# Billiard 5

*Final project in computer science planning -*

*- and programming object-oriented systems 5*

*Study units on the UWP platform.*

| | |
|---|---|
| Project Name | Billiard |
| Programming Language: | C#, XAML, SQL |
| Developer Name | Noam Erlich |
| ID: | ******** |
| School: | Ort Naomi Shemer |
| Teacher Name: | Oleg Katsnelson |
| Year: | 2024 |

# Table of Contents

# 1. Introduction

## 1.1 Background

The project I choose to make is "Billiard", I choose this project because I used to play a lot of billiard on a phone app called '8 Ball Pool' and I had a lot of fun, also I wanted to challenge myself to make a difficult project with a lot of physics, math, and thinking inside of it.
While making the project I had to study a bunch of new material and re-study a lot of physics from last year (II'th grade) and learn even more, just to make the physics in my game as close to real as possible, mostly because my work environment (UWP) isn't really suited for that kind of project and for a lot of physics.

My game was built on the Universal Windows Platform (UWP), using Three programming languages:

Xmal – Responsible for designing the project interface.

C# – Responsible for the logic part of the game.

SQL – Intended for the database in the project.

1.2 Game Logic

- Game Background: 'Billiard', also known as pool, is a popular tabletop game played on a rectangular table with pockets in each corner and along the sides. Players use a cue stick to strike colored balls, aiming to pocket them into the table's pockets according to specific rules. It's a game of skill, precision, and strategy, enjoyed by players of all ages around the world for centuries.

- Game Target: Each player needs to push all of his balls (solid balls or striped balls) into the pockets on the table (Holes), and then the black ball, to win.
If somebody enters the black ball by accident before entering all of his balls, he loses, and the other player wins.

- Game process:
In the game there is a table, on the table there are 15 balls that are placed in a shape of a triangle (7 solid balls, 7 striped balls, and 1 black ball), and more one white ball in front of the triangle.

There are 2 players in the game, the goal of each player is to move all the balls of their type (solid or striped) into the holes on the table frame.

Selecting the type of balls for each player is usually determined in the second turn of the game, when in the first turn one of the players hits with the white ball all the other balls in the triangle together and then all of the balls are scattered on the board, if not even one ball enter the holes, the turn goes to the next player, if some of the balls entered, the player gets another turn and if he puts a ball in the holes, the first type of ball that he puts in will be his type of ball.

Removing the balls from the game is done by using a stick that pushes the white ball with a blow, the player determines the power of the blow with the help of the power bar, with the white ball the players have to hit the balls and put their balls in the holes.

A player's turn continues as long as he puts at least one ball of his type into one of the holes on his turn.

The game ends when one player has put all of his balls into the holes, and finally the black ball, this player is the winner.

**Money** – After every billiard match, the user earns $100 that he can spend in the shop, buying new table colors, or save.

**There are several clear rules in the game:**

- If a player puts in the black ball before he has put in the rest of his balls, he automatically loses.

- It is mandatory to hit the white ball in order to move the balls on the table.

- If a player accidentally puts the white ball into the holes, both his turn is over and the other player is allowed to decide where he puts the white ball on the table.

- If a player hits a ball of the other player's type with the white ball first before hitting any other balls then his turn is also over and the other player can place the cue ball wherever he wants.

- On a player's last turn, if he puts the black ball together with the white, he automatically loses.

**How to play(buttons):**

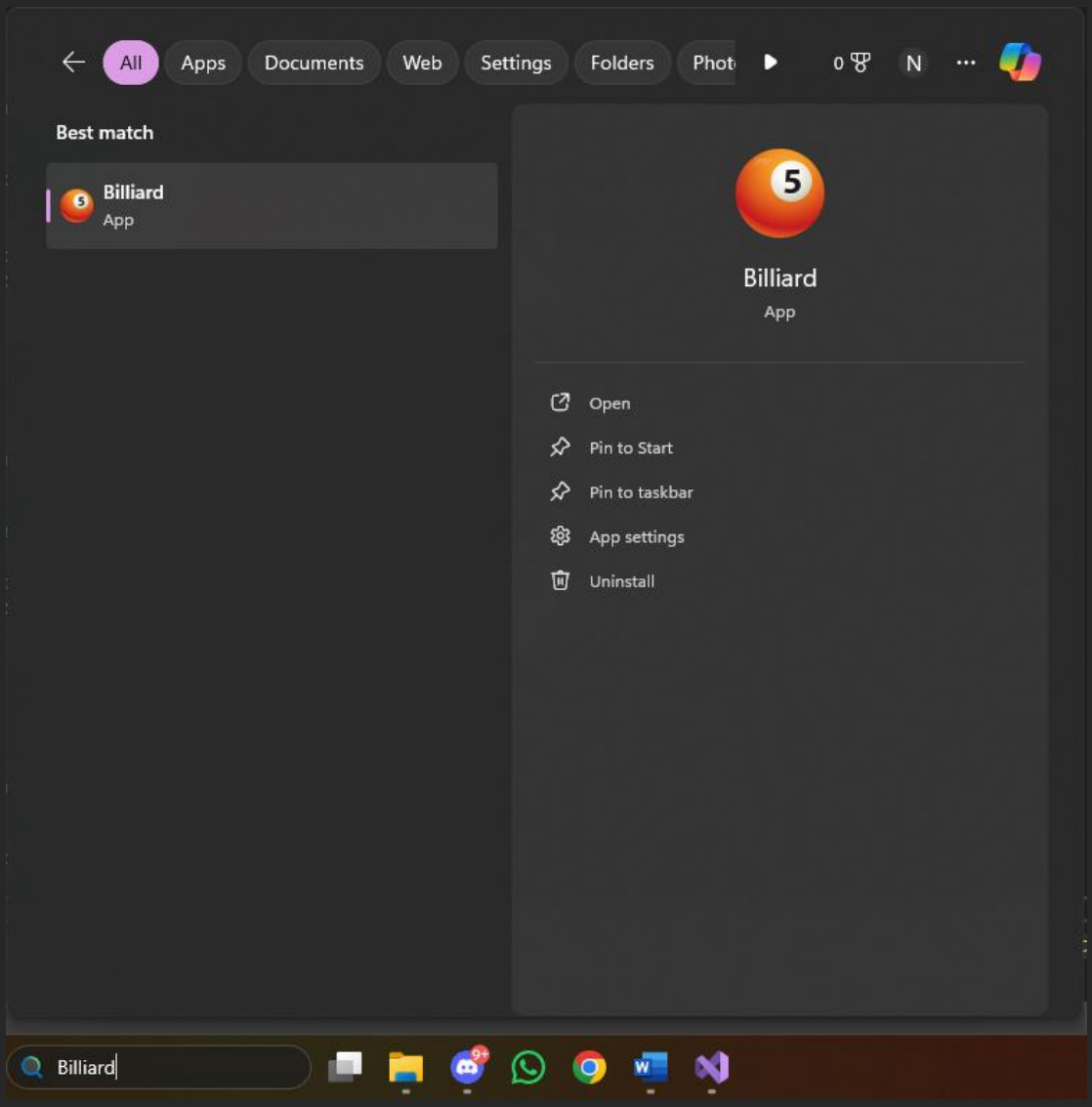
#Use the left button on the mouse in order to set the stick(cue) angle.

#Use the scroll wheel on the mouse in order to set the stick(cue) power blow.

#Use the right button on the mouse or the space bar key in order to give a blow to the white ball.

## 1.3 Launch Game

In order to start up the game, the user needs to open the search bar, type the game name- "Billiard" and click on it.

