CS 221 : Programming-2

Circus of Plates Project Report

Names :
Ahmed Moustafa El-Naggar (16)

Muhammed Khamis Ibrahim (67)

Hisham Osama Ghazy (81)

Introduction

- Circus of plates is a two player-game (one may use keyboard, and the other uses the mouse) in which each clown carry two stacks of plates, and there are a set of colored plates queues that end up falling down. 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. You are free to put rules ending the game.

Design

- The Game design was mainly divided into 3 main components through the usage of the MVC design pattern : Model, View, and Controller.
- The Model contains the raw Classes to be used the controller during the game.
    - Shape : abstract class representing any shape to be used instead of plates. The Shapes are dynamically loaded through dynamic loading. Any class that extends the Shape class will be enlisted as a shape to be used

as a plate. There must be an image named after each class available in a specific folder.

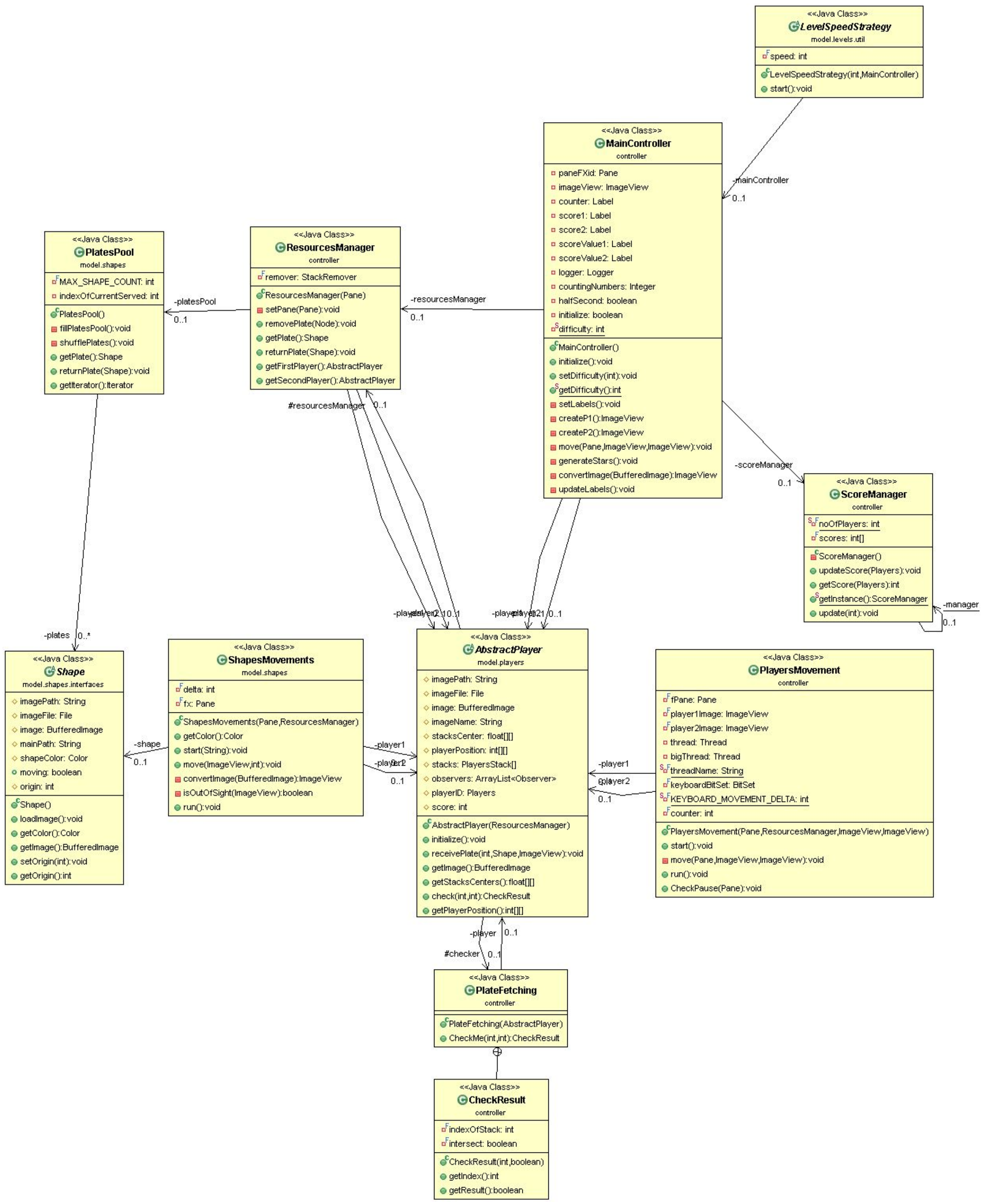
  - AbstractPlayer : abstract class to be used as a representation of the two players (clowns) that will take part in the game.

- Controller contains all class that take part in manipulating the model classes and display them through the View.
  - Main Controller : derives the whole process of GUI manipulation.
  - Resources Manager : manages the resources from the Model to be ready for use by the Main Controller.
  - Score Manager : keeps track of the score changes to be displayed in the GUI.
  - PlayersMovement/ShapesMovement : contains threads that control the movement of shapes in parallel to other processes.

