Programming II – Final Project

Circus of Plates

Team:

1- Sophia Samir Tawfik Arida - 7428

2- Jana Essam Abdelmoneim Soliman Elnagar – 7623

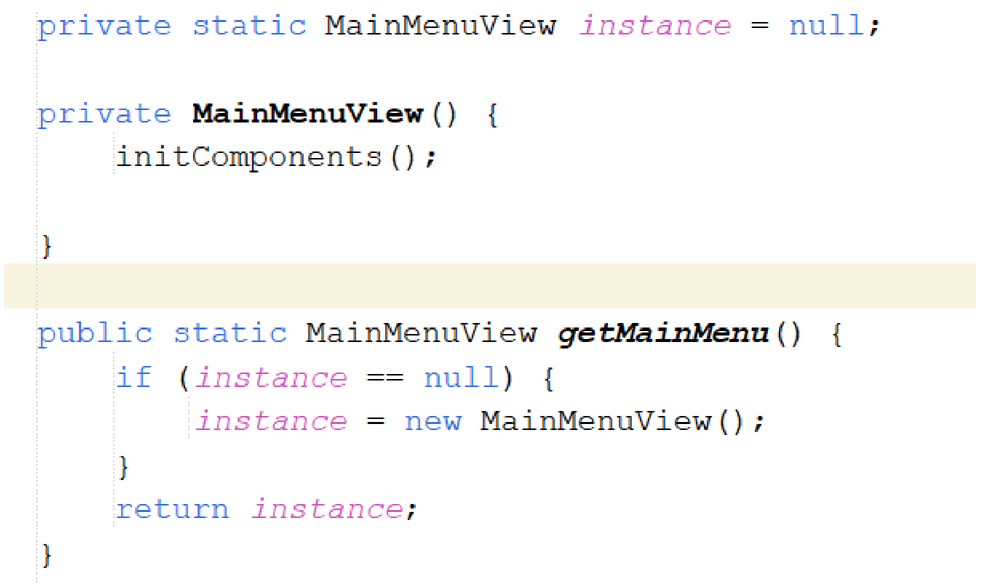
3- Hana Waleed Farouk Mohamed Mohamed Ghanem – 7599

4- Mennatallah Mahmoud Elsayed Mansour – 7618

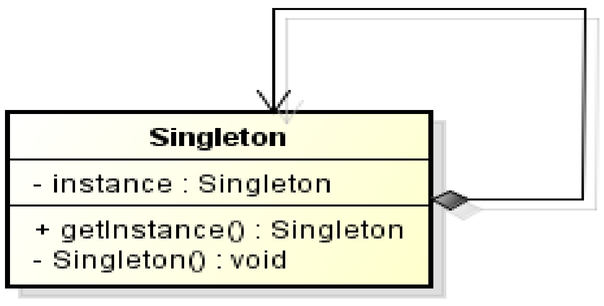
Part 1 – Design Patterns

1) Singleton Pattern

The singleton design pattern was used throughout the project, especially in the view frames

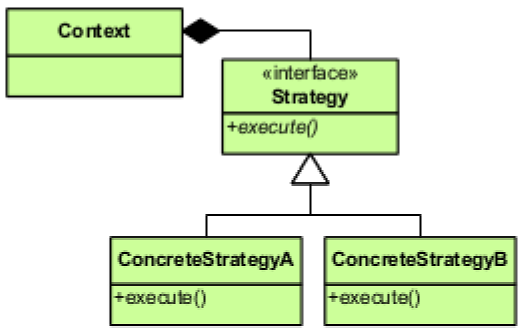


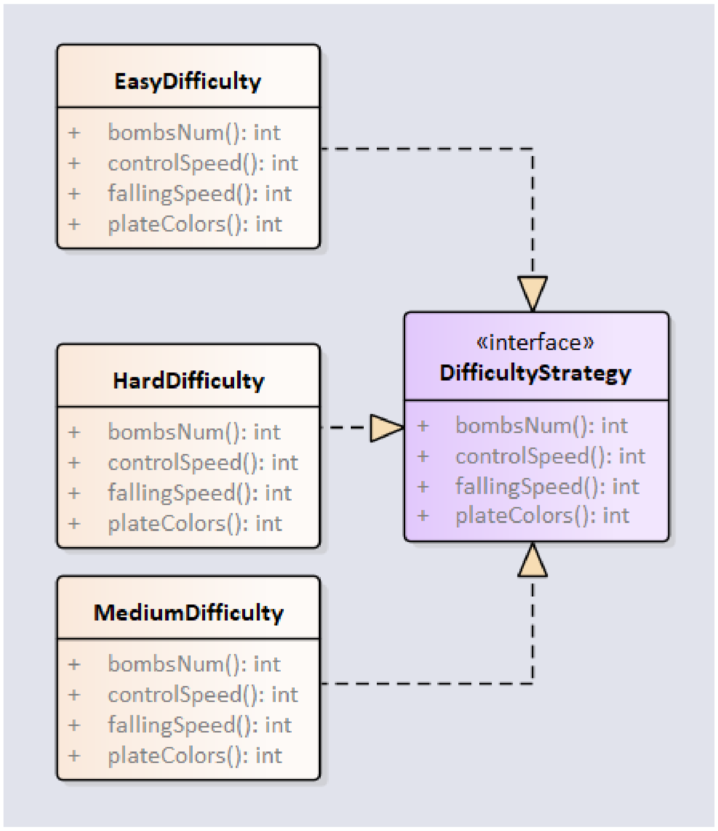
```
private static MainMenuView instance = null;

private MainMenuView() {
    initComponents();

}

public static MainMenuView getMainMenu() {
    if (instance == null) {
        instance = new MainMenuView();
    }
    return instance;
}
```

