Circus of Plates

Name :Amr El-Begawy

Id : 45

Name : Hamza hassan Mohammed

Id : 26

Name : Mahmoud Ebrahim Elsayed

Id : 58

Name : Mohanad Ayman

Id : 65

Content

1- Description

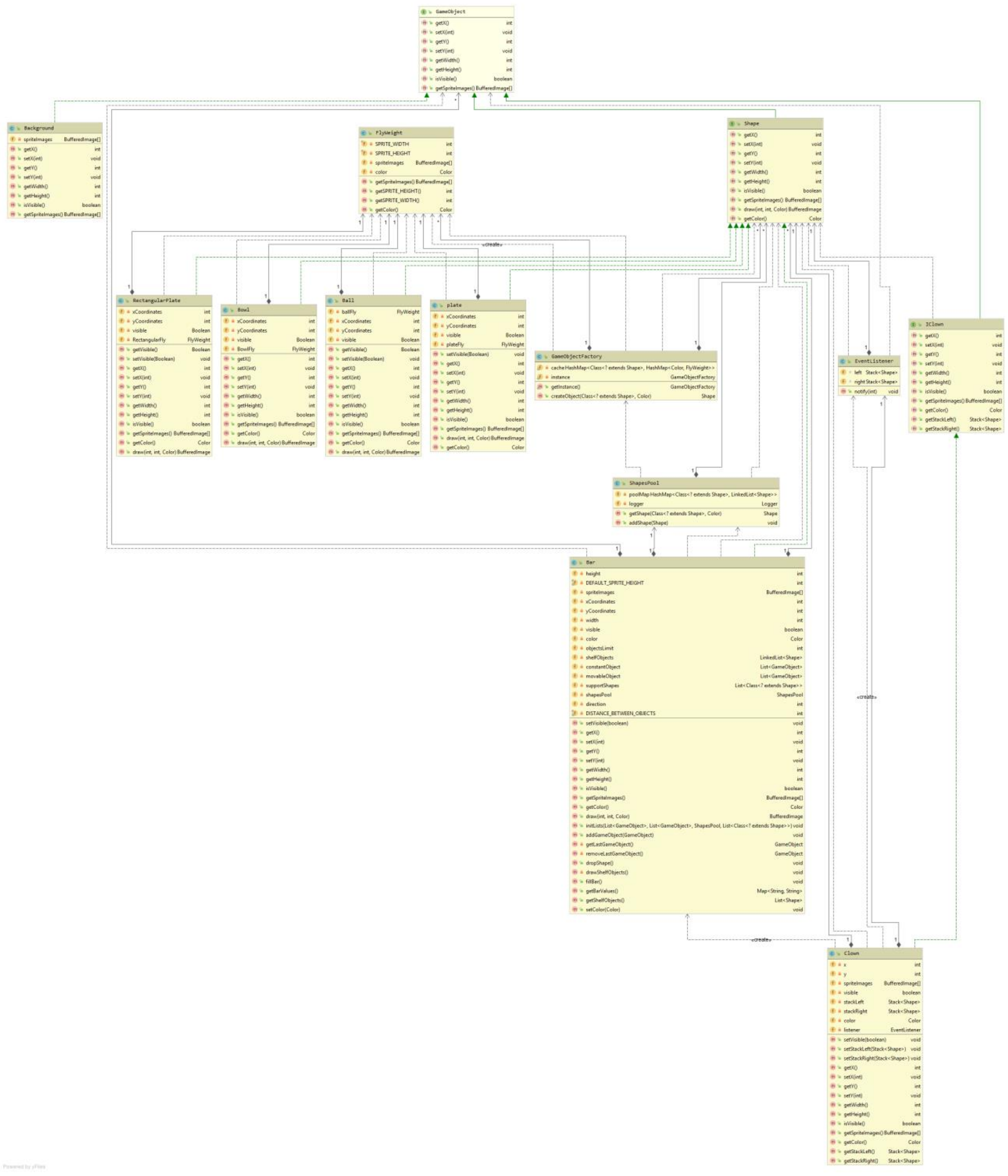
- It is single player-game in which a clown carries two stacks of plates, and there are a set of colored plates queues that fall and he tries to catch them, if he manages to collect three consecutive plates of the same color, then they are vanished and his score increases.

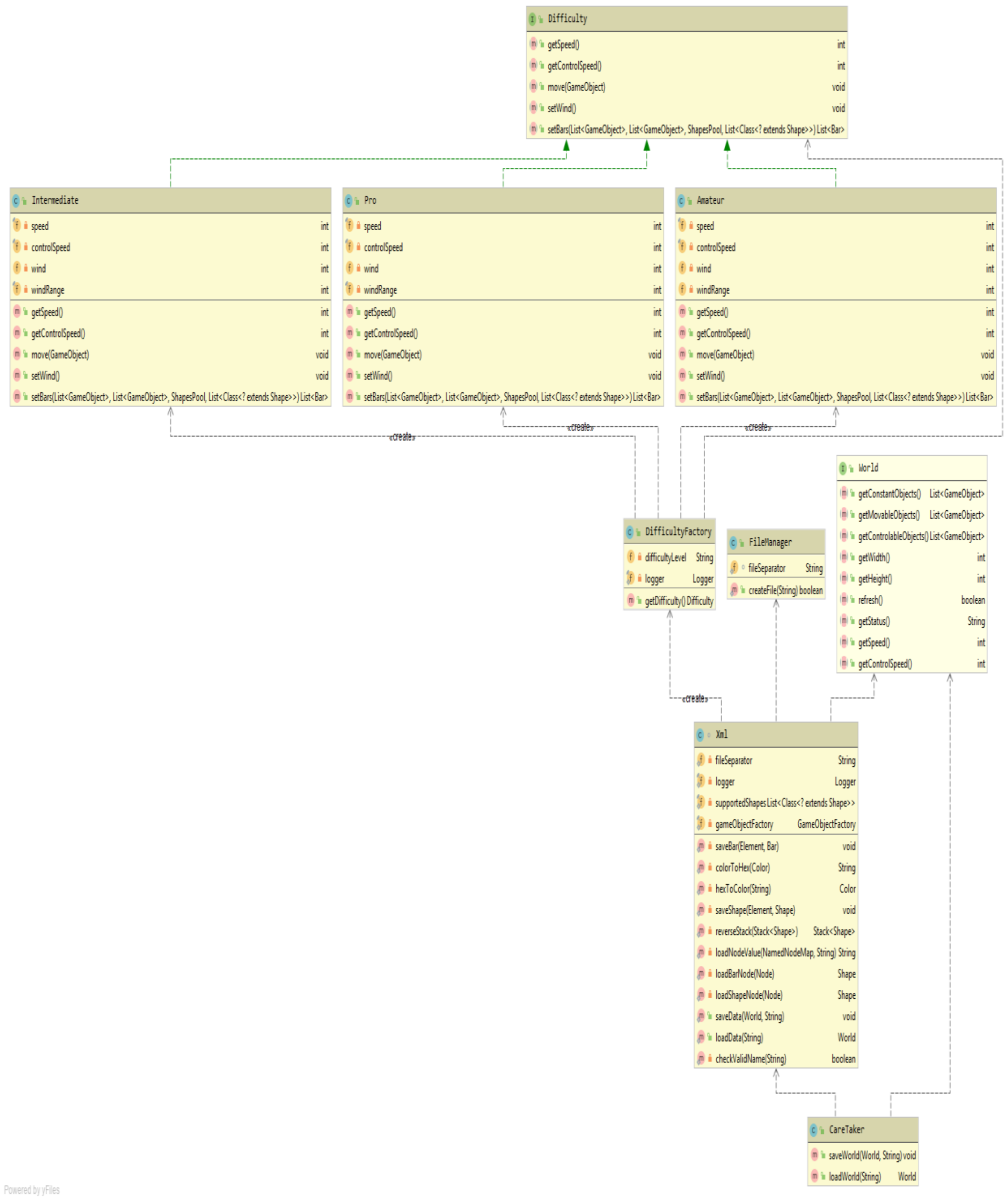
- If the length of the stack exceeds the red line the user loses the Game