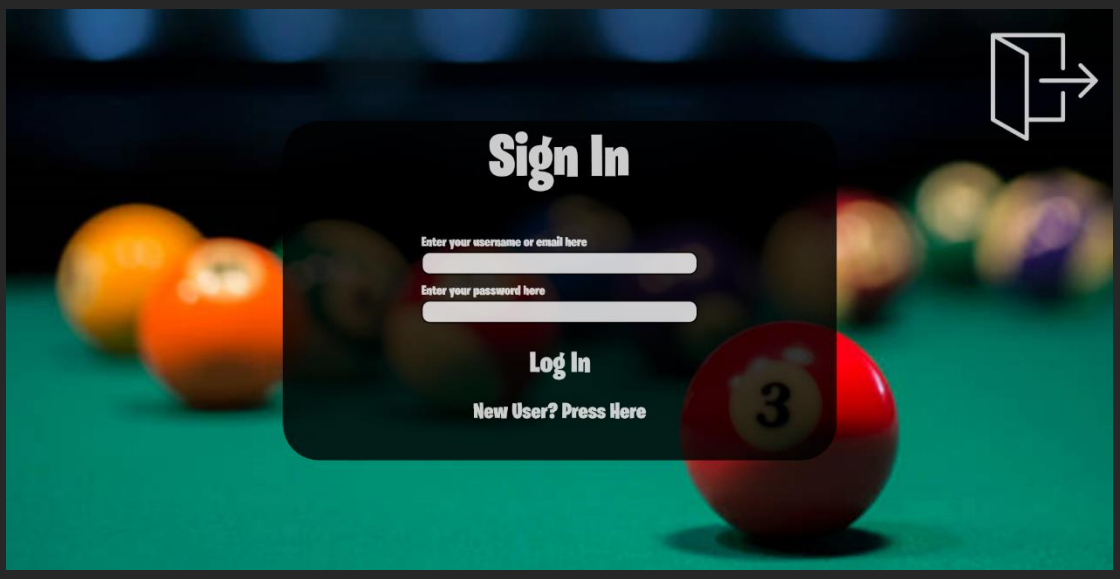


# 2. User's Guide

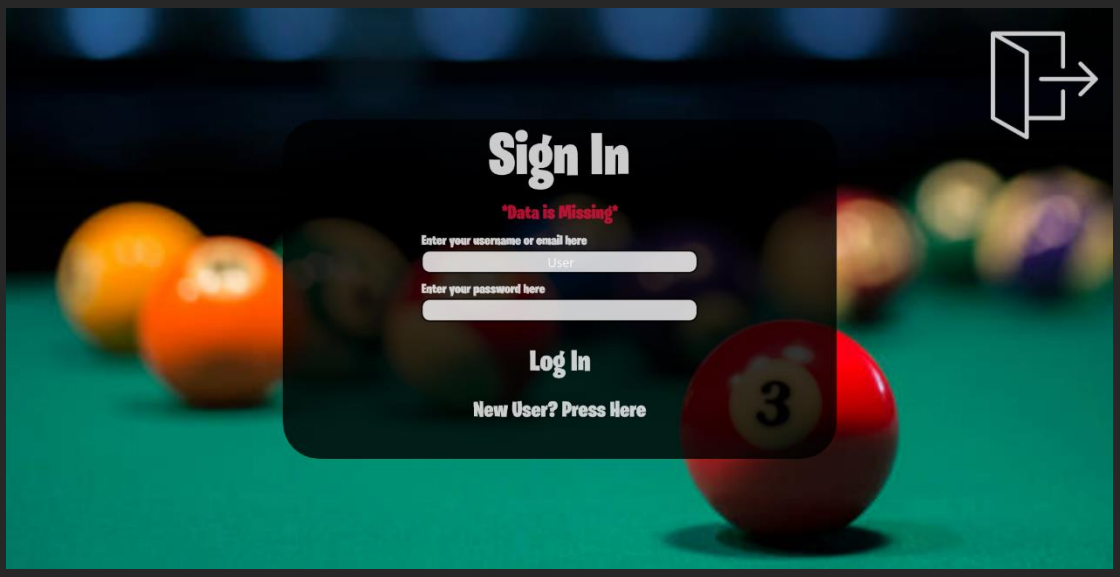
## 2.1 Sign In\Register Page

### Sign In

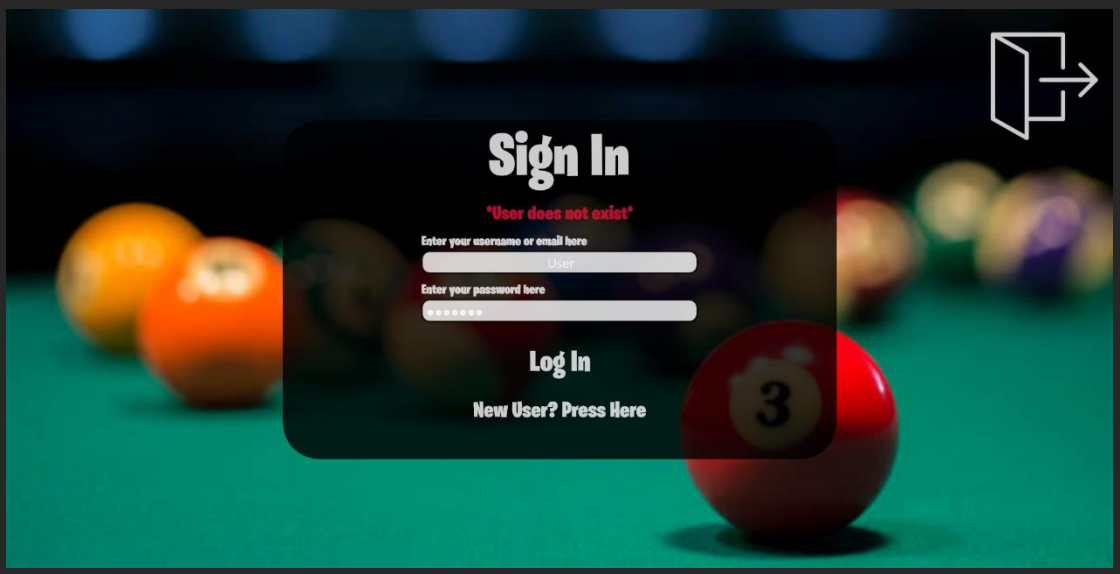
The first page that loads up for the user, the game asks the user to sign in.
The top right door button is used to exit the game.



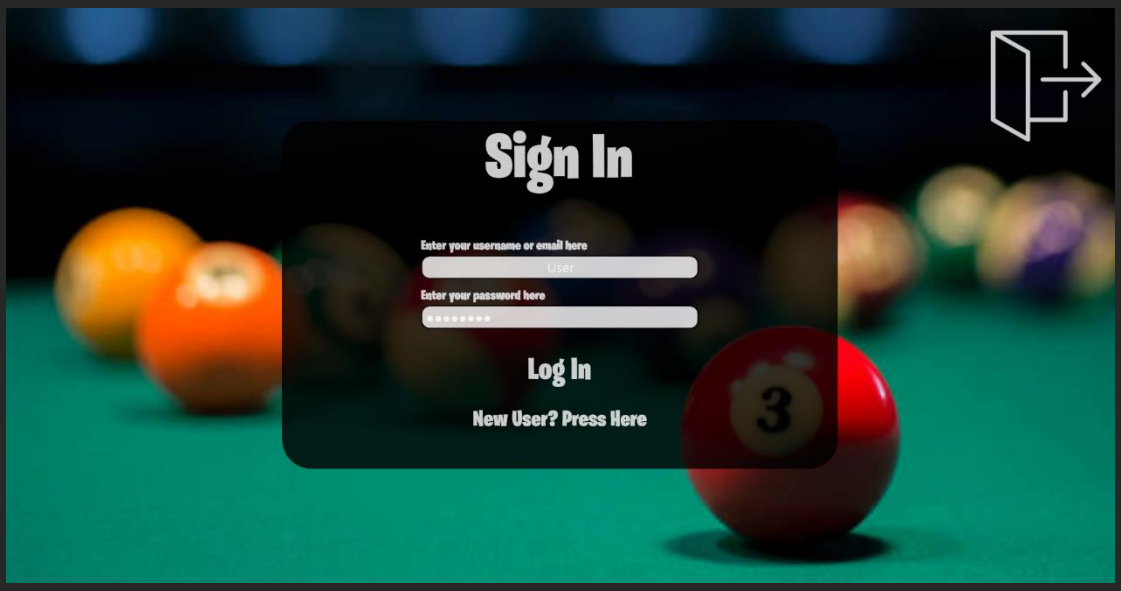

The error message that shows up when user didn't fill all boxes.



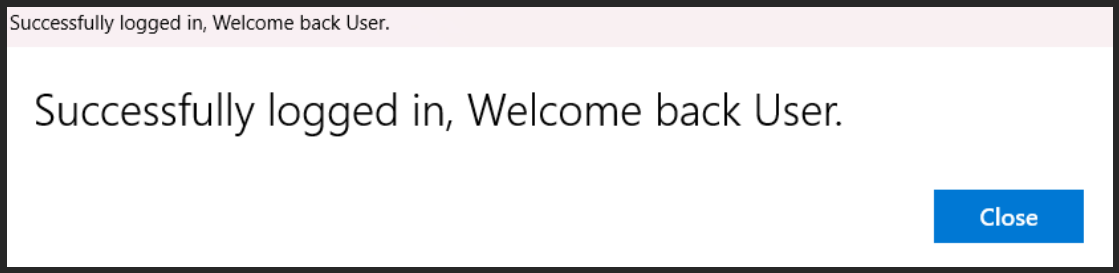
The error message that shows up when user don't exist in data base.




The correct way to sign in without any errors.

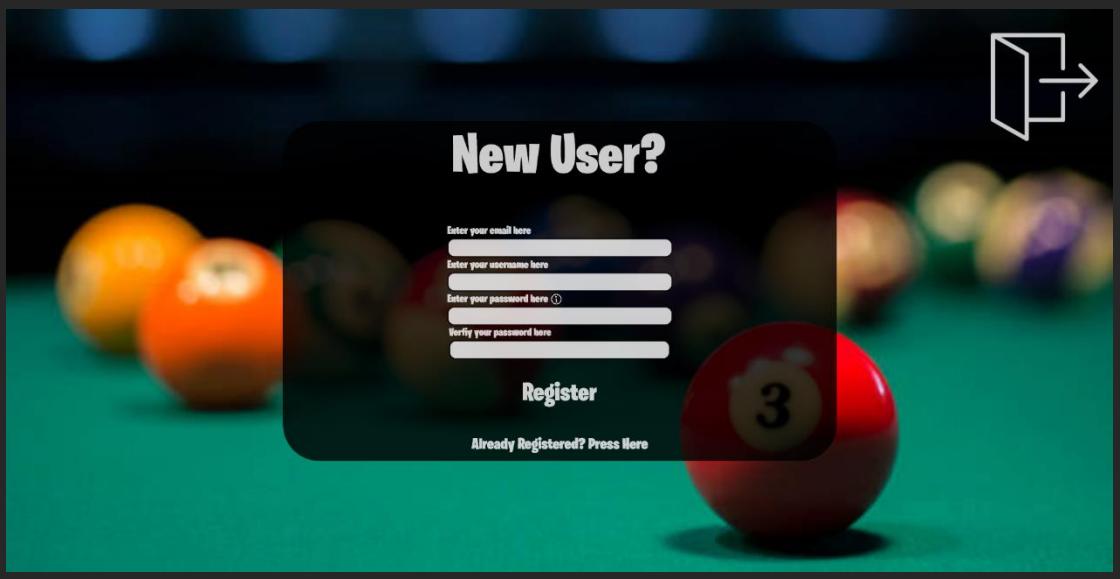



The message that shows up if player successfully logged in.