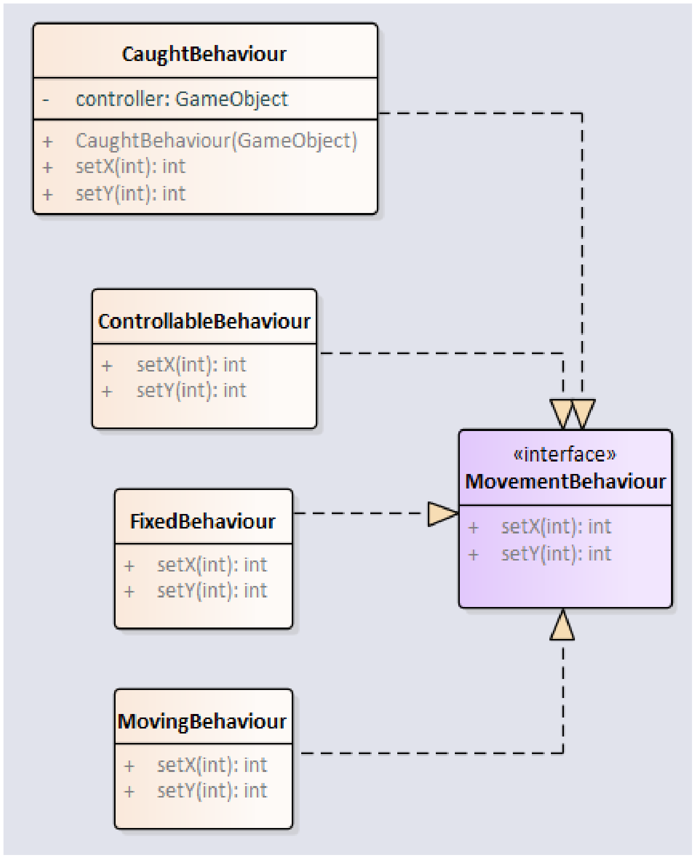


2) Strategy Design Pattern

The strategy pattern was used to determine the game difficulty (easy, medium, & hard), and also the gameObject’s movement behavior (fixed, moving, caught, & controllable)



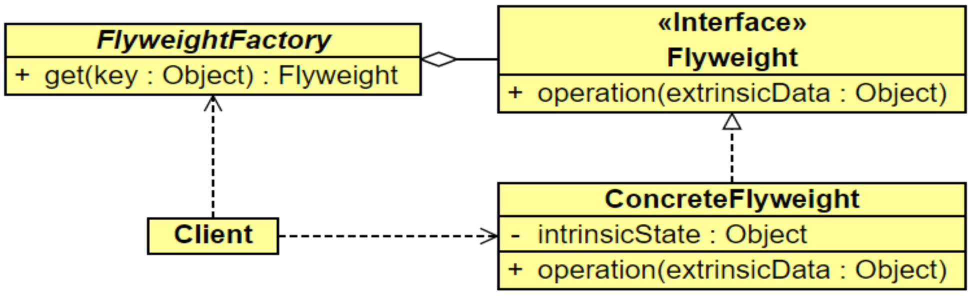

EasyDifficulty
+ bombsNum(): int
+ controlSpeed(): int
+ fallingSpeed(): int
+ plateColors(): int
HardDifficulty
+ bombsNum(): int
+ controlSpeed(): int
+ fallingSpeed(): int
+ plateColors(): int
MediumDifficulty
+ bombsNum(): int
+ controlSpeed(): int
+ fallingSpeed(): int
+ plateColors(): int
«interface»
DifficultyStrategy
+ bombsNum(): int
+ controlSpeed(): int
+ fallingSpeed(): int
+ plateColors(): int

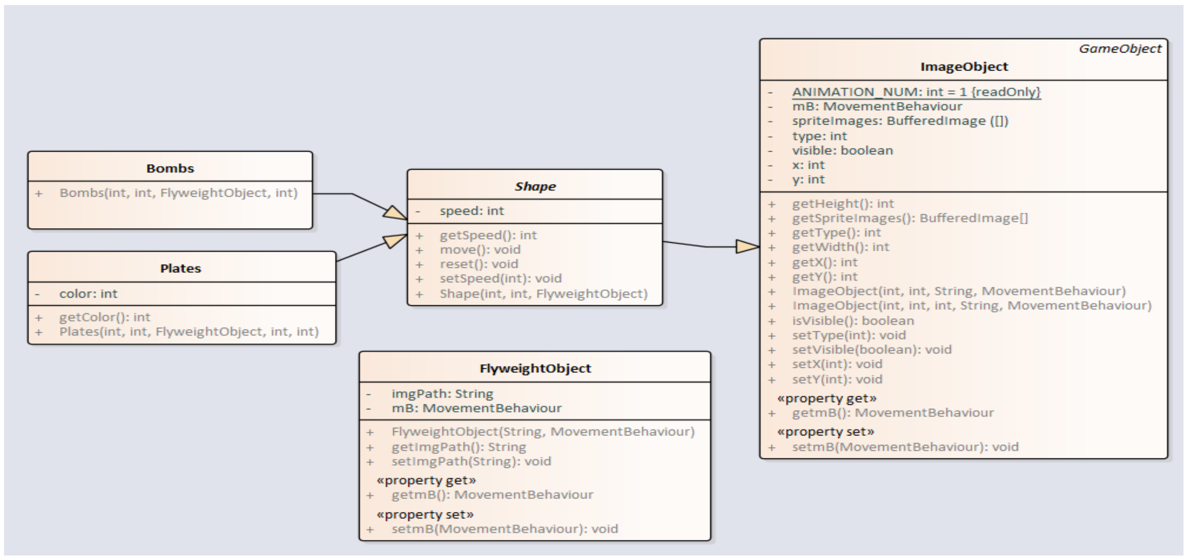

CaughtBehaviour
- controller: GameObject
+ CaughtBehaviour(GameObject)
+ setX(int): int
+ setY(int): int
ControllableBehaviour
+ setX(int): int
+ setY(int): int
FixedBehaviour
+ setX(int): int
+ setY(int): int
MovingBehaviour
+ setX(int): int
+ setY(int): int
«interface»
MovementBehaviour
+ setX(int): int
+ setY(int): int

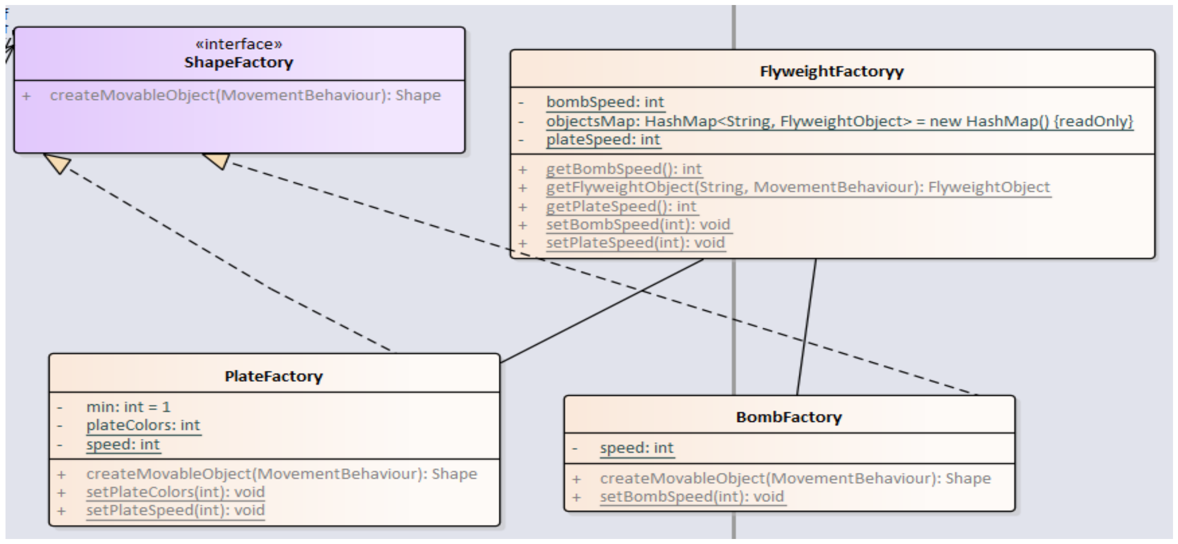
3 & 4) Pool/Factory Pattern and Flyweight Pattern