2- Game features

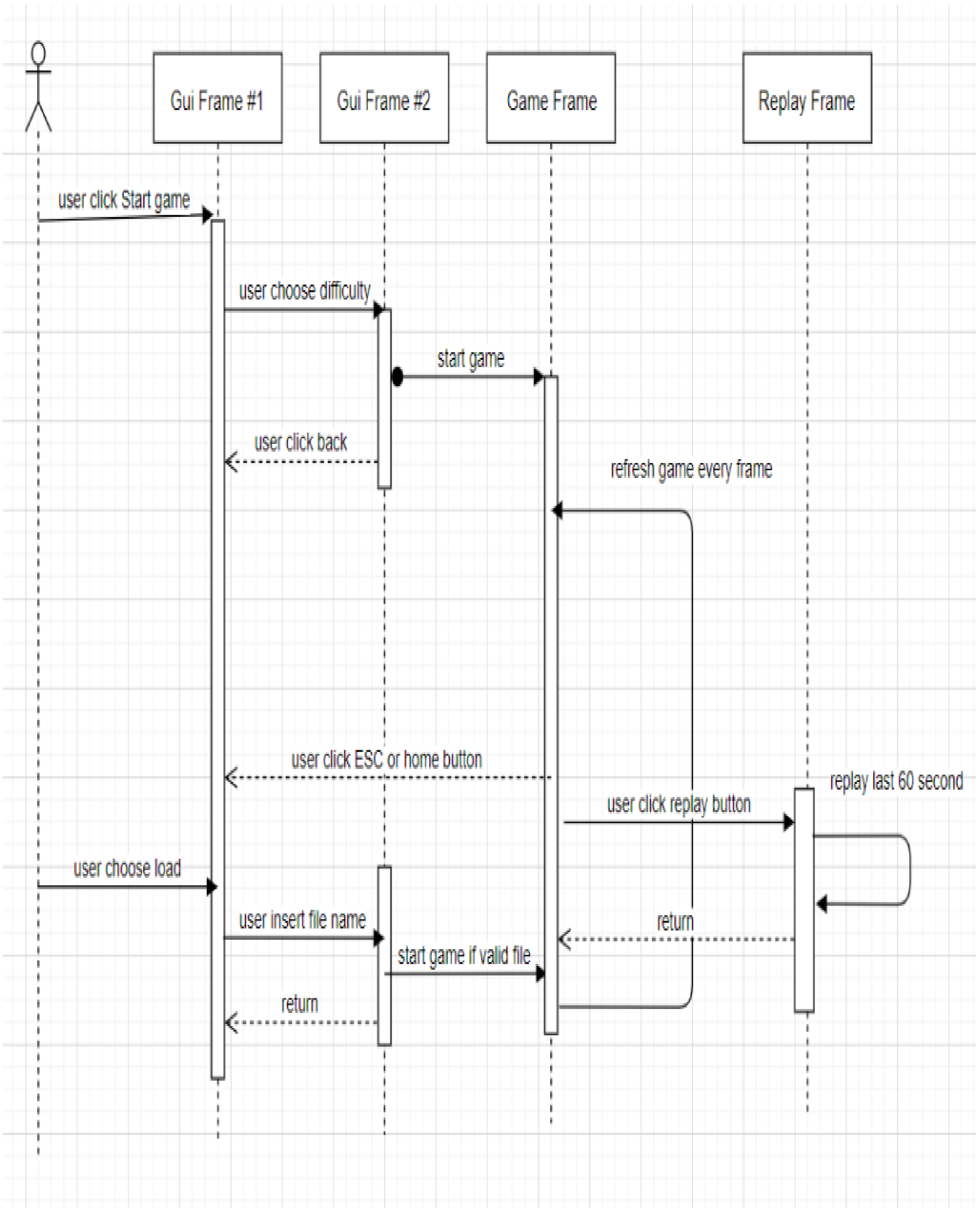
- A user can save his current state and load it later
- A user can load an external shape
- A user can choose a clown from two different clown supported
- A user can use the keyboard and mouse for interaction with the interface.

3- UML class diagram




Difficulty
getSpeed() int
getControlSpeed() int
move(GameObject) void
setWind() void
setBars(List<GameObject>, List<GameObject>, ShapesPool, List<Class<? extends Shape>>) List<Bar>
Intermediate
speed int
controlSpeed int
wind int
windRange int
getSpeed() int
getControlSpeed() int
move(GameObject) void
setWind() void
setBars(List<GameObject>, List<GameObject>, ShapesPool, List<Class<? extends Shape>>) List<Bar>
Pro
speed int
controlSpeed int
wind int
windRange int
getSpeed() int
getControlSpeed() int
move(GameObject) void
setWind() void
setBars(List<GameObject>, List<GameObject>, ShapesPool, List<Class<? extends Shape>>) List<Bar>
Amateur
speed int
controlSpeed int
wind int
windRange int
getSpeed() int
getControlSpeed() int
move(GameObject) void
setWind() void
setBars(List<GameObject>, List<GameObject>, ShapesPool, List<Class<? extends Shape>>) List<Bar>
«create»
«create»
«create»
World
getConstantObjects() List<GameObject>
getMovableObjects() List<GameObject>
getControlableObjects() List<GameObject>
getWidth() int
getHeight() int
refresh() boolean
getStatus() String
getSpeed() int
getControlSpeed() int
DifficultyFactory
difficultyLevel String
logger Logger
getDifficulty() Difficulty
FileManager
fileSeparator String
createFile(String) boolean
«create»
Xml
fileSeparator String
logger Logger
supportedShapes List<Class<? extends Shape>>
gameObjectFactory GameObjectFactory
saveBar(Element, Bar) void
colorToHex(Color) String
hexToColor(String) Color
saveShape(Element, Shape) void
reverseStack(Stack<Shape>) Stack<Shape>
loadNodeValue(NamedNodeMap, String) String
loadBarNode(Node) Shape
loadShapeNode(Node) Shape
saveData(World, String) void
loadData(String) World
checkValidName(String) boolean
CareTaker
saveWorld(World, String) void
loadWorld(String) World
Powered by yFiles

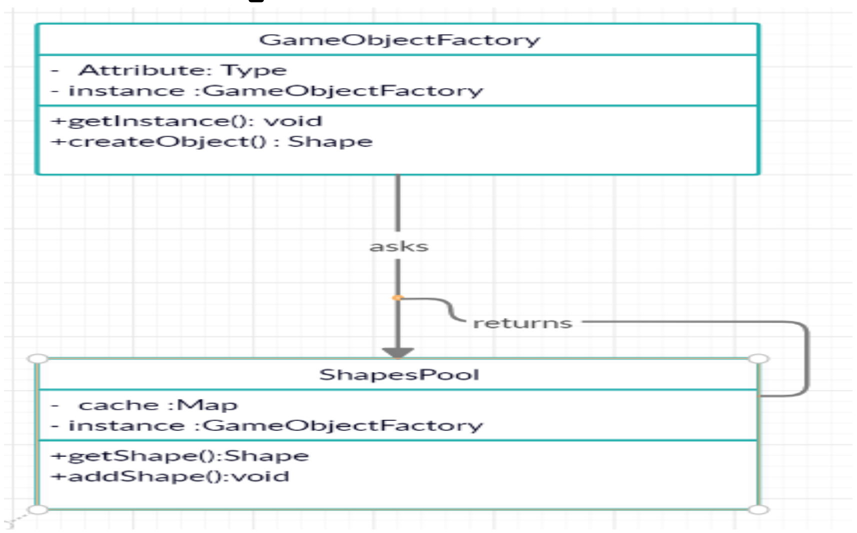
4- Sequence diagram



5- Design patterns

1- Singleton

- **Class diagram**



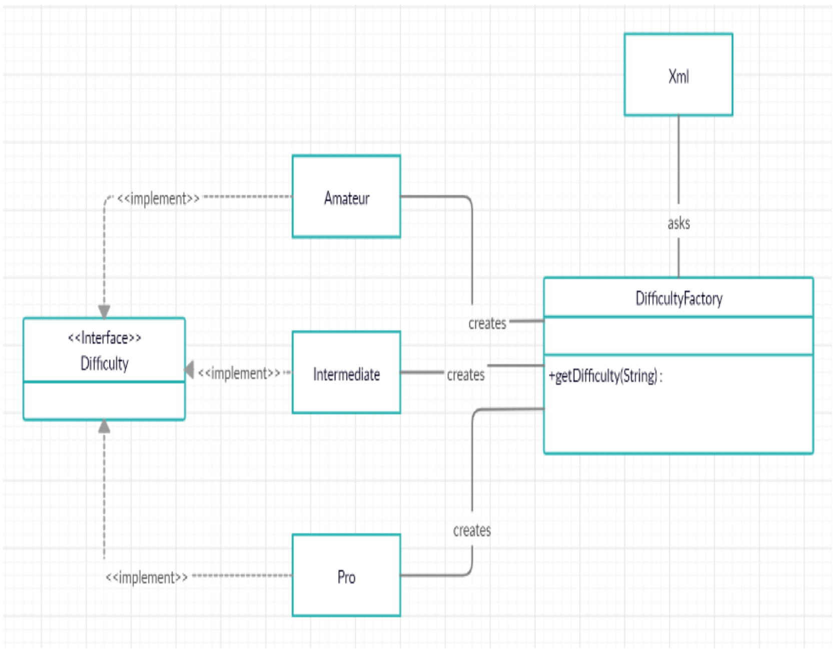
- **Usage**

In case that we need to create a new shape , we call shapesPool to create us a new shape or let us use an existing one

In case that we did not find an existing shape the shapesPool calls GameObjectFactory to create a new shape for it, hence we need only one instance to reduce memory usage

2- <u>Factory method</u>

- **Class diagram**



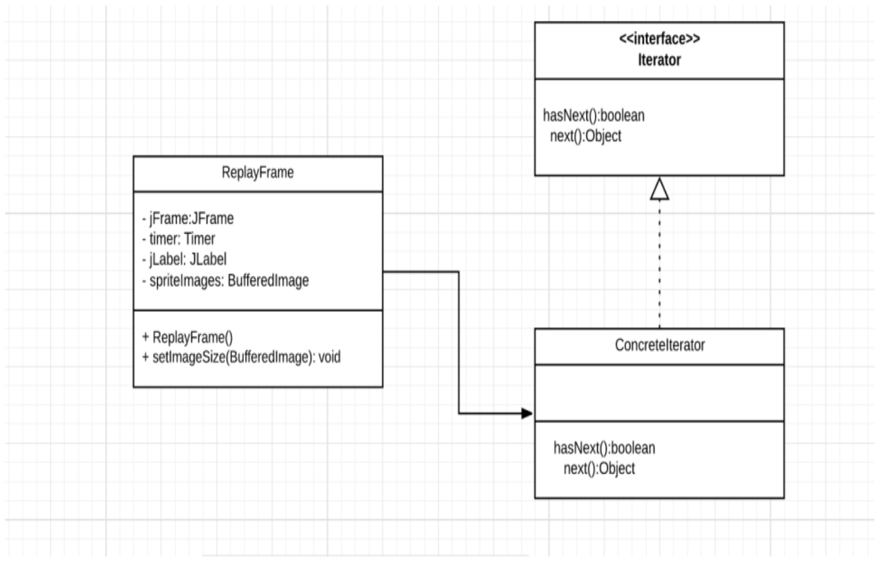
- **Usage**

  We use factory to encapsulate creating of a difficulty class from the concrete classes that need to create one.

