Student Names-IDs:	Abo Bakr AbdElAzeez Ahmad-2 Elsayed Akram Elsayed-16 Seif Eldin Ehab Mostafa-33 Fares Medhat ElSaadawy-47
Lab Title:	Circus of Plates - Game
Drs:	Dr/Khaled Nagi
TA:	Eng/Abdelrahman Hany

Computer and Systems Engineering Department

1 Introduction

It is single player-game in which each 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.

2 Game Rules

2.1 Score Increment

The score increases only when the user catches 3 consecutive plates of the same color on the same stick regardless of their shape.
The score increases by 10 points for every plate which makes it 30 point total for each collection.

2.2 Score Decrement

The score decreases only when the user drops a plate from his stick which happens when a plate touches the corner of the plate below it.
The score decreases by 10 points for each plate he drops in condition that the score never gets below zero.

2.3 Game Over

The game ends in 2 ways:

1. If the timer reaches zero.
2. If the plates on either of the sticks reach the maximum height.

After the game ends the screen shows the user his score along with options for starting a new game or exit the game.

3 Design Description

3.1 Flow

1. Fetching the screen resolution in order to use the width and height dynamically in every class.
2. Creating main menu user interface which generate buttons and wait for the user to take an action.
3. After the user chooses game level it creates option menu which let the user chooses the shape of the plate in order to be loaded dynamically using its JAR file.
4. Initializing the game after creating the circus world using the chosen strategy with suitable objects.
5. Creating menu bar for the game engine.

6. Starting the game using the game engine.
7. After the game ends it creates end menu giving the user options to start new game or exit.

3.2 MVC Architectural Pattern

3.3 Model

- Using the suitable design patterns for creating suitable objects and organizing the code.
- In Circus class initialize main objects.
- Refresh function which:
    1. checks if the time has ended
    2. checks if plates height reached the top
    3. iterate through each plate determine its state if it is MOVING set its new Y and release back to top
    4. if it reached the bottom then checks if the clown caught a new plate after that it checks
    5. if top 3 plates are of the same color
    6. if it is CAUGHT it sets its new X
    7. if a plate drops on another plate touching small part of it, then the plate state is changed to FALLING and falls from the clown stick
    8. if it is FALLING set the new Y without checking for intersection

3.4 View

1. Generate main menu used at the beginning.
2. Generate buttons used in the menus.
3. Generate option menu for the plate shape.
4. Generate menu bar for the engine to be used in the game.
5. Generate end menu after the game ends.

