Antonio Croissy, Marco Puig, Hana Segura, Laura Waldron
Group 17
December 5, 2023
COP 4331

Super Groovy Pong

Github link: *https://github.com/Marco-Puig/SuperGroovyPong.git*
Demo video link: *https://www.youtube.com/watch?v=bCYxaVjbn9Y*

Introduction

A fast-paced Pong game implemented in Java using the LibGDX library. The game offers a classic Pong experience with a twist of increased speed and challenging gameplay. We will incorporate free movement until a certain point. The ball speed increases as the player keeps hitting the ball, going from a cold color to a warmer color. We have one player mode, with a basic AI as the opponent, and the players navigate with the wasd or arrow keys. We are also including sound effects, and possibly a leaderboard. We will have screens for instructions and also when the game ends. Players are able to restart the game, or to pause/resume in the middle of the game. Overall, this will be an engaging pong game where players test their skills.

Roles:

Antonio Croissy:
- –AI player mode
- –Player 2 paddle
- –Collisions
- –Start Screen
- –End Screen
- –Video Demo

Marco Puig:
- –Group lead
- –Particle trail
- –Basic game setup
- –Paddles
- –Scoring
- –StartScreen

Hana Segura:
- –Start Screen
- –End Screen

Laura Waldron:
- –Collisions

–Ball movement
–Documentation

1. Requirement Specification:

Design Patterns & Model View Controller:

Application Adapter Design Pattern:

LibGDX has an application adapter class, which we employed in our program. This class allows us to create a new game instance, with several methods, defined in the table below. It gave us a starting point for our code. The design pattern can be found in SuperGroovyPong.java (or the main file of the program). The following table defines the methods in the Application Adapter Design Pattern:

Name in Design Pattern:	**Description of Usage:**
Class: Super Groovy Pong	***Extends:*** Application Adapter
Method: create()	Initializes code, creating an instance of the game with resources
Method: render()	Renders the code logic, updating the gaming world continuously
Method: dispose()	Disposes of the resources upon completion
Initialization: libRequired()	Library setup method to initialize the components
Rendering: update()	Updates the game; paddles, ball, and collisions.

Singleton Pattern

We used the Singleton Pattern in reference to our sounds. The sounds were loaded by the GDX sound library. Below is a table showing the different parts of this pattern.

Name in Design Pattern:	**Actual Name:**
Class	SGPSounds()
Single Instance	private Music music
Static Method	play()

The above pattern is found in chapter 9 of the textbook:

http://wisenet.fau.edu/class/cop4331/notes/Ch10/Ch10.html

Observer Pattern

The Observer Pattern was used in controlling the paddle using the WASD keys. This was specifically implemented in the Paddle.java file. Below is a table showing how this pattern works.

Name in Design Pattern	**Actual Name** (moveKeys)
Subject	Paddle
Observer	moveInputKeys()
ConcreteObserver	moveInputWASD()
attach()	moveAI()
notify()	checkCollide()

The above pattern is found in chapter 5 of the textbook:
http://wisenet.fau.edu/class/cop4331/notes/Ch5/Ch5.html

Decorator Pattern

The decorator pattern is used in creating the ball so that it changes colors upon hitting a wall or a paddle. It changes between blue and red. Below is a table showing how this pattern is applied.

Name in Design Pattern	**Actual Name**(Ball)
Component	ShapeRenderer
ConcreteComponenet	Circle
Decorator	Color
Method()	render()

The above pattern is found in chapter 5 of the textbook:
http://wisenet.fau.edu/class/cop4331/notes/Ch5/Ch5.html

State Pattern

The state pattern is used in changing between the AI, Player 1, and Player 2 states. The update method updates this pattern. Below is a table showing this pattern in use.

Name in Design Pattern	**Actual Name**(Player States)
State	Player1, Player2, PlayerAI

ConcreteState	moveInputeWASD(), moveInputKeys(), moveAI()

The above pattern can be found in the following document: https://www.geeksforgeeks.org/state-design-pattern/

Model View Controller:

For our project, the model is SCP.java (Super Groovy Pong.java). This model runs the main game file, which includes paddles, ball, and collisions. The View is a libGDX render via update, inside the Super Groovy Pong (the Desktop Launcher). This allows the user to view the program, by popping up a window showing the Super Groovy Pong home screen. The Controller is the paddle.java file. This file is the paddle, which controls how many points the player will win.

Use Cases:

1 Player Controls Movement of the Paddle:
Actor: Player
–the player moves their paddle to defend their goal
–the player presses arrow keys to move their paddle around ½ of the screen

2 Player Hits Ball:
Actor: Player
–the player hits the ball with their paddle
–the paddle intercepts the ball to send it in the right direction
–the ball's speed increases with each successful hit
–a basic line render controls the paddle speed
–we start with a cold color and increment to hotter colors
–AI responds to the ballby hitting it back to also defend its goal.

3 Pause/Resume:
–the player wants to pause the game
–pressing the "Pause" button, or the P key will temporarily halt the game
–press the same button to continue the game

4 Scoring:
Actor: Player
–the player scores one point per round
–when the ballpasses the opponent's paddle & enters the opponent's goal, they score a point.
–score is updated on the screen

5 Game Over:
Actor: Player & AI

–the game ends when the player reaches best of 5 (first to 3).
–the “Game Over” screen is displayed (Player or AI wins).
–players can return to the main menu, or restart the game

6 Game Restart:
–the player clicks “Restart” to begin a new game, for convenience.
–after the game ends, they can click the “Restart” button or use the key (R key) to initiate a new game

7 Game Exit:
–the player decides to exit the game
–the player can click the “Exit Game” button or use the (Esc) key to close the game.
–the game closes and players are returned to their desktop screen

8 Game Instructions:
–the player must understand how to play
–the player can access an “Instructions” screen providing information on controlling paddles, scoring points, or special game rules

Platform:

For platform, we will be developing our application using LibGDX. LibGDX is a library for Java, written in Java, to aid in Game Development. It comes with a sprite rendering engine, window handling, and additionally built in libraries that will help us develop our game/project.

OS:

For Operating System, since we are developing our project using LibGDX, our project will be targeted for the Windows Operating System. The project can also be used on Mac, but is easier employed on Windows. Two of our group members used Windows, while the other two used Mac.

Glossary

Player: The human using the computer and controlling the motions of the paddle in the game.
Opponent Player: Either an AI player or another human player, depending on the choice, who competes with the player to score as many Goals as possible.
Paddle: The item the player uses to hit the puck away from their goal.
Ball: A object the player will hit back and forth with their paddle. The ball changes colors upon impact.
Goal: The place the player is trying to put the ball. When the goal reaches 4 points, the respective player wins.
Restart: The button that resets the game to its original state
Pause: The button that pauses the game, or resumes the game if already paused
Multiplayer: The state of the game involving two players

2. Design Specification:

<u>Use Case 1 Player controls movement of the paddle–Laura Waldron</u>

CRC Card: Laura Waldron
Actor: Player (human)

Responsibilities:
- Control the movement of the paddle

Collaborators:
- Class Paddle

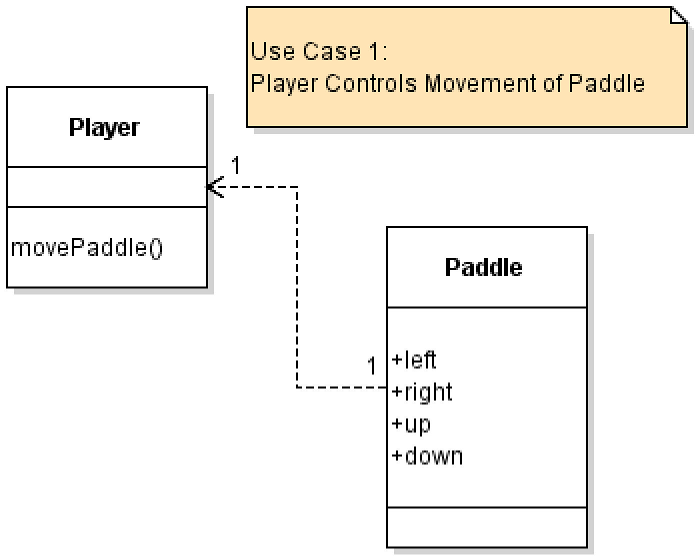
Methods:
- movePaddle()

Class Paddle

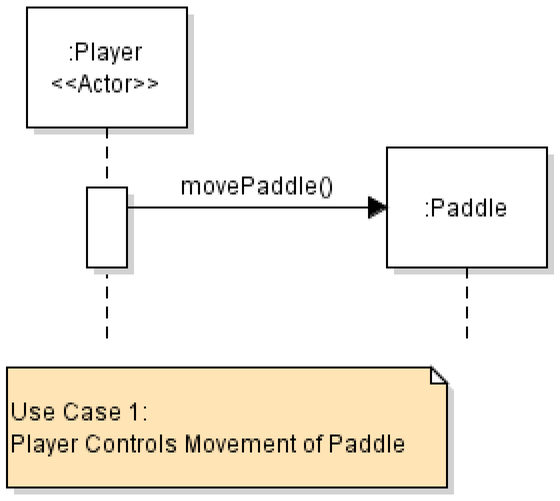
Responsibilities:
- Move the puck from one location to another