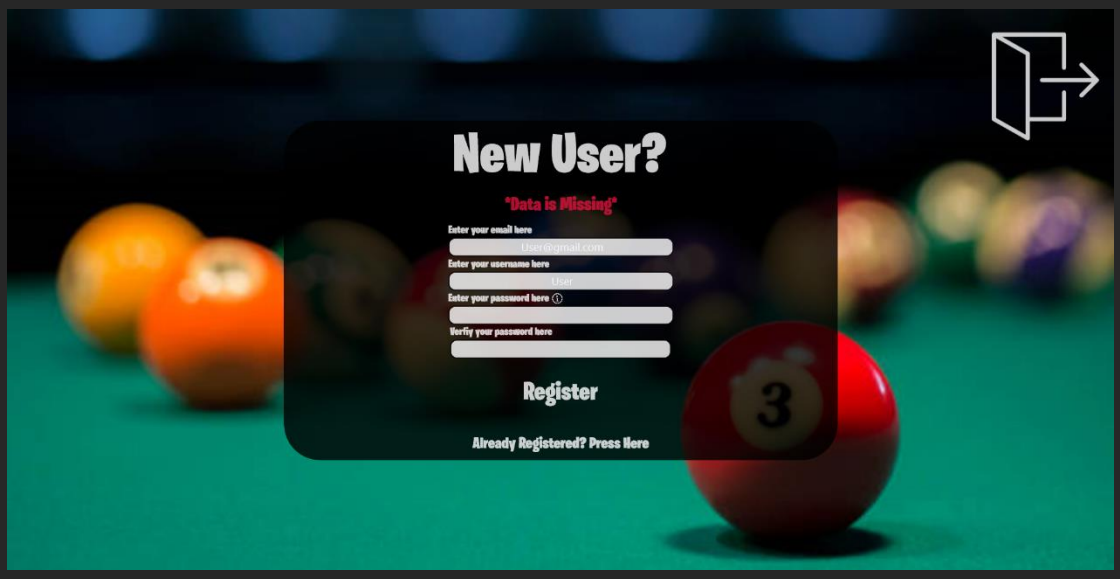


## Register

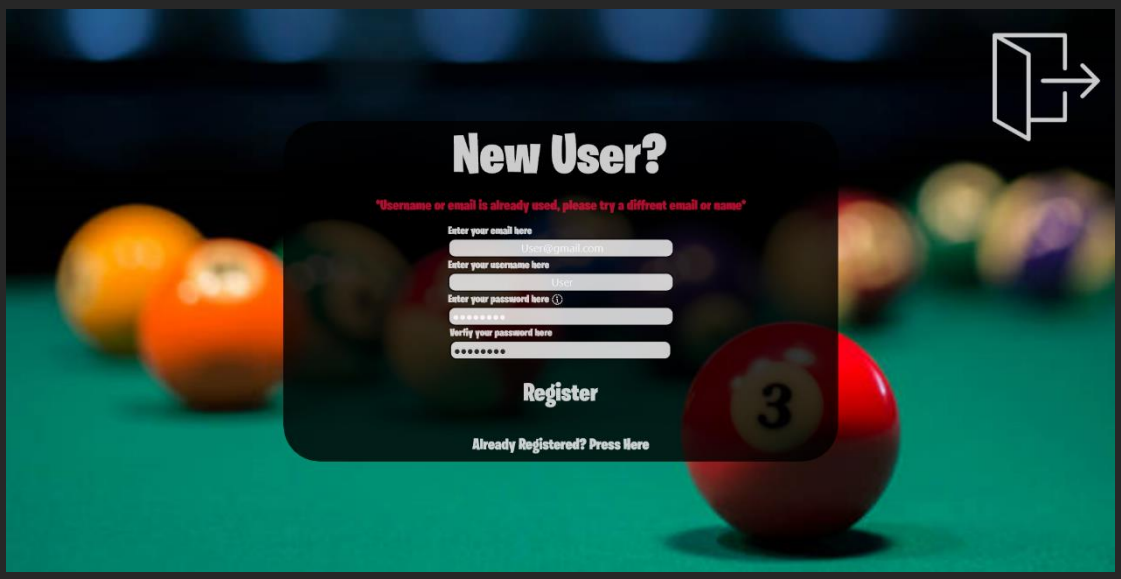
Registration page for new users, the game asks the user to sign up.



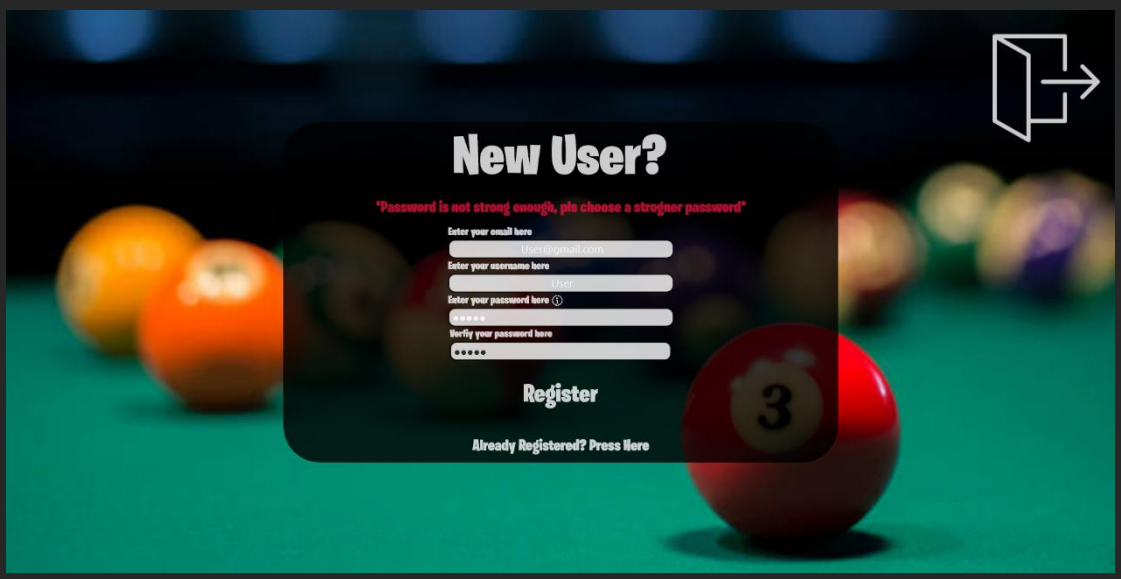

The error message that shows up when user didn't fill all boxes.



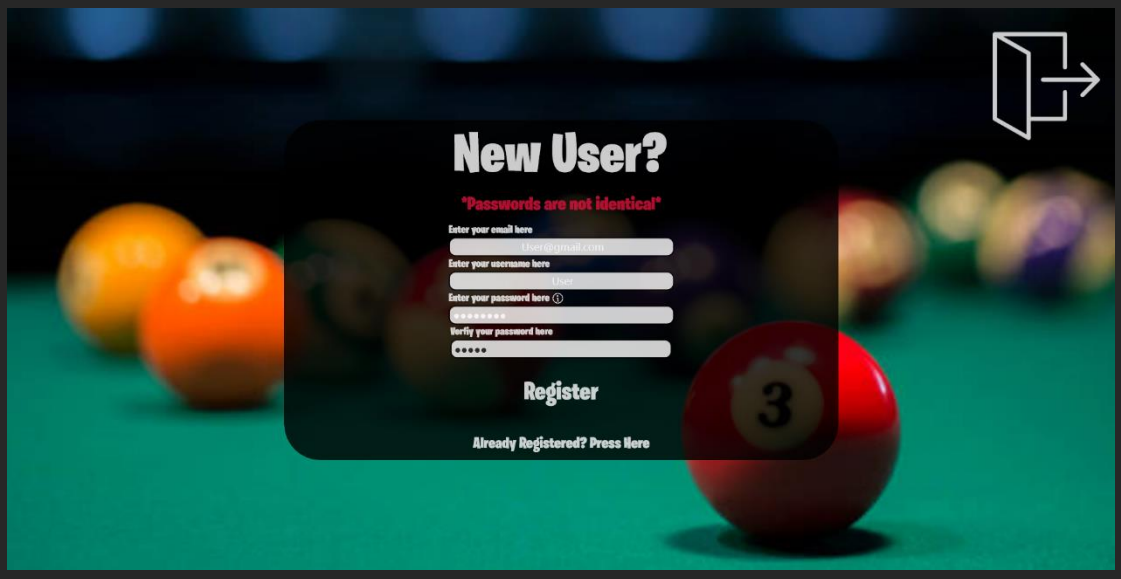
The error message that shows up when username or email is already used.



