# <u>Circus of Plates - Game</u>

**Prepared by:**

**Youssef Ahmed Ibrahem 87**

**Mohamed said Ibrahem 68**

# Design Patterns&Description

**Description:**

The game starts with the main frame "GameFrame" which initializes the frame and makes Object from the "MainMenu".

there you can choose from different options (new game , high scores , load page , exit the game).

New game option opens the themes menu where you can choose one theme out of 3.once you choose it you go throw a small popup message that asks  you to input players names.

then the actual game begins ,shapes are loaded and timer starts to count. as you go through the game you will counter 3 levels of difficulties as time progresses , where number of colors increase, the bars move and the queue gets crowded.

the game ends when at least one of the players get his two stacks (left and right) full so he can't get more shapes.

a game over screen appears with a label that announces the winner with his score . if the score was high it's added to the high scores file.

finally you have to options there: exit the game , or play another game .

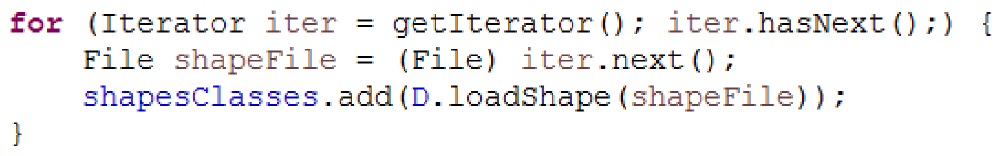
**Singleton**: There are more menus in the game all of them use Singleton so that when you navigate through them you're using the objects that are lastly created without create multiple menus every time. Additionally the game main view "Circus" is also using Singleton, so that you're in one circus and never create a new one unless you choose a "new game" button.

**ObjectPool**: Shapes that are falling on the ground are being reused again instead if creating new ones, so when we want to create a shape we address the request to the pool first , if it has reusable shapes we get them otherwise we order the factory to get them.

**Factory**: as we mention above in the "ObjectPool" section. When we create a shape we "order" it from the factory "ShapesFactory" that has the shapes collection and a function "orderShape()"to get a random shape from them and then set its values (i.e coloe,position .. ) and return a shape.

**Iterator**: We have a collection of Data which are Classes of shapes in an external directory with unknown representation, so we used Iterator to iterate through those files one by one and loading them dynamically.

"ShapesIterator" is the class that implements the "Iterator" interface. It's an inner class in the "shapes collection" that implements "container" to get an Iterator for the processes of gathering data.



```
for (Iterator iter = getIterator(); iter.hasNext();) {
    File shapeFile = (File) iter.next();
    shapesClasses.add(D.loadShape(shapeFile));
}
```

**Dynamic Loading**: in order to make the game more dynamic there is an abstract class " shapes" that all shapes inherit from , so if any shape follows the class abstraction can be loaded dynamically to the game as a valid shape that we can use and manipulate its values,
So the shapes classes are dynamically loaded at the start
Of the execution from a specific folder.

**State**: in this game we consider that the shape has two states share a common interface that has one method "move " which defines the way a shape moves according to its state ,which is either "OnBar" which means it's moving horizontally on a bar. Or "falling" which means it's outside the bar and ready to be caught or destroyed.

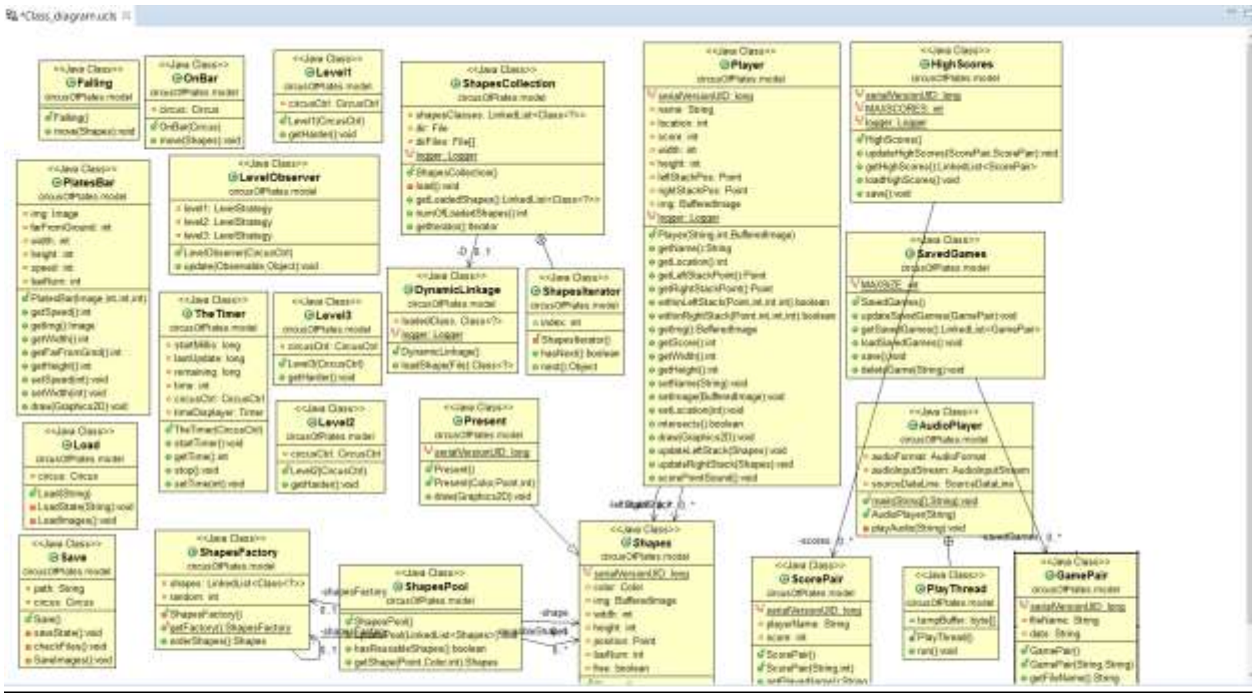
**Strategy:** strategy pattern is used when there is a common behavior but different implementations,
Here we have 3 level of difficulties in the game hence 3 strategies for each (level1,level2,level3) , all of them share a common interface "levelStrategy" that has a method "getHarder()" which leverage the level of difficulty.