Collaborators:
- Class Puck

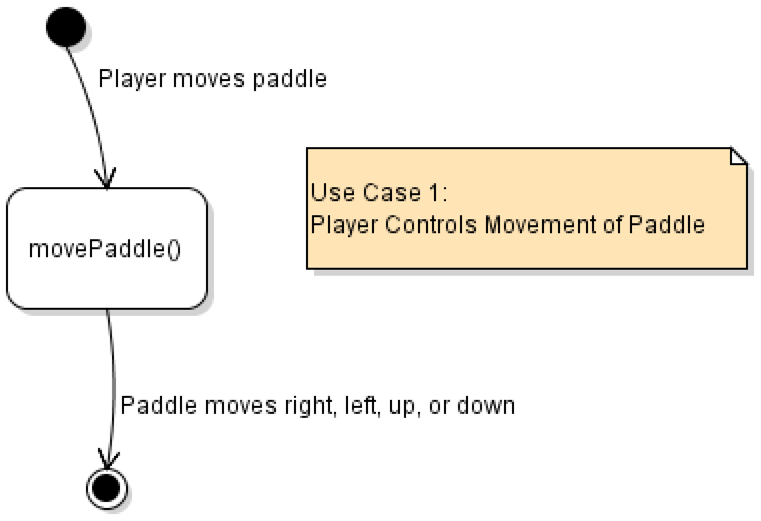
Class Diagram: Laura Waldron



Sequence Diagram: Laura Waldron

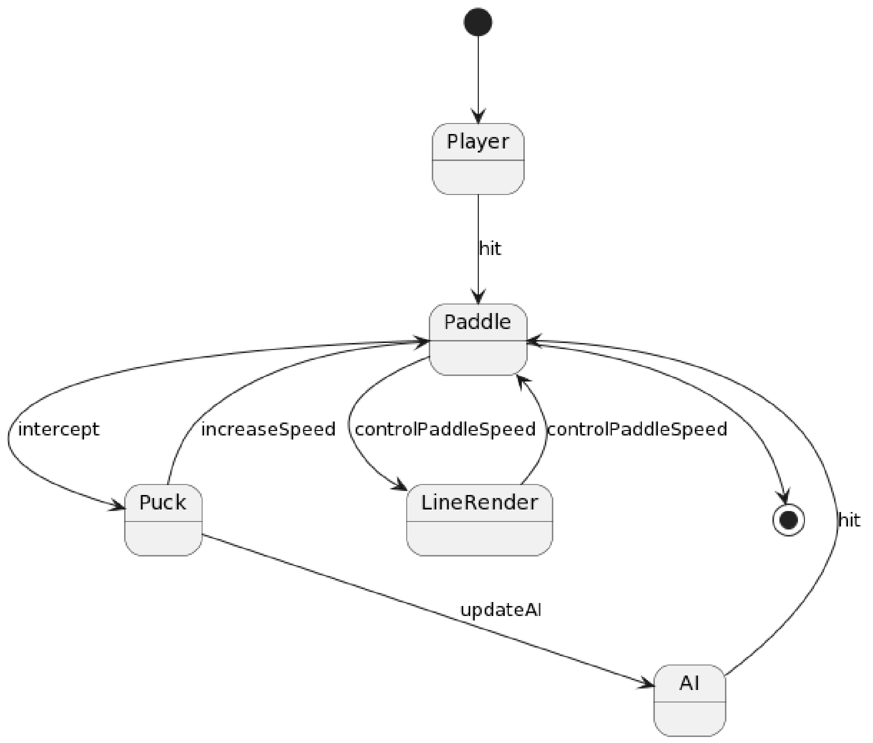


State Diagram: Laura Waldron

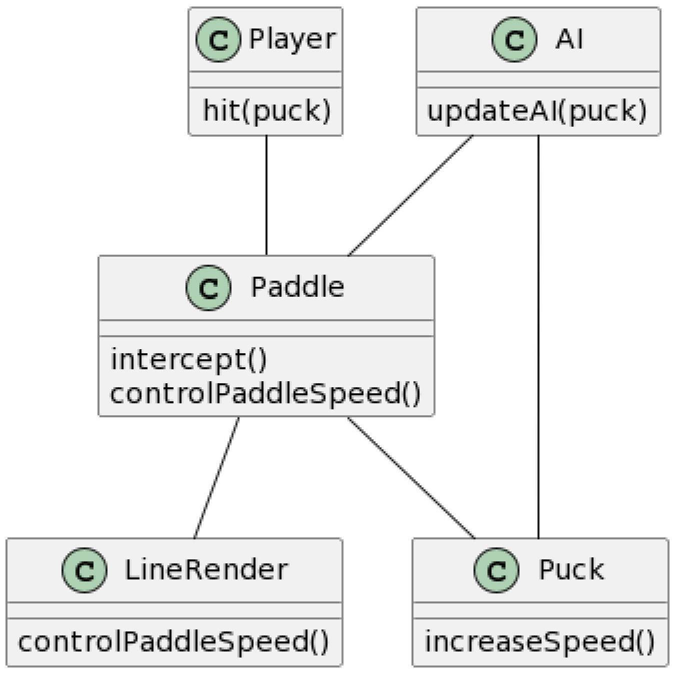

Player moves paddle
movePaddle()
Use Case 1:
Player Controls Movement of Paddle
Paddle moves right, left, up, or down

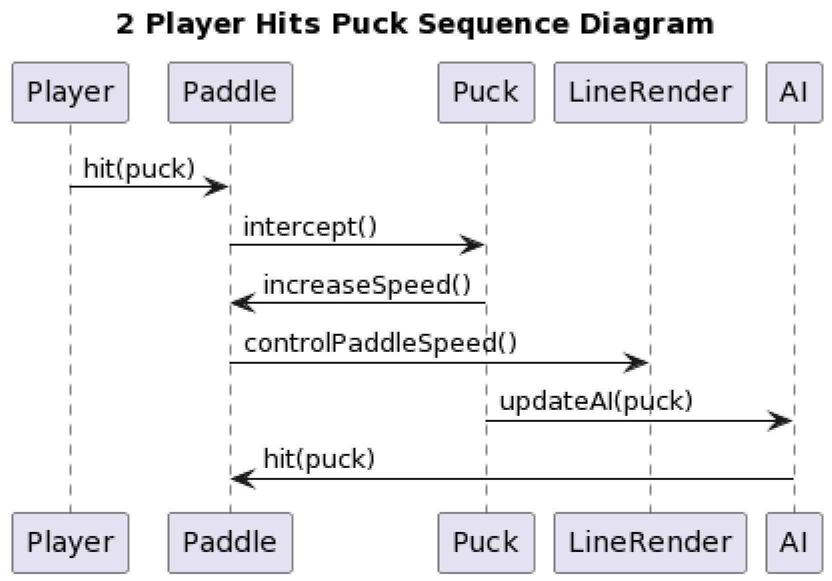
Use Case 2 Player hits puck - Marco Puig

2 Player Hits Puck State Diagram



2 Player Hits Puck Class Diagram





Use Case: Player Hits Pucks
Class Name: Player

Responsibilities:
- HitPuck()
- IncreasePuckSpeed()