  This makes us never modify our concrete classes in case of adding or removing something in difficulty

3- <u>Iterator</u>

- **Class diagram**

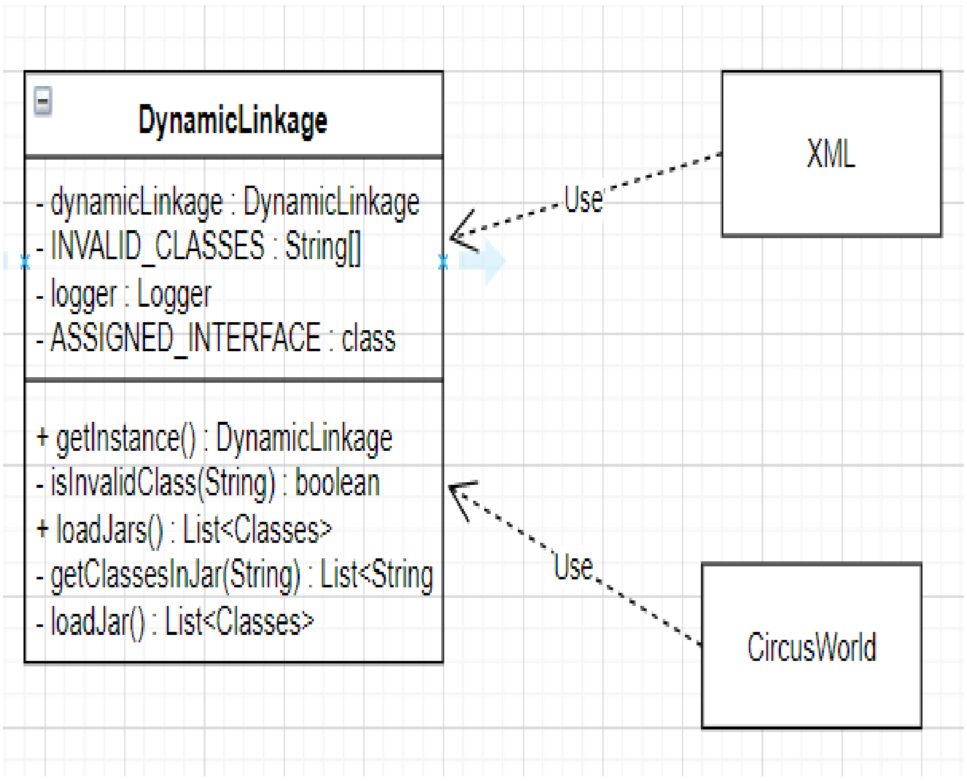


- **Usage**

  We use iterator design pattern in the *SpriteImages* class to iterate on it every second to display the photo

4- Dynamic Linkage

- **Class diagram**

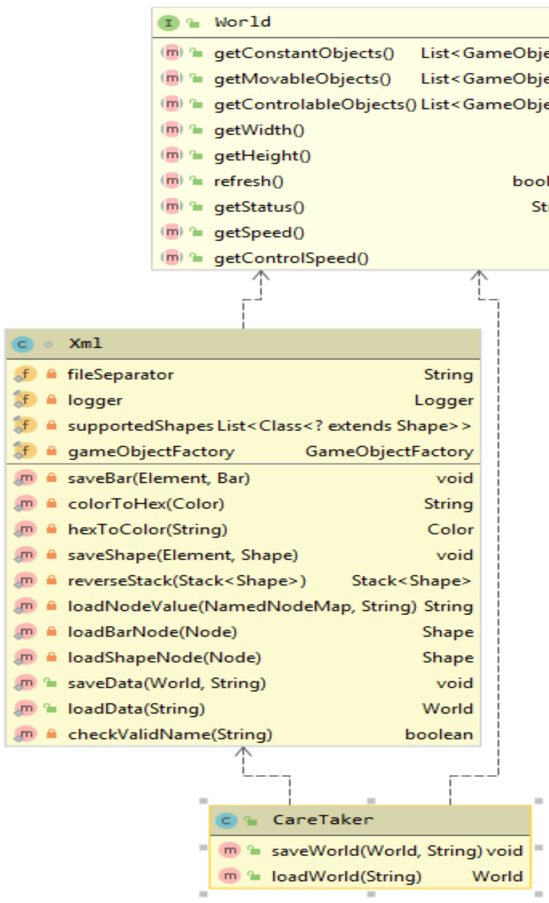

DynamicLinkage
- dynamicLinkage : DynamicLinkage
- INVALID_CLASSES : String[]
- logger : Logger
- ASSIGNED_INTERFACE : class
+ getInstance() : DynamicLinkage
- isInvalidClass(String) : boolean
+ loadJars() : List<Classes>
- getClassesInJar(String) : List<String
- loadJar() : List<Classes>
Use
XML
Use
CircusWorld

- **Usage**

  Dynamic linkage as can be seen is used by 2 different classes, First the XML is used by dynamic linkage to determine all the different type of shapes that can be loaded, And it's also used by the Gameworld during runtime to load the possible shapes for the game, Also dynamic Linkage is a singleton class because just 1 instantiation is needed.

5- Momento

- **Class diagram**


World
getConstantObjects() List<GameObject>
getMovableObjects() List<GameObject>
getControlableObjects() List<GameObject>
getWidth() int
getHeight() int
refresh() boolean
getStatus() String
getSpeed() int
getControlSpeed() int
Xml
fileSeparator String
logger Logger
supportedShapes List<Class<? extends Shape>>
gameObjectFactory GameObjectFactory
saveBar(Element, Bar) void
colorToHex(Color) String
hexToColor(String) Color
saveShape(Element, Shape) void
reverseStack(Stack<Shape>) Stack<Shape>
loadNodeValue(NamedNodeMap, String) String
loadBarNode(Node) Shape
loadShapeNode(Node) Shape
saveData(World, String) void
loadData(String) World
checkValidName(String) boolean
CareTaker
saveWorld(World, String) void
loadWorld(String) World

- **Usage**

We use it for saving a certain state and load it later by the user

Three main parts

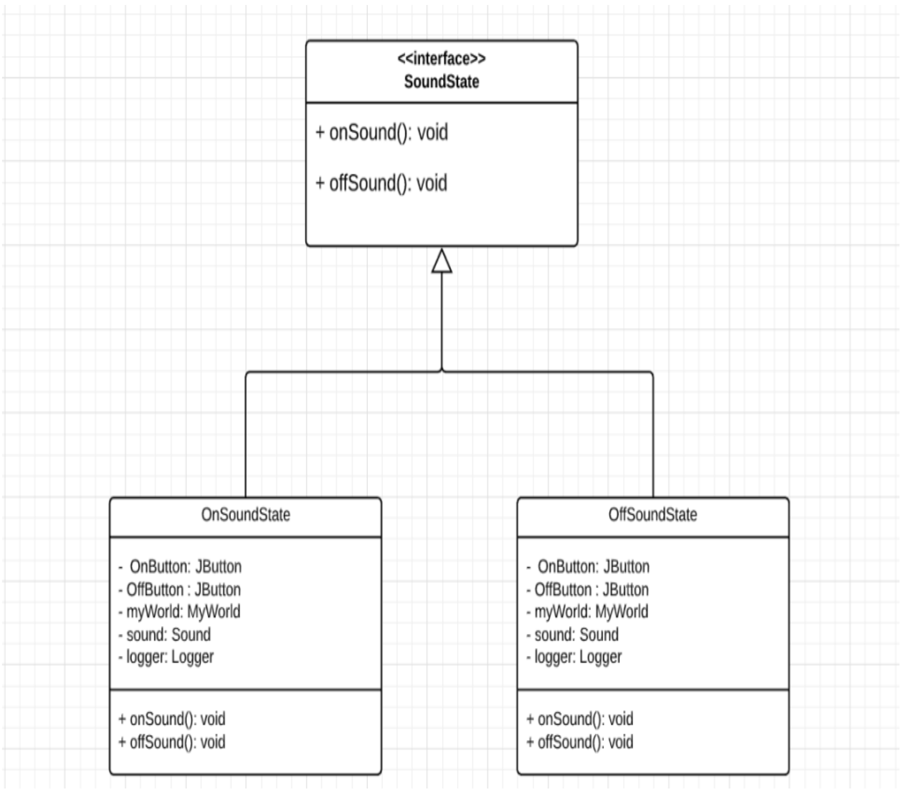
Momento -> World

Originator -> Xml

CareTaker -> CareTaker

6- <u>State</u>

- **Class diagram**

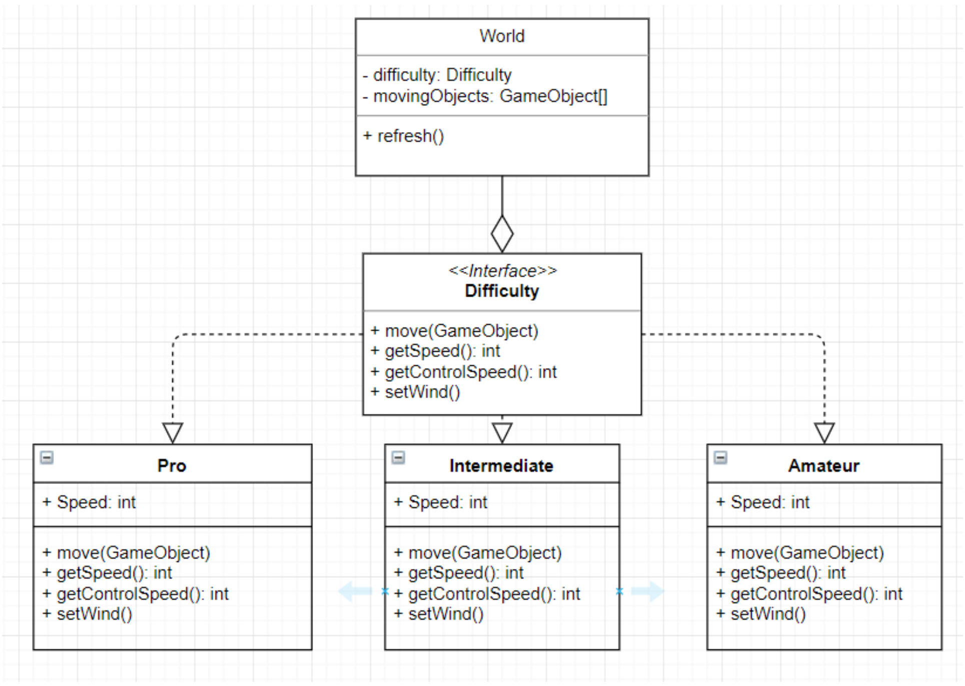


- **Usage**

Use it for turn on and off the sound the first state of the sound is off and change the state every time.