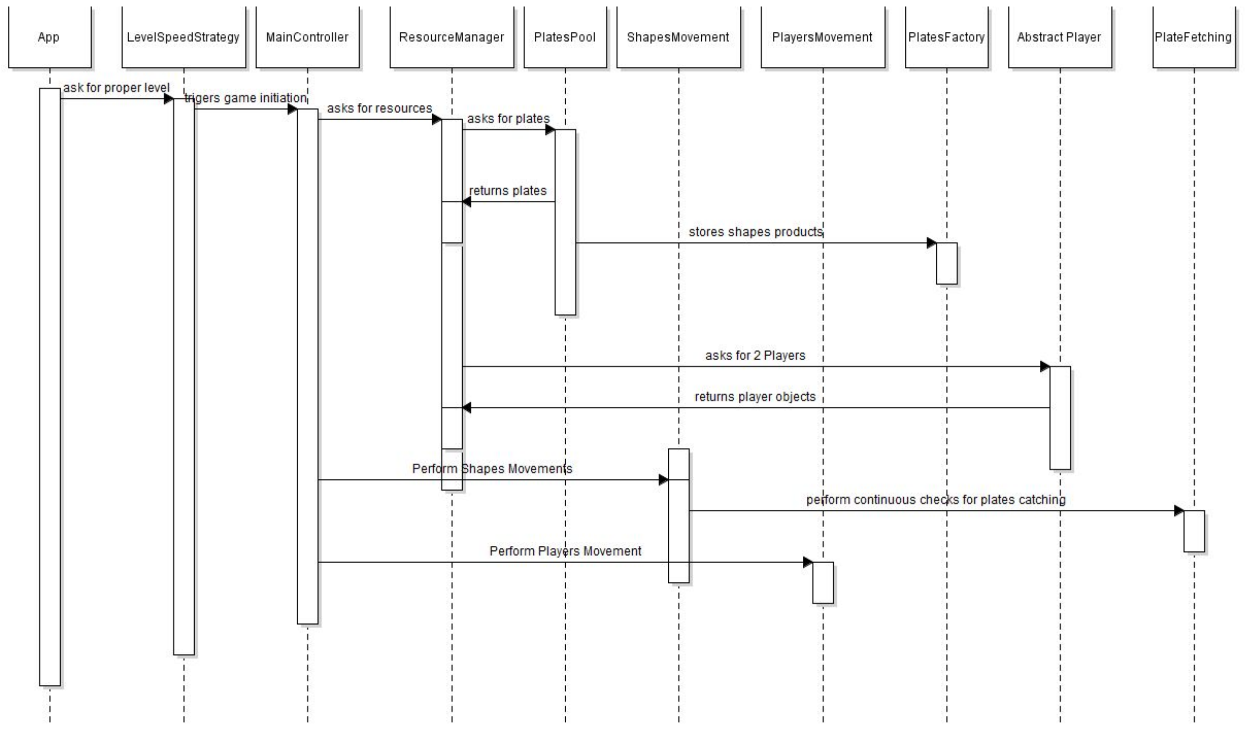
- View contains one class (App) that handles GUI display and start menus.

UML Class Diagrams:

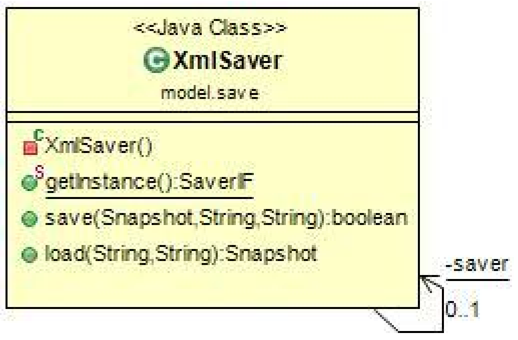
1 - Full Game Design



Game Sequence Diagram:



Design Patterns

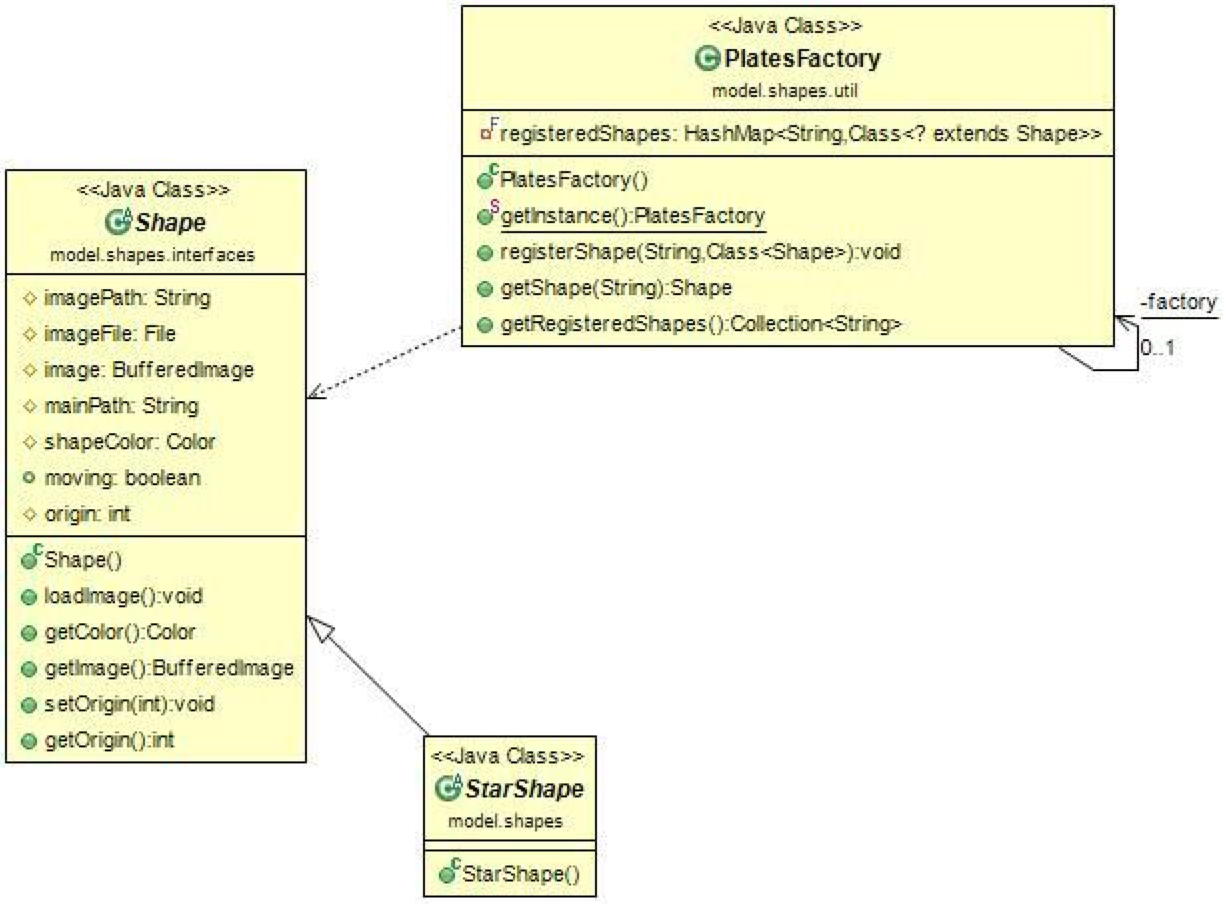


1. Singleton

XML Saver was made using the Singleton design pattern to ensure that only one writer is being used.