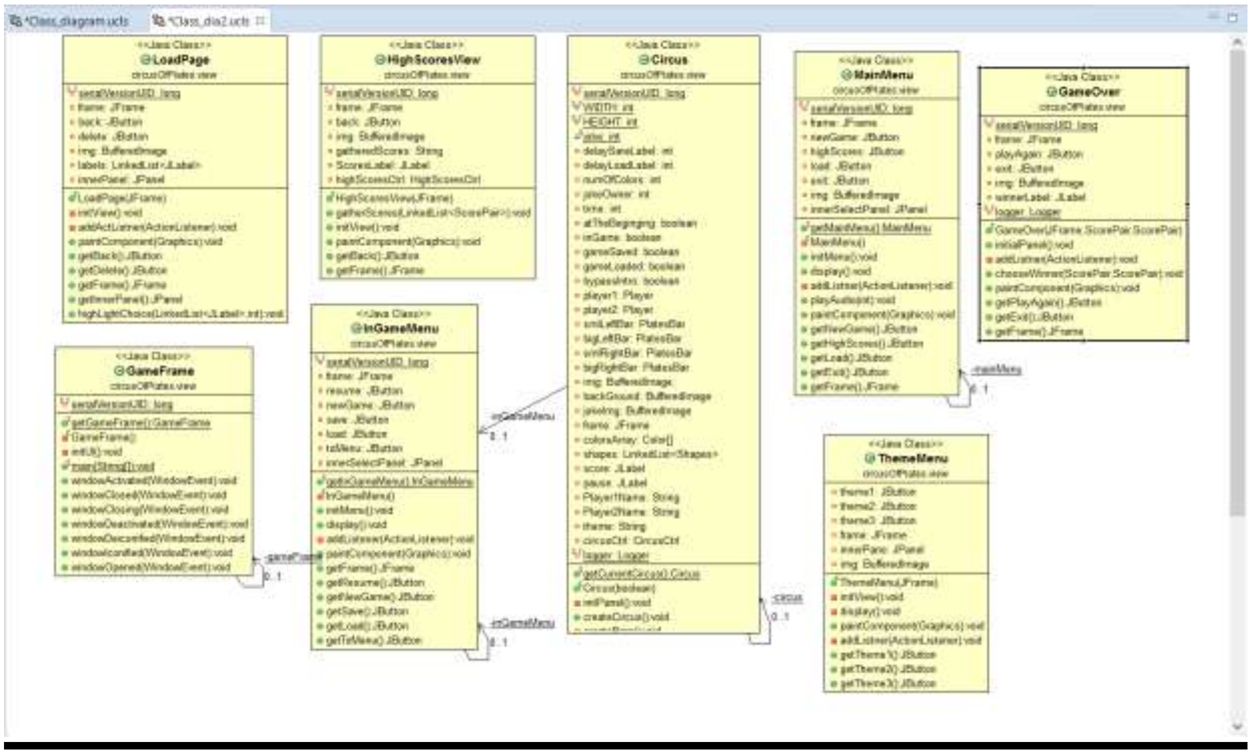
**Observer**: there is a Timer class which is the observed that extends the observable class, it notifies the observer "LevelObserver" that implements the observer interface each time it changes. The Update method in "LevelObserver" checks the time to determine the level hence choose the difficulty.

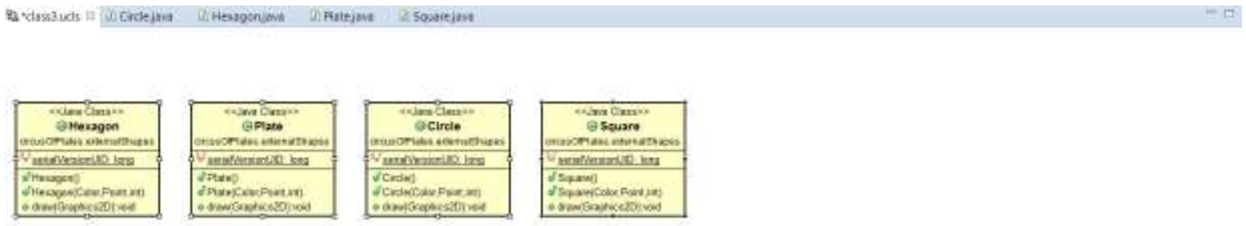
**MVC**: in the sake of simplicity, the model, view and controller need to be separated , there are 3 package of them ,where the model has the data and the operations (functionality) , and the view deals only with the UI . here comes the controller role where it provide communication between the model and the view . for the view there are different menus and a main panel where the actual game reside, each menu has it's controller which uses the needed objects to obtain the need functionality for those panels.

**Façade**: we used a high level unified interface "CircusCtrl" that interacts with the subsystem and make the design simple so that the game main view (Circus) deals with only one interface which in turn handles all the requests and manages the main game view through objects of another subsystems like Timer calculator, creating shapes, investigating the pool, factory and so on.