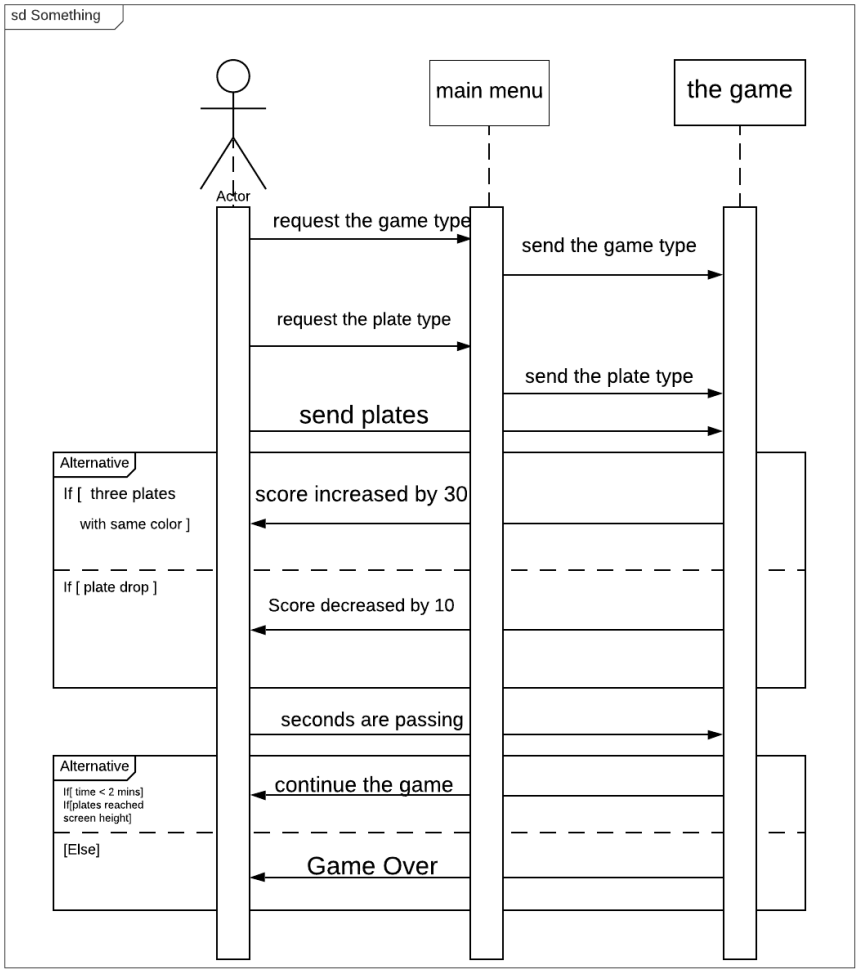
3.5 Controller

Creates game menu which takes the input from the user taken through the View and sends to the model to create the skeleton.

4 Class Diagram

Part of class diagram is displayed in Design Patterns Used section and the file ClassDiagram.ucls is in project folder.

5 Sequence Diagram



6 Design Patterns Used

****Design decisions are bold****
Assumptions are italic

6.1 Singleton

The singleton design pattern is used to get only one instance of multiple classes like:

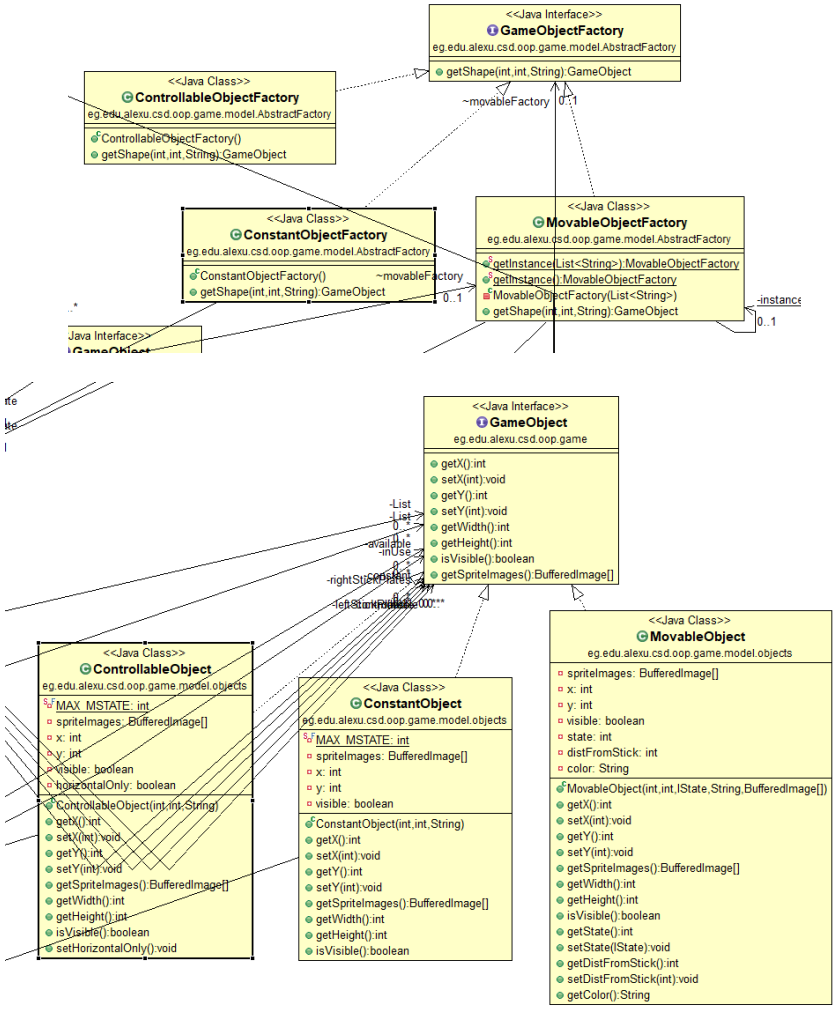
1. GameLogger which generates log of the operations done on the game.
2. **FlyweightShapeFactory which must be instantiated once to load jars through DynamicLinkage class only one time.**