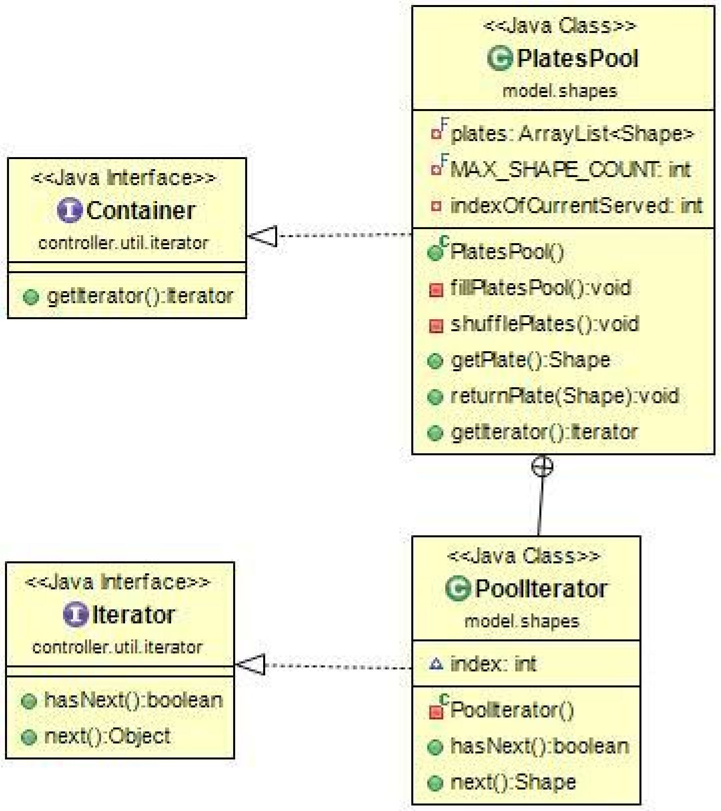
2. Factory

Plates Factory was designed using the Factory design pattern to systemize the production of plates.



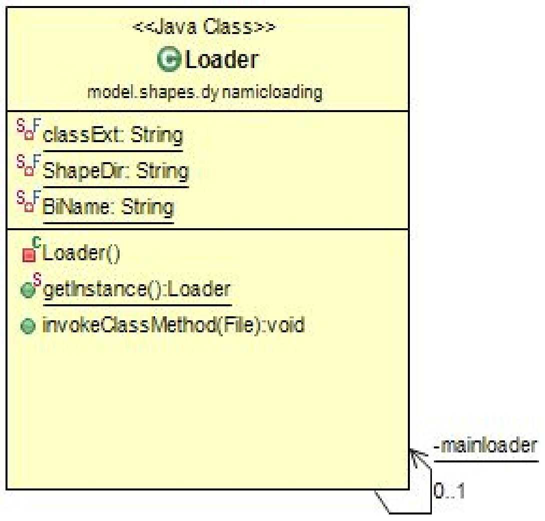
3. Iterator

Iterator design pattern was used to iterate on the plates available in the plates pool.



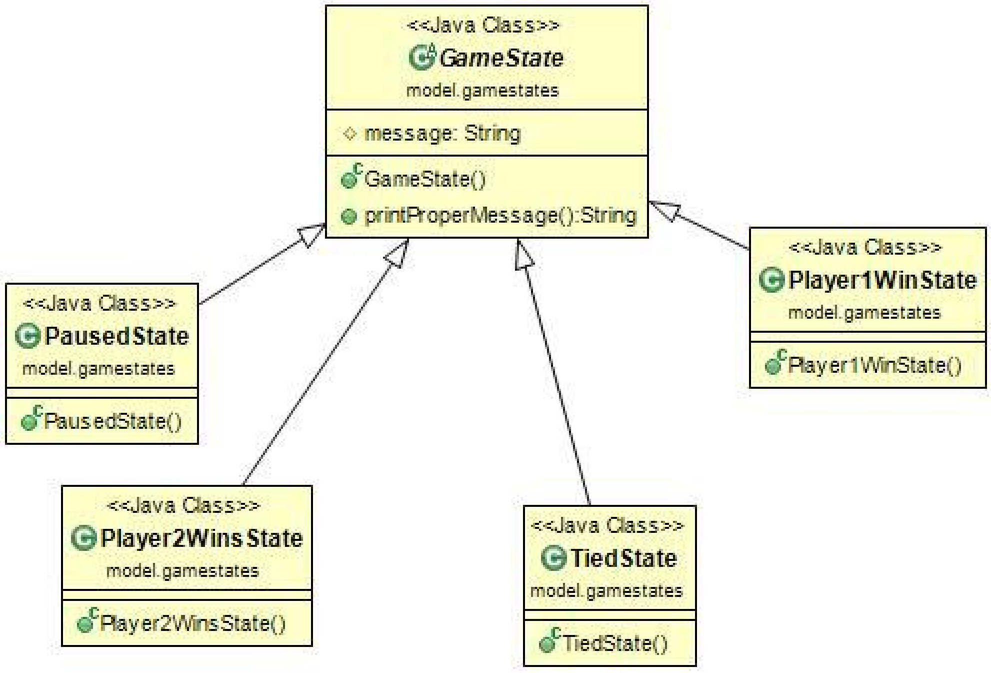
4. Dynamic Linkage

Dynamic Linkage pattern was used to load shapes from external files and use them to increase the variety of the plates shapes.



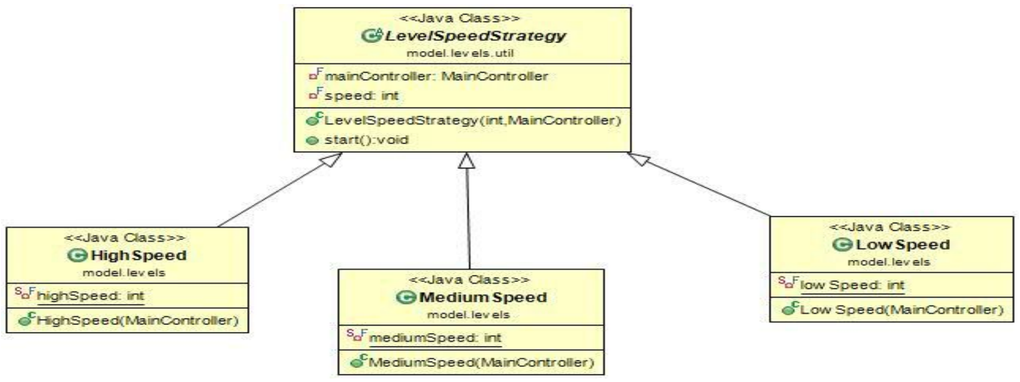
5. State

State design pattern was used to represent game states; (Paused, Player Win, etc.);