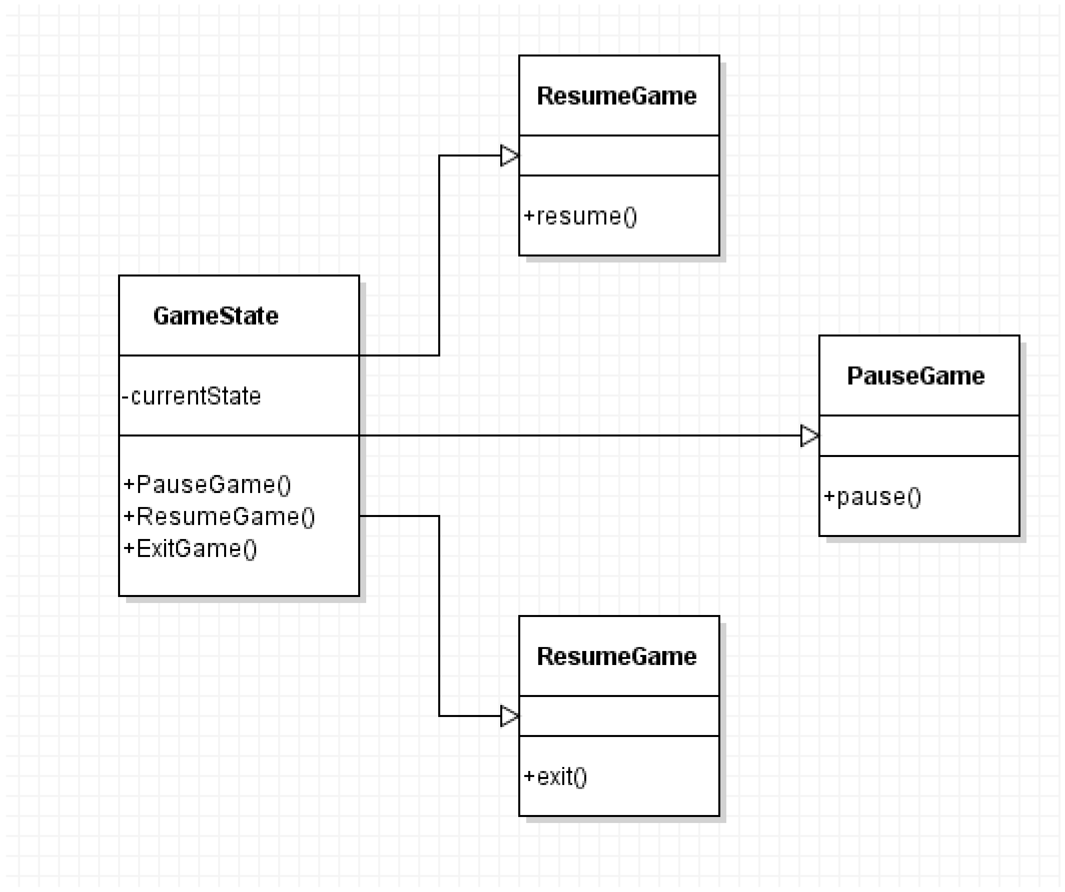
Collaborators:
- Puck

<u>Use Case 3 Pause/Resume – Hana Segura</u>
CRC cards

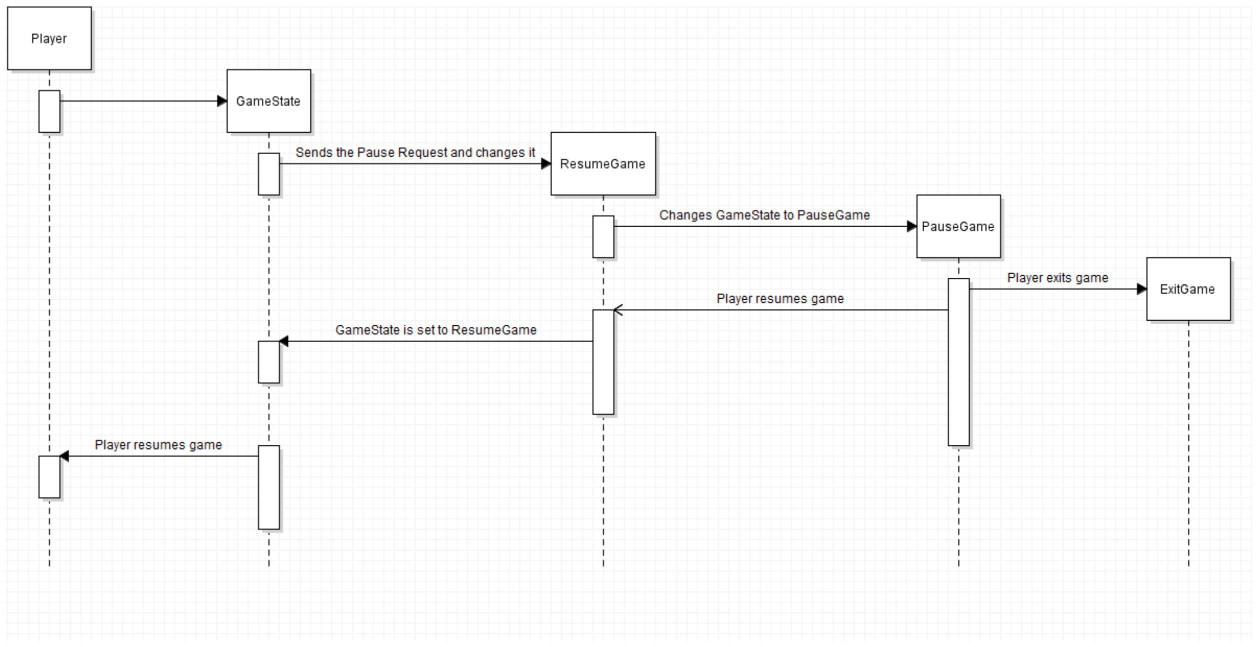
- Class: GameState
  - Responsibilities:
    - Maintain the current state of the game.
    - Allow changes to the game state.
  - Collaborators:
    - PauseGame
    - ResumeGame
    - ExitGame
- Class: ResumeGame
  - Responsibilities:
    - Handle the logic for resuming the game.
  - Collaborators:
    - GameState
- Class: PauseGame

    - Responsibilities:
        - Handle the logic for pausing the game.
    - Collaborators:
        - GameState
- Class: ExitGame
    - Responsibilities:
        - Handle the logic for exiting the game.
    - Collaborators:
        - GameState

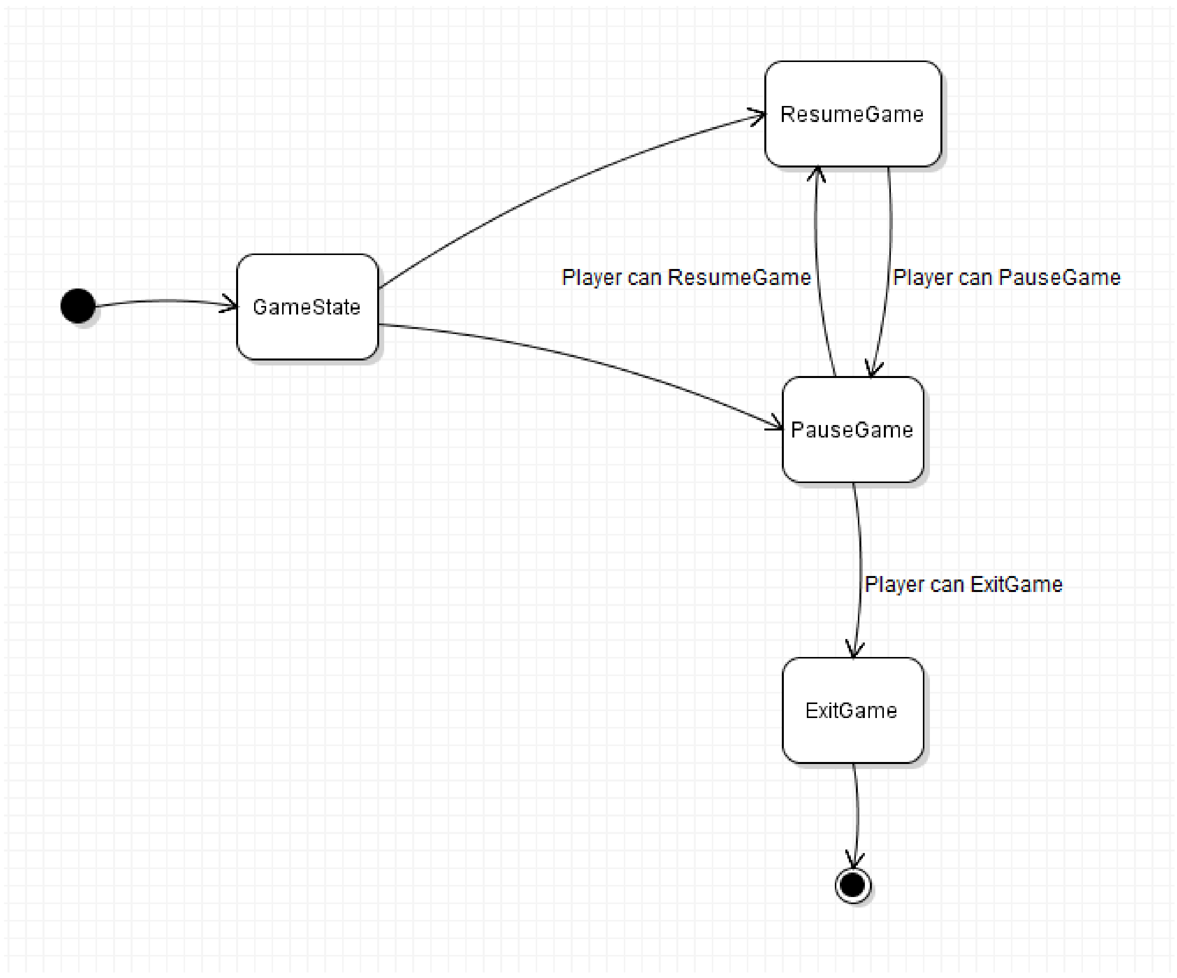
Class Diagram:



Sequence Diagram:


Player
GameState
Sends the Pause Request and changes it
ResumeGame
Changes GameState to PauseGame
PauseGame
Player exits game
ExitGame
Player resumes game
GameState is set to ResumeGame
Player resumes game

State Diagram:



<u>Use Case 4 Scoring –Laura Waldron</u>

CRC cards: Laura Waldron

Actor: Player (human)

Responsibilities:

- Score one point (max) per round

Collaborators:

- Class Puck

Methods:

- hitPuck()

Class Puck

Responsibilities:

- Enter the opponent’s goal to score a point

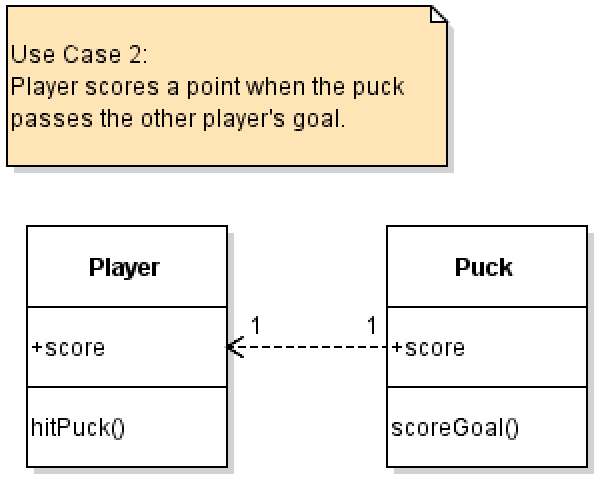
Collaborators:

- Class Player

Methods:

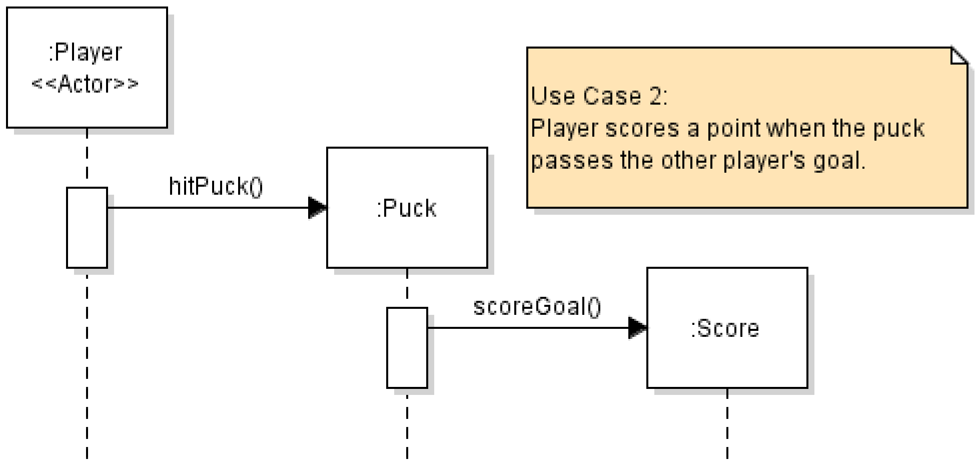
- scoreGoal()