7- <u>Strategy</u>

- **Class diagram**

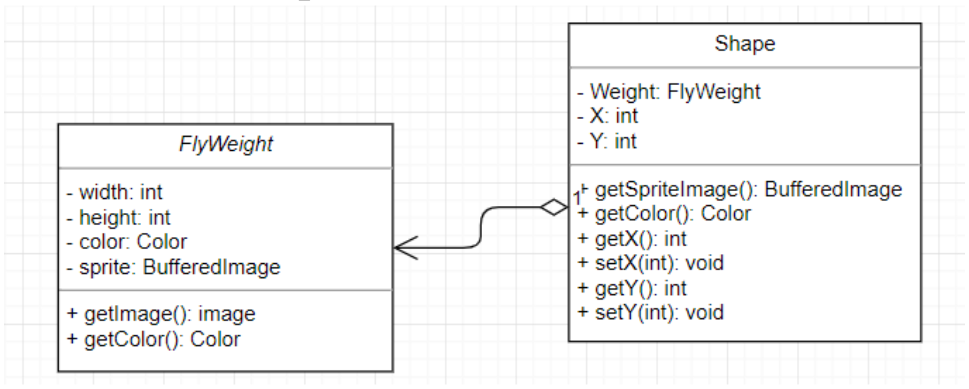


- **Usage**

A strategy design pattern is used to change the movement of objects according to chosen difficulty by varying the speed and wind. Furthermore, the initialization of the game is determined according to the concrete strategy deciding the number of bars in the game.

8- <u>Flyweight</u>

- **Class diagram**

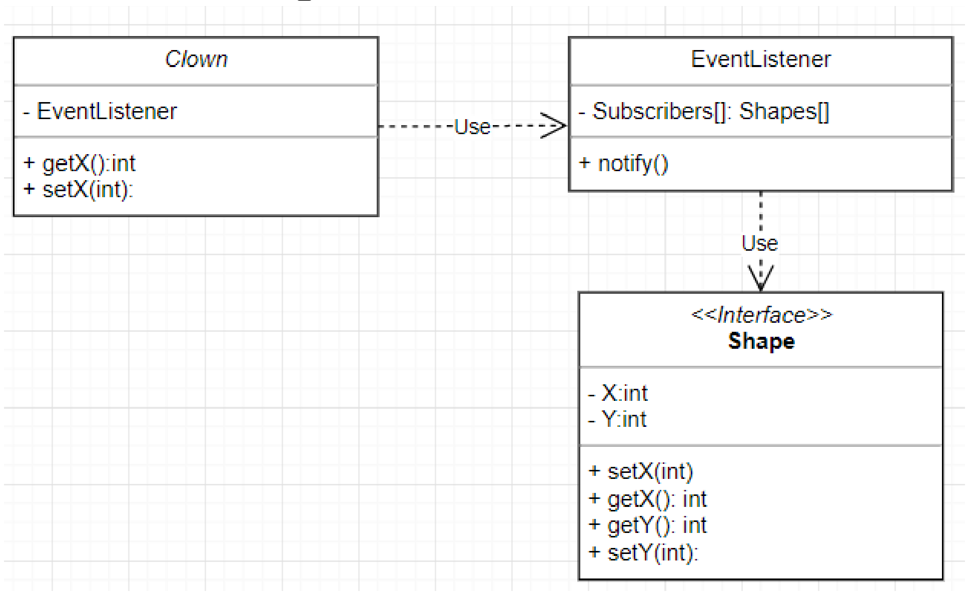


- **Usage**

Since all shapes of the same type share the same image, color and other common properties. A flyweight object is used to carry the common properties and a reference to it is shared among all objects of the same type saving memory.

9- Observer

- **Class diagram**

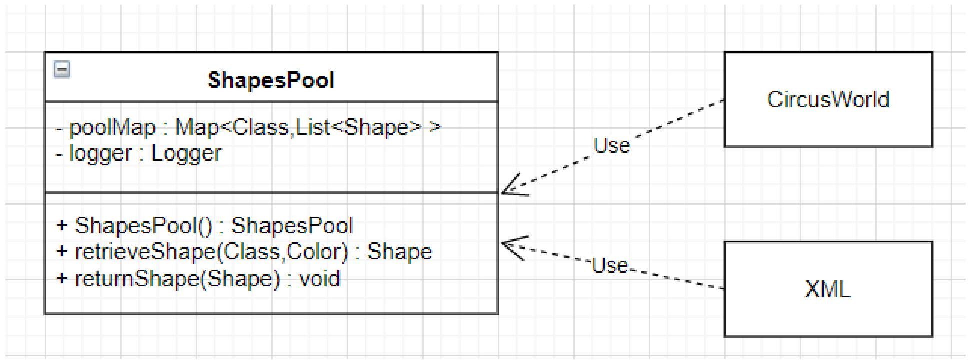


- **Usage**

A change in the clown's coordinates is observed by the event listener, notifying all the shapes held on the two stacks of the clown, changing their coordinates accordingly.

10- <u>Pool</u>

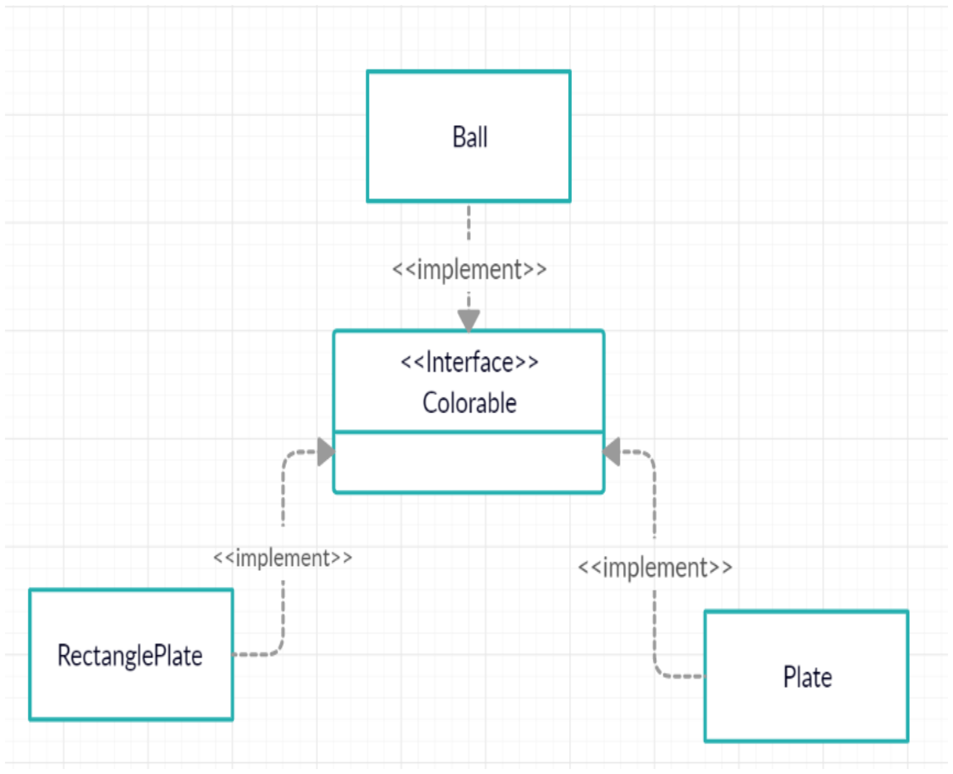
- **Class diagram**


ShapesPool
- poolMap : Map<Class,List<Shape> >
- logger : Logger
+ ShapesPool() : ShapesPool
+ retrieveShape(Class,Color) : Shape
+ returnShape(Shape) : void
CircusWorld
XML
Use
Use

- **Usage**
  - This specific design pattern is used to decrease the time consumption of making new objects, by saving previous unused created objects and saving them in a map and whenever needed we would retrieve them from the pool & put hold on this object until it is no longer used then returned back to the pool.