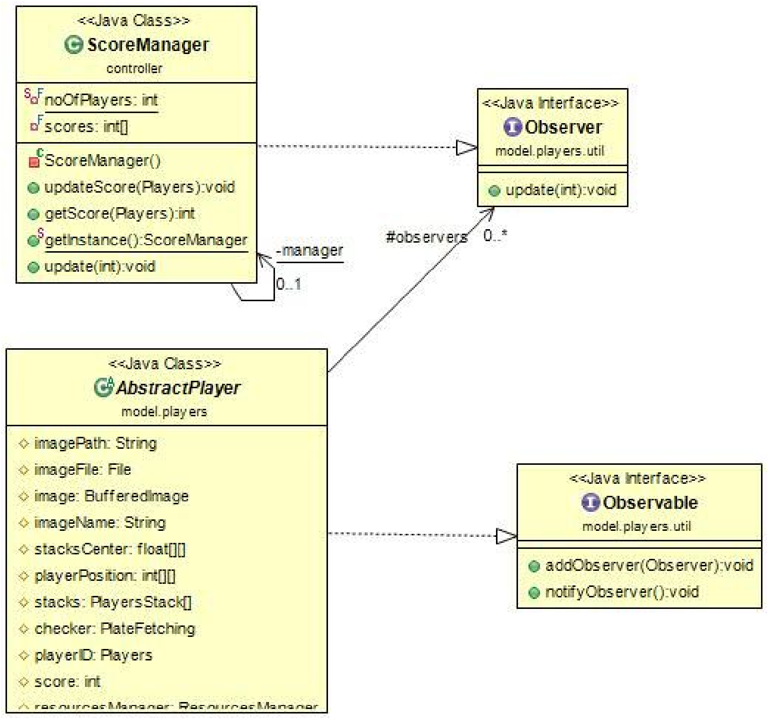
6 . Strategy

To enable multiple game levels (difficulties), Strategy design pattern was used. One main abstract class indicates the main strategy and different levels strategies extend that class.



7. Observer

Observer design pattern was used to build the connection between players and score managers to notify score manager in case of any score update to any of the players.



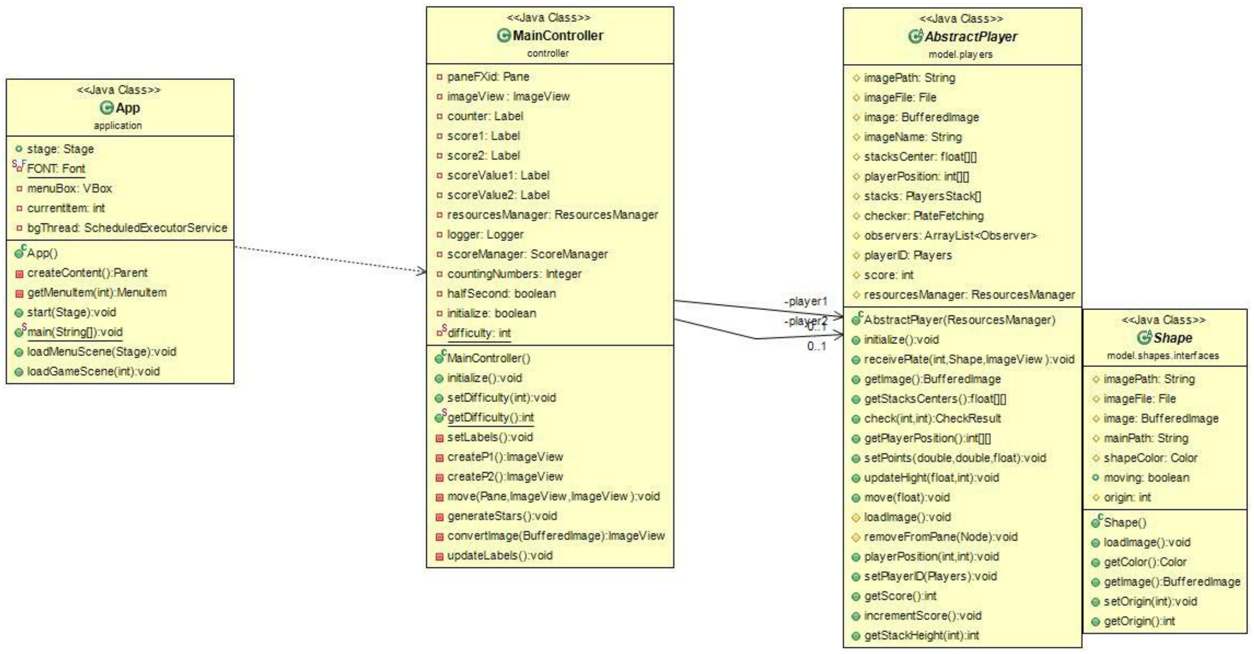
8. Model View Controller (MVC)

MVC is used to separate application's concerns.

Model : represented mainly by the Player and the Shape Classes.

View : represented by the App Class that displays the GUI.

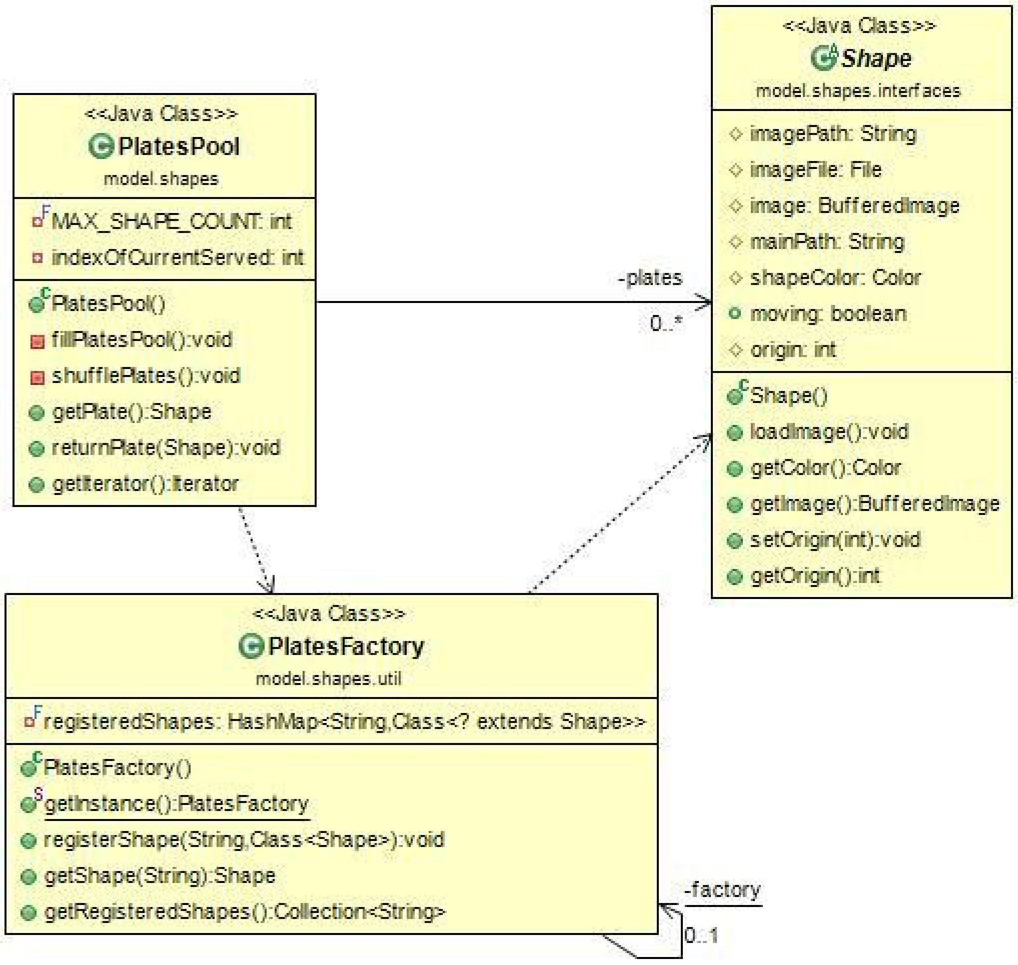
Controller : represented by the MainController and other Helping Classes.