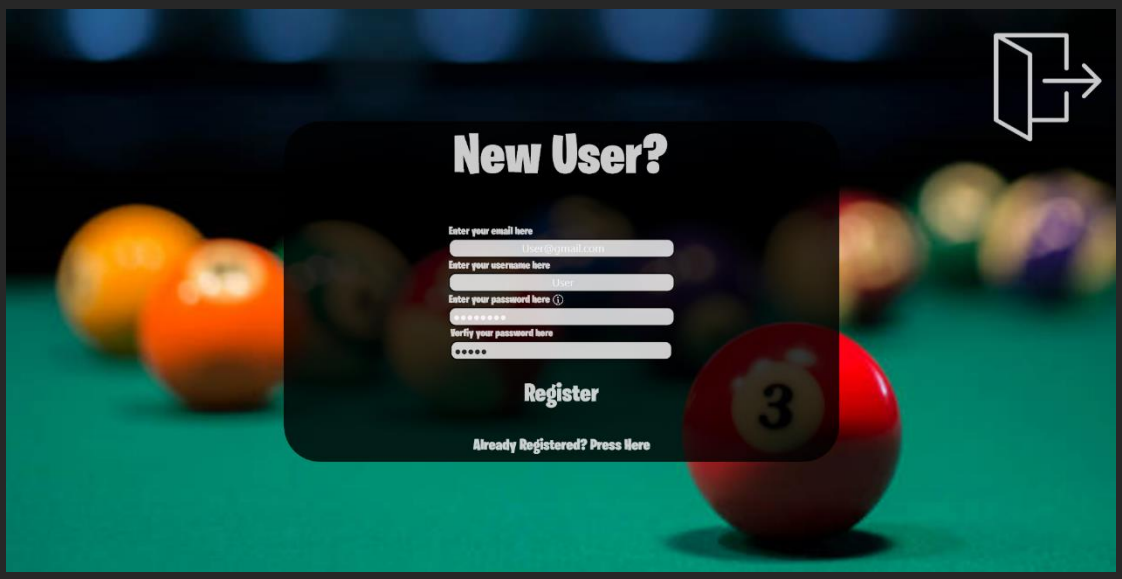

The error message that shows up when password is not strong enough.



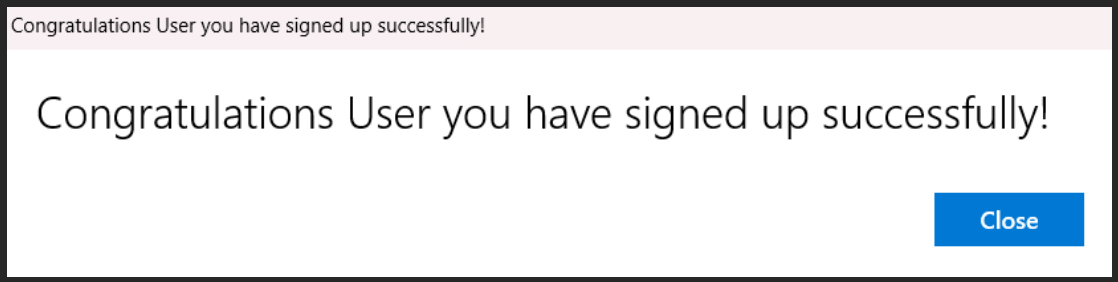
The error message that shows up when passwords are not identical.




The correct way to signed up without any errors.




The message that shows up if player successfully signed up.

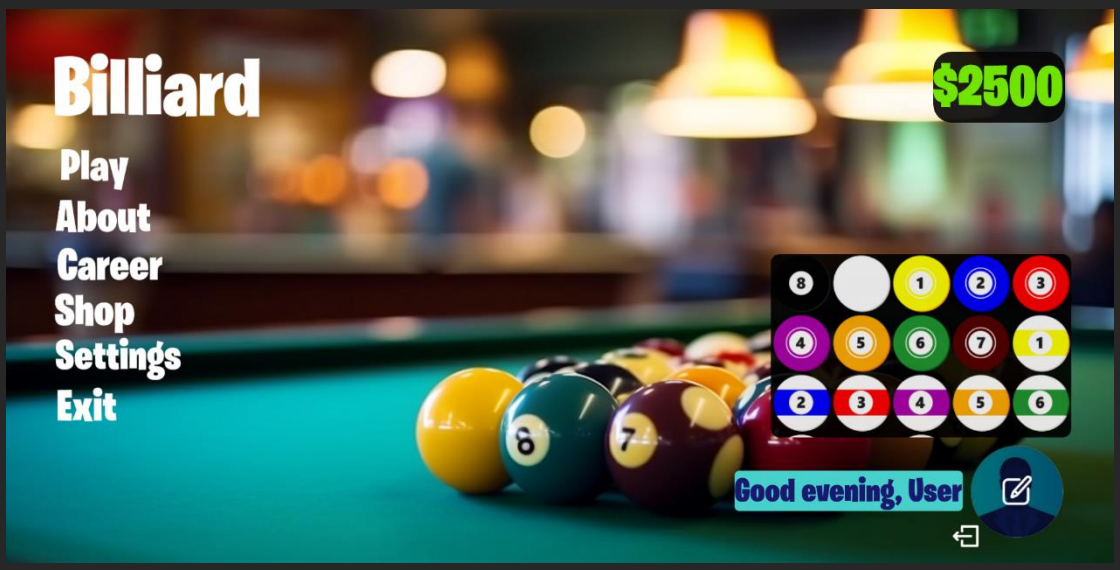


## 2.2 Menu Page

The menu page is the main page of the game, from there the player can navigate to any page he wants, change settings, and exit the game.


## Edit Avatar

The window that shows up if player pressed on the avatar logo, this window is used to change profile picture(not working yet, just an idea).



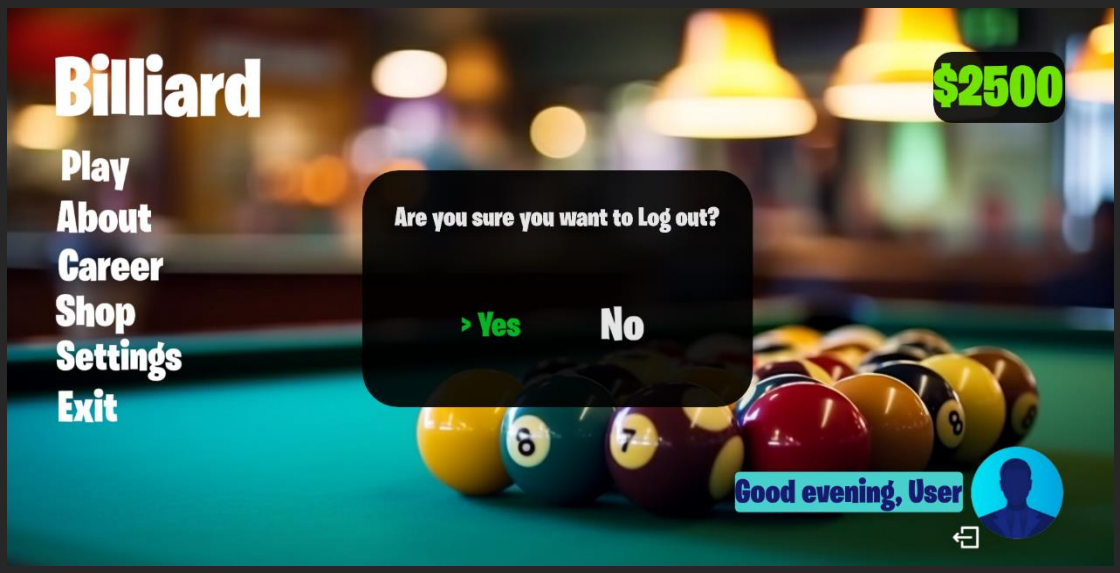
## Settings

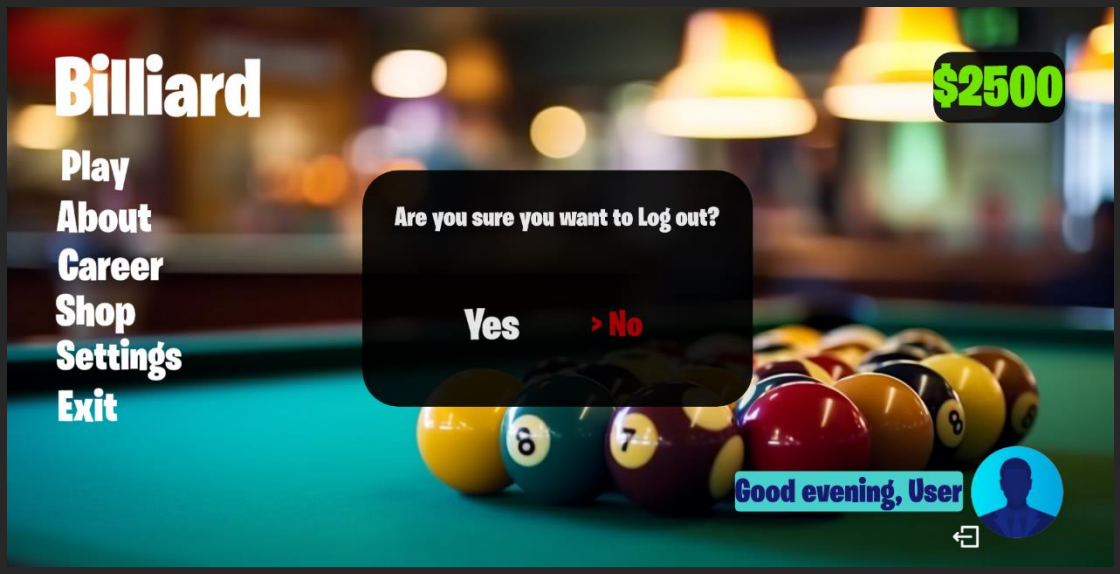
The settings window that shows up when player presses the settings button, in this window, player can turn on/off the music and sound effects (sfx) and adjust the volume of both.