11- <u>Marker</u>

- **Class diagram**

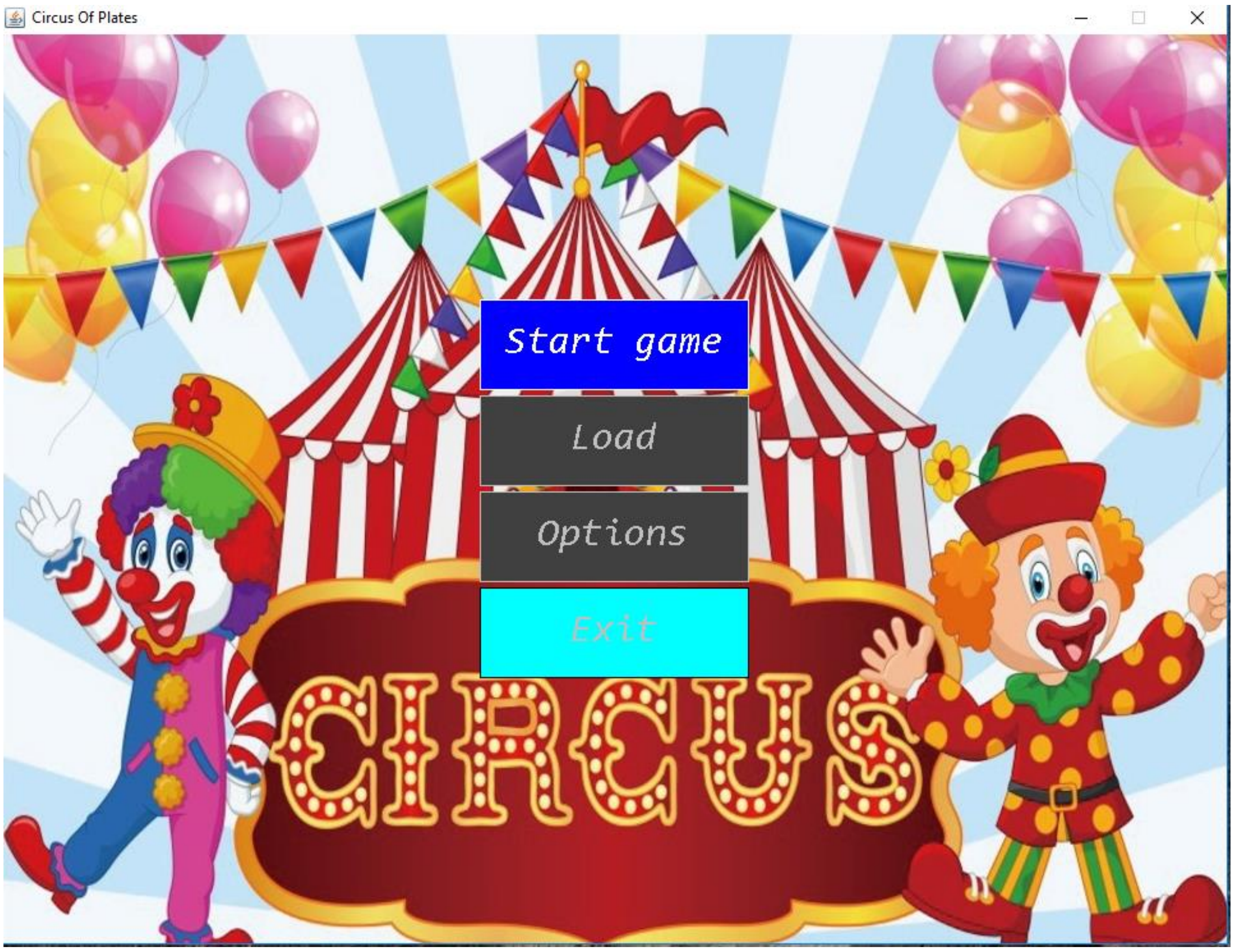


- **Usage**

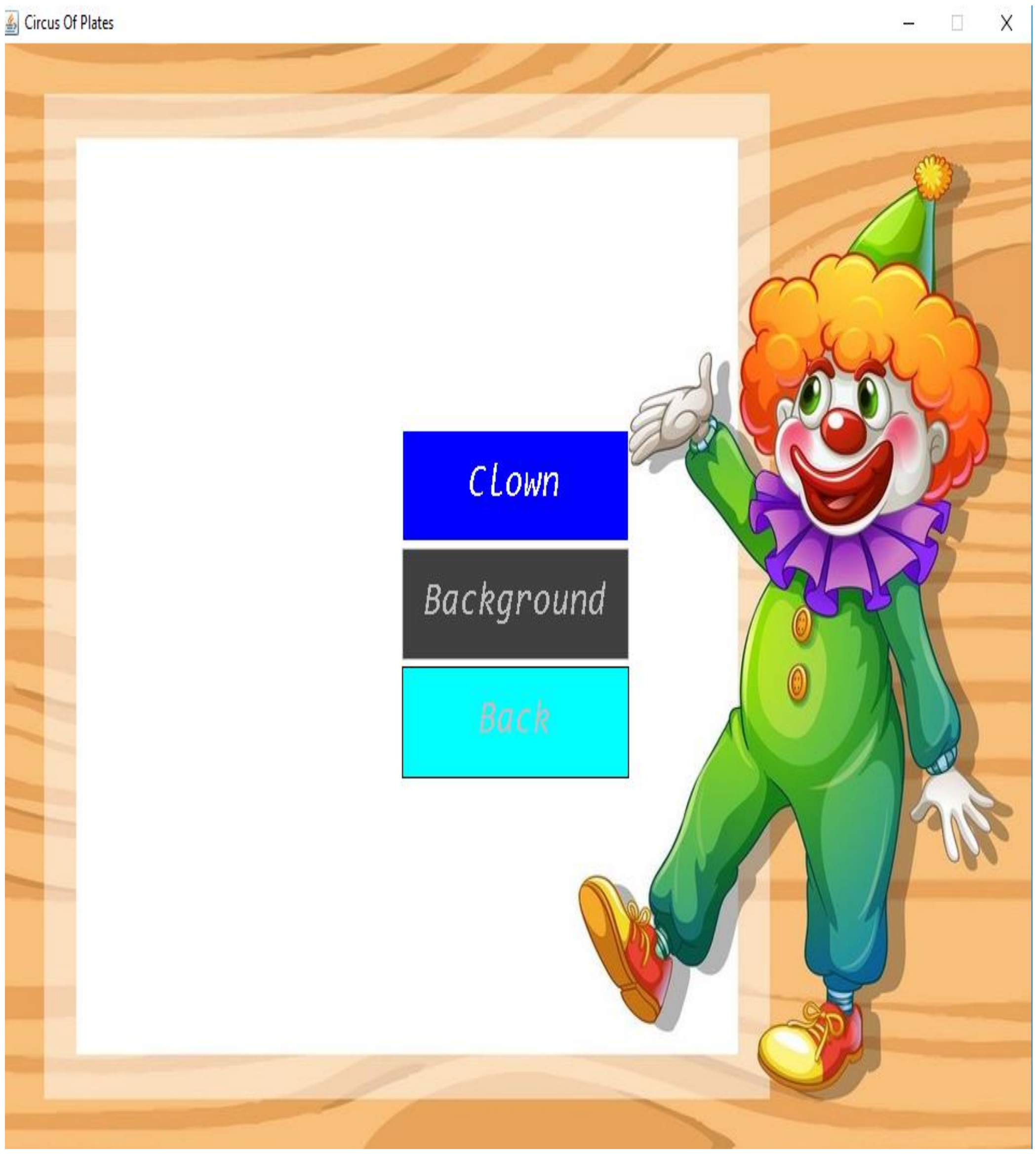
In case of adding a new difficulty in which shapes' colors changes during falling down we need to mark those shapes whose color should never change such as bombs or gifts
And here where markable design patterns comes with the suitable solution for that problem