Both the factory and flyweight patterns were integrated together to create an object factory for plates and bombs that returns only one object of each type to be continuously reused.

Two factories were created, plateFactory and bombFactory, and they call on the flyweightFactory passing to it the common attributes (imagePath and movementBehavior) and the flyweightFactory returns the created flyweight object.

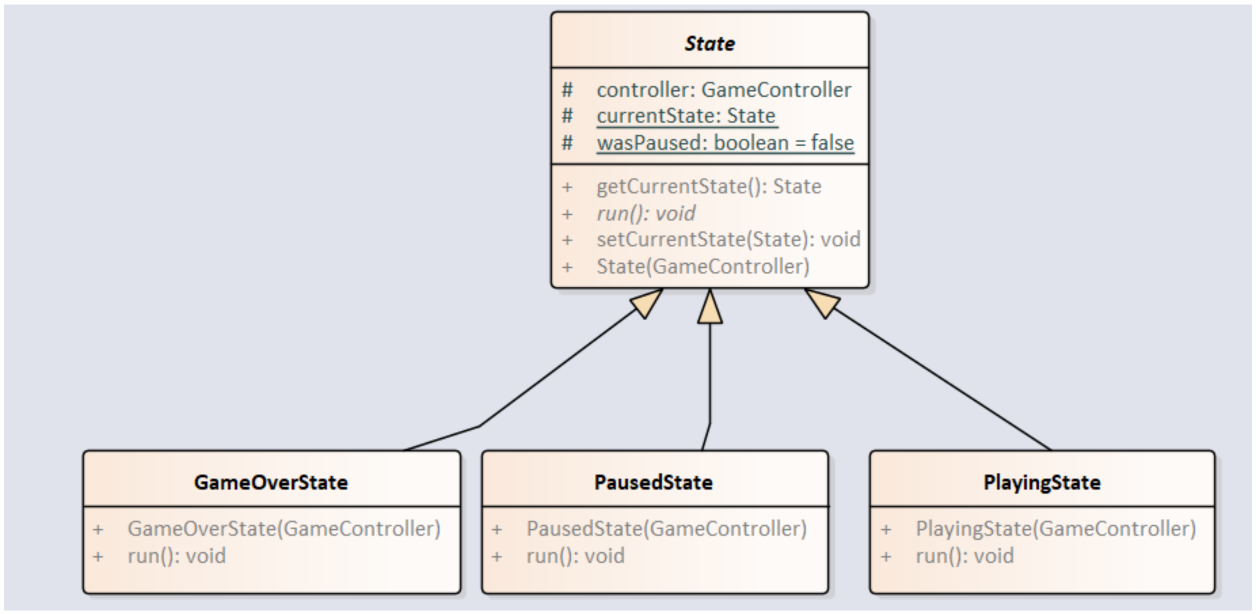

FlyweightFactory
+ get(key : Object) : Flyweight
«Interface»
Flyweight
+ operation(extrinsicData : Object)
ConcreteFlyweight
- intrinsicState : Object
+ operation(extrinsicData : Object)
Client


Bombs
+ Bombs(int, int, FlyweightObject, int)
Plates
- color: int
+ getColor(): int
+ Plates(int, int, FlyweightObject, int, int)
Shape
- speed: int
+ getSpeed(): int
+ move(): void
+ reset(): void
+ setSpeed(int): void
+ Shape(int, int, FlyweightObject)
FlyweightObject
- imgPath: String
- mB: MovementBehaviour
+ FlyweightObject(String, MovementBehaviour)
+ getImgPath(): String
+ setImgPath(String): void
«property get»
+ getmB(): MovementBehaviour
«property set»
+ setmB(MovementBehaviour): void
GameObject
ImageObject
- ANIMATION_NUM: int = 1 {readOnly}
- mB: MovementBehaviour
- spriteImages: BufferedImage ([])
- type: int
- visible: boolean
- x: int
- y: int
+ getHeight(): int
+ getSpriteImages(): BufferedImage[]
+ getType(): int
+ getWidth(): int
+ getX(): int
+ getY(): int
+ ImageObject(int, int, String, MovementBehaviour)
+ ImageObject(int, int, int, String, MovementBehaviour)
+ isVisible(): boolean
+ setType(int): void
+ setVisible(boolean): void
+ setX(int): void
+ setY(int): void
«property get»
+ getmB(): MovementBehaviour
«property set»
+ setmB(MovementBehaviour): void


«interface»
ShapeFactory
+ createMovableObject(MovementBehaviour): Shape
FlyweightFactoryy
- bombSpeed: int
- objectsMap: HashMap<String, FlyweightObject> = new HashMap() {readOnly}
- plateSpeed: int
+ getBombSpeed(): int
+ getFlyweightObject(String, MovementBehaviour): FlyweightObject
+ getPlateSpeed(): int
+ setBombSpeed(int): void
+ setPlateSpeed(int): void
PlateFactory
- min: int = 1
- plateColors: int
- speed: int
+ createMovableObject(MovementBehaviour): Shape
+ setPlateColors(int): void
+ setPlateSpeed(int): void
BombFactory
- speed: int
+ createMovableObject(MovementBehaviour): Shape
+ setBombSpeed(int): void

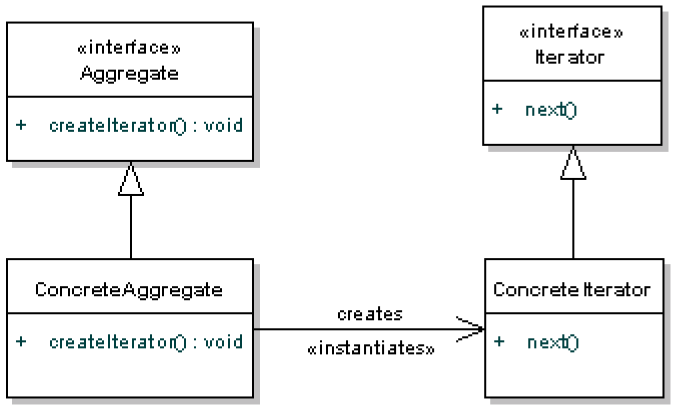
5) State Design Pattern

The state design pattern was used to determine the game's state, whether it was running, paused, or Game Over (ended).