Class Diagram: Laura Waldron

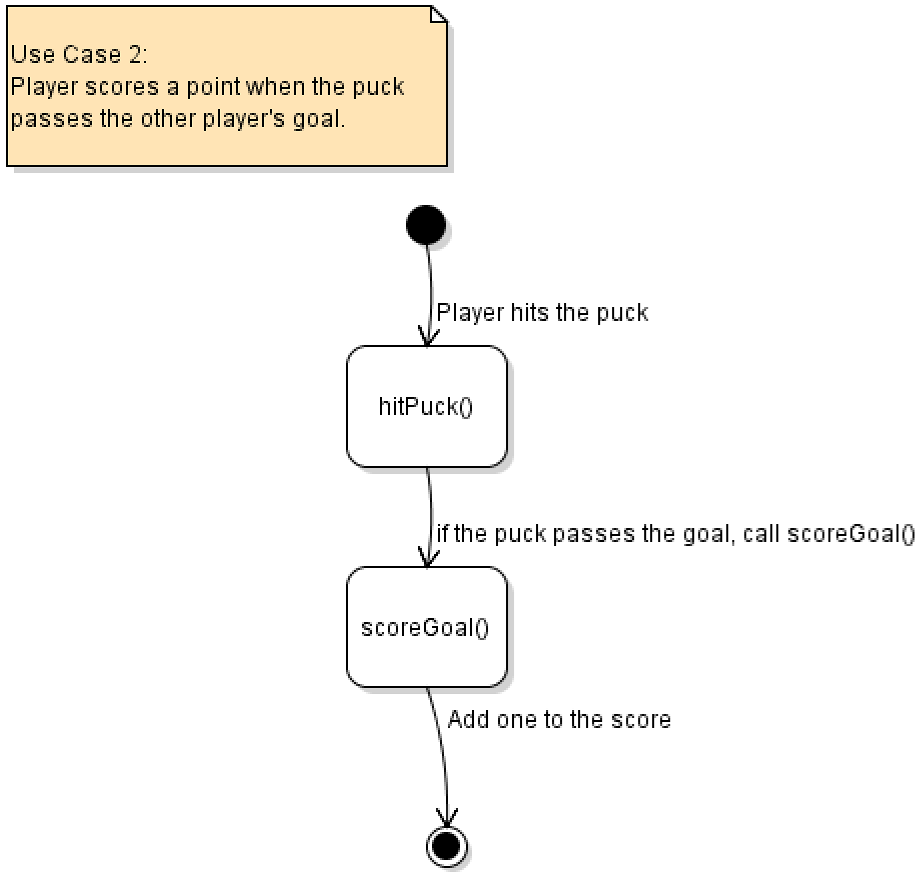


Use Case 2:
Player scores a point when the puck passes the other player's goal.

Sequence Diagram: Laura Waldron


:Player
<<Actor>>
hitPuck()
:Puck
scoreGoal()
:Score
Use Case 2:
Player scores a point when the puck passes the other player's goal.

State Diagram: Laura Waldron


Use Case 2:
Player scores a point when the puck passes the other player's goal.
Player hits the puck
hitPuck()
if the puck passes the goal, call scoreGoal()
scoreGoal()
Add one to the score

<u>Use Case 5 Game Over - Antonio Croissy</u>

CRC Card
gameOver
  responsibilities:

- display game over screen

Collaborators:

- game

Actor: Player (human)

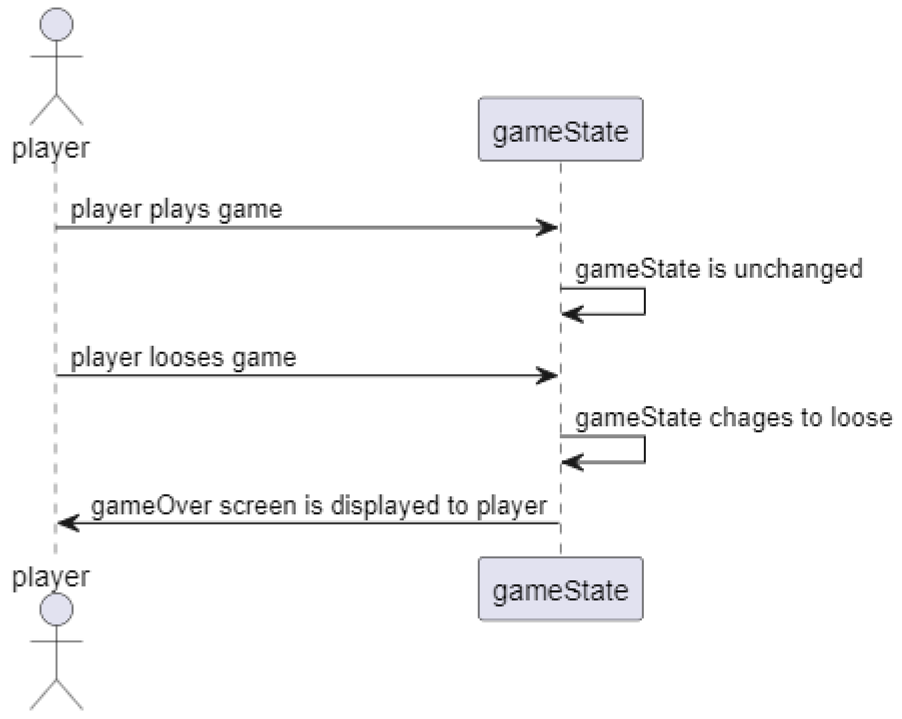
responsibilities:

- plays game

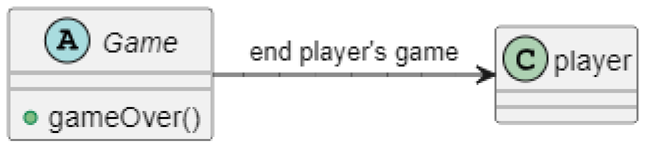
Collaborators:

- game
- game interface

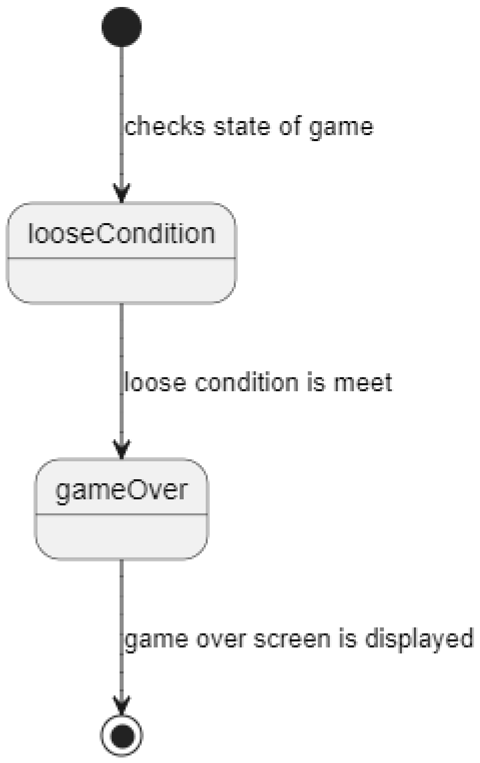
Sequence



Class



State



<u>Use Case 6 Game Restart - Antonio Croissy</u>

CRC Card
gameRestart
responsibilities:
- reset game back to initial conditions

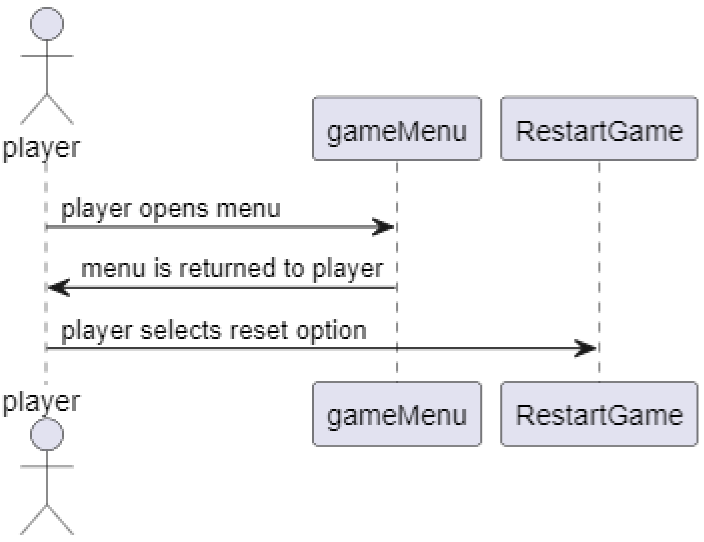
Collaborators:
- game menu

Methods:
- Restart()

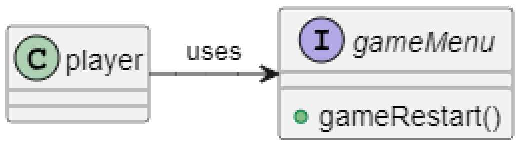
Actor: Player (human)
responsibilities:
- pick option from game menu