An example when music and sfx are off and volume is changed.

## Log Out

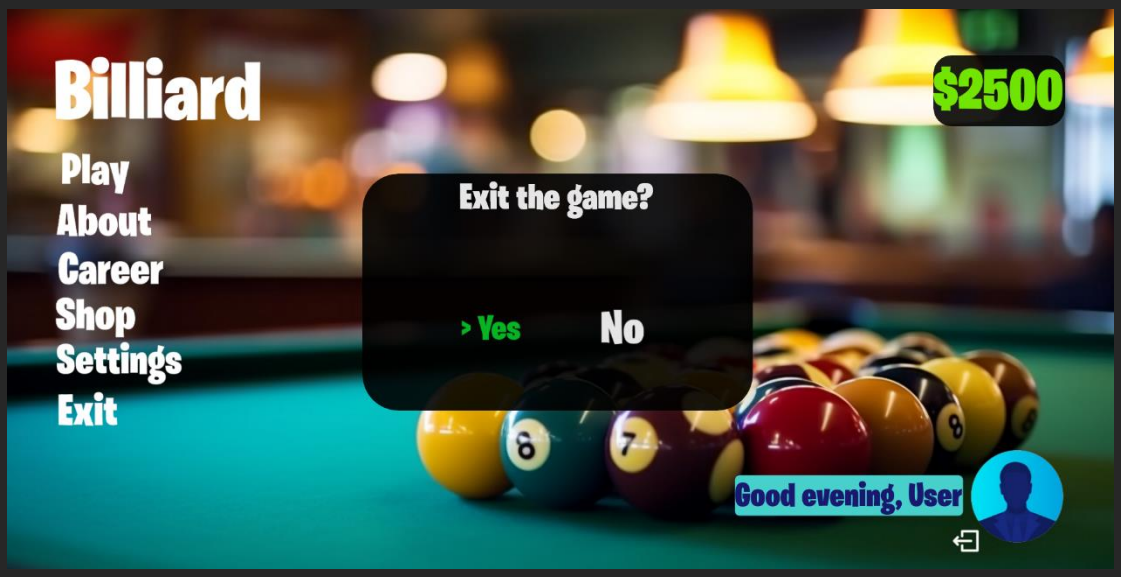
The log out window that shows up when player presses the exit door button next to the profile picture, in this window, player choose if he wants to log out or not.

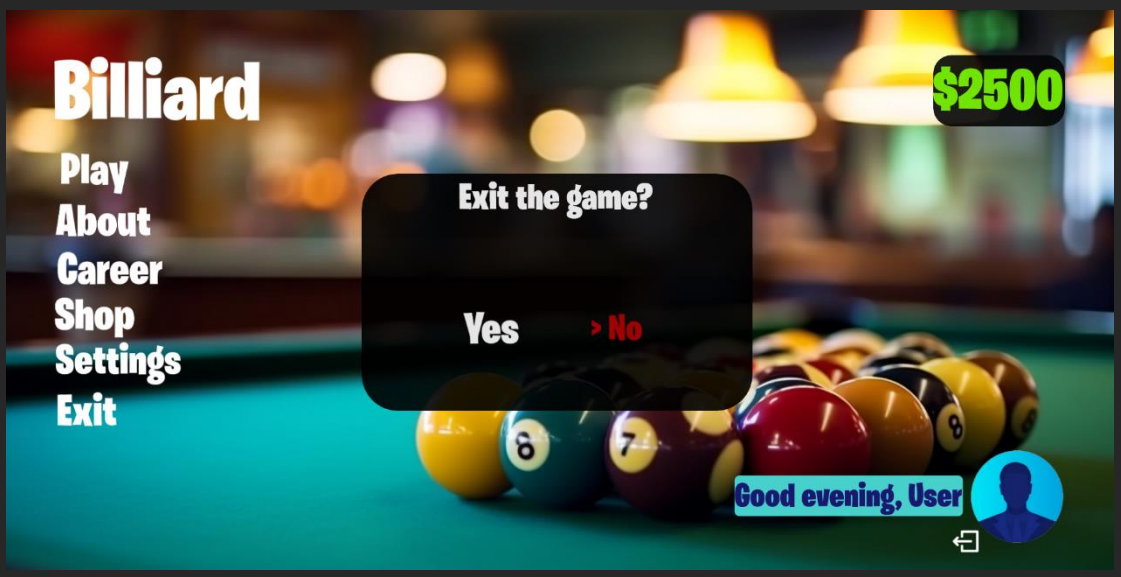


## Exit

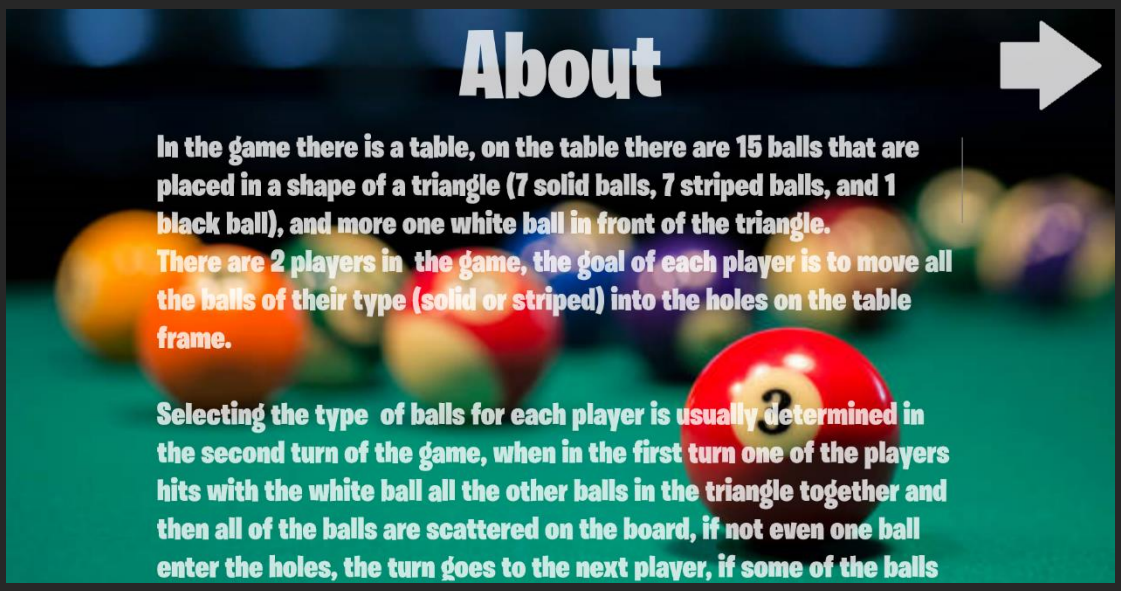
The exit window that shows up when player presses the exit button, in this window, player choose if he wants to exit the game or not.