<<Java Class>>
App
application
stage: Stage
FONT: Font
menuBox: VBox
currentItem: int
bgThread: ScheduledExecutorService
App()
createContent():Parent
getMenuItem(int):MenuItem
start(Stage):void
main(String[]):void
loadMenuScene(Stage):void
loadGameScene(int):void
<<Java Class>>
MainController
controller
paneFXid: Pane
imageView : ImageView
counter: Label
score1: Label
score2: Label
scoreValue1: Label
scoreValue2: Label
resourcesManager: ResourcesManager
logger: Logger
scoreManager: ScoreManager
countingNumbers: Integer
halfSecond: boolean
initialize: boolean
difficulty: int
MainController()
initialize():void
setDifficulty(int):void
getDifficulty():int
setLabels():void
createP1():ImageView
createP2():ImageView
move(Pane,ImageView,ImageView):void
generateStars():void
convertImage(BufferedImage):ImageView
updateLabels():void
-player1
0..1
-player2
0..1
<<Java Class>>
AbstractPlayer
model.players
imagePath: String
imageFile: File
image: BufferedImage
imageName: String
stacksCenter: float[][]
playerPosition: int[][]
stacks: PlayersStack[]
checker: PlateFetching
observers: ArrayList<Observer>
playerID: Players
score: int
resourcesManager: ResourcesManager
AbstractPlayer(ResourcesManager)
initialize():void
receivePlate(int,Shape,ImageView):void
getImage():BufferedImage
getStacksCenters():float[][]
check(int,int):CheckResult
getPlayerPosition():int[][]
setPoints(double,double,float):void
updateHight(float,int):void
move(float):void
loadImage():void
removeFromPane(Node):void
playerPosition(int,int):void
setPlayerID(Players):void
getScore():int
incrementScore():void
getStackHeight(int):int
<<Java Class>>
Shape
model.shapes.interfaces
imagePath: String
imageFile: File
image: BufferedImage
mainPath: String
shapeColor: Color
moving: boolean
origin: int
Shape()
loadImage():void
getColor():Color
getImage():BufferedImage
setOrigin(int):void
getOrigin():int

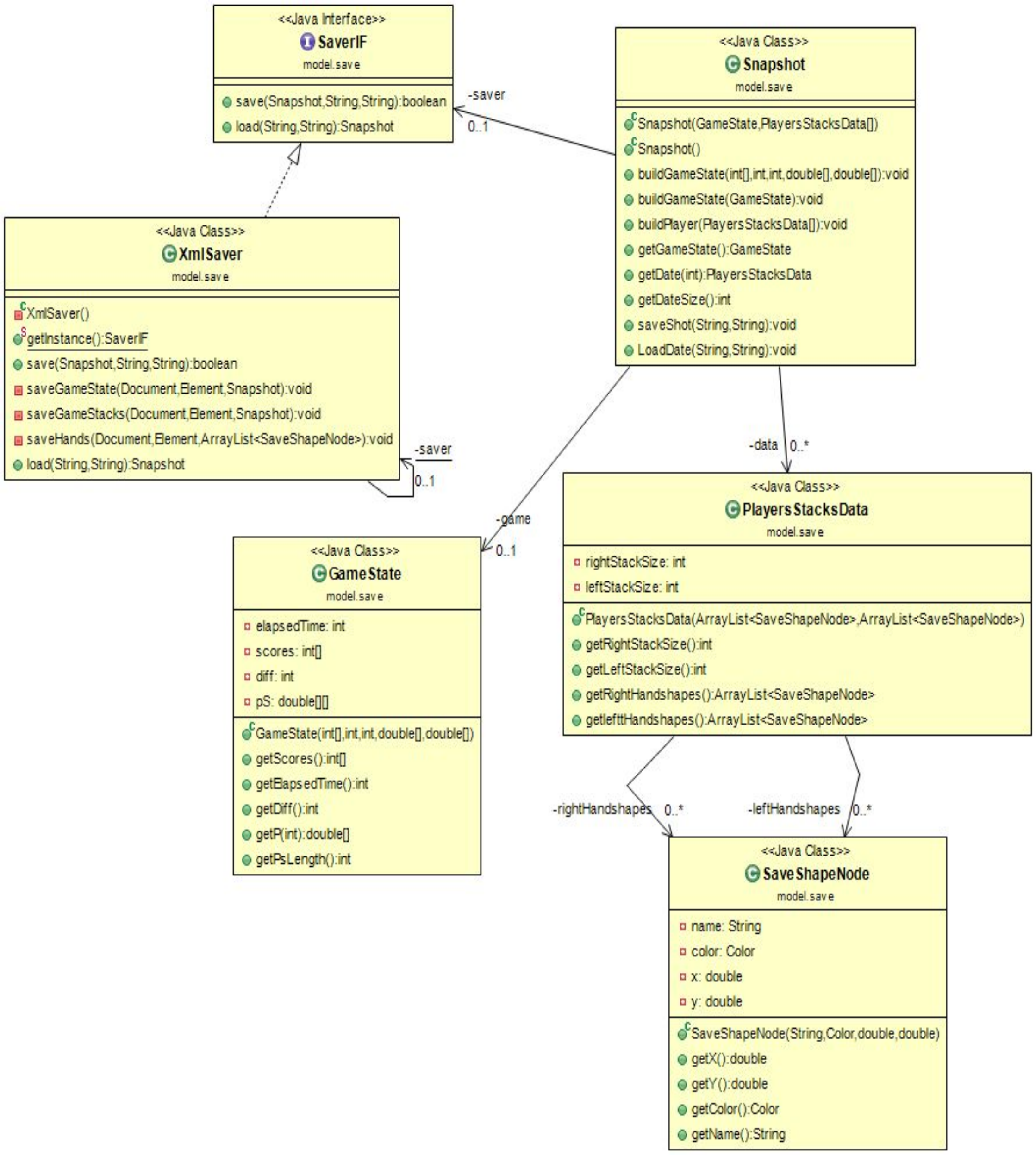
9. Object Pool

Pool design pattern was used to manage the pool of plates displayed in the GUI in cooperation with the Factory Class.