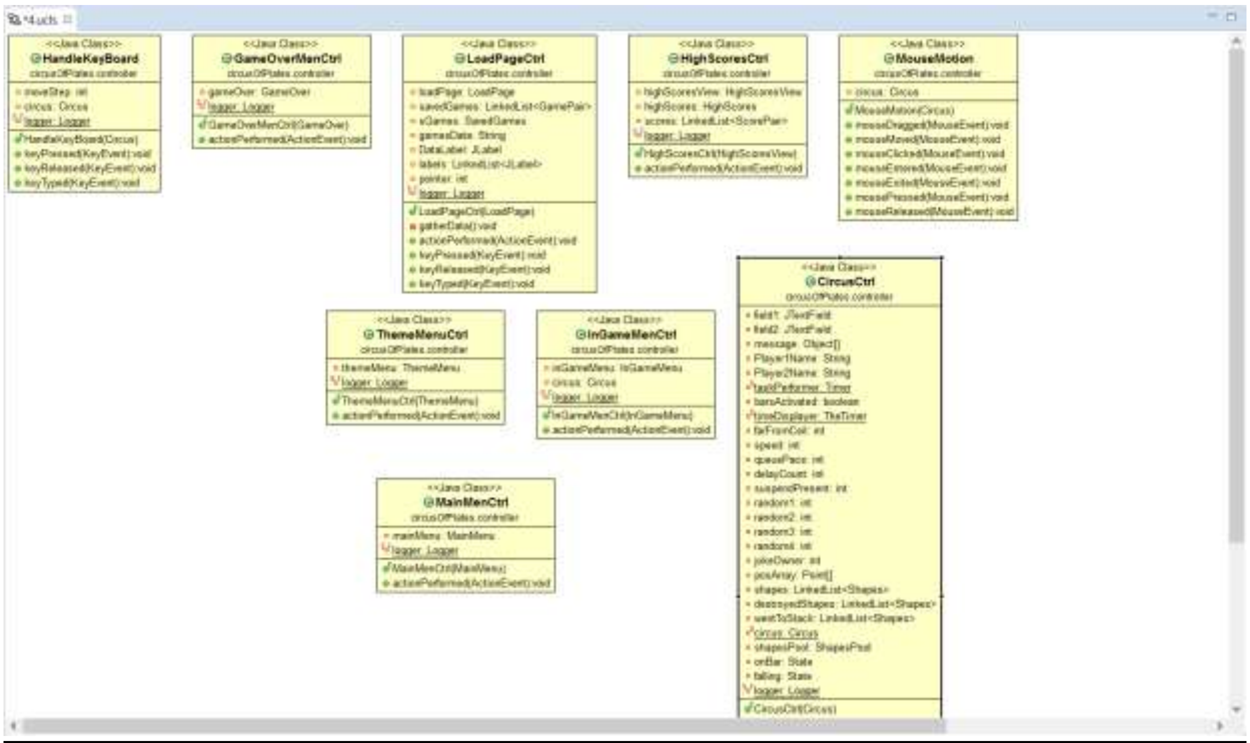
# Class diagram





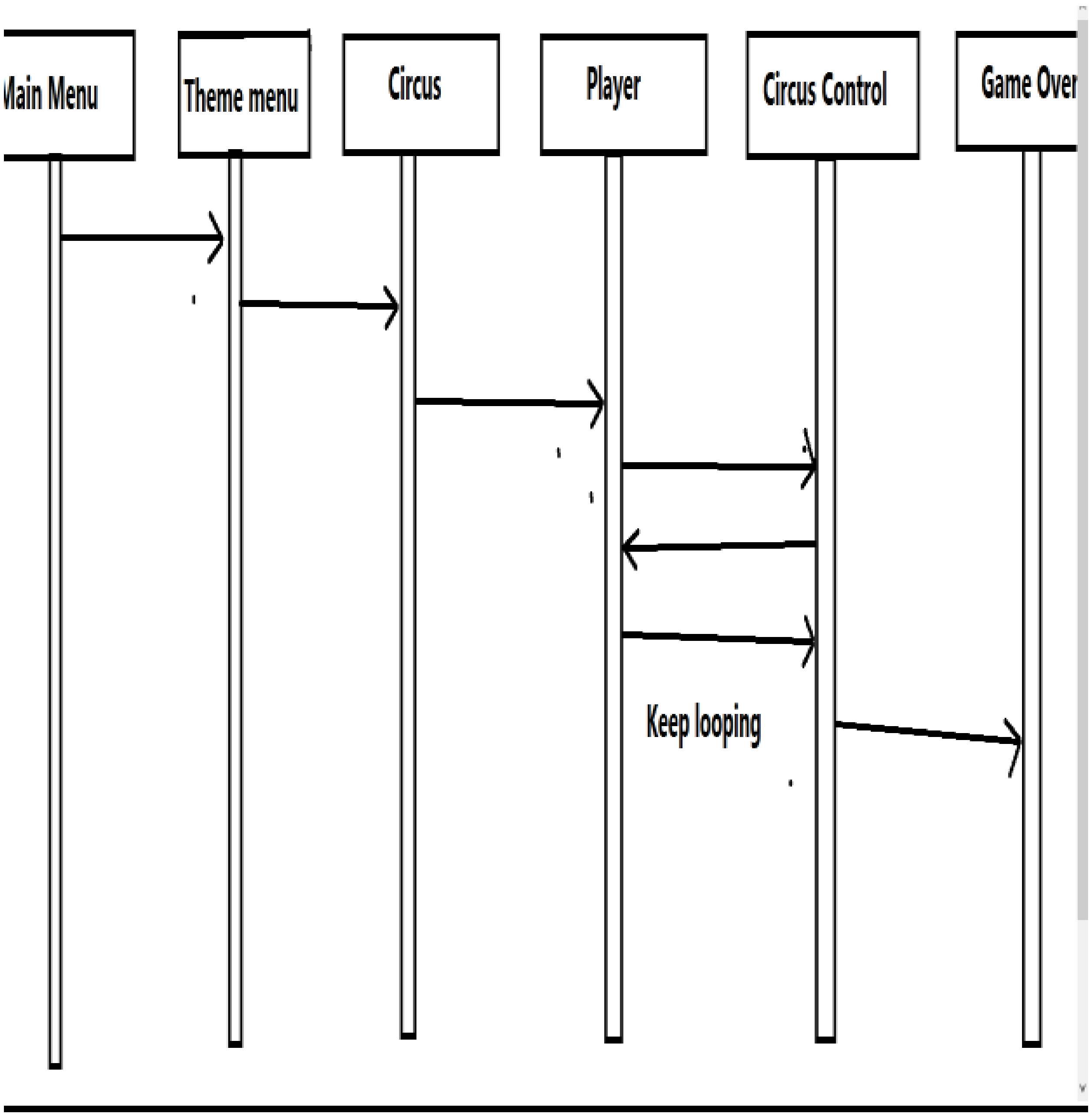


```
*class3.ucls   Circle.java   Hexagon.java   Plate.java   Square.java

<<Java Class>>
Hexagon
circusOfPlates.externalShapes
serialVersionUID: long
Hexagon()
Hexagon(Color,Point,int)
draw(Graphics2D):void

<<Java Class>>
Plate
circusOfPlates.externalShapes
serialVersionUID: long
Plate()
Plate(Color,Point,int)
draw(Graphics2D):void

<<Java Class>>
Circle
circusOfPlates.externalShapes
serialVersionUID: long
Circle()
Circle(Color,Point,int)
draw(Graphics2D):void

<<Java Class>>
Square
circusOfPlates.externalShapes
serialVersionUID: long
Square()
Square(Color,Point,int)
draw(Graphics2D):void
```