State
# controller: GameController
# currentState: State
# wasPaused: boolean = false
+ getCurrentState(): State
+ run(): void
+ setCurrentState(State): void
+ State(GameController)
GameOverState
+ GameOverState(GameController)
+ run(): void
PausedState
+ PausedState(GameController)
+ run(): void
PlayingState
+ PlayingState(GameController)
+ run(): void

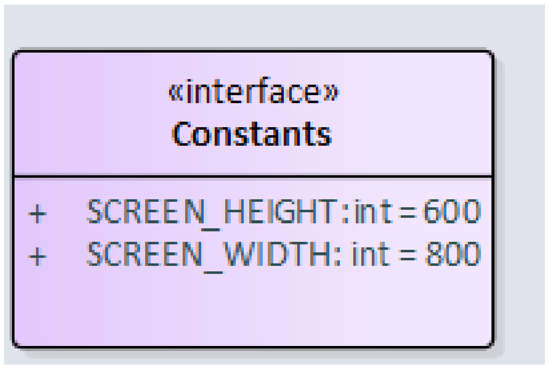
6) Iterator

The iterator was used to iterate over arraylists of objects easily.

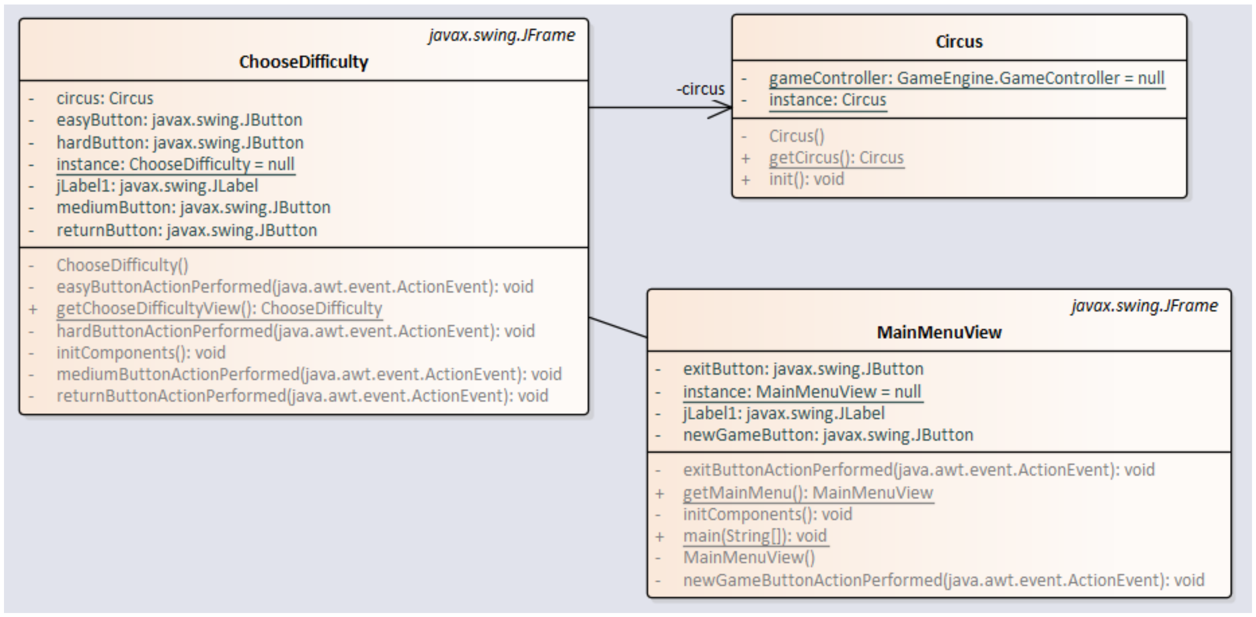


Part 2 – Class Diagrams

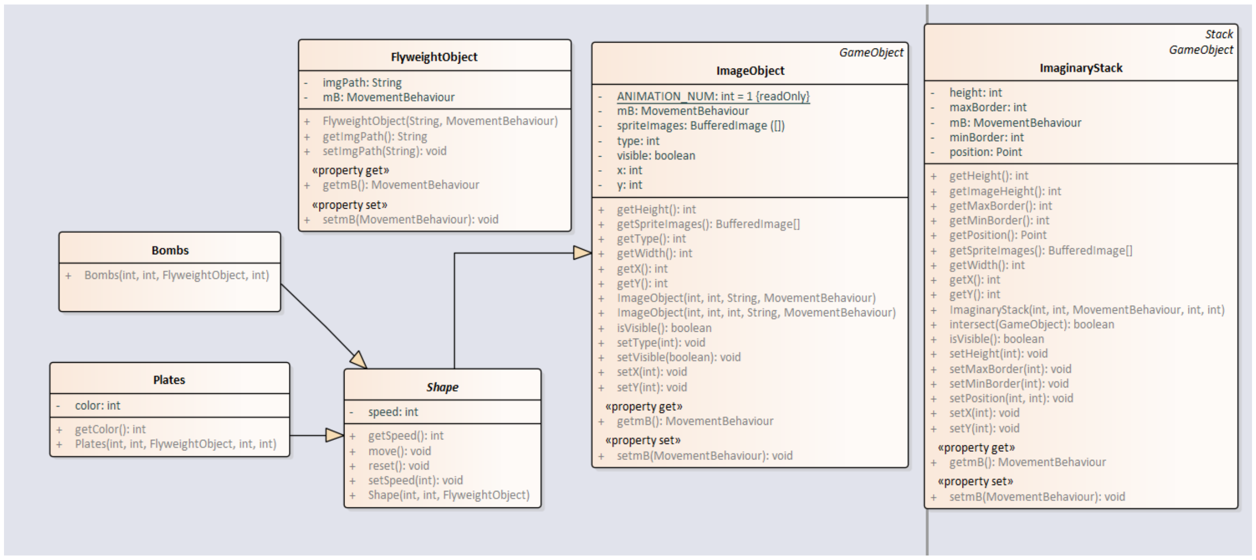
I. Contstants Package



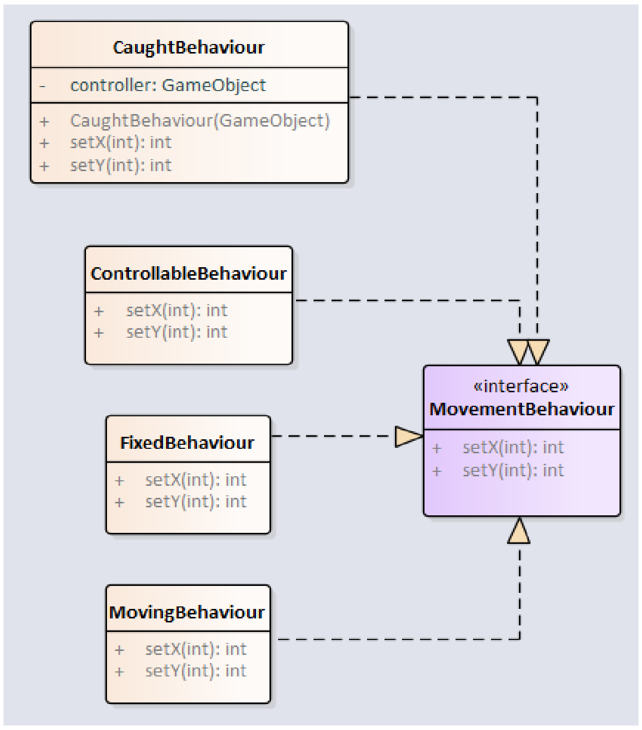
II. View Package

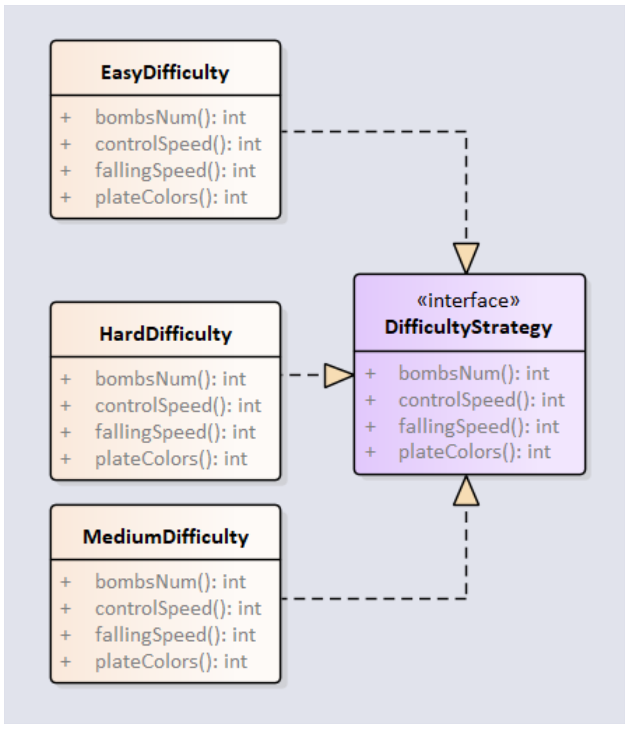

javax.swing.JFrame
ChooseDifficulty
- circus: Circus
- easyButton: javax.swing.JButton
- hardButton: javax.swing.JButton
- instance: ChooseDifficulty = null
- jLabel1: javax.swing.JLabel
- mediumButton: javax.swing.JButton
- returnButton: javax.swing.JButton
- ChooseDifficulty()
- easyButtonActionPerformed(java.awt.event.ActionEvent): void
+ getChooseDifficultyView(): ChooseDifficulty
- hardButtonActionPerformed(java.awt.event.ActionEvent): void
- initComponents(): void
- mediumButtonActionPerformed(java.awt.event.ActionEvent): void
- returnButtonActionPerformed(java.awt.event.ActionEvent): void
-circus
Circus
- gameController: GameEngine.GameController = null
- instance: Circus
- Circus()
+ getCircus(): Circus
+ init(): void
javax.swing.JFrame
MainMenuView
- exitButton: javax.swing.JButton
- instance: MainMenuView = null
- jLabel1: javax.swing.JLabel
- newGameButton: javax.swing.JButton
- exitButtonActionPerformed(java.awt.event.ActionEvent): void
+ getMainMenu(): MainMenuView
- initComponents(): void
+ main(String[]): void
- MainMenuView()
- newGameButtonActionPerformed(java.awt.event.ActionEvent): void