Collaborators:
- game interface

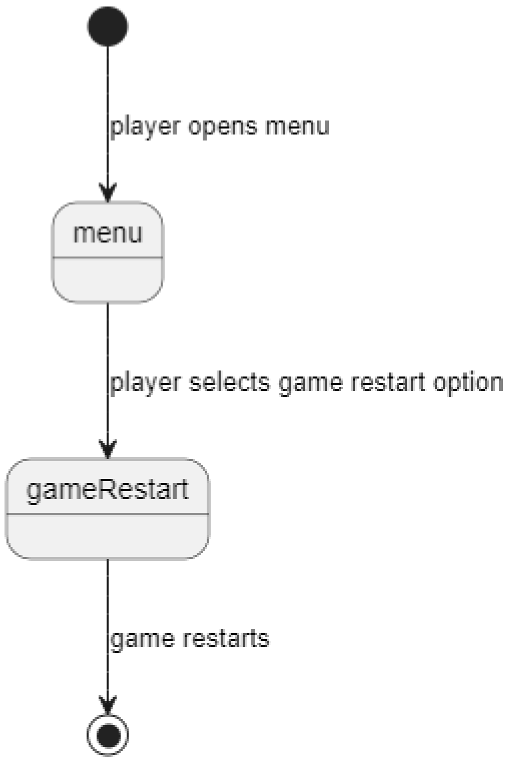
Sequence



Class



State

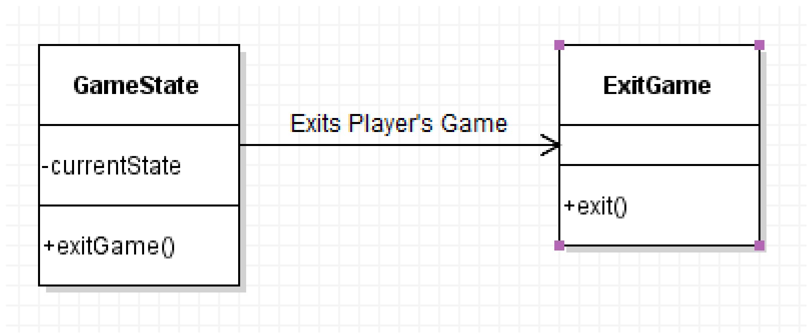


<u>Use Case 7 Game Exit – Hana Segura</u>

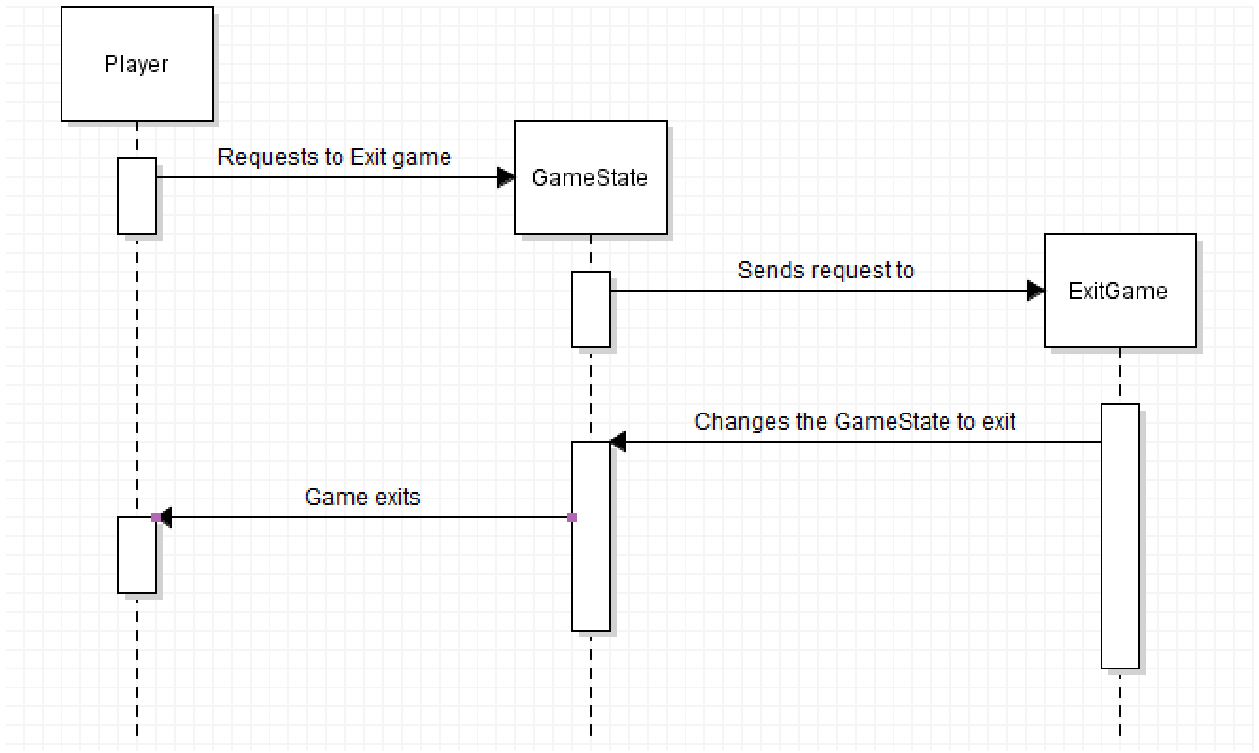
CRC cards

- Class: GameState
  - Responsibilities:
    - Maintain the current state of the game.
    - Allow changes to the game state.
  - Collaborators:
    - ExitGame
- Class: ExitGame
  - Responsibilities:
    - Handle the logic for exiting the game.
  - Collaborators:
    - GameState

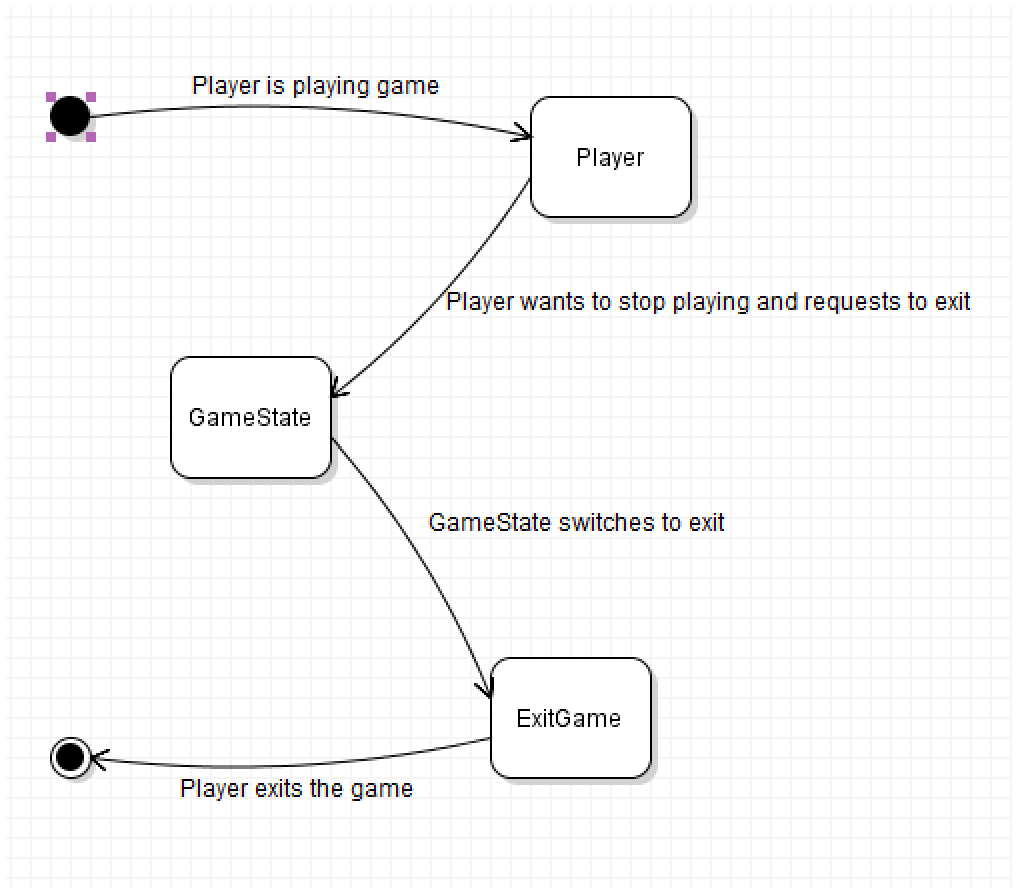
Class Diagram:



Sequence Diagram:



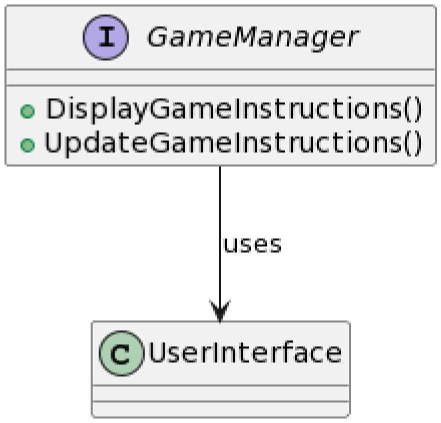
State Diagram:


Player is playing game
Player
Player wants to stop playing and requests to exit
GameState
GameState switches to exit
ExitGame
Player exits the game

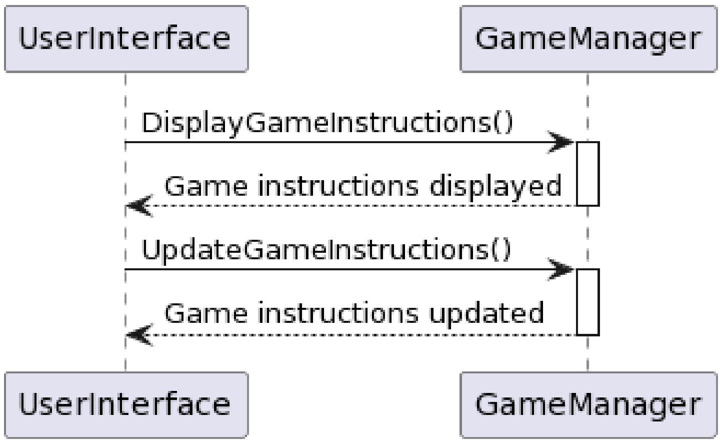
<u>Use Case 8 Game Instructions - Marco Puig</u>