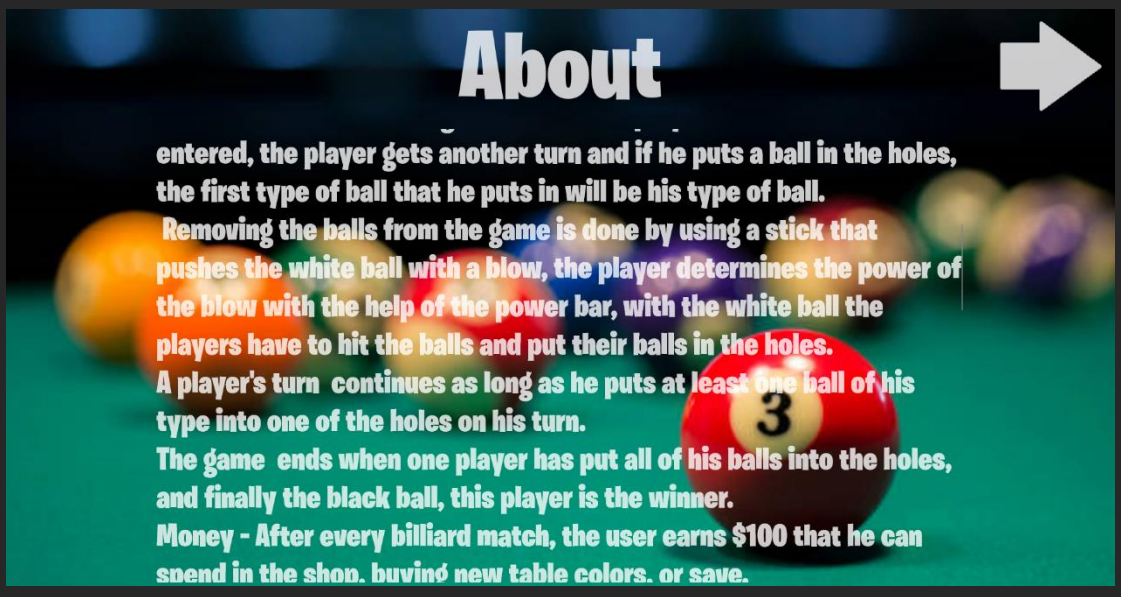


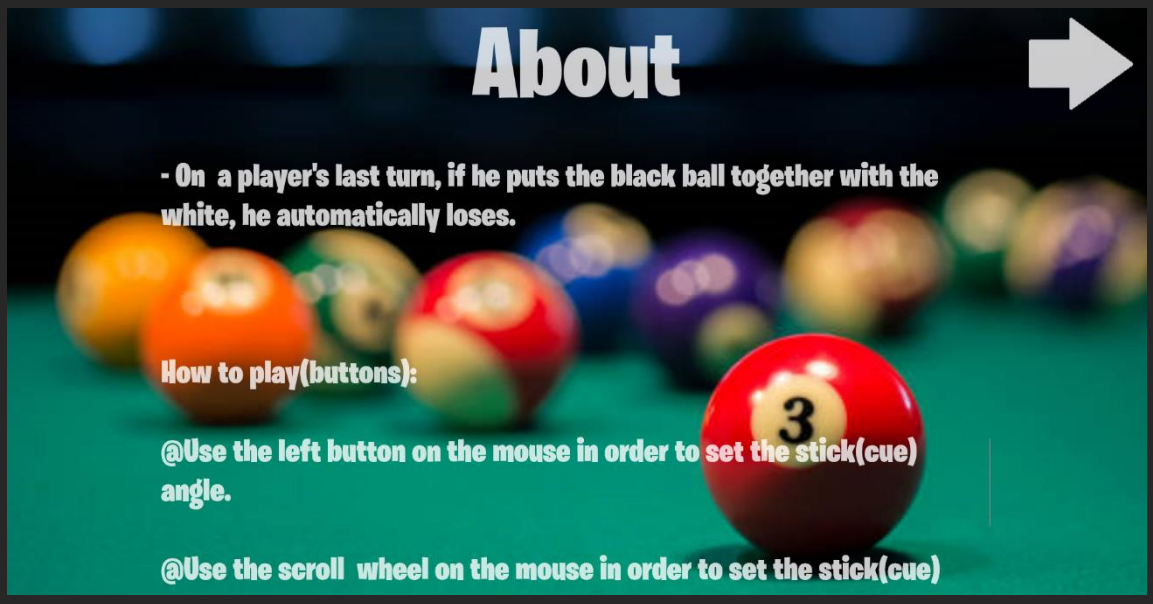
## 2.3 About Page

This page is "About Page", it has all the info about the game, rules, buttons, how to play and extra information about the game. To return to main menu player need to press on the top right arrow.




About
There are several clear rules in the game:
- If a player puts in the black ball before he has put in the rest of his balls, he automatically loses.
- It is mandatory to hit the white ball in order to move the balls on the table.
- If a player accidentally puts the white ball into the holes, both his turn is over and the other player is allowed to decide where he puts the white ball on the table.


About
- On a player's last turn, if he puts the black ball together with the white, he automatically loses.
How to play(buttons):
@Use the left button on the mouse in order to set the stick(cue) angle.
@Use the scroll wheel on the mouse in order to set the stick(cue)

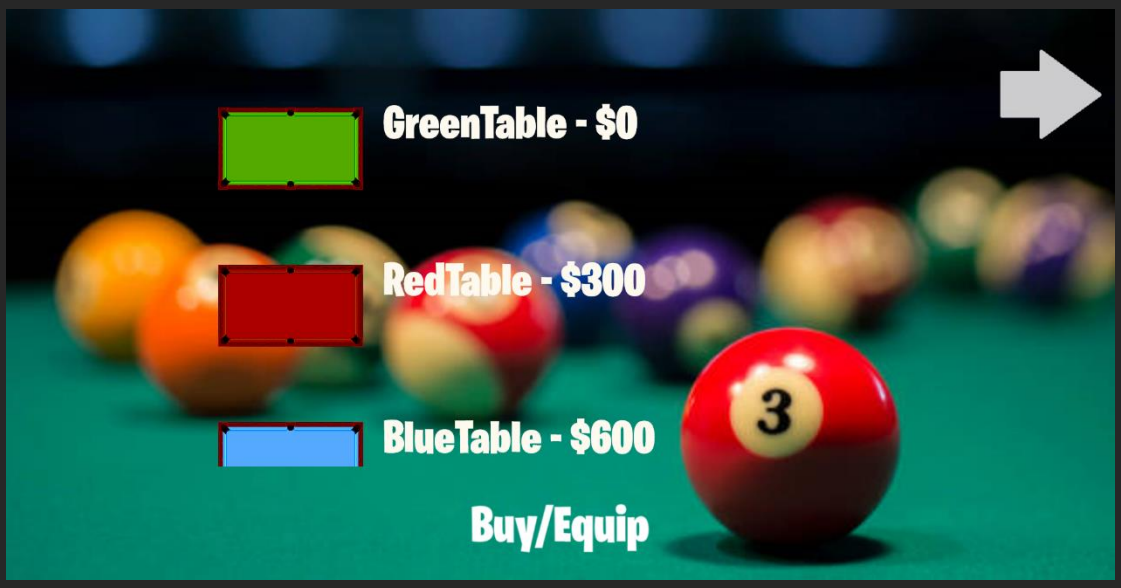

About
@Use the left button on the mouse in order to set the stick(cue) angle.
@Use the scroll wheel on the mouse in order to set the stick(cue) power blow.
@Use the right button on the mouse or the space bar key in order to give a blow to the white ball.
Special thanks: Liel Kigel, Avihai Aharon, Liam Mark, David Vuller, Oleg Katsnelson, Margalit Golkarov

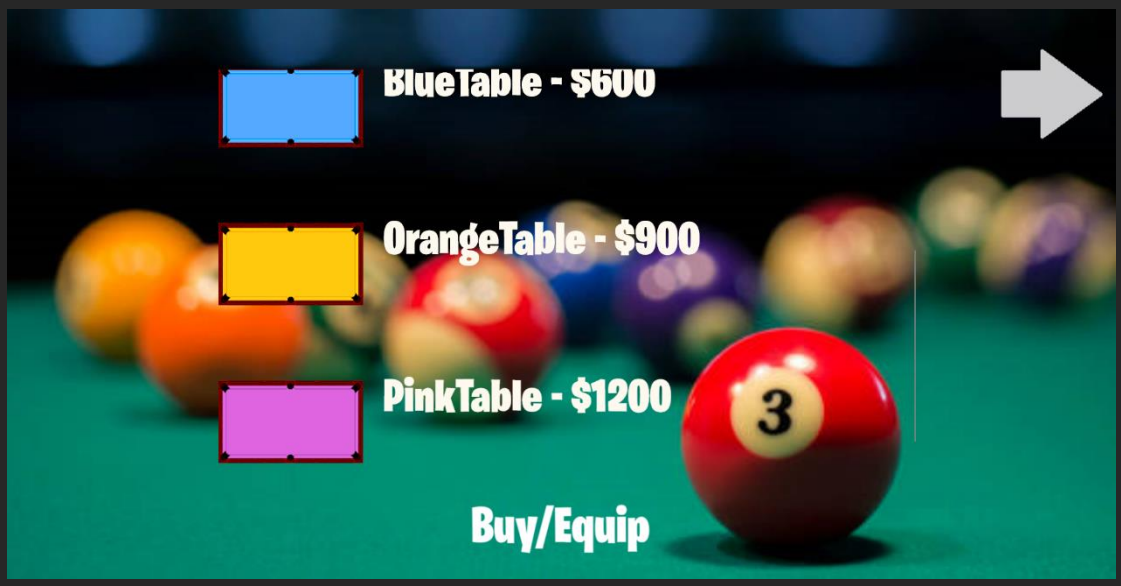
## 2.4 Carrer Page

This page is "Career Page", it shows a leaderboard of all the registered players in the game and shows their placement based on who has the most money.
To return to main menu player need to press on the top right arrow.

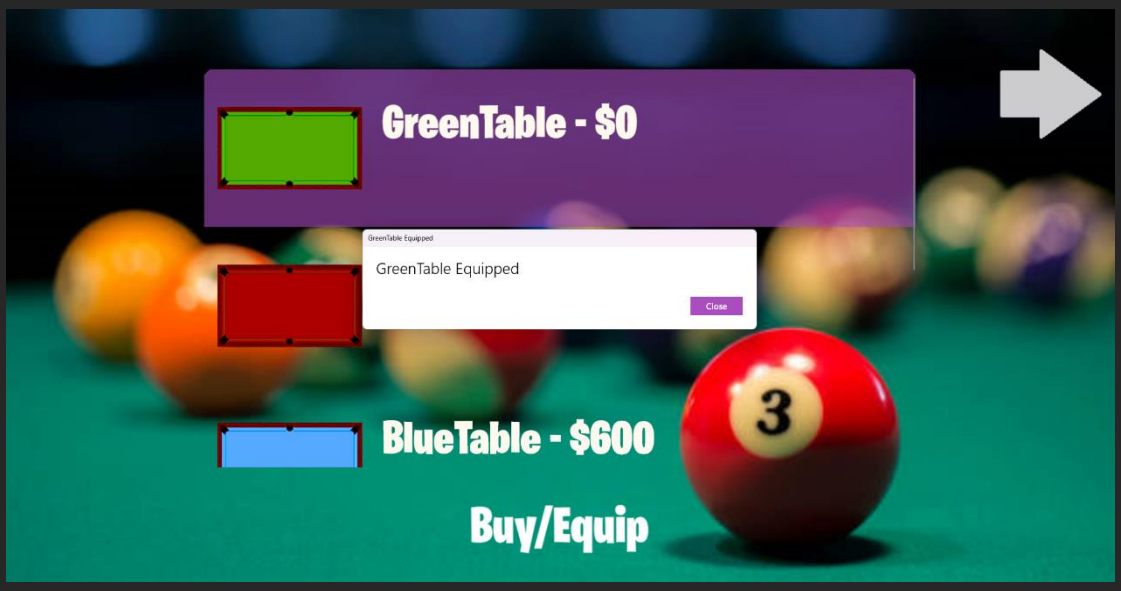
## 2.5 Shop Page

This page is "Shop Page", it shows all the items that player can purchase if they have enough money or equipped if they already own them.
To return to main menu player need to press on the top right arrow.