6- User Guide

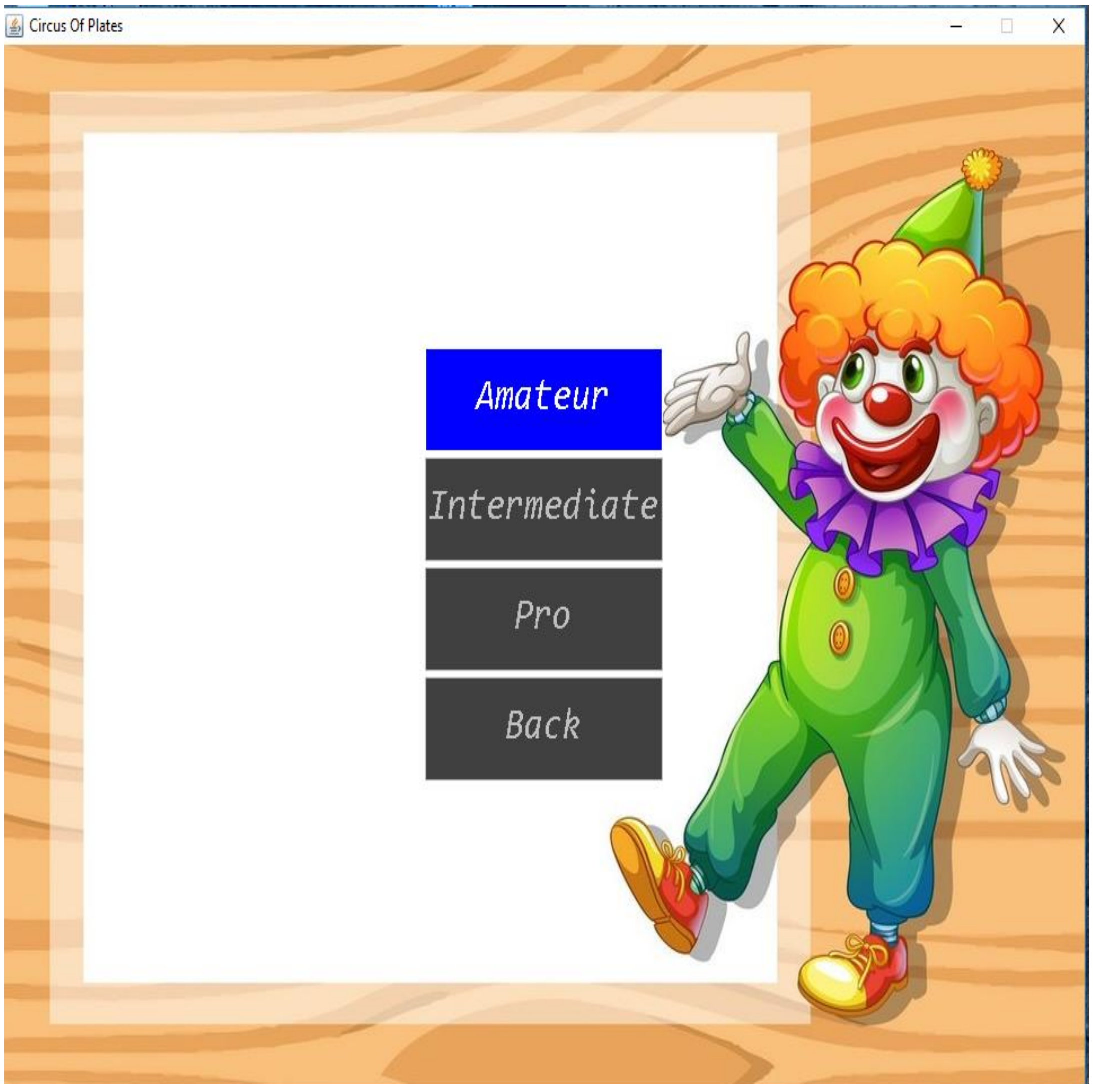
1- The start screen of the game



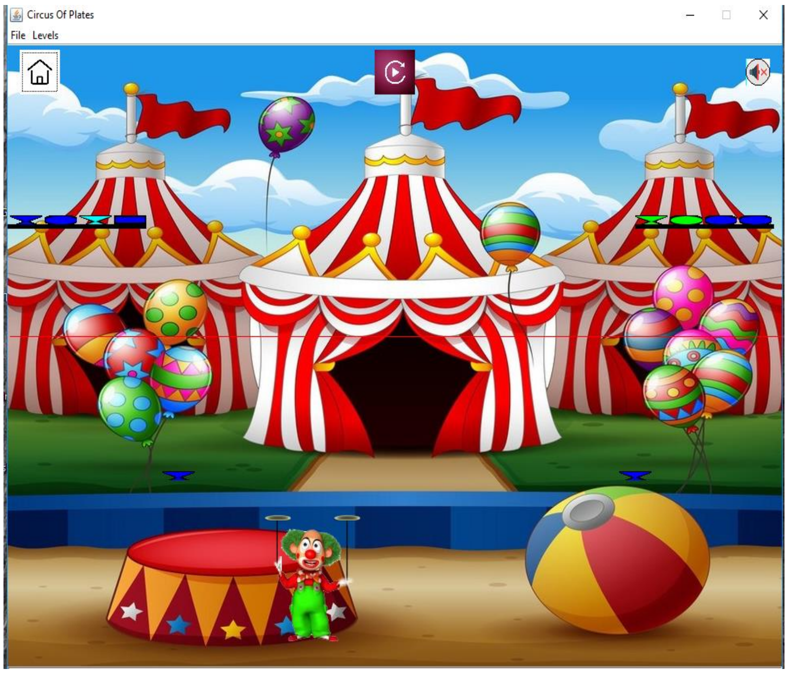
2- The options of the game
There are two clowns to choose from



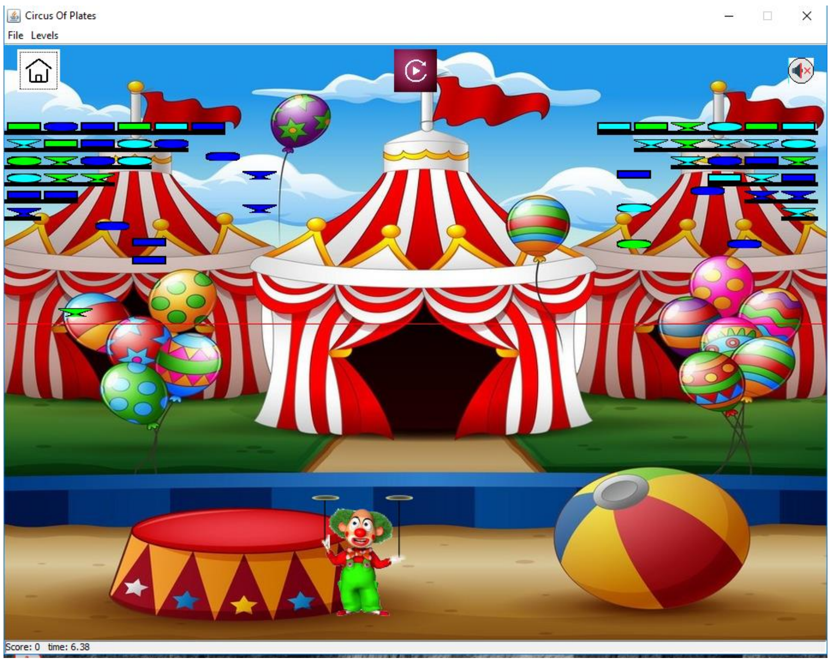
3- On clicking play game the user can choose one of the three difficulties available in the game (Extensible)