Class Diagram

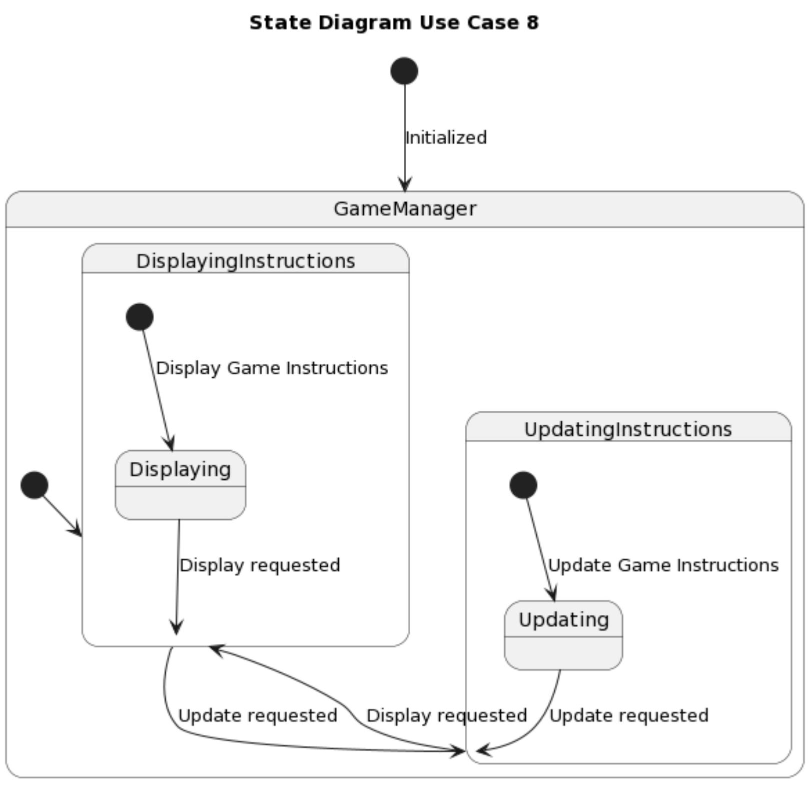
Use Case 8 Game Instructions



Sequence Diagram



UserInterface
GameManager
DisplayGameInstructions()
Game instructions displayed
UpdateGameInstructions()
Game instructions updated
UserInterface
GameManager



Use Case: Game Instructions
Class Name: GameManager

Responsibilities:
- DisplayGameInstructions()
- UpdateGameInstructions()

Collaborators:
- UserInterface

3. Project Code:

DesktopLauncher.java

```
package com.pong.game;


import com.badlogic.gdx.backends.lwjgl3.Lwjgl3Application;

import com.badlogic.gdx.backends.lwjgl3.Lwjgl3ApplicationConfiguration;
```

```
/**
 * Class to run the desktop launcher for the game
 * View: for the MVC
 * @author Marco Puig
 */
// Please note that on macOS your application needs to be started with the
-XstartOnFirstThread JVM argument
public class DesktopLauncher {
	public static void main (String[] arg) {
		Lwjgl3ApplicationConfiguration config = new Lwjgl3ApplicationConfiguration();
		config.setForegroundFPS(60);
		config.setTitle("SuperGroovyPong");
		new Lwjgl3Application(new SuperGroovyPong(), config);
	}
}
```

SuperGroovyPong.java

```
package com.pong.game;
import com.badlogic.gdx.ApplicationAdapter;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.pong.game.Paddle.State;
//import com.badlogic.gdx.Screen;
//import com.badlogic.gdx.scenes.scene2d.ui.List;
```

```java
import com.badlogic.gdx.scenes.scene2d.ui.Skin;



/**
 * Main Super Groovy Pong Class, using the Application Adapter
 * Model: for the MVC
 * Design Pattern: Application Adapter
 * @author Marco Puig
 */
public class SuperGroovyPong extends ApplicationAdapter {
    private SpriteBatch batch;
    private ShapeRenderer shapeRenderer;
    private int screenWidth, screenHeight;
    private Paddle paddle1, paddle2;
    private Ball ball;
    private BitmapFont font;
    Skin skin;

    //music instance
    private SGPSounds SGPSounds;

    //Flags
    private boolean isPaused;
    private boolean gameOver;

    public enum gameScreen {StartScreen, PlayScreen, PauseScreen, EndScreen}
    public gameScreen CurrentScreen;


    private StartScreen startScreen;


    /**
     * Method to create the new game
     * Create method of Application Adapter
     * Initializes code, creating an instance of the game with resources
     */
    // On Start
    @Override
    public void create () {
        // Call needed dependencies
        libRequired();


        // Initialize the StartScreen instance
        startScreen = new StartScreen(this);


        //Set flags related to game states and screens
        CurrentScreen = gameScreen.StartScreen;
        gameOver = false;
        isPaused = false;


        // Initialize Objects once we go past Start Screen
        paddle1 = new Paddle(20, screenHeight / 2 - 40, 20, 80, State.playerOne);
        ball = new Ball(screenWidth / 2, screenHeight / 2, 20, 20);
```

```
        //load and play the audio in the background
        SGPSounds = new SGPSounds("audio\\game-theme.mp3");
        SGPSounds.play();
    }

    /**
     * Method to render the screen
     * Render method of Application Adapter
     * Renders the code logic, updating the gaming world continuously
     */
    // Render on every frame
    @Override
    public void render () {

        // Graphics lib
        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        if (CurrentScreen == gameScreen.StartScreen)
        {
            startScreen.render(Gdx.graphics.getDeltaTime());
            CurrentScreen = gameScreen.StartScreen;
            if (startScreen.start) {
                CurrentScreen = gameScreen.PlayScreen;
                if (!startScreen.SinglePlayer()) {
                    paddle2 = new Paddle(screenWidth - 40, screenHeight / 2 - 40, 20, 80,
State.playerTwo); // can also say State.playerAI
```

```
            } else {
                paddle2 = new Paddle(screenWidth - 40, screenHeight / 2 - 40, 20, 80,
State.playerAI); // can also say State.playerAI
            }
        }
    }

    // Update logic
    checkGameState();

    if (CurrentScreen == gameScreen.PlayScreen)
    {
        paddle1.update();
        paddle2.update();
        paddle2.trackBall(ball);
        ball.update(paddle1, paddle2, screenWidth, screenHeight);
    }

    // Draw objects
    batch.begin();
    shapeRenderer.begin(ShapeRenderer.ShapeType.Filled);

    if (CurrentScreen == gameScreen.PlayScreen)
    {
        paddle1.render(shapeRenderer);
        paddle2.render(shapeRenderer);
        ball.render(shapeRenderer);
        // Draw Score
        font.draw(batch, Integer.toString(ball.getScorePlayerLeft()), screenWidth / 2 - 50, 50);
        font.draw(batch, Integer.toString(ball.getScorePlayerRight()), screenWidth / 2 + 25, 50);
    }

    // Needed calls for render
    batch.end();
    shapeRenderer.end();
}


/**
 * Method to dispose of the screen when the game is over
 * dispose method of Application Adapter
 * Disposes of the resources upon completion
 */
// Dispose when completed
@Override
public void dispose () {
    batch.dispose();
    shapeRenderer.dispose();
    font.dispose();
    SGPSounds.dispose();
}

/**
```

```
 * runs various game checks and load screen based on CurrentScreen value
 * @author Antonio Croissy
 */
public void checkGameState() {

    gameOverCheck();
    pauseCheck();

    switch (CurrentScreen) {
        case StartScreen:
            startScreen.render(Gdx.graphics.getDeltaTime());
            break;
        case PauseScreen:
            PauseScreen pauseScreen = new PauseScreen(this);
            pauseScreen.render(Gdx.graphics.getDeltaTime());
            break;
        case EndScreen:
            EndScreen endScreen = new EndScreen(this, ball.getScorePlayerLeft());
            endScreen.render(Gdx.graphics.getDeltaTime());
            break;
        default:
            break;
    }
}

/**
 * checks score of both player to determine if game should be over
```

```
     * @author Antonio Croissy
     */
    private void gameOverCheck() {
        if (gameOverOnThreshold()) {
            gameOver = true;
            CurrentScreen = gameScreen.EndScreen;
        }
    }

    /**
     * Allows for for game to be paused by changing currentScreen to pauseScreen
     * @author Antonio Croissy
     */
    private void pauseCheck() {
        if (!gameOver) {
            if (Gdx.input.isKeyJustPressed(Input.Keys.ESCAPE) ||
Gdx.input.isKeyJustPressed(Input.Keys.ENTER)) {
                isPaused = !isPaused;
            }

            if (isPaused) {
                // Handle any pause-related logic (e.g., stop animations or background music)
                CurrentScreen = gameScreen.PauseScreen;
                SGPSounds.stop();
            } else {
                // Handle resume logic (e.g., restart animations or resume background music)
                if (CurrentScreen != gameScreen.StartScreen)
                CurrentScreen = gameScreen.PlayScreen;
            SGPSounds.play();
        }
    }
}

public void showEndScreen(int finalScore) {
    // End screen should just display then have a button to redirect to menu screen
    //setScreen(new EndScreen(this, finalScore));
}

// Check to if a player has scored more than 10 points, if so, end the game
public boolean gameOverOnThreshold()
{
    if (ball.getScorePlayerLeft() >= 10)
    {
        return true;
    }
    else if (ball.getScorePlayerRight() >= 10)
    {
        return true;
    }

    return false;
}

/**
```

```
     * Needed requirements for the game
     * libRequired method of Application Adapter
     * Library setup method to initialize the components
     * @author Marco Puig
     */
    public void libRequired()
    {
        batch = new SpriteBatch();
        shapeRenderer = new ShapeRenderer();
        font = new BitmapFont();
        screenWidth = Gdx.graphics.getWidth();
        screenHeight = Gdx.graphics.getHeight();
    }
}
```