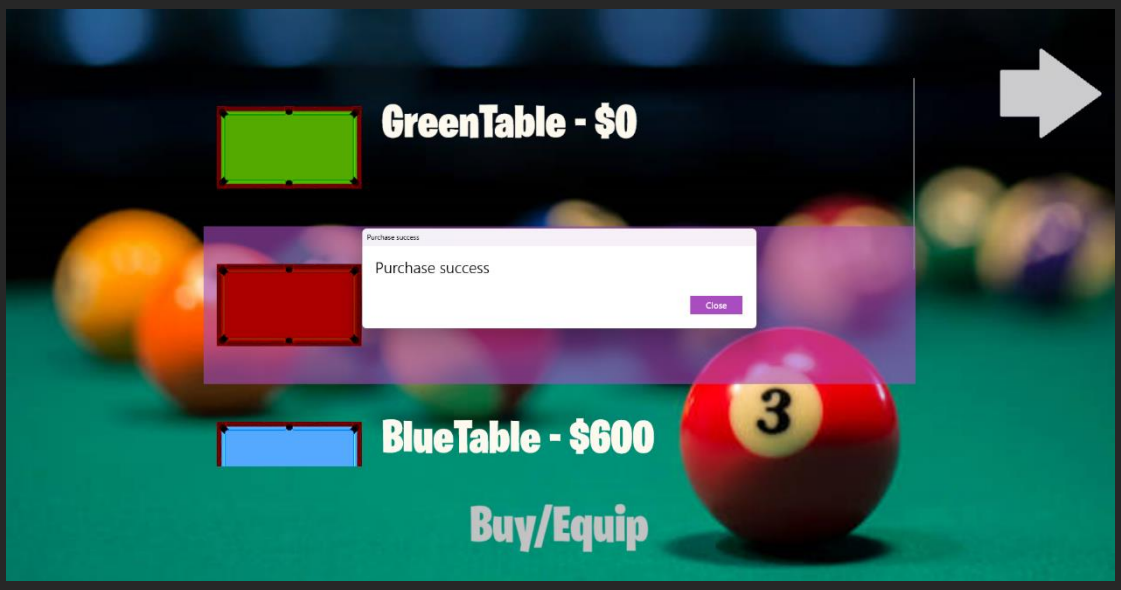


The message that shows up if player owns the selected item.




The message that shows up if player bought the selected item.



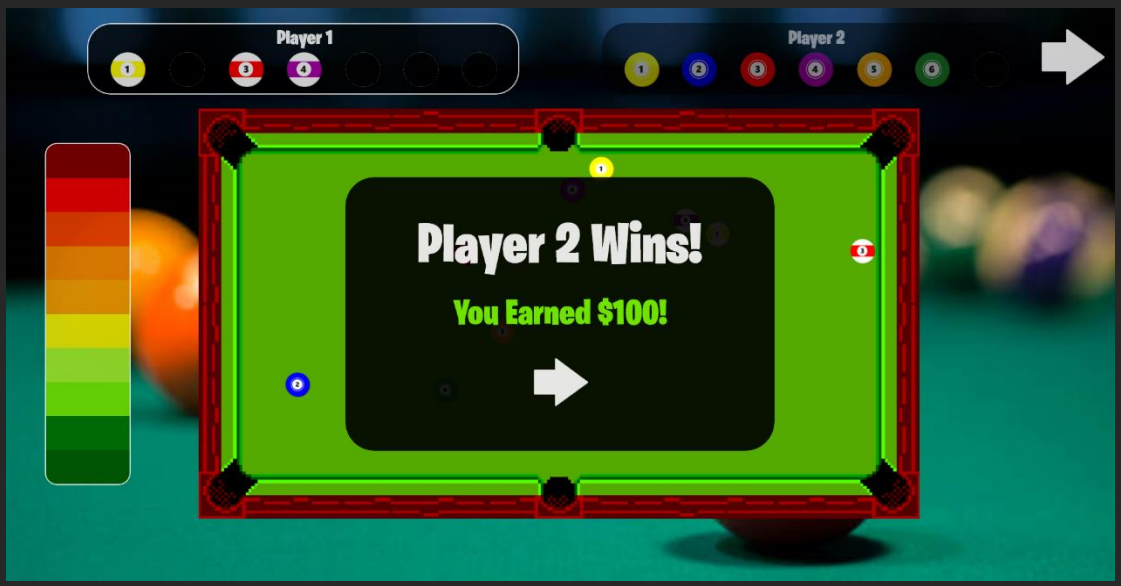
## 2.6 Game Page

This page is "Game Page", this is the main game, where the players can play against each other and earn money to upgrade the table or to save it and be the user with the most money.
To return to main menu player need to press on the top right arrow.


In order to play, user needs to scroll up or down to adjust the power bar which sets the blow power on the white ball.
Left click is used to choose the angle that the ball will move at, and pressing right click or the space bar will shoot the ball with those settings (angle, power).

A picture from the middle of the game: (after players has their unique balls style and after solid ball 1 and 4 got out from the game- was pushed to the holes)


The window that shows up when one of the players has won (in this picture you can see that at player one turn, he entered the black ball accidently and therefore he lost the match).
The logged user earned $100 for finishing the match.



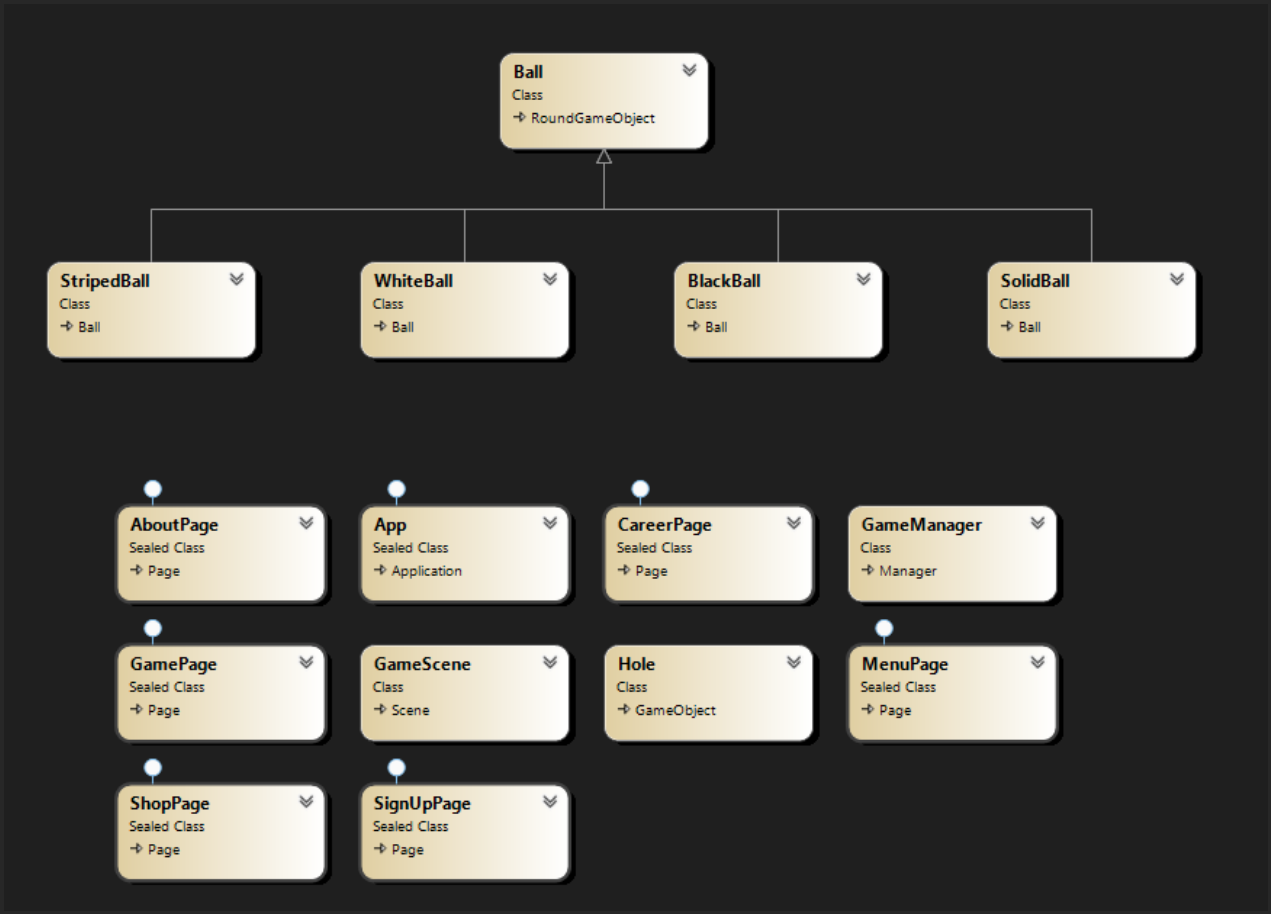
A picture from the middle of the game: (after player one entered all of his ball except the black one, if he enters it, he wins).