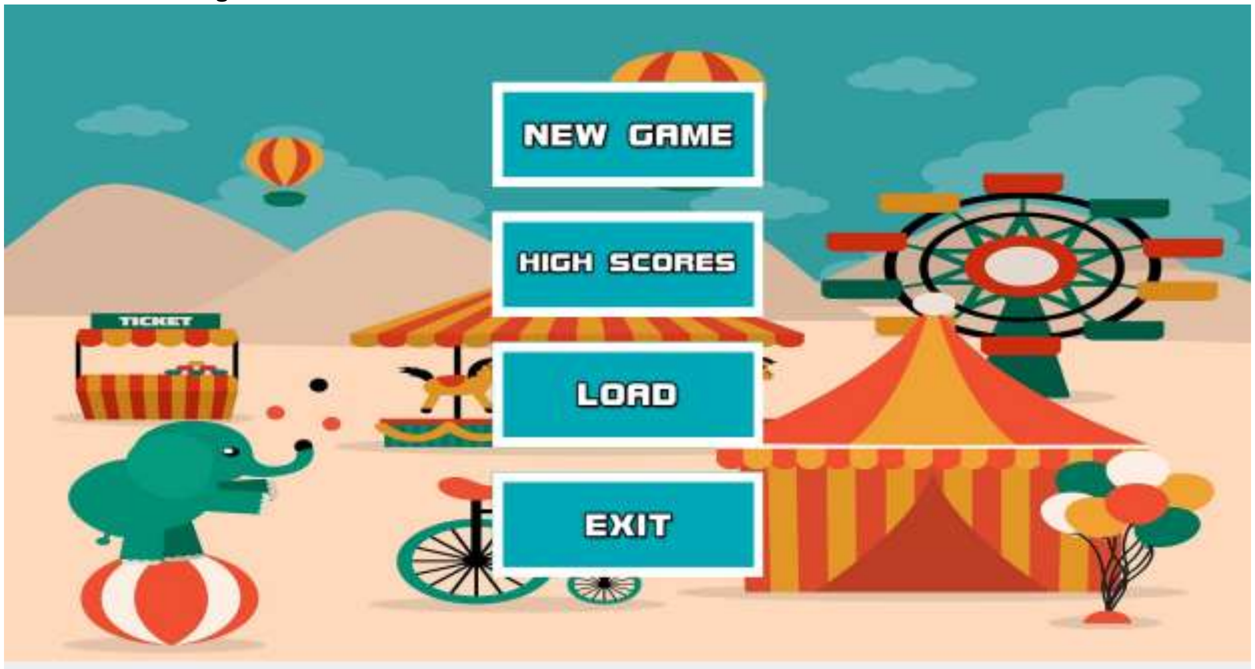
```
*4.ucls

<<Java Class>>
HandleKeyBoard
circusOfPlates.controller
moveStep: int
circus: Circus
logger: Logger
HandleKeyBoard(Circus)
keyPressed(KeyEvent):void
keyReleased(KeyEvent):void
keyTyped(KeyEvent):void

<<Java Class>>
GameOverMenCtrl
circusOfPlates.controller
gameOver: GameOver
logger: Logger
GameOverMenCtrl(GameOver)
actionPerformed(ActionEvent):void

<<Java Class>>
LoadPageCtrl
circusOfPlates.controller
loadPage: LoadPage
savedGames: LinkedList<GamePair>
sGames: SavedGames
gamesData: String
DataLabel: JLabel
labels: LinkedList<JLabel>
pointer: int
logger: Logger
LoadPageCtrl(LoadPage)
gatherData():void
actionPerformed(ActionEvent):void
keyPressed(KeyEvent):void
keyReleased(KeyEvent):void
keyTyped(KeyEvent):void

<<Java Class>>
HighScoresCtrl
circusOfPlates.controller
highScoresView: HighScoresView
highScores: HighScores
scores: LinkedList<ScorePair>
logger: Logger
HighScoresCtrl(HighScoresView)
actionPerformed(ActionEvent):void

<<Java Class>>
MouseMotion
circusOfPlates.controller
circus: Circus
MouseMotion(Circus)
mouseDragged(MouseEvent):void
mouseMoved(MouseEvent):void
mouseClicked(MouseEvent):void
mouseEntered(MouseEvent):void
mouseExited(MouseEvent):void
mousePressed(MouseEvent):void
mouseReleased(MouseEvent):void

<<Java Class>>
ThemeMenuCtrl
circusOfPlates.controller
themeMenu: ThemeMenu
logger: Logger
ThemeMenuCtrl(ThemeMenu)
actionPerformed(ActionEvent):void

<<Java Class>>
InGameMenCtrl
circusOfPlates.controller
inGameMenu: InGameMenu
circus: Circus
logger: Logger
InGameMenCtrl(InGameMenu)
actionPerformed(ActionEvent):void

<<Java Class>>
MainMenCtrl
circusOfPlates.controller
mainMenu: MainMenu
logger: Logger
MainMenCtrl(MainMenu)
actionPerformed(ActionEvent):void

<<Java Class>>
CircusCtrl
circusOfPlates.controller
field1: JTextField
field2: JTextField
message: Object[]
Player1Name: String
Player2Name: String
taskPerformer: Timer
barsActivated: boolean
timeDisplayer: TheTimer
farFromCol: int
speed: int
queuePace: int
delayCount: int
suspendPresent: int
random1: int
random2: int
random3: int
random4: int
jokeOwner: int
posArray: Point[]
shapes: LinkedList<Shapes>
destroyedShapes: LinkedList<Shapes>
wentToStack: LinkedList<Shapes>
circus: Circus
shapesPool: ShapesPool
onBar: State
falling: State
logger: Logger
CircusCtrl(Circus)
```