3. **MovableFactory class which creates objects of moving game objects and get them from FlyweightShapeFactory, though it must instantiated once to pass jars the required jars.**

6.2 Abstract Factory

The abstract factory design pattern is used to provide an interface (GameObjectFactory) for creating families of dependent objects without specifying their concrete classes like:

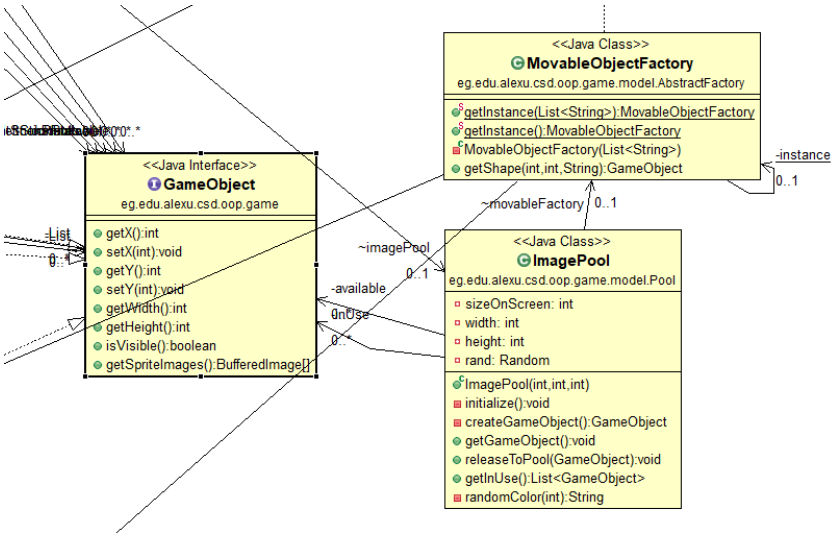
1. Constant Object Factory.
2. Controllable Object Factory.
3. Movable Object Factory.



6.3 Object Pool

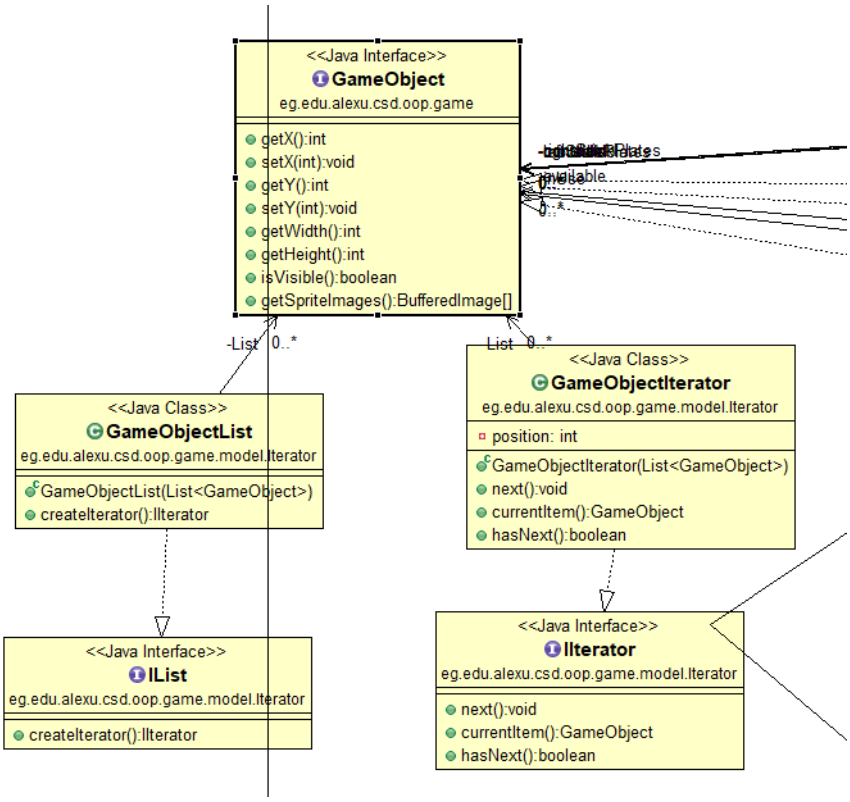
The object pool design pattern is used to keep movable objects in game ready to use, rather than allocating and destroying them on demand.
They are initialized in a "Pool".