After he entered the black ball and all of his other balls at the previous turn, he won, and the logged user earned $100 for finishing the match.

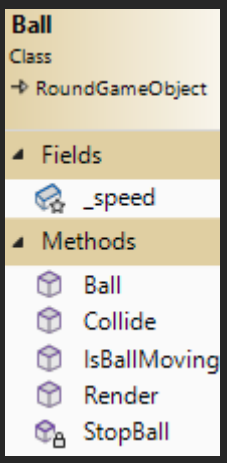
# 3. Programmer's guide

## 3.1 Billiard

### 3.1.1 Project Diagram



## 3.1.2 Ball Class




This class represents a ball object in a billiard game. It includes methods for checking if the ball is moving, stopping its movement, rendering its movement and handling collisions with scene boundaries or other game objects like balls and holes. It encapsulates the behavior and interactions of a ball within the billiard game environment.

Ball inherits from RoundGameObject.

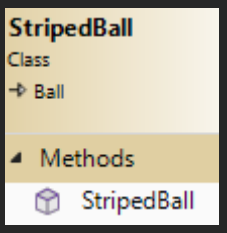
### Class Fields:

```
protected double _speed; //The initial speed of the ball
```

### Class Methods:

```
/// <summary>
/// Initializes a new instance of the <see cref="Ball"/> class.
/// </summary>
/// <param name="scene">The scene where the ball exists.</param>
/// <param name="fileName">The file name of the image representing the ball.</param>
/// <param name="speed">The initial speed of the ball.</param>
/// <param name="width">The width of the ball.</param>
/// <param name="placeX">The initial X coordinate of the ball.</param>
/// <param name="placeY">The initial Y coordinate of the ball.</param>
4 references
public Ball(Scene scene, string fileName, double speed, double width, double placeX, double placeY) ...

/// <summary>
/// Checks if the ball is currently moving.
/// </summary>
/// <returns>True if the ball is moving, otherwise false.</returns>
1 reference
public bool IsBallMoving()...

/// <summary>
/// Stops the movement of the ball.
/// </summary>
1 reference
private void StopBall()...

/// <summary>
/// Renders the ball's movement and handles collisions with the scene boundaries.
/// </summary>
9 references
public override void Render()...

/// <summary>
/// Handles collisions between the ball and other game objects, such as other balls and holes.
/// If the ball collides with another ball, it adjusts the speeds of both balls and fixes their positions to prevent overlap.
/// If the ball collides with a hole, it updates game state variables accordingly, removes the ball from the scene, and stops its movement.
/// </summary>
/// <param name="gameObject">The game object the ball collides with.</param>
3 references
public override void Collide(GameObject gameObject)...
```

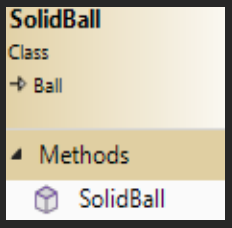
### 3.1.3 StripedBall Class




Represents a striped ball in the game.
StripedBall inherits from Ball.

Class Methods:

```
/// <summary>
/// Initializes a new instance of the <see cref="StripedBall"/> class.
/// </summary>
/// <param name="scene">The scene where the striped ball exists.</param>
/// <param name="fileName">The file name of the image representing the striped ball.</param>
/// <param name="speed">The initial speed of the striped ball.</param>
/// <param name="width">The width of the striped ball.</param>
/// <param name="placeX">The initial X coordinate of the striped ball.</param>
/// <param name="placeY">The initial Y coordinate of the striped ball.</param>
1 reference
public StripedBall(Scene scene, string fileName, double speed, double width, double placeX, double placeY)
    : base(scene, fileName, speed, width, placeX, placeY)
{
}
```



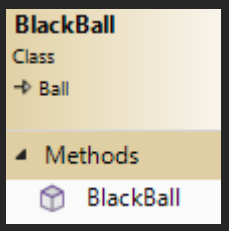

## 3.1.4 SolidBall Class

Represents a solid ball in the game.
SolidBall inherits from Ball.

### Class Methods:

```
/// <summary>
/// Initializes a new instance of the <see cref="SolidBall"/> class.
/// </summary>
/// <param name="scene">The scene where the solid ball exists.</param>
/// <param name="fileName">The file name of the image representing the solid ball.</param>
/// <param name="speed">The initial speed of the solid ball.</param>
/// <param name="width">The width of the solid ball.</param>
/// <param name="placeX">The initial X coordinate of the solid ball.</param>
/// <param name="placeY">The initial Y coordinate of the solid ball.</param>
1 reference
public SolidBall(Scene scene, string fileName, double speed, double width, double placeX, double placeY)
    : base(scene, fileName, speed, width, placeX, placeY)
{
}
```

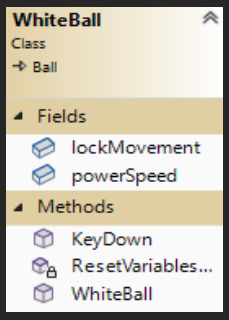
### 3.1.5 BlackBall Class




Represents a black ball in the game.
BlackBall inherits from Ball.

## Class Methods:

```
/// <summary>
/// Initializes a new instance of the <see cref="BlackBall"/> class.
/// </summary>
/// <param name="scene">The scene where the black ball exists.</param>
/// <param name="fileName">The file name of the image representing the black ball.</param>
/// <param name="speed">The speed of the black ball.</param>
/// <param name="width">The width of the black ball.</param>
/// <param name="placeX">The initial X coordinate of the black ball.</param>
/// <param name="placeY">The initial Y coordinate of the black ball.</param>
1 reference
public BlackBall(Scene scene, string fileName, double speed, double width, double placeX, double placeY
    : base(scene, fileName, speed, width, placeX, placeY)
{
}
```




## 3.1.6 WhiteBall Class

This class represents the white ball in a billiard game. It handles its initialization, movement control via key presses, and resetting variables and settings related to the white ball and game turn.
WhiteBall inherits from Ball.

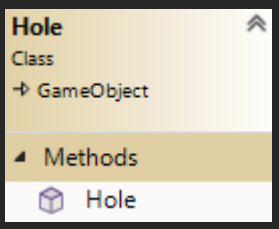
### Class Fields:

```
public static bool lockMovement; // Indicates whether the movement of the white ball is locked.

public static double powerSpeed; // The power speed of the white ball.
```

### Class Methods:

```
/// <summary>
/// Initializes a new instance of the <see cref="WhiteBall"/> class.
/// </summary>
/// <param name="scene">The scene where the white ball exists.</param>
/// <param name="fileName">The file name of the image representing the white ball.</param>
/// <param name="speed">The initial speed of the white ball.</param>
/// <param name="width">The width of the white ball.</param>
/// <param name="placeX">The initial X coordinate of the white ball.</param>
/// <param name="placeY">The initial Y coordinate of the white ball.</param>
3 references
public WhiteBall(Scene scene, string fileName, double speed, double width, double placeX, double placeY)...

/// <summary>
/// Resets variables and settings related to the white ball and to game turn.
/// </summary>
1 reference
private void ResetVariablesAndSettings()...

/// <summary>
/// Handles the key down event for controlling the white ball's movement.
/// </summary>
/// <param name="key">The virtual key that is pressed.</param>
2 references
public void KeyDown(VirtualKey key)...
```

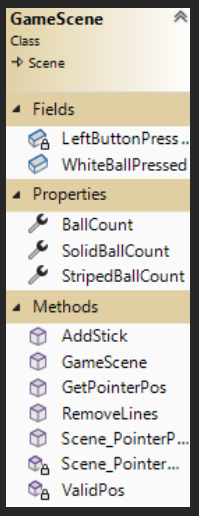
## 3.1.7 Hole Class




Represents a Hole in the table in the game.

### Class Methods:

```
/// <summary>
/// Initializes a new instance of the <see cref="Hole"/> class.
/// </summary>
/// <param name="scene">The scene where the hole exists.</param>
/// <param name="fileName">The file name of the image representing the hole.</param>
/// <param name="width">The width of the hole.</param>
/// <param name="height">The height of the hole.</param>
/// <param name="placeX">The initial X coordinate of the hole.</param>
/// <param name="placeY">The initial Y coordinate of the hole.</param>
6 references
public Hole(Scene scene, string fileName, double width, double height, double placeX, double placeY)
    : base(scene, fileName, width, height, placeX, placeY)
{
}
```




## 3.1.8 GameScene Class

This class manages the scene of a billiard game, handling user interactions such as pointer movements and clicks. It allows adjustment of shot power via mouse wheel, enables interaction with the white ball through mouse clicks, and visualizes the stick angle for ball movement. Additionally, it validates the position of the white ball to prevent overlaps with other objects in the scene. Overall, it ensures smooth gameplay and user interaction within the billiard game environment.

### Class Fields & Properties:

```
public static bool WhiteBallPressed; // Indicates whether the white ball has been pressed by the user.
/// <summary>
/// Gets the total count of balls in the scene.
/// </summary>
0 references
public int BallCount => _gameObjectsSnapshot.Count(x => x is Ball);

/// <summary>
/// Gets the count of solid balls in the scene.
/// </summary>
0 references
public int SolidBallCount => _gameObjectsSnapshot