10. Snapshot

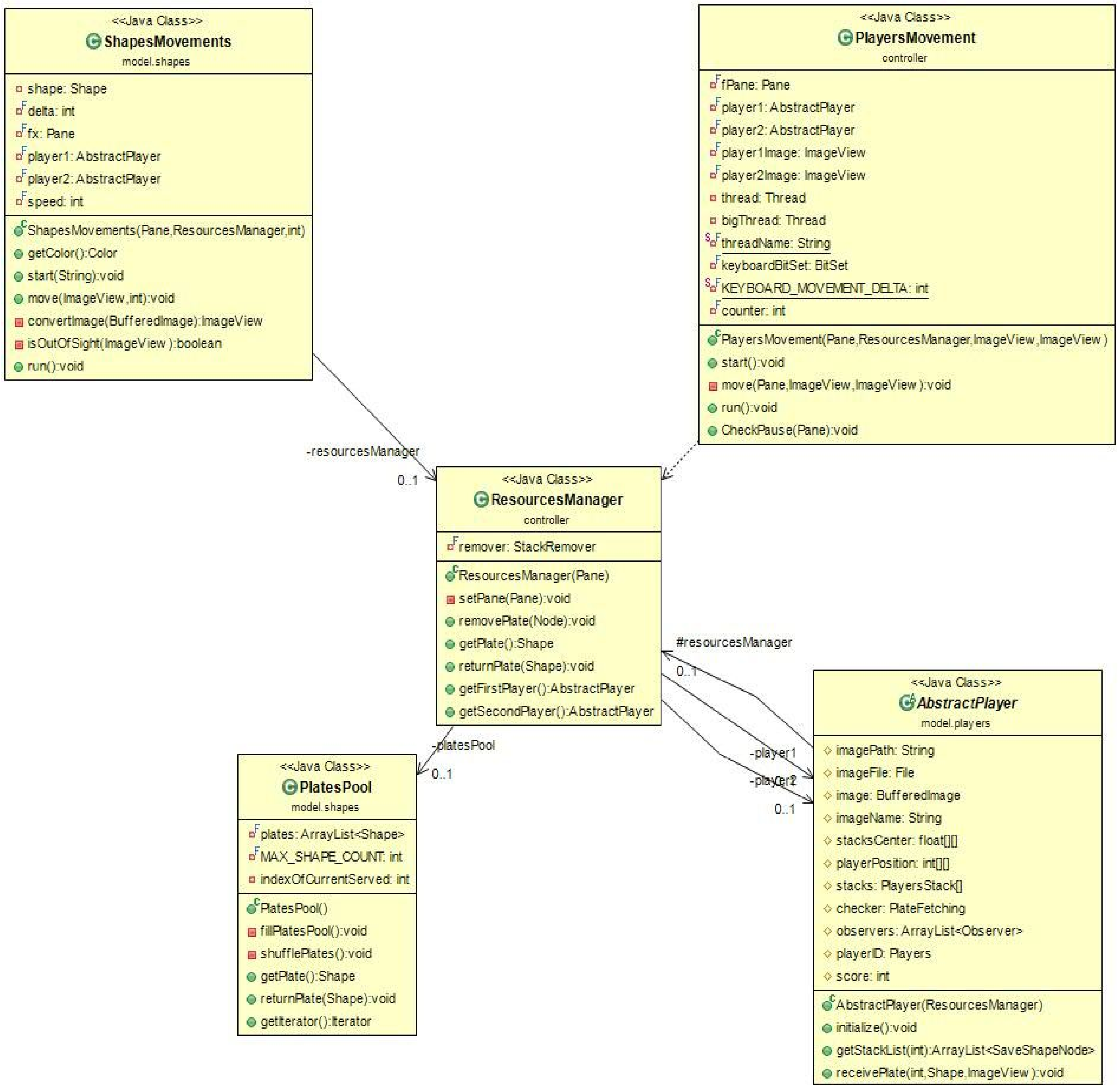
Snapshot design pattern was used to save and load the game.