- **The object is released back to the "pool" when a movable object reaches the bottom of the screen.**
- **Or when there are three movable objects of same color above each other on a clown stick.**
- **New movable objects are get from the pool when there are not enough game objects in game.**



6.4 Iterator

The iterator design pattern is used to provide a way to access the elements of movable objects sequentially without exposing the underlying representation of these game objects.

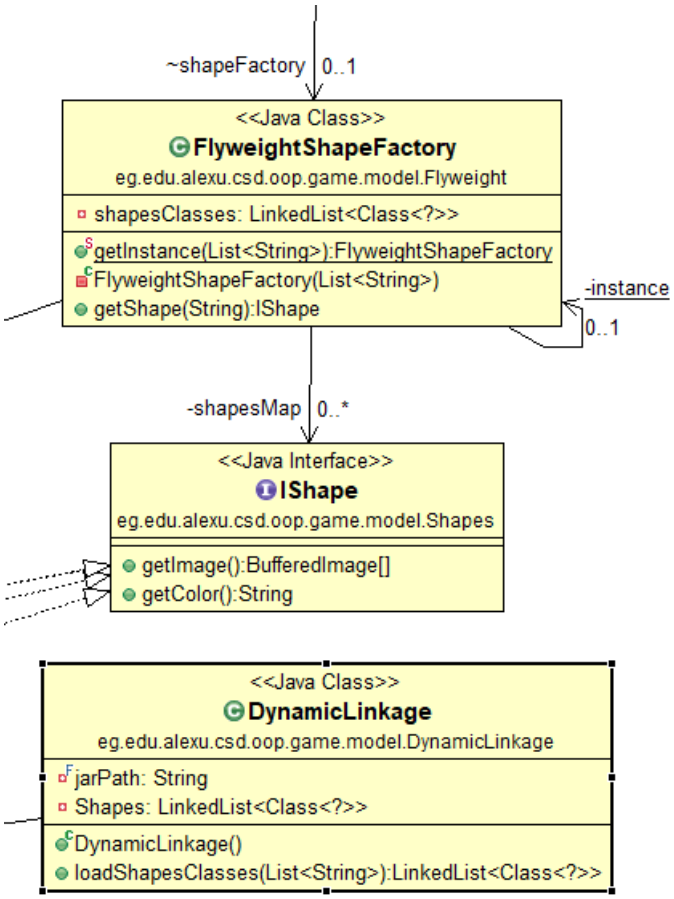


6.5 Dynamic Linkage

The dynamic linkage design pattern is used to allow the game to load and use shapes as arbitrary classes that implement the IShape interface like:

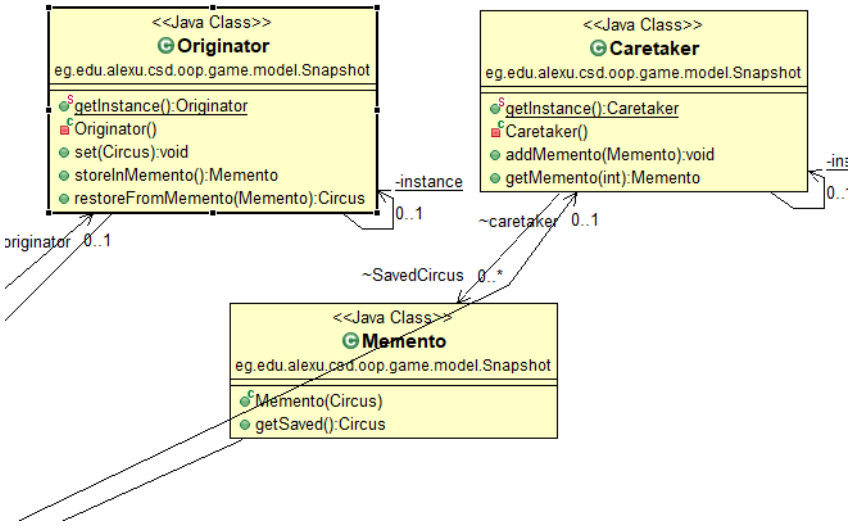
1. Plate with base shape.
2. Plate with deep base shape.
3. Plate without base shape.
4. Pot shape.