# Sequence diagram

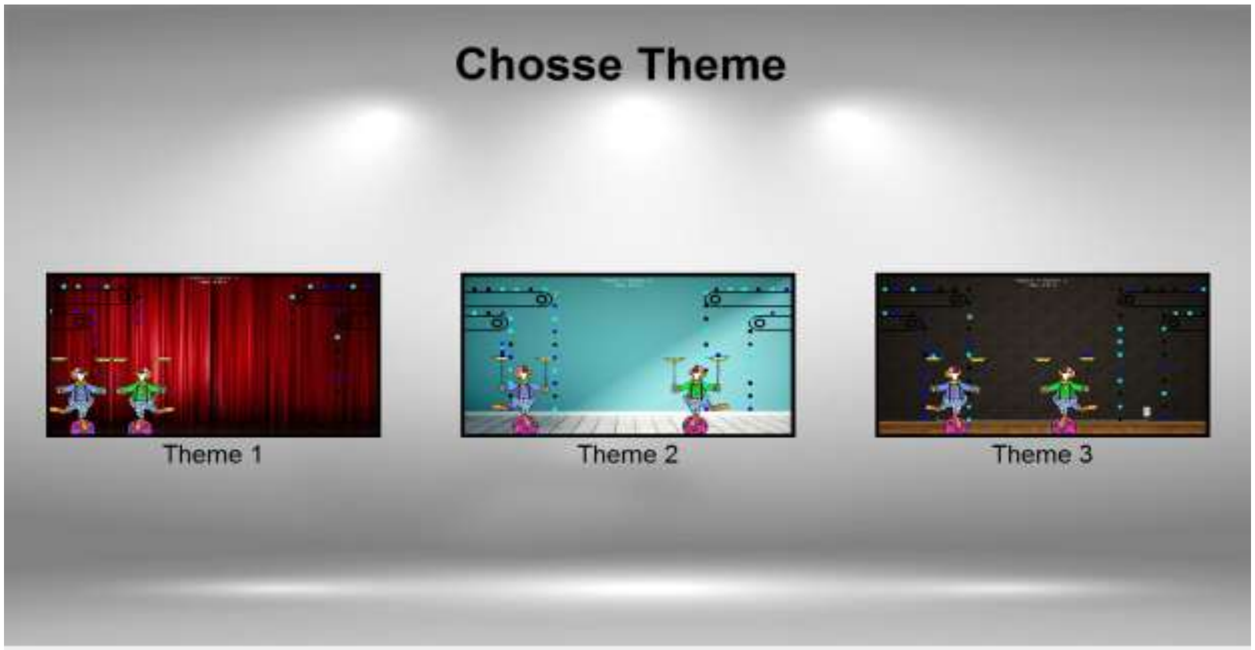


# User guide

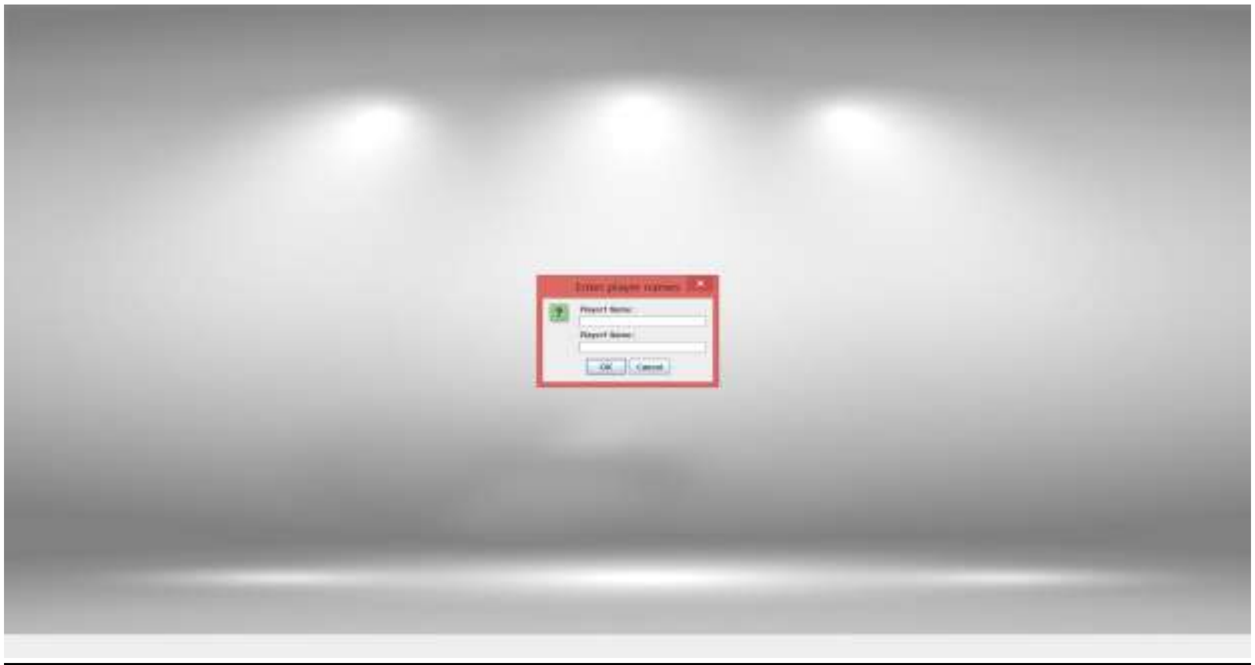
In the start of the game you can choose between start a new game, load a last game, getting the High scores or exit the game.



