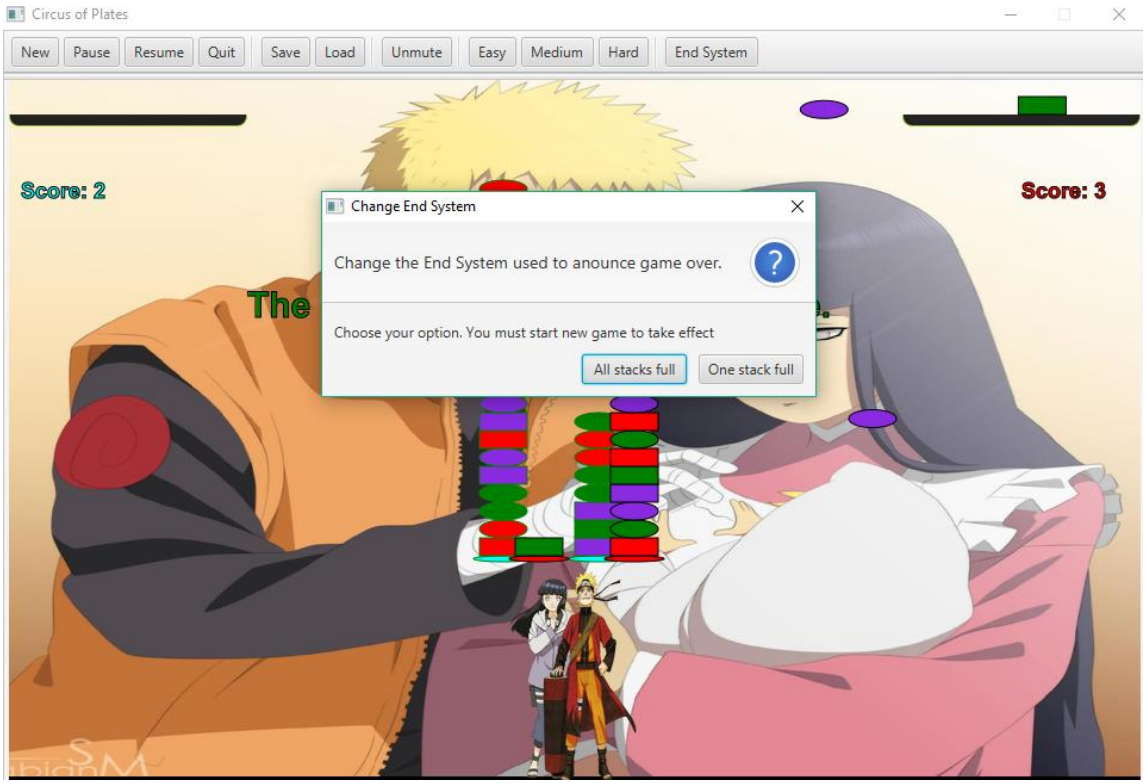
- Sample Run



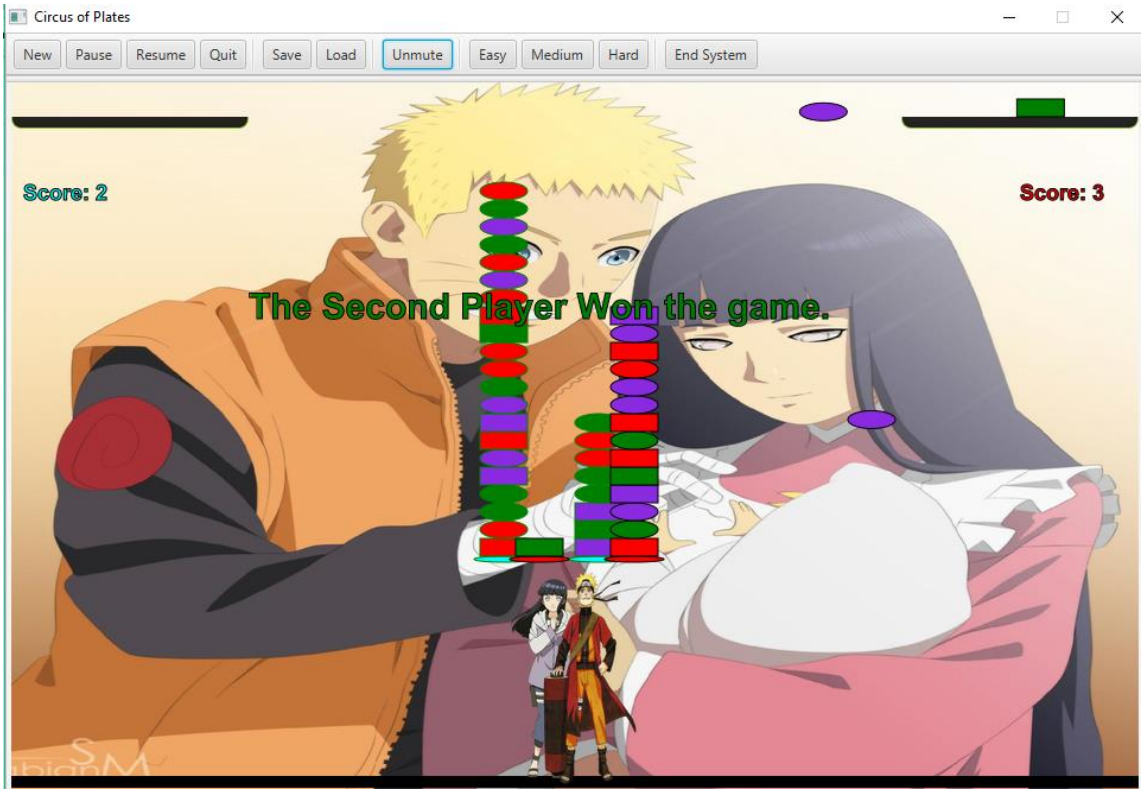
- All-Stack-Full Mode



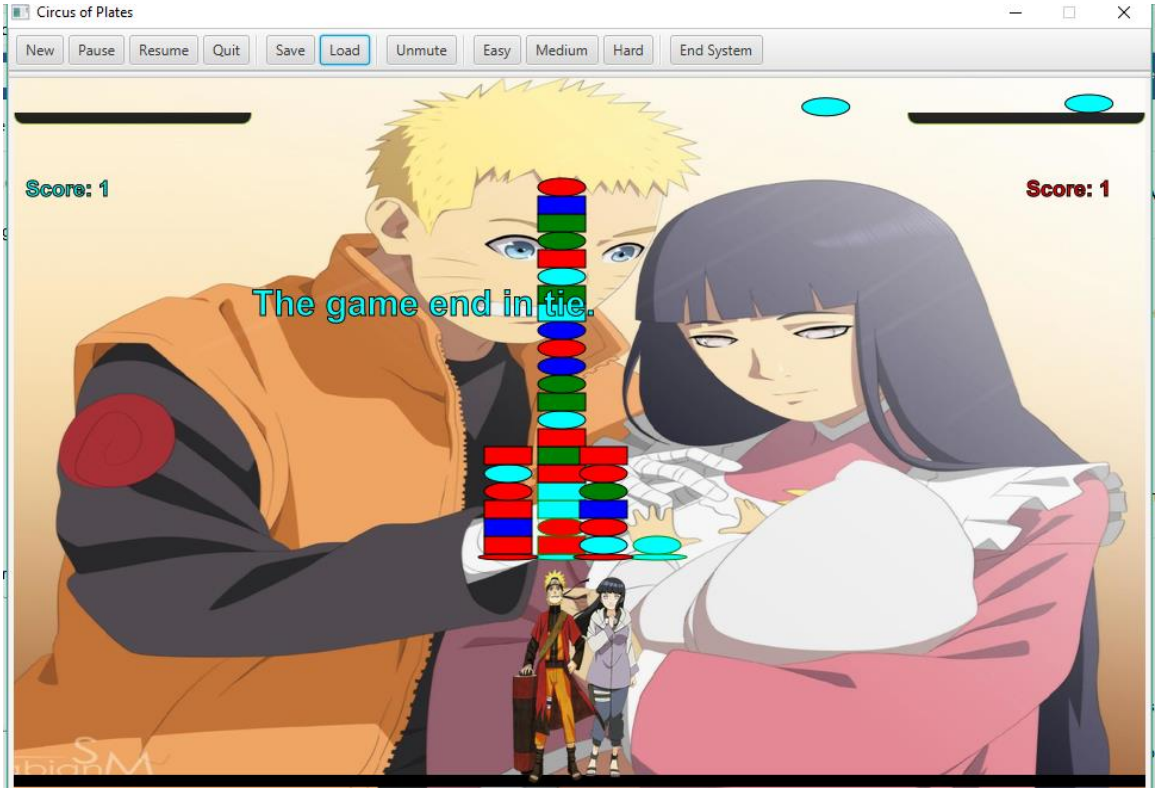
- Change game mode



- One-Stack-Full Mode



- Tie game


Circus of Plates
New
Pause
Resume
Quit
Save
Load
Unmute
Easy
Medium
Hard
End System
Score: 1
Score: 1
The game end in tie.