Paddle.java

```
package com.pong.game;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.glutils.ShapeRenderer;
import com.badlogic.gdx.math.Rectangle;

/**
 * Class to handle the paddle
 * Design Pattern: Observer Pattern
 * @author Marco Puig
 */
```

```
public class Paddle {

    private int x, y, width, height; //box width height

    private static int speed = 5;

    private Rectangle boundingBox;

    public State playerState;

    private int ballYposition, ballXposition;


    /**

     * Method to define a paddle

     * Subject of Observer Pattern

     * @param x the x coordinate

     * @param y the y coordinate

     * @param width the width of the paddle

     * @param height the height of the paddle

     * @param playerState the state of the player

     */

    public Paddle(int x, int y, int width, int height, State playerState) {

        this.x = x;

        this.y = y;

        this.width = width;

        this.height = height;

        this.boundingBox = new Rectangle(x, y, width, height);

        this.playerState = playerState;

    }


    /**

     * Method to update the state
```

```
 * Design Pattern: State Pattern
 * State can be two players or an ai player
 */
public void update() {
    // Movement code based on state

    // If main player
    if (playerState == State.playerOne) moveInputWASD();
    if (playerState == State.playerTwo) moveInputKeys();

    // call ai for State.playerAI
    if (playerState == State.playerAI) moveAI();

    // Update the bounding box position
    boundingBox.setPosition(x, y);
}
/**
 * Method to create a paddle (rectangle)
 * @param shapeRenderer create the paddle
 */
public void render(ShapeRenderer shapeRenderer) {
    shapeRenderer.setColor(Color.WHITE);
    shapeRenderer.rect(x, y, width, height);
}

/**
 * Method to create a bounding box
 * @return the bounding box
 */
public Rectangle getBoundingBox() {
    return boundingBox;
}

/**
 * Method to create a rectangle
 * @return the rectangle
 */
public Rectangle getBoundingRectangle(){
    return boundingBox;
}

/**
 * Method to move the input keys (WASD)
 * Observer for Observer Pattern
 * ConcreteState1 for State Pattern
 * @author Antoinio Croissy
 */
public void moveInputKeys()
{
    // Get Input
    if (Gdx.input.isKeyPressed(Input.Keys.UP)) {
        y += speed;
    }
    if (Gdx.input.isKeyPressed(Input.Keys.DOWN)) {
        y -= speed;
    }
    if (Gdx.input.isKeyPressed(Input.Keys.RIGHT)) {
        x += speed;
    }
    if (Gdx.input.isKeyPressed(Input.Keys.LEFT)) {
        x -= speed;
    }

    if (x < (2*(Gdx.graphics.getWidth() / 3))) {
        x = 2*(Gdx.graphics.getWidth() / 3);
    }

    checkCollide();
}

/**
 * Method to move the WASD keys
 * ConcreteObserver for Observer Pattern
 * ConcreteState2 for State Pattern
 * @author Marco Puig
 */
public void moveInputWASD()
{
    // Get Input
    if (Gdx.input.isKeyPressed(Input.Keys.W)) {
        y += speed;
    }
    if (Gdx.input.isKeyPressed(Input.Keys.S)) {
        y -= speed;
    }
    if (Gdx.input.isKeyPressed(Input.Keys.D)) {
        x += speed;
    }
    if (Gdx.input.isKeyPressed(Input.Keys.A)) {
        x -= speed;
    }

    if (x > Gdx.graphics.getWidth() / 3) {
        x = Gdx.graphics.getWidth() / 3;
    }

    checkCollide();
}

/**
 * Method to move the AI Player
 * attach() method for Observer pattern
 * ConcreteState3 for State pattern
 * @author Antoinio Croissy
 */
public void moveAI() {
    if (y < ballYposition) {
        y += speed;
    }
    if (y > ballYposition) {
        y -= speed;
    }

    checkCollide();
}

/**
 * Method to track the ball
 * @author Marco Puig
 * @param ball the moving ball
 */
public void trackBall(Ball ball) {
    ballYposition = ball.getBallY();
    ballXposition = ball.getBallX();
}

/**
 * Method to check for a collision
 * notify() for Observer pattern
 * @author Marco Puig
 */
public void checkCollide() {
    // Add boundary checking to prevent the paddle from going off-screen (Collision)

    // Y-axis boundary checking
```

```
        if (y < 0) {
            y = 0;
        }
        if (y > Gdx.graphics.getHeight() - height) {
            y = Gdx.graphics.getHeight() - height;
        }

        // X-axis boundary checking (limit movement to one-third of the screen)
        if (x < 0) {
            x = 0;
        }
        if (x > Gdx.graphics.getWidth()) {
            x = Gdx.graphics.getWidth() - width;
        }
    }

    /**
     * State declaration (3 states)
     * States for State Pattern
     * @author Marco Puig
     */
    public enum State {
        playerOne, //state 1 -- player one
        playerTwo, //state 2 -- player two
        playerAI //state 3 -- AI player
    }
}
```

Ball.java

```java
package com.pong.game;

import com.badlogic.gdx.graphics.Color;

import com.badlogic.gdx.graphics.glutils.ShapeRenderer;

import com.badlogic.gdx.math.Circle;

import com.badlogic.gdx.math.Intersector;

import com.badlogic.gdx.math.Rectangle;

//import libraries to control the ball

import com.badlogic.gdx.scenes.scene2d.ui.List;

import java.util.ArrayList;

import java.util.Random; //use the random library to make the ball move randomly

import java.util.Iterator;


import com.badlogic.gdx.Gdx;




/**

 * Class handling the Ball's definition

 * Design Pattern: Decorator Pattern

 * @author Marco Puig

 */

public class Ball {

    //initialize variables to handle the ball

    private int x, y, width, height;

    private int speed = 5;

    private int velocityX, velocityY;

    private Circle boundingCircle;
private int score1 = 0;

private int score2 = 0;


/**

 * Method to declare a ball

 * @param x the x position of the ball

 * @param y the y position of the ball

 * @param width the width of the ball

 * @param height the height of the ball

 */

public Ball(int x, int y, int width, int height) {

    this.x = x;

    this.y = y;

    this.width = width;

    this.height = height;

    this.boundingCircle = new Circle(x, y, width / 2);

    resetSpeed();

}


/**

 * Method to reverse the ball's x velocity

 * @author Laura Waldron

 */

public void reverseVelocityX(){

    velocityX = -velocityX;

}

/**
```

```
 * Method to reverse the ball's y velocity
 * @author Laura Waldron
 */
public void reverseVelocityY(){
    velocityY = -velocityY;
}
/**
 * Method to get the X value of the ball
 * @return the x value of the ball
 */
public float getX(){
    return x;
}
/**
 * Method to get the Y value of the ball
 * @return the y value of the ball
 */
public float getY(){
    return y;
}
/**
 * Method to get the width of the ball
 * @return the width of the ball
 */
public float getWidth(){
    return width;
}
```

```
/**
 * Method to set the position of the ball
 * @param x the x coordinate
 * @param y the y coordinate
 */
public void setPosition(float x, float y){
    this.x = (int)x;
    this.y = (int)y;
    boundingCircle.setPosition(x,y);
}
/**
 * Method to get the circle (ball)
 * @return the circle
 */
public Circle getBoundingCircle(){
    return boundingCircle;
}

/**
 * Method to update the ball's position upon collision
 * @author Laura Waldron
 * @param paddle1 the first paddle
 * @param paddle2 the second paddle
 * @param screenWidth the width of the screen
 * @param screenHeight the height of the screen
 */
public void update(Paddle paddle1, Paddle paddle2, int screenWidth, int screenHeight) {
```

```
//update the ball position based on the velocity
x += velocityX;
y += velocityY;

//check to see if it collided with a wall and on which side to update the score
if ((x + width / 2) > screenWidth){
    //reverse the horizontal velocity
    reverseVelocityX();
    //give the right player a point
    resetSpeed();
    // play sound on scored
    Gdx.audio.newSound(Gdx.files.internal("audio/scored.wav")).play();
    score1++;
    //reset the ball position to the center again
    resetBall();
}
else if((x - width / 2) < 0){
    //reverse the horizontal velocity
    reverseVelocityX();
    //give the left player a point
    resetSpeed();
    // play sound on scored
    Gdx.audio.newSound(Gdx.files.internal("audio/scored.wav")).play();
    score2++;
    //reset the ball position to the center
    resetBall();
}
```

```
        // Check for collisions with paddles
        Rectangle ballRectangle = new Rectangle(x - width / 2, y - width / 2, width, height);

        if (Intersector.overlaps(ballRectangle, paddle1.getBoundingRectangle()) ||
            Intersector.overlaps(ballRectangle, paddle2.getBoundingRectangle())) {
            // Reverse the horizontal velocity if the ball hits a paddle
            reverseVelocityX();
            // play sound on hit
            Gdx.audio.newSound(Gdx.files.internal("audio/hit.wav")).play();
            increaseSpeed();
        }

        //check for a collsision in the top and bottom walls
        if (y - width / 2 < 0 || y + width / 2 > screenHeight) {
            // Reverse the vertical velocity if the ball hits the top or bottom wall
            reverseVelocityY();
        }

        //update the BoundingCircle.setPosition
        boundingCircle.setPosition(x, y);
    }

    /**
     * Method to create the ball
     * render() method of decorator pattern
     * @author Marco Puig
```

```
 * @param shapeRenderer the shapeRenderer option to make the ball
 */
public void render(ShapeRenderer shapeRenderer) {
    // Render the ball

    // Calculate color based on which player has last hit the ball
    //Decorator (color) of Decorator Pattern
    float red = -velocityX;
    float blue = velocityX;
    float green = 0.0f;

    //shapeRenderer: Component of Decorator Pattern
    shapeRenderer.setColor(red, green, blue, 1);
    //circle: ConcreteComponent of Decorator Pattern
    shapeRenderer.circle(x, y, width / 2);
}

/**
 * Get the y coordinate of the ball
 * @return the y coordinate
 */
public int getBallY() {
    return y;
}

/**
 * Get the y coordinate of the ball
 * @return the y coordinate
 */
public int getBallX() {
    return x;
}

/**
 * Method to get the score of the left player
 * @return the score of the left player
 */
public int getScorePlayerLeft() {
    return score1;
}

/**
 * Method to get the score of the right player
 * @return the score of the right player
 */
public int getScorePlayerRight() {
    return score2;
}

/**
 * Method to increase the speed of the ball
 * @author Marco Puig
 */
public void increaseSpeed()
```

```
{
    velocityX *= 1.2;
    velocityY *= 1.2;
}

/**
 * Method to reset the speed of the ball
 * @author Marco Puig
 */
public void resetSpeed()
{
    Random random = new Random();
    velocityX = speed * (random.nextBoolean() ? 1 : -1);
    velocityY = speed * (random.nextBoolean() ? 1 : -1);
}


/**
 * Method to reset the position of the ball
 * @author Marco Puig
 */
private void resetBall()
{
    x = Gdx.graphics.getWidth() / 2;
    y = Gdx.graphics.getHeight() / 2;
}
```

```
}
```