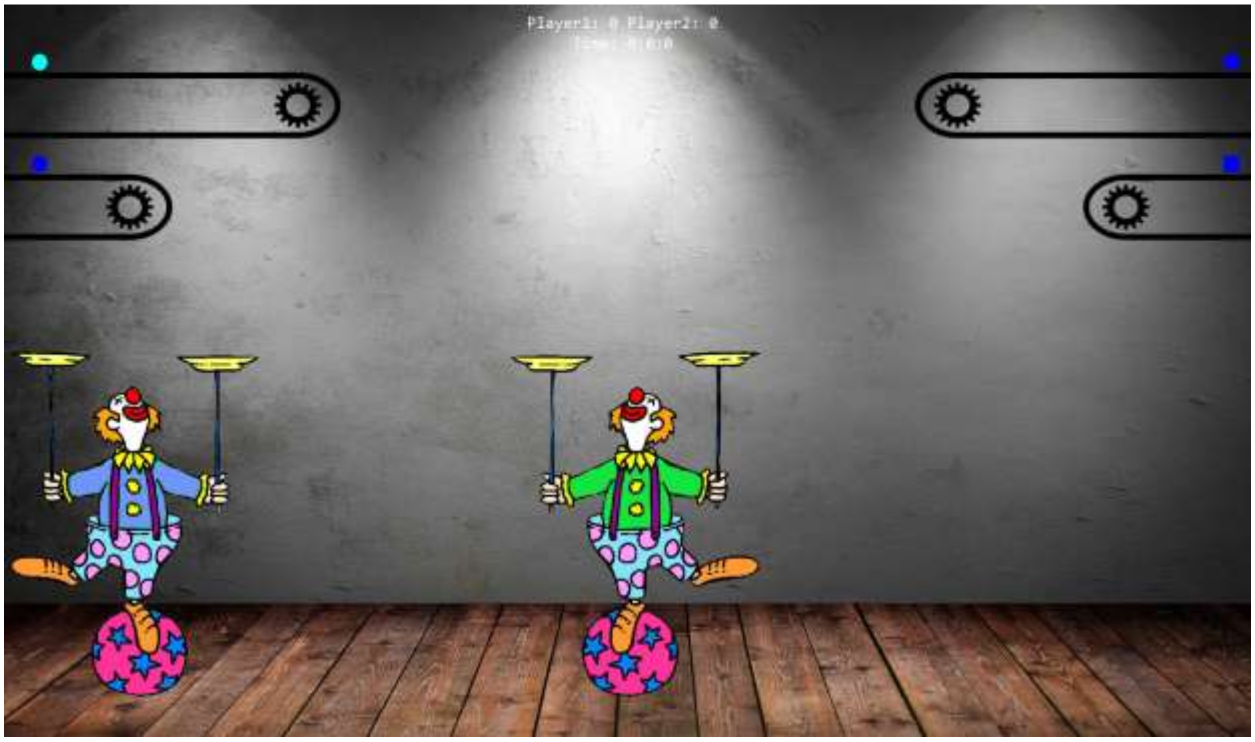
If we choose new game you have to choose a theme from 3 themes for the game.