III. Model Package

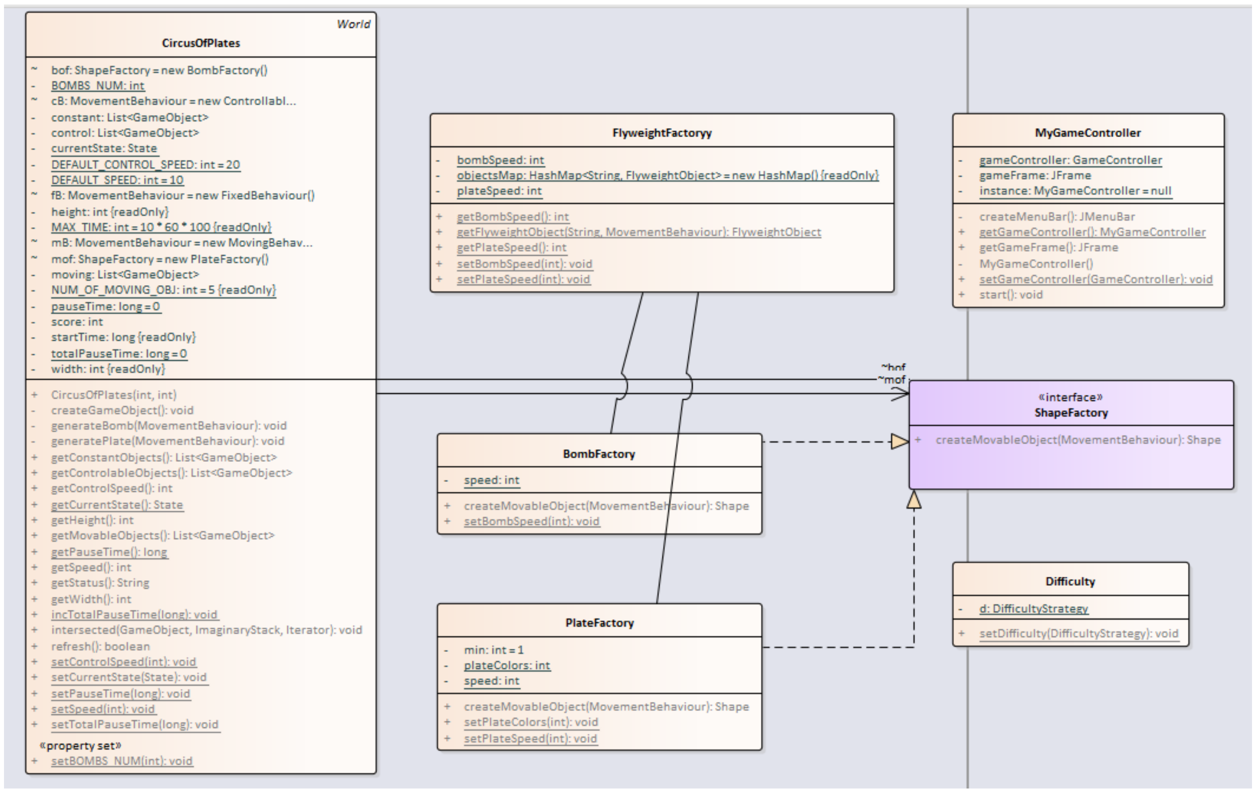


a) Strategy in Model

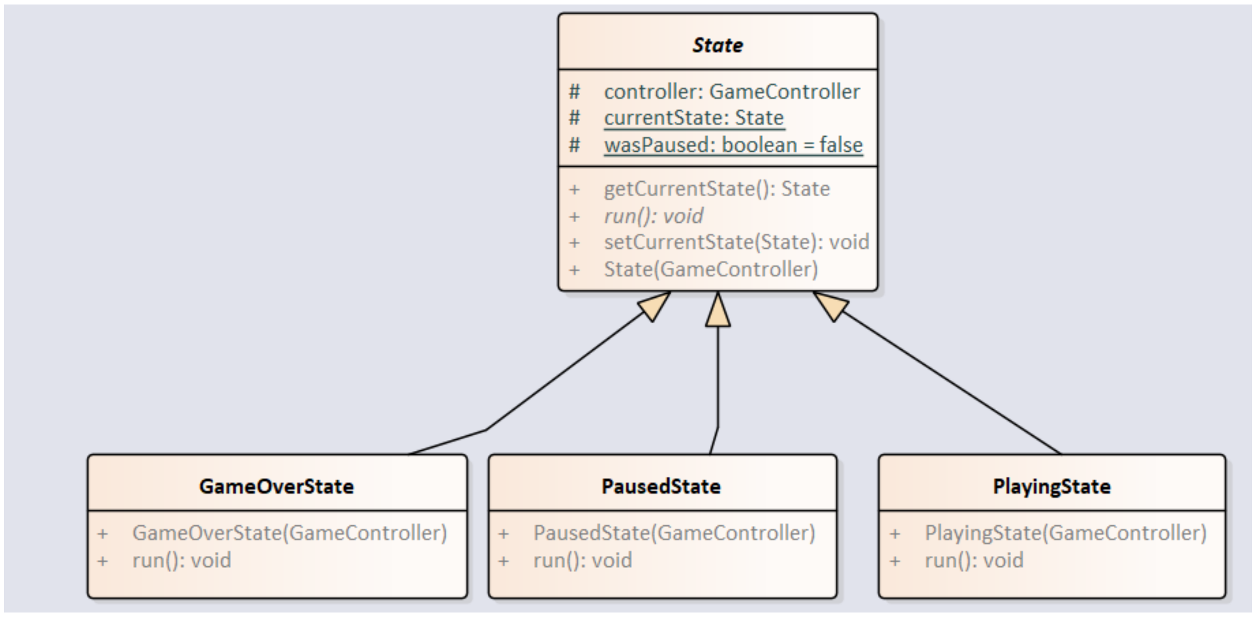




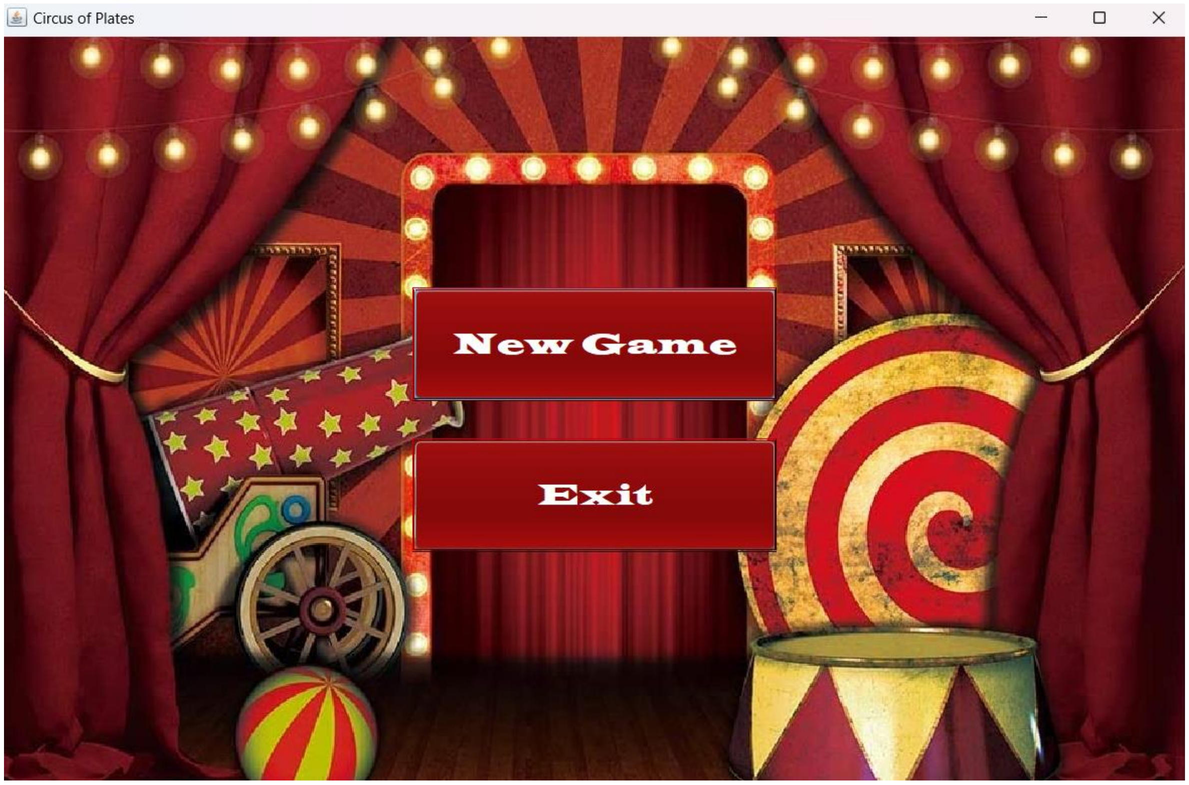
IV. Controller

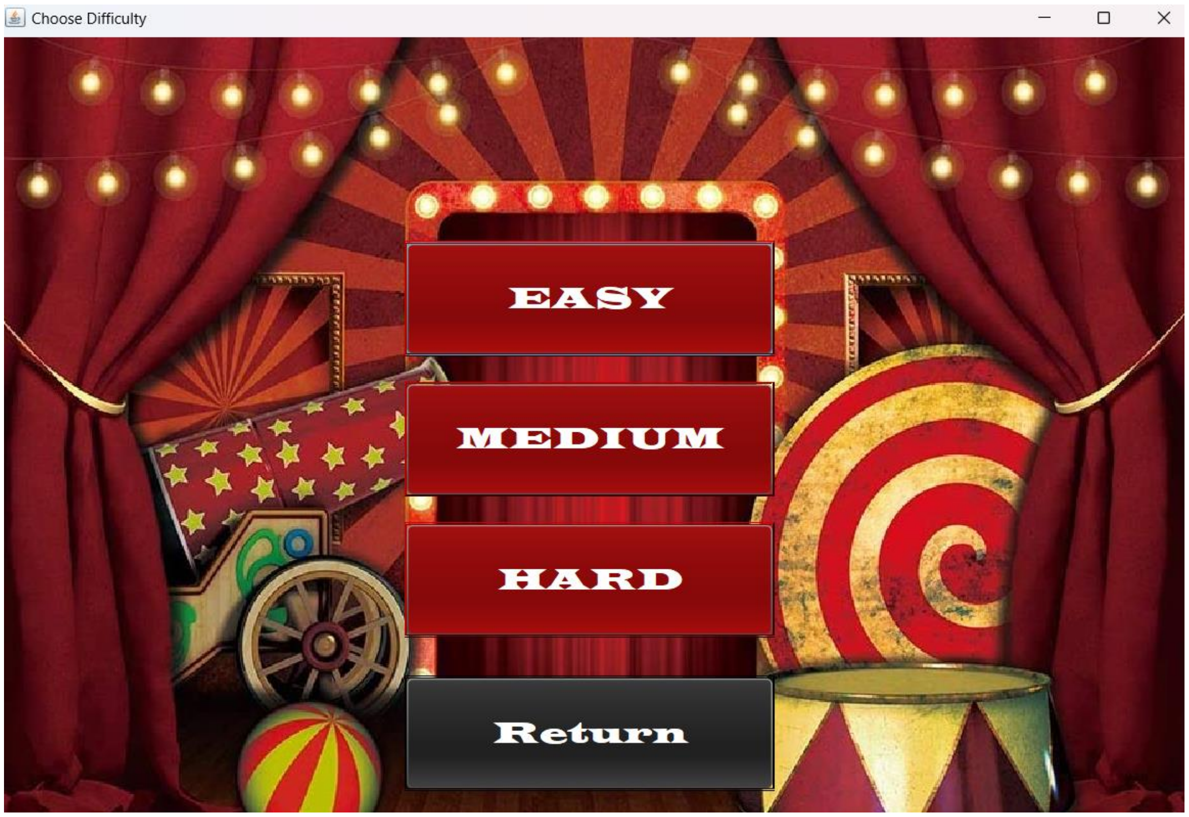

World
CircusOfPlates
~ bof: ShapeFactory = new BombFactory()
- BOMBS_NUM: int
~ cB: MovementBehaviour = new Controllabl...
- constant: List<GameObject>
- control: List<GameObject>
- currentState: State
- DEFAULT_CONTROL_SPEED: int = 20
- DEFAULT_SPEED: int = 10
~ fB: MovementBehaviour = new FixedBehaviour()
- height: int {readOnly}
- MAX_TIME: int = 10 * 60 * 100 {readOnly}
~ mB: MovementBehaviour = new MovingBehav...
~ mof: ShapeFactory = new PlateFactory()
- moving: List<GameObject>
- NUM_OF_MOVING_OBJ: int = 5 {readOnly}
- pauseTime: long = 0
- score: int
- startTime: long {readOnly}
- totalPauseTime: long = 0
- width: int {readOnly}
+ CircusOfPlates(int, int)
- createGameObject(): void
- generateBomb(MovementBehaviour): void
- generatePlate(MovementBehaviour): void
+ getConstantObjects(): List<GameObject>
+ getControlableObjects(): List<GameObject>
+ getControlSpeed(): int
+ getCurrentState(): State
+ getHeight(): int
+ getMovableObjects(): List<GameObject>
+ getPauseTime(): long
+ getSpeed(): int
+ getStatus(): String
+ getWidth(): int