CollisionUtils.java

```java
package com.pong.game;


import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.math.Circle;
import com.badlogic.gdx.audio.Sound;
//libGDX Intersector class
import com.badlogic.gdx.math.Intersector; //use intersector class for collision detection


/**
 * Class to handle collisions between the paddle and the ball
 * @author Laura Waldron, Marco Puig, Antonio Croissy
 */
public class CollisionUtils {
    //sounds instance
    private static Sound hitSound;


    /**
     * Method to initialize the sound when the ball hits
     * @author Laura Waldron
     */
    public static void initializeSound(){
        hitSound = Gdx.audio.newSound(Gdx.files.internal("assets\\scored.wav"));
    }


    /**
```

```
     * Method to handle a collision between a paddle and the ball
     * @author Laura Waldron
     * @param paddle the paddle
     * @param ball the ball
     * @return a collision between the two circles (true or false)
     */
    public static boolean collides(Paddle paddle, Ball ball) {
        //use intersector classs to check this
        // Convert Rectangle to Circle for collision check
        Circle paddleCircle = new Circle(paddle.getBoundingBox().getX() +
paddle.getBoundingBox().getWidth() / 2,
        paddle.getBoundingBox().getY() + paddle.getBoundingBox().getHeight() / 2,
        Math.max(paddle.getBoundingBox().getWidth(), paddle.getBoundingBox().getHeight()) / 2);

        // Check for collision between the converted Circle and the Ball's Circle
        return Intersector.overlaps(paddleCircle, ball.getBoundingCircle());
    }

    /**
     * Method to handle a collision detected before
     * @author Laura Waldron
     * @param paddle the paddle
     * @param ball the ball
     */
    public static void handleCollision(Paddle paddle, Ball ball) {
        //reverse the ball's velocity upon a collision
        ball.reverseVelocityX();
```

```java
    playHitSound();
}

/**
 * Method to handle if the ball goes out of bounds
 * @author Marco Puig
 * @param ball the ball
 * @param screenWidth the width of the screen
 * @param screenHeight the height of the screen
 * @return false if ball is out of bounds
 */
public static boolean isOutOfBounds(Ball ball, int screenWidth, int screenHeight) {
    return ball.getX()-ball.getWidth()/2 < 0 ||
    ball.getX() + ball.getWidth() / 2 > screenWidth ||
    ball.getY() - ball.getWidth() / 2 < 0 ||
    ball.getY() + ball.getWidth() / 2 > screenHeight;
}

/**
 * Method to reset the ball to the middle of the screen
 * @author Antoinio Croissy
 * @param ball the ball
 * @param screenWidth the width of the screen
 * @param screenHeight the height of the screen
 */
public static void resetBall(Ball ball, int screenWidth, int screenHeight) {
    //reset the ball to the center of the screen
```

```
        ball.setPosition(screenWidth/2, screenHeight/2);
    }

    /**
     * Method to play the sound when the ball hits
     * @author Laura Waldron
     */
    private static void playHitSound(){
        //method to play the sound
        if(hitSound != null){
            hitSound.play();
        }
    }
}
```

SGPSounds.java

```
package com.pong.game;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.audio.Music;

/**
 * Class to handle the sounds of the game
 * Design Pattern: Singleton Pattern
 * @author Laura Waldron
 */
public class SGPSounds {
    private Music music; //create a private music instance
```

```
/**
 * Method to find the music file path
 * @param musicFilePath the string of the file path
 */
public SGPSounds(String musicFilePath) {
    music = Gdx.audio.newMusic(Gdx.files.internal(musicFilePath));
}

/**
 * Method to play the music
 * Static method
 */
public void play() {
    if (music != null) {
        music.setLooping(true); //allow it to loop
        music.play();
    }
}

/**
 * Method to stop the music
 */
public void stop() {
    if (music != null) {
        music.stop();
    }
    }


    /**
     * Method to dispose of the music
     */
    public void dispose() {
        if (music != null) {
            music.dispose();
        }
    }
}
```

PauseScreen.java

```
package com.pong.game;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.ScreenAdapter;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.pong.game.SuperGroovyPong.gameScreen;

/**
 * Class to render the pausing of the game
 * Design Pattern: Singleton Pattern
 * @author Antonio Croissy
 */
public class PauseScreen extends ScreenAdapter {

    private final SuperGroovyPong game;
    private OrthographicCamera camera;
    private SpriteBatch batch;
    private BitmapFont font;

    public PauseScreen(final SuperGroovyPong game) {
        this.game = game;

        camera = new OrthographicCamera();
        camera.setToOrtho(false, 800, 400); // Adjust dimensions
        batch = new SpriteBatch();
        font = new BitmapFont();
        font.getData().setScale(2.75f);
    }

    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        camera.update();
        batch.setProjectionMatrix(camera.combined);

        batch.begin();
        font.draw(batch, "Game Paused", 100, 300);
        font.draw(batch, "Press Esc or Enter to unpause", 100, 200);
        batch.end();

        if (game.CurrentScreen != gameScreen.PauseScreen) {
            dispose();
        }
    }

    @Override
    public void dispose() {
        // Dispose of resources when the screen is no longer shown
        batch.dispose();
        font.dispose();
    }
}
```

StartScreen.java

```
package com.pong.game;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Input;
import com.badlogic.gdx.ScreenAdapter;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

/**
 * Class to handle the start of the game
```

```java
 * Design Pattern: Singleton Pattern
 * @author Hana Segura, Antonio Croissy
 */
public class StartScreen extends ScreenAdapter {
   private final SuperGroovyPong game;
   private OrthographicCamera camera;
   private SpriteBatch batch;
   private BitmapFont font;
   public Boolean start = false;
   private boolean singlePlayer;

   public StartScreen(final SuperGroovyPong game) {
      this.game = game;

      camera = new OrthographicCamera();
      camera.setToOrtho(false, 800, 400);
      batch = new SpriteBatch();
      font = new BitmapFont();
      font.getData().setScale(2.15f);
   }

   @Override
   public void render(float delta) {
      Gdx.gl.glClearColor(0, 0, 0, 1);
      Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

      camera.update();
      batch.setProjectionMatrix(camera.combined);

      batch.begin();
      font.draw(batch, "Welcome to Super Groovy Pong", 100, 300);
      font.draw(batch, "Press '1' for Single Player or '2' for Two Players", 100, 250);
      batch.end();


      // DO THE CHANGE PLAYER STATE HERE::::::::::
      if (Gdx.input.isKeyJustPressed(Input.Keys.NUM_1)) {
         singlePlayer = true;
         start = true;
         dispose();
      } else if (Gdx.input.isKeyJustPressed(Input.Keys.NUM_2)) {
         singlePlayer = false;
         start = true;
         dispose();
      }
   }
```

```java
    public boolean SinglePlayer() {
        return singlePlayer;
    }

    @Override
    public void dispose() {
        batch.dispose();
        font.dispose();
    }
}
```

EndScreen.java

```java
package com.pong.game;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.ScreenAdapter;
import com.badlogic.gdx.graphics.GL20;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;

/**
 * Class to build an ending screen
 * @author Laura Waldron, Hana Segura, Antonio Croissy
 */
public class EndScreen extends ScreenAdapter {

    private final SuperGroovyPong game;
    private OrthographicCamera camera;
    private SpriteBatch batch;
    private BitmapFont font;
    private int score;

    public EndScreen(final SuperGroovyPong game, int score) {
        this.game = game;
        this.score = score;

        camera = new OrthographicCamera();
        camera.setToOrtho(false, 800, 400); // Adjust dimensions
        batch = new SpriteBatch();
        font = new BitmapFont();
        font.getData().setScale(2.75f);
    }
    @Override
    public void show() {
        // Setup screen when shown
    }

    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

        camera.update();
        batch.setProjectionMatrix(camera.combined);

        batch.begin();
        font.draw(batch, "Game Over", 100, 300);
        font.draw(batch, "Score: " + score, 100, 250);
        font.draw(batch, "Tap the screen to play again", 100, 200);
        batch.end();

        if (Gdx.input.isTouched()) {
            // Reset the game and switch back to the main screen
            game.dispose();
            game.create();
            dispose();
        }
    }

    @Override
    public void dispose() {
        // Dispose of resources when the screen is no longer shown
        batch.dispose();
        font.dispose();
    }
}
```

Testing file: SuperGroovyPongTest.java

```
import org.junit.Before;
import org.junit.Test;
import static org.junit.Assert.*;
/**
 * Test method for the super groovy pong
 * @author Marco Puig
 */
public class SuperGroovyPongTest {
```

```java
private SuperGroovyPong game;

@Before
public void setUp() {
    game = new SuperGroovyPong();
    game.create(); // Initialize the game
}

@Test
public void testPaddle1Update() {
    // Move paddle1 to a new position
    game.paddle1.setPosition(50, 100);

    // Trigger the update method
    game.render();

    // Check if the paddle1's position has been updated
    assertEquals(50, game.paddle1.getX(), 0);
    assertEquals(100, game.paddle1.getY(), 0);
}

@Test
public void testPaddle2Update() {
    // Move paddle2 to a new position
    game.paddle2.setPosition(500, 200);

    // Trigger the update method
    game.render();

    // Check if the paddle2's position has been updated
    assertEquals(500, game.paddle2.getX(), 0);
    assertEquals(200, game.paddle2.getY(), 0);
}

@Test
public void testBallUpdate() {
    // Move the ball to a new position
    game.ball.setPosition(300, 400);

    // Trigger the update method
    game.render();

    // Check if the ball's position has been updated
    assertEquals(300, game.ball.getX(), 0);
    assertEquals(400, game.ball.getY(), 0);
}
```

```
    @Test
    public void testStartGame() {
        // Call the startGame method
        game.startGame();
    }

    @Test
    public void testShowEndScreen() {
        // Call the showEndScreen method with a mock finalScore
        game.showEndScreen(100);
    }

}
```

References:

https://chat.openai.com/
http://wisenet.fau.edu/class/cop4331/notes/
https://www.geeksforgeeks.org/state-design-pattern/
https://canvas.fau.edu/courses/143042/modules