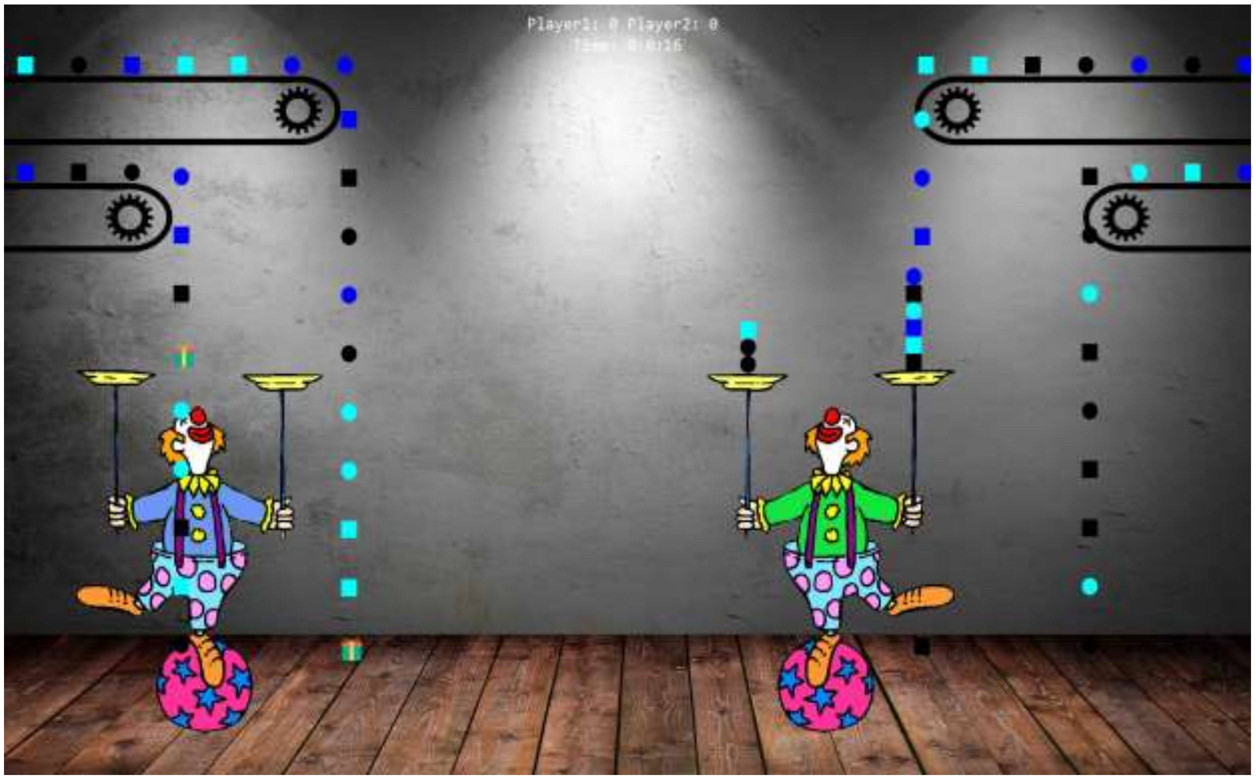


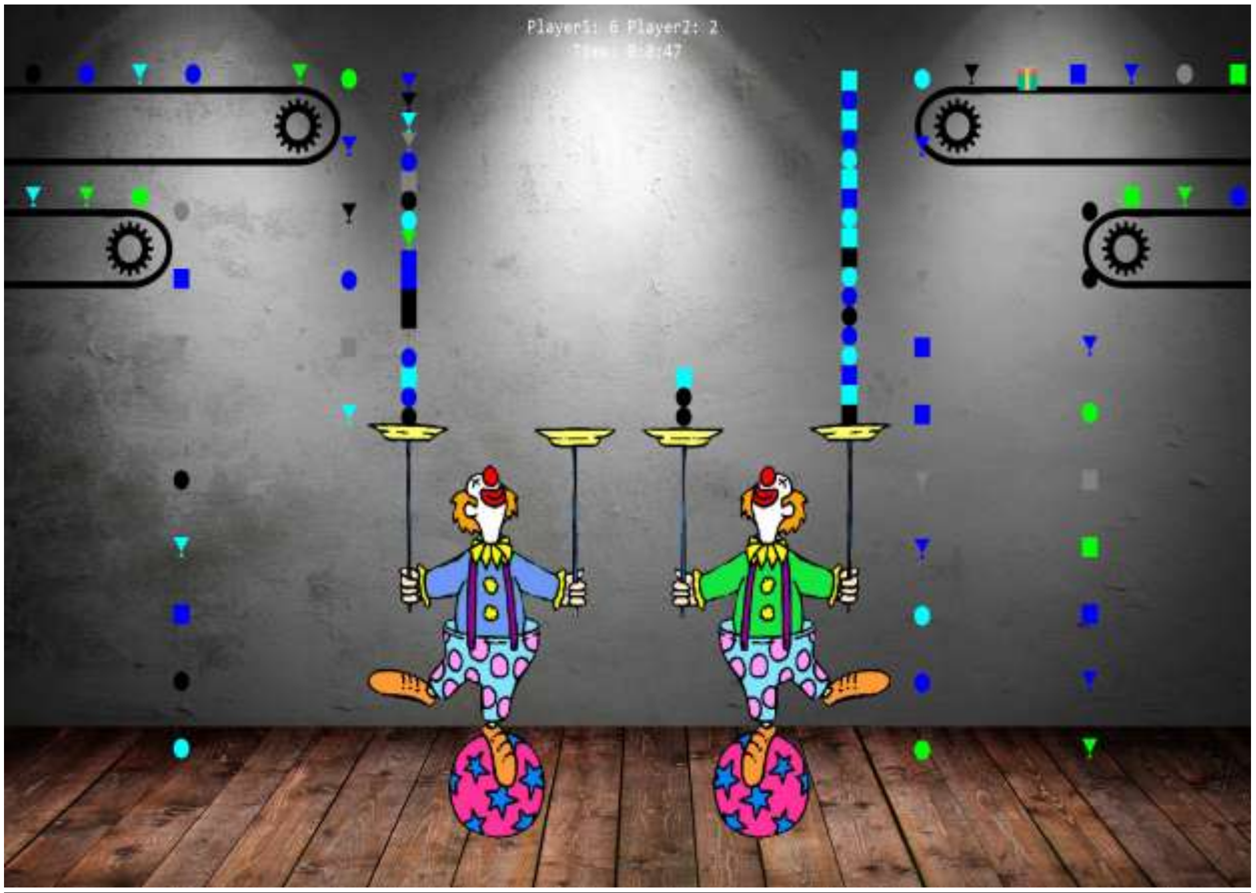
Then you have to enter the two players name and if not it will be default names.