+ incTotalPauseTime(long): void
+ intersected(GameObject, ImaginaryStack, Iterator): void
+ refresh(): boolean
+ setControlSpeed(int): void
+ setCurrentState(State): void
+ setPauseTime(long): void
+ setSpeed(int): void
+ setTotalPauseTime(long): void
«property set»
+ setBOMBS_NUM(int): void
FlyweightFactoryy
- bombSpeed: int
- objectsMap: HashMap<String, FlyweightObject> = new HashMap() {readOnly}
- plateSpeed: int
+ getBombSpeed(): int
+ getFlyweightObject(String, MovementBehaviour): FlyweightObject
+ getPlateSpeed(): int
+ setBombSpeed(int): void
+ setPlateSpeed(int): void
MyGameController
- gameController: GameController
- gameFrame: JFrame
- instance: MyGameController = null
- createMenuBar(): JMenuBar
+ getGameController(): MyGameController
+ getGameFrame(): JFrame
- MyGameController()
+ setGameController(GameController): void
+ start(): void
~bof
~mof
«interface»
ShapeFactory
+ createMovableObject(MovementBehaviour): Shape
BombFactory
- speed: int
+ createMovableObject(MovementBehaviour): Shape
+ setBombSpeed(int): void
Difficulty
- d: DifficultyStrategy
+ setDifficulty(DifficultyStrategy): void
PlateFactory
- min: int = 1
- plateColors: int
- speed: int
+ createMovableObject(MovementBehaviour): Shape
+ setPlateColors(int): void
+ setPlateSpeed(int): void