The jar files are located in a folder called Jars in project.



6.6 Snapshot

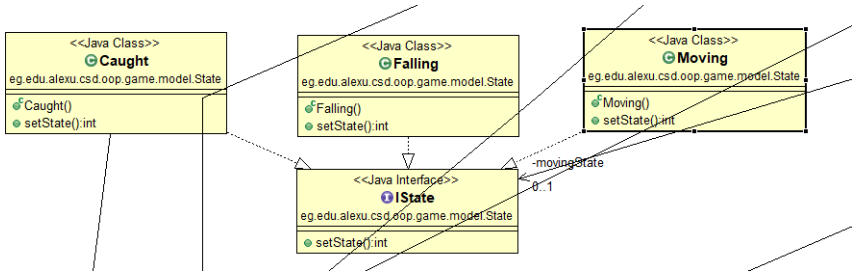
The memento design pattern is used to capture and externalize the circus class (which implements the world interface) internal state so that it can be restored to this state later without violating encapsulation.



6.7 State

The state design pattern is used to set the current state of the any movable plate either it is MOVING, CAUGHT or FALLING which determines the methods used on the plate according to its state.

- **MOVING state is set when the plate is originally moving downwards through the screen waiting to be caught by the clown.**
- **CAUGHT state is set when the moving plate has been caught by the clown using his stick or the last plate on the stick.**
- **FALLING state is set when the plate falls from the stick if new plates touches it from the corner.**



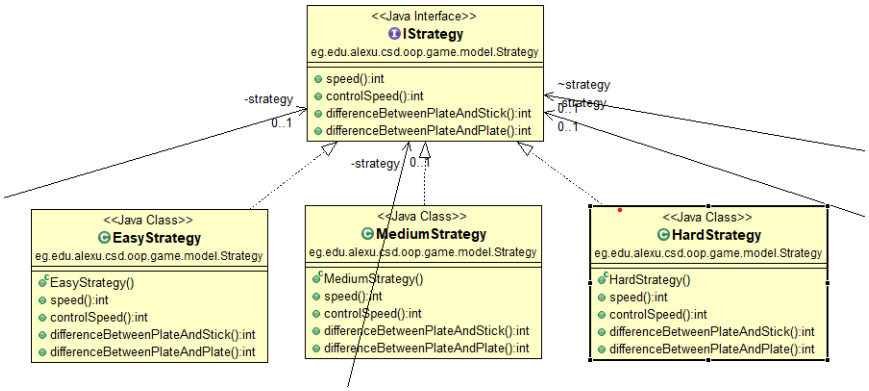
6.8 Strategy

The strategy design pattern is used in order to decide the level of the game which decides:

1. The speed of the falling plates
2. The speed of controlling the clown

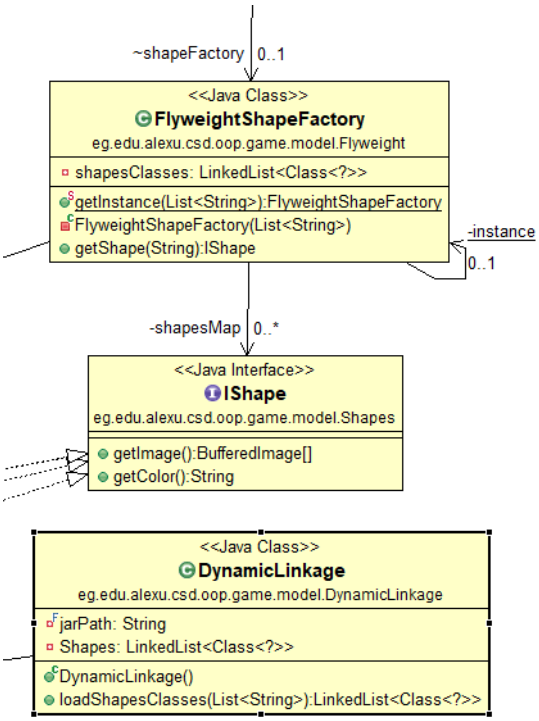
3. The difference between the the plate and stick in order to be caught
4. Or the difference between the plate and plate in order to be caught

Easy Strategy for easy level, Medium Strategy for normal level and Hard Strategy for hard level.



6.9 Flyweight

The flyweight design pattern is used to store the loaded shapes from jar files and use sharing to support large numbers of movable objects efficiently.



6.10 Observer

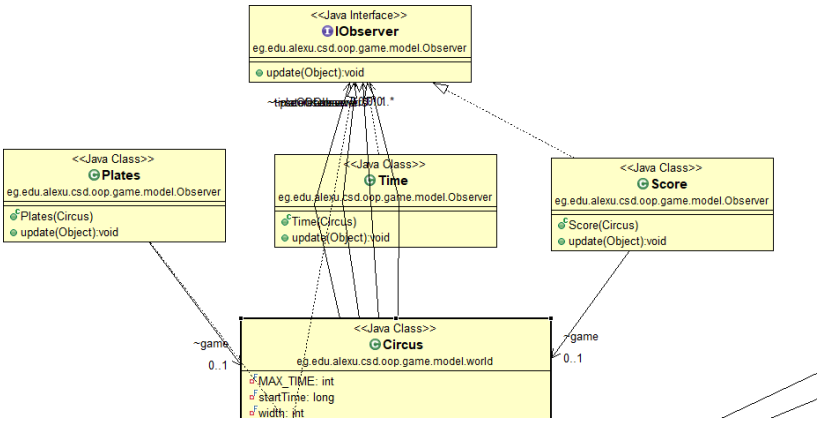
The observer design pattern is used define a one-to-many dependency between objects like:

- Circus class

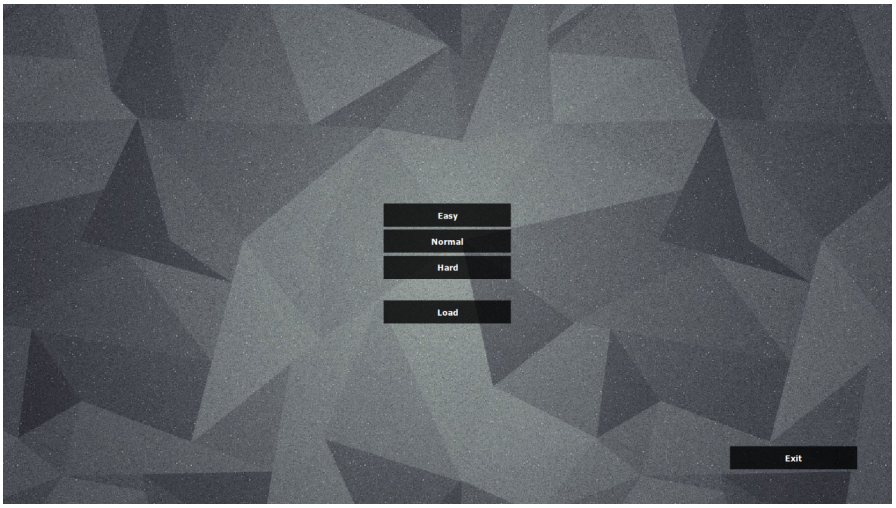
so that when the world class changes state, all its dependents like:

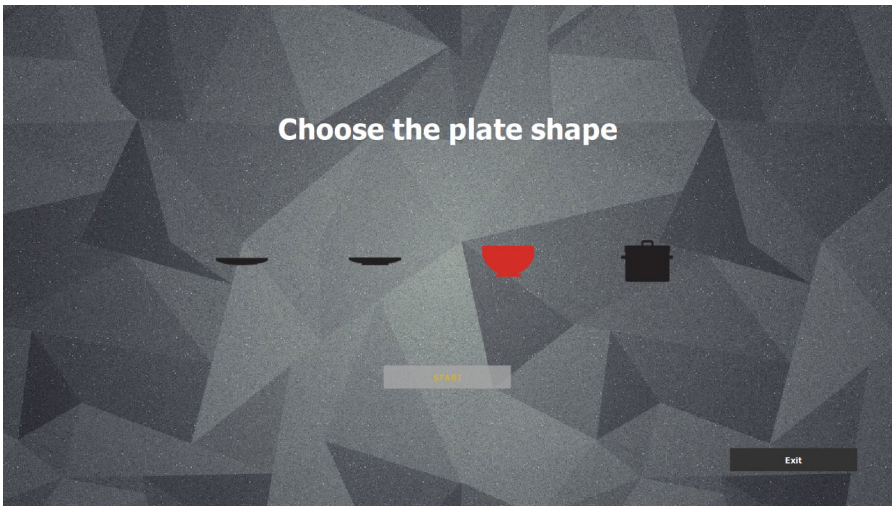
1. Score
2. Time
3. Plates

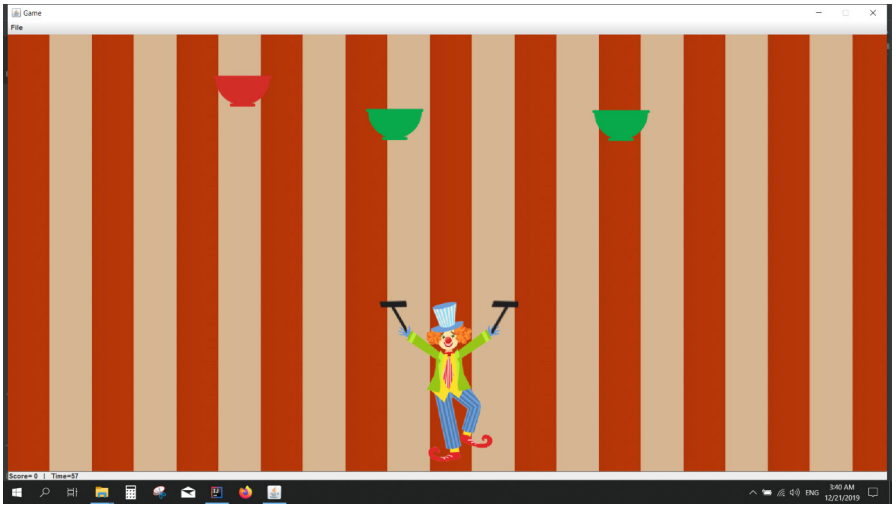
are notified and updated automatically.

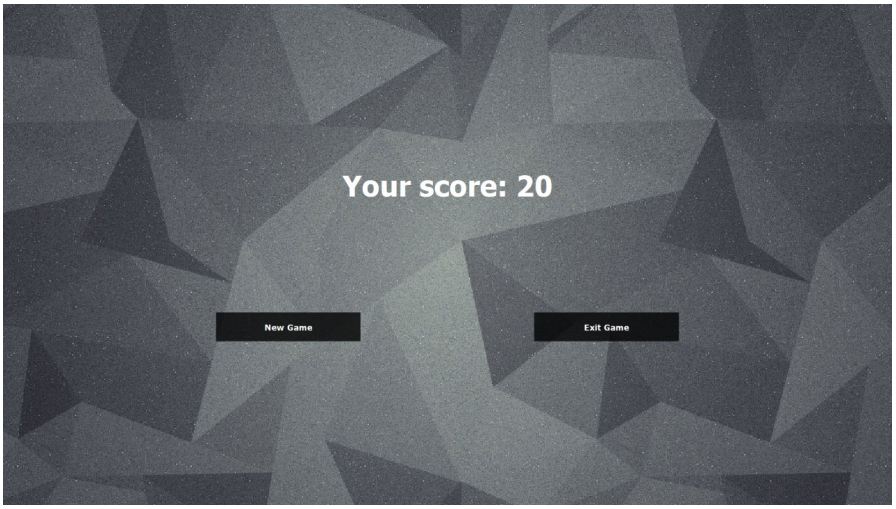


7 Sample Runs




Choose the plate shape
START
Exit


Game
File
Score= 0 | Time=57
ENG
3:40 AM
12/21/2019


Your score: 20
New Game
Exit Game