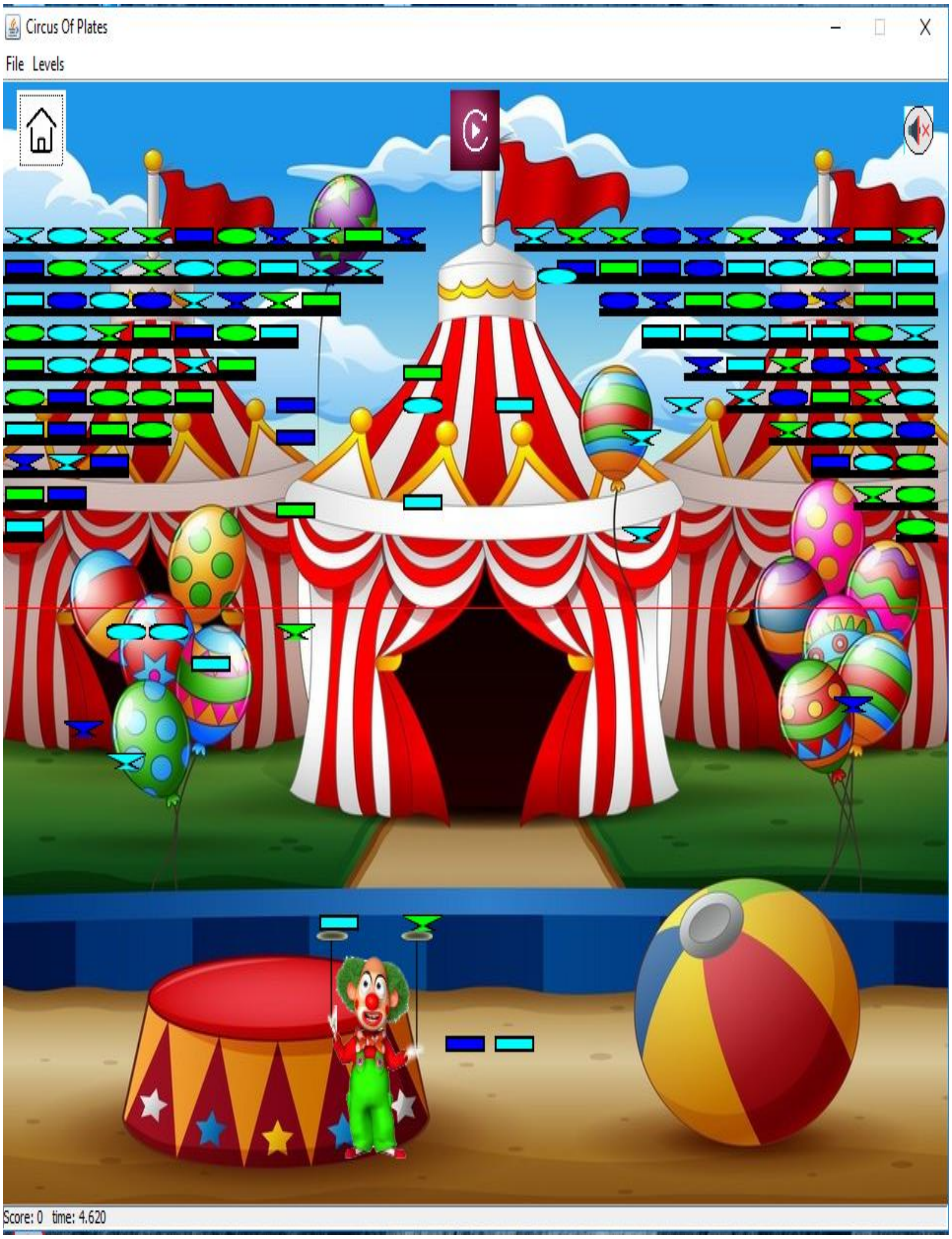
- **Amateur Mode**



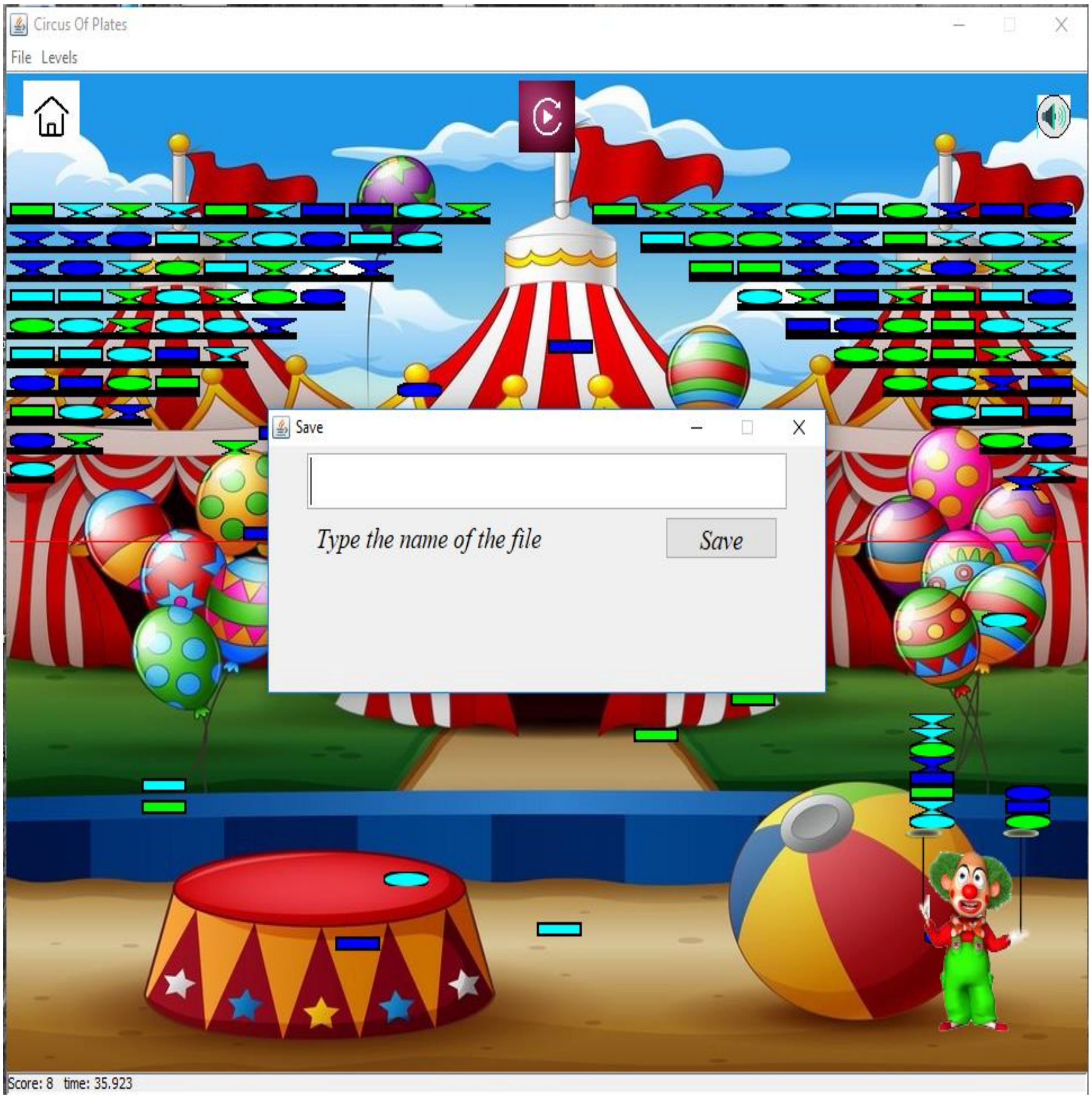
- **Intermediate Mode**



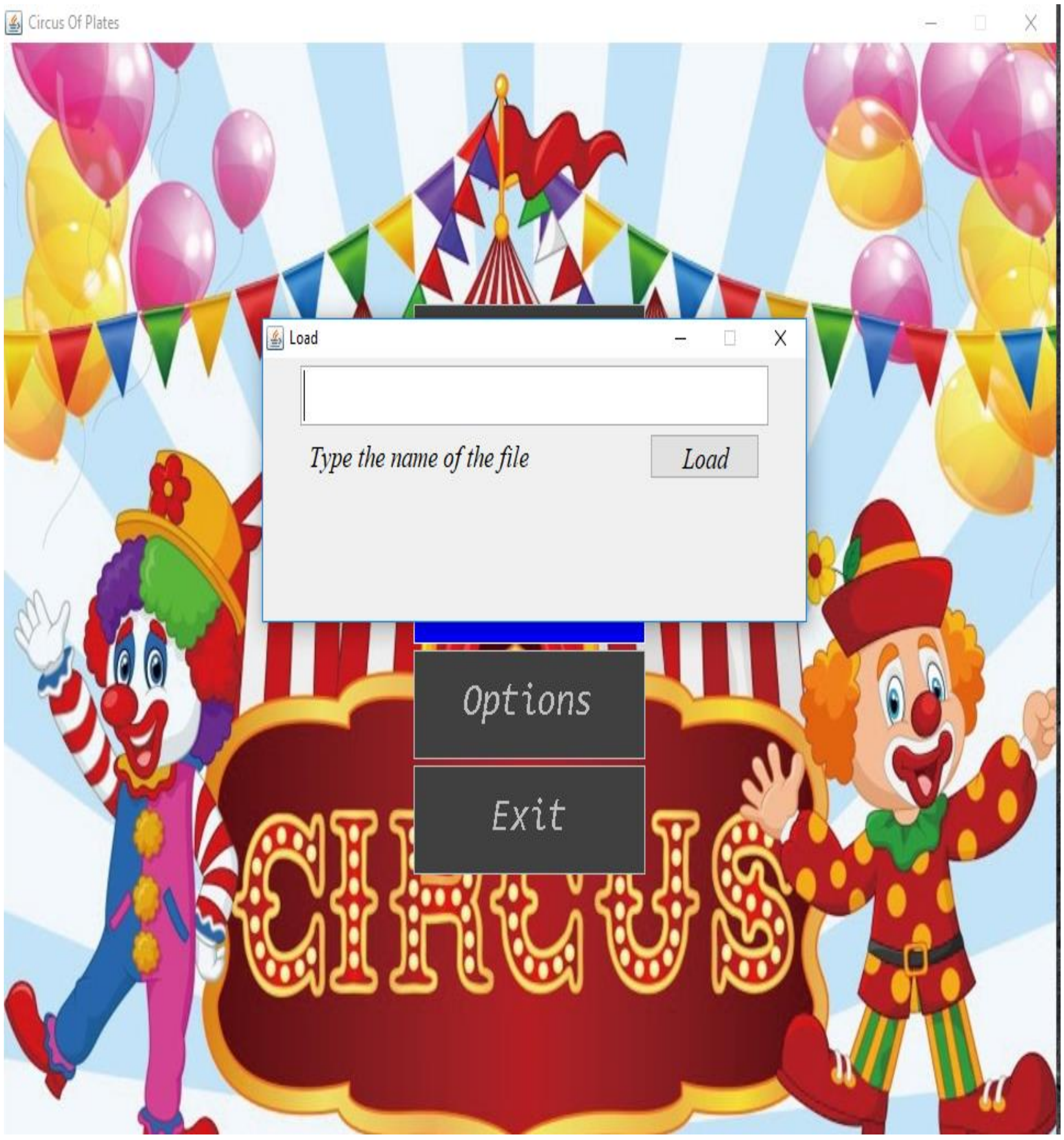
- ## Pro Mode



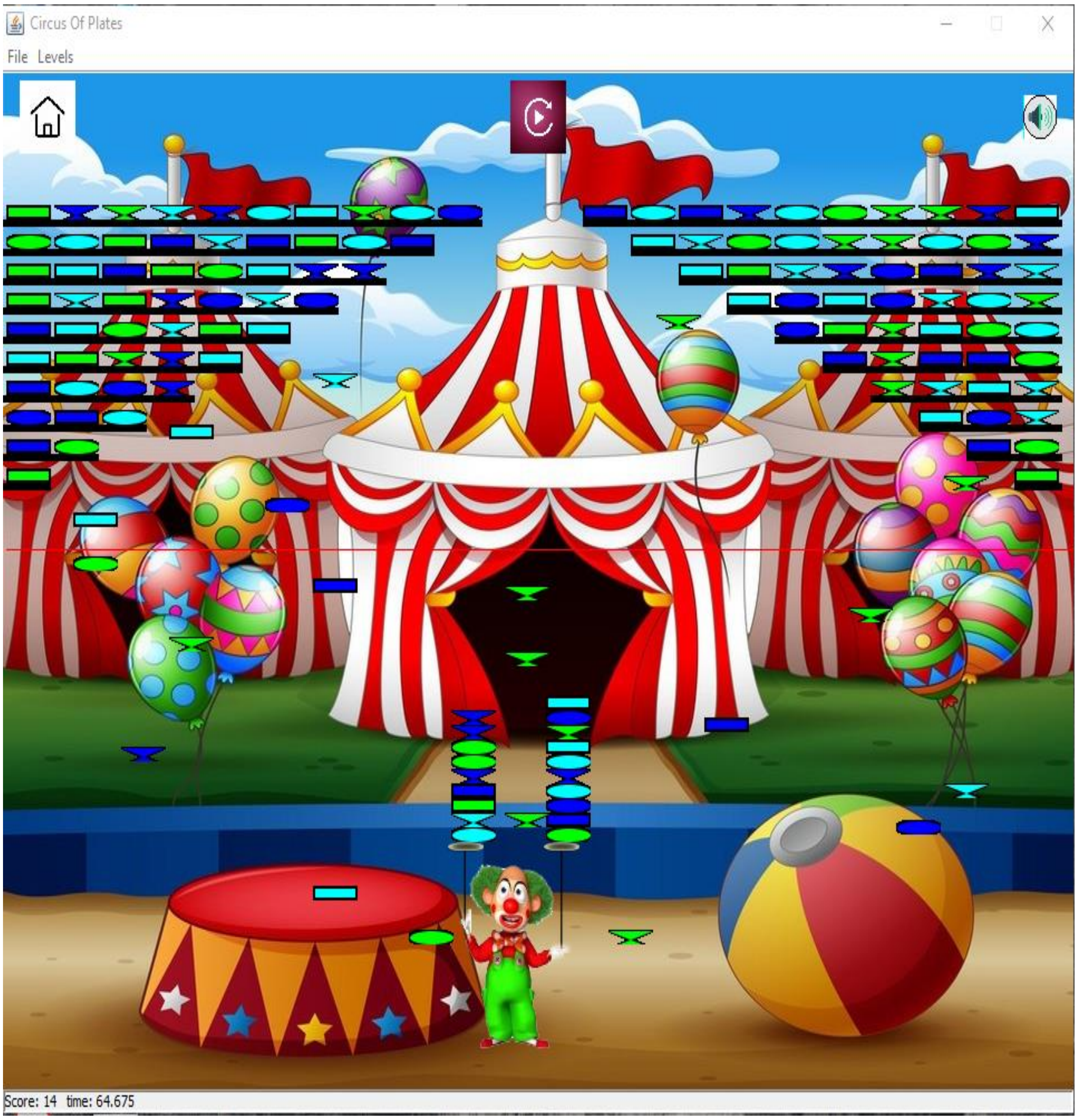
4- At any moment the user may want to save the current state