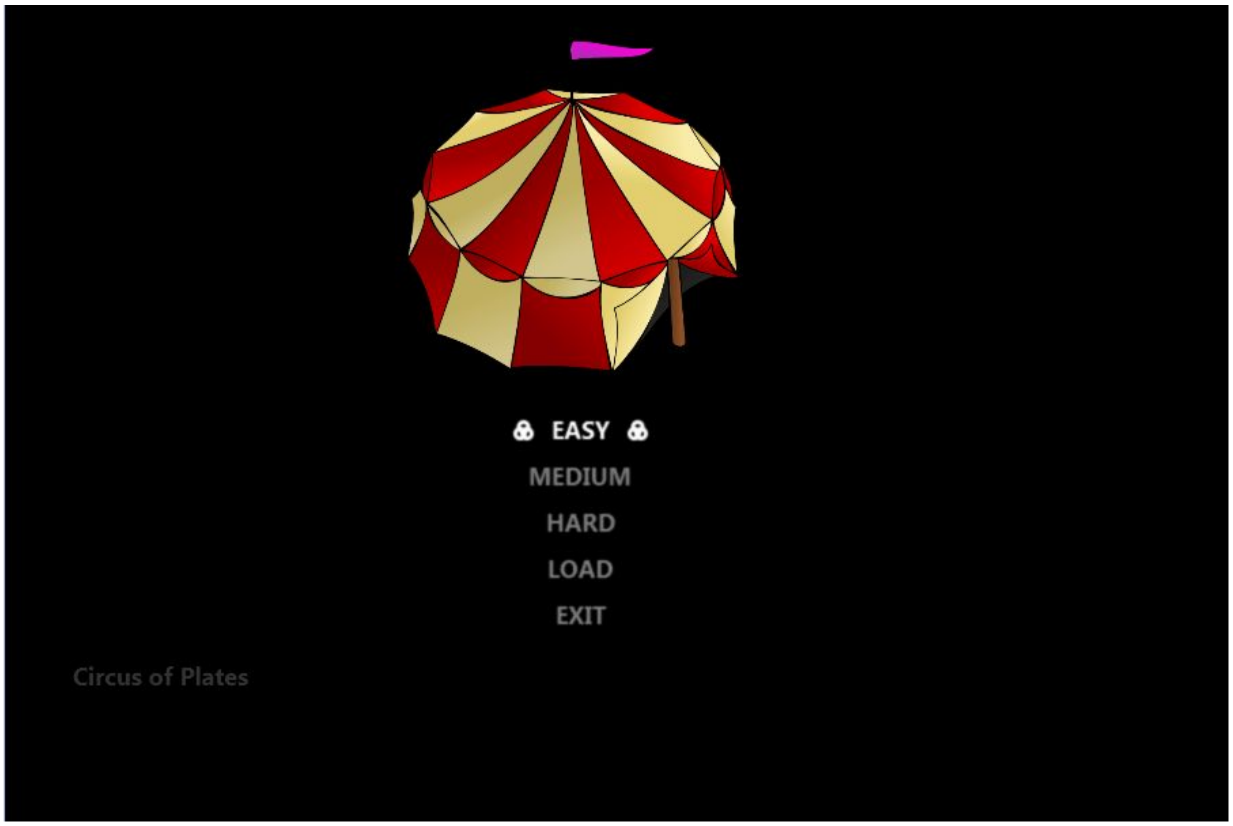
<<Java Interface>>
SaverIF
model.save
save(Snapshot,String,String):boolean
load(String,String):Snapshot
<<Java Class>>
Snapshot
model.save
Snapshot(GameState,PlayersStacksData[])
Snapshot()
buildGameState(int[],int,int,double[],double[]):void
buildGameState(GameState):void
buildPlayer(PlayersStacksData[]):void
getGameState():GameState
getDate(int):PlayersStacksData
getDateSize():int
saveShot(String,String):void
LoadDate(String,String):void
-saver
0..1
<<Java Class>>
XmlSaver
model.save
XmlSaver()
getInstance():SaverIF
save(Snapshot,String,String):boolean
saveGameState(Document,Element,Snapshot):void
saveGameStacks(Document,Element,Snapshot):void
saveHands(Document,Element,ArrayList<SaveShapeNode>):void
load(String,String):Snapshot
-saver
0..1
-data
0..*
-game
0..1
<<Java Class>>
GameState
model.save
elapsedTime: int
scores: int[]
diff: int
pS: double[][]
GameState(int[],int,int,double[],double[])
getScores():int[]
getElapsedTime():int
getDiff():int
getP(int):double[]
getPsLength():int
<<Java Class>>
PlayersStacksData
model.save
rightStackSize: int
leftStackSize: int
PlayersStacksData(ArrayList<SaveShapeNode>,ArrayList<SaveShapeNode>)
getRightStackSize():int
getLeftStackSize():int
getRightHandshapes():ArrayList<SaveShapeNode>
getlefttHandshapes():ArrayList<SaveShapeNode>
-rightHandshapes
0..*
-leftHandshapes
0..*
<<Java Class>>
SaveShapeNode
model.save
name: String
color: Color
x: double
y: double
SaveShapeNode(String,Color,double,double)
getX():double
getY():double
getColor():Color
getName():String

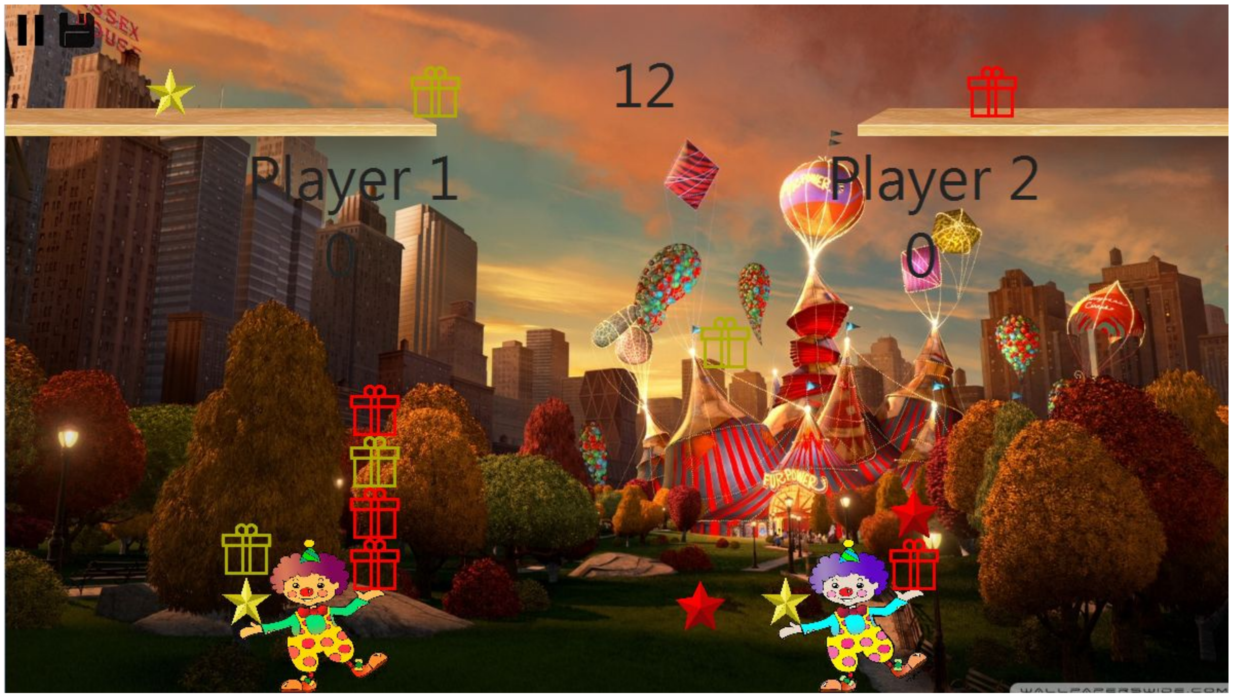
11. Facade (Additional)