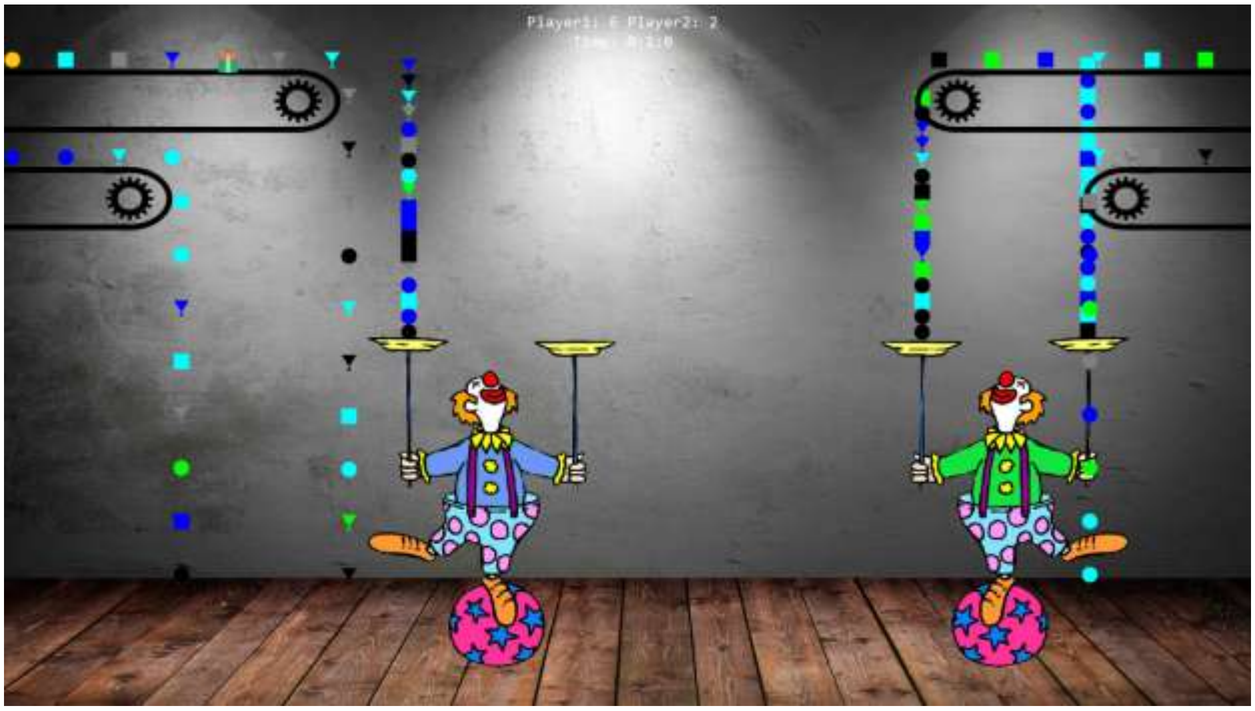


The game starts by two clowns and four bars which you can collect the falling elements.

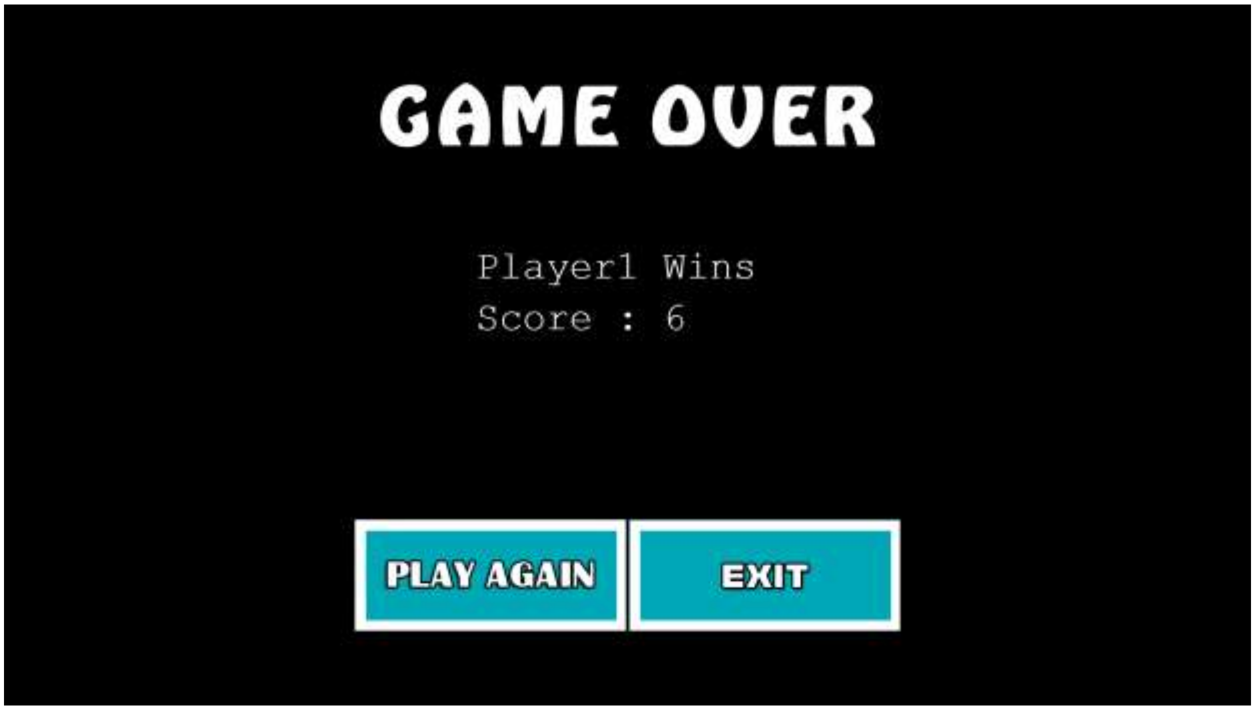



Player1: 0 Player2: 0
Time: 0:0:16


Player1: 6 Player2: 2
Time: 0:0:47



If the both stacks for any player get failed the game will end and you will get the results.



Player1 Wins
Score : 6