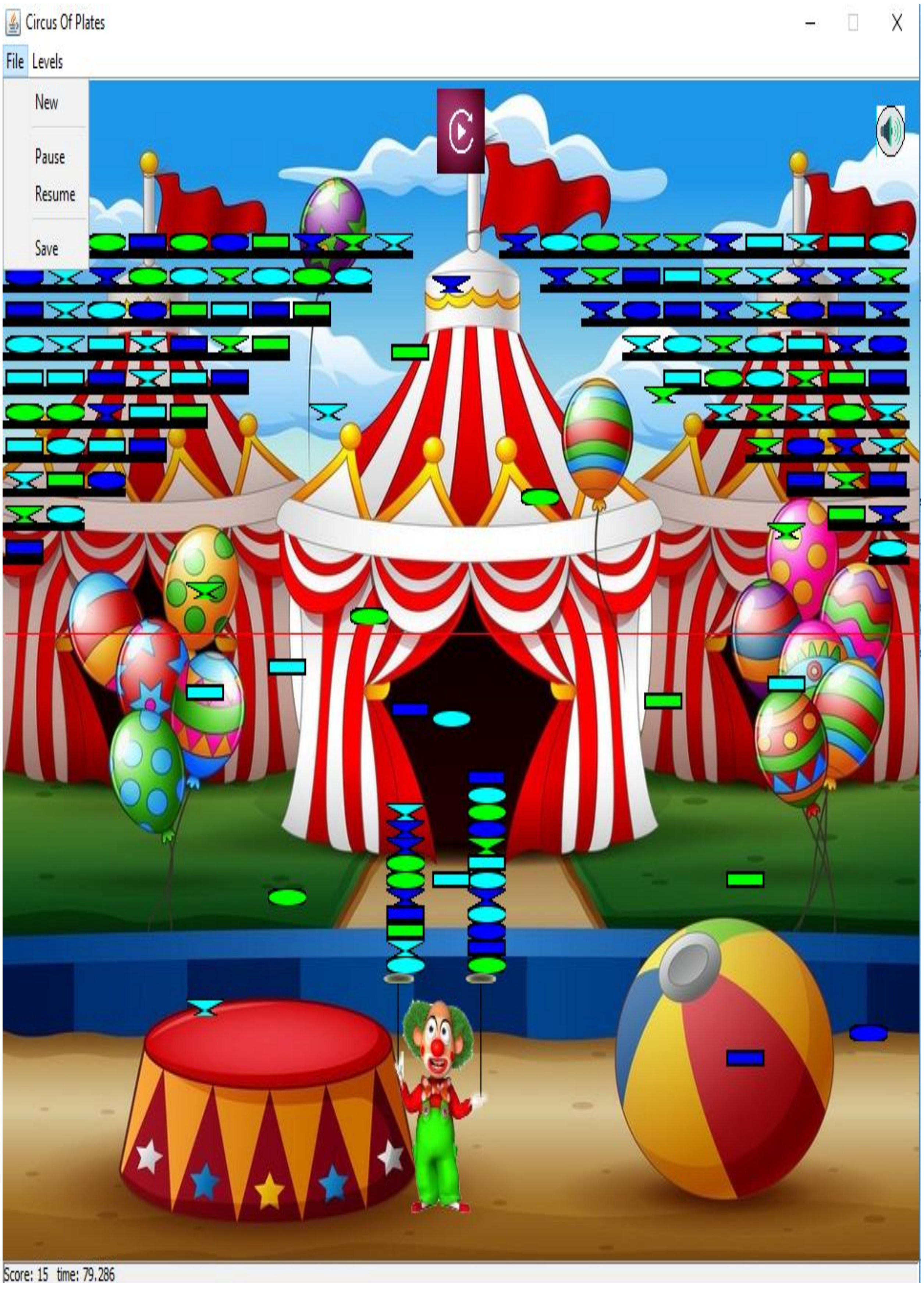


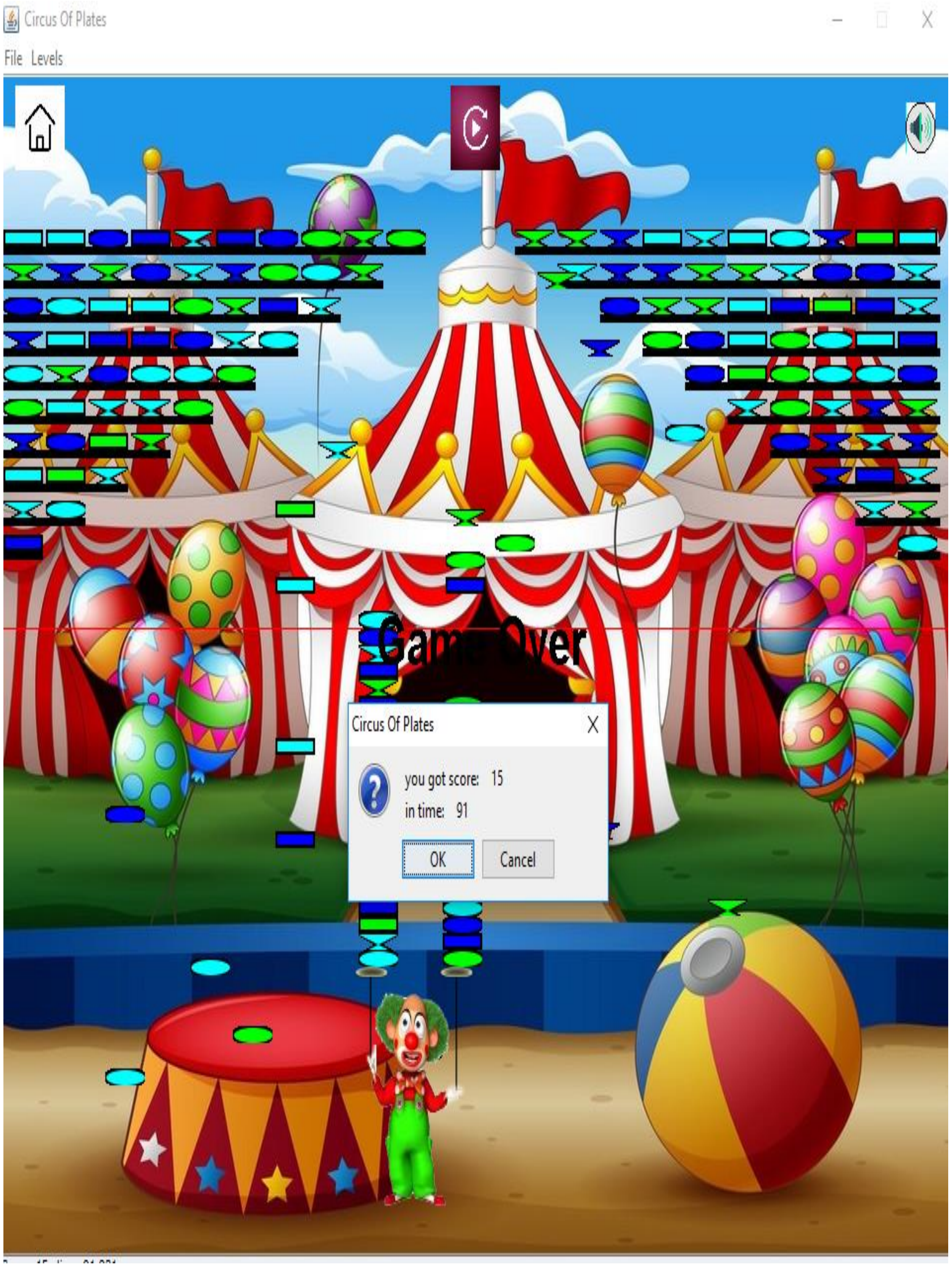
5- For loading



7- Sample Runs




Circus Of Plates
File
Levels
New
Pause
Resume
Save
Score: 15 time: 79.286



Circus Of Plates

you got score: 15

in time: 91