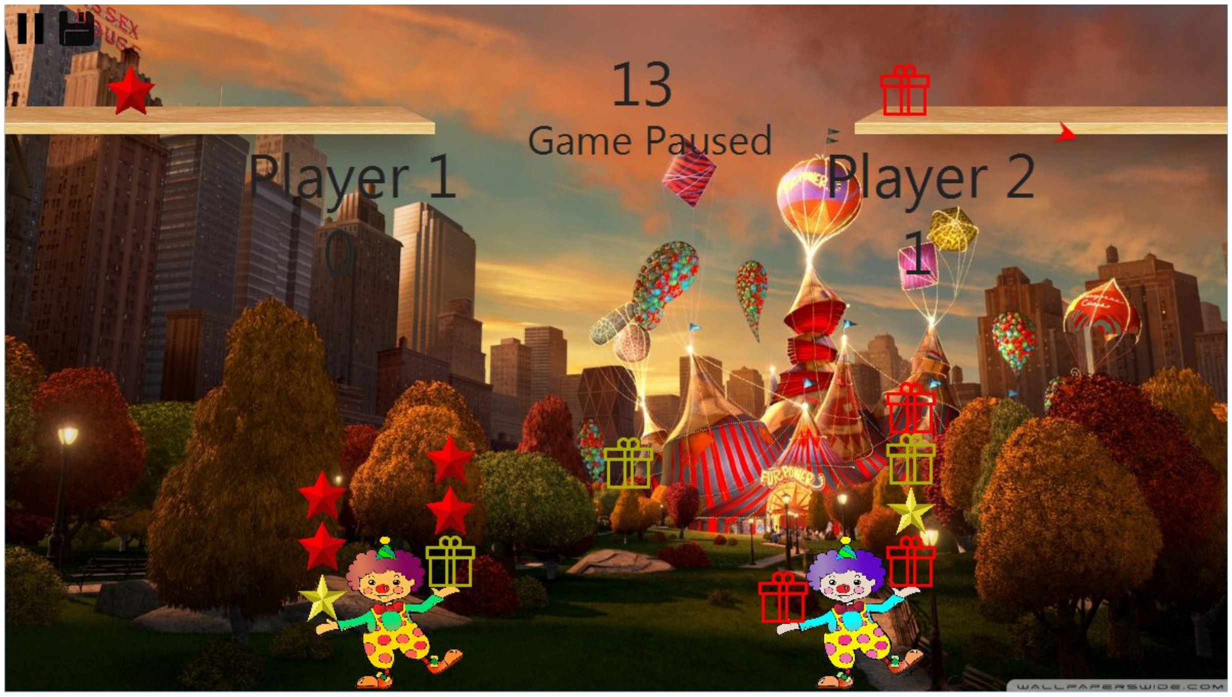
Facade design pattern was used in the Resources Manager Class to facilitate handling of multiple requests of resources in one class. Requests of plates or players or other services.

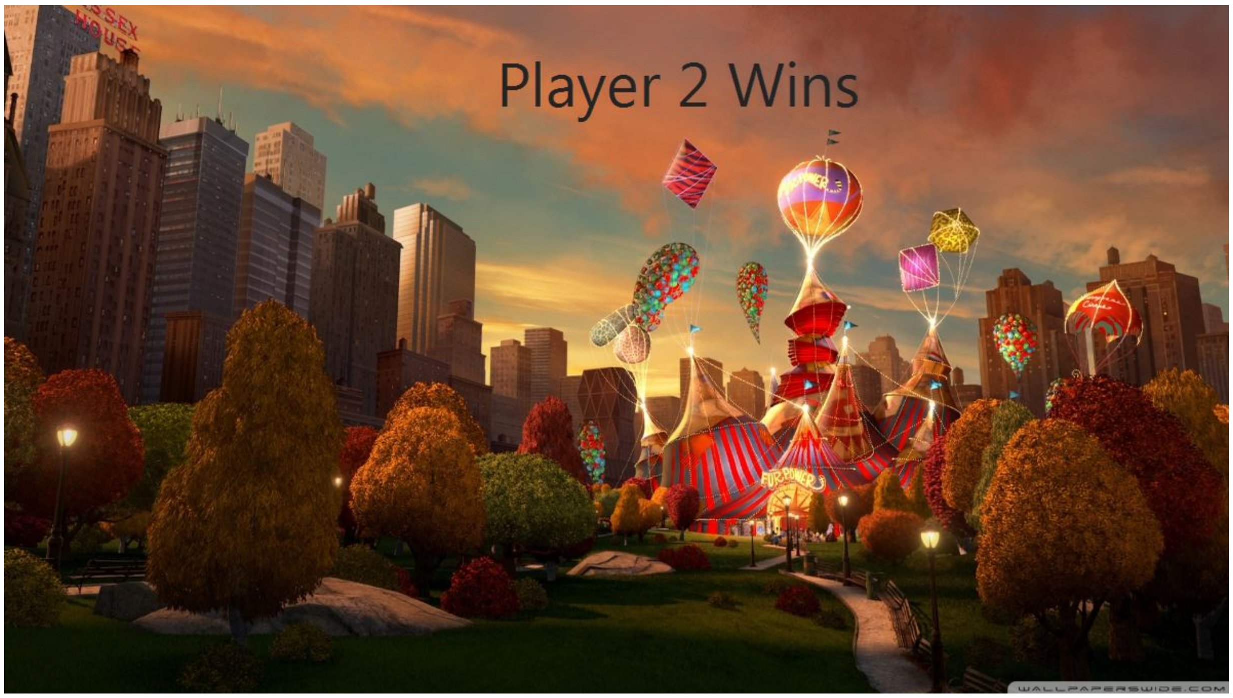


Snapshots:


EASY
MEDIUM
HARD
LOAD
EXIT
Circus of Plates


12
Player 1
0
Player 2
0


13
Game Paused
Player 1
0
Player 2
1


Player 2 Wins

User guide:

1. On running the application, you are allowed to choose one of the three main options:
    - Selecting a new game on easy, medium or hard mode
    - Loading previously saved game
    - Closing the application
2. Selecting or Loading a game will lead you to the main game scene where you and your competitor will start playing as follows:
    - Player one will take control of the first clown placed on right side of the screen and start moving it using “Right” and “Left” keys of the keyboard
    - Player two will take control of the second clown placed on left side of the screen and start moving it using “D” and “A” keys of the keyboard
    - Shapes of different colors will start to fall from upper part of the screen and you should manage to collect shapes using the two hands of your clown during 60 seconds of time
    - You’ll get a point whenever you collect 3 shapes of the same color and these shapes will be removed from your stack
3. During the game you can click on the 2 buttons provided on the screen:
    - Save Button : this will the save the game at the moment of pressing save button to be loaded later

        You can just press “S” button from the keyboard to save the game

    - Pause Button : this will freeze the game, clicking this button again will unfreeze the game and you’ll continue playing

        You can just press “P” button from the keyboard to pause the game

4. After 60 seconds the game will be over and a screen displaying the result will be shown depending on the number of points of each player

Design Decisions:

- For the addition of any shape, a class that extends the Shape abstract class should be available as a JAR file and an image with the same name as the Class should be added to a specified images folder.
- Difficulties vary as plates speed varies.