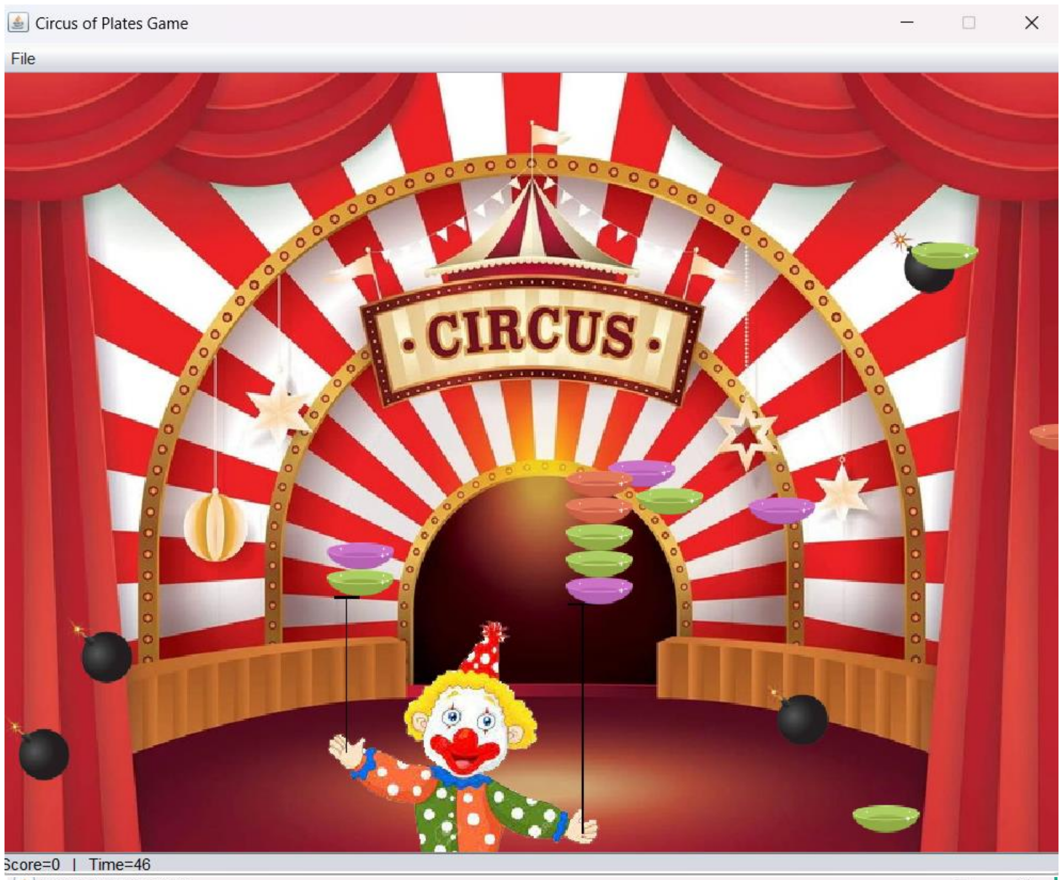
a) State in Controller


State
# controller: GameController
# currentState: State
# wasPaused: boolean = false
+ getCurrentState(): State
+ run(): void
+ setCurrentState(State): void
+ State(GameController)
GameOverState
+ GameOverState(GameController)
+ run(): void
PausedState
+ PausedState(GameController)
+ run(): void
PlayingState
+ PlayingState(GameController)
+ run(): void

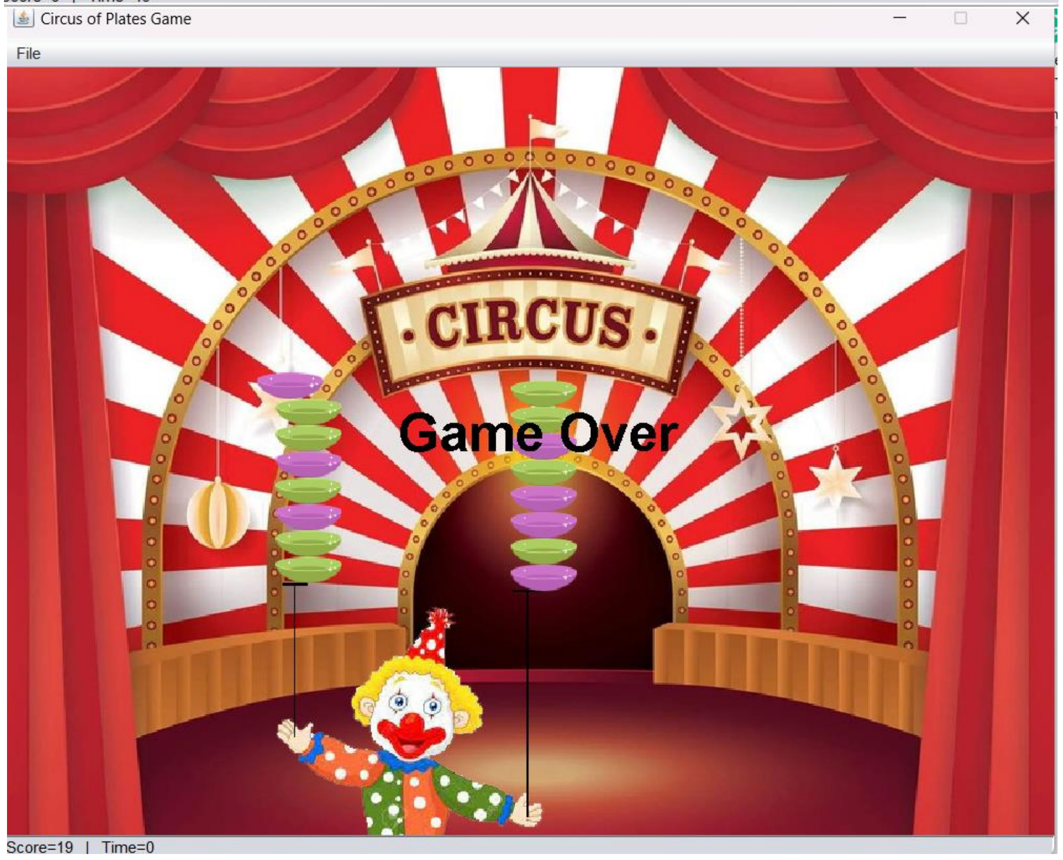
Part 3 – Screenshots







Score=0 | Time=46



Game Over

Score=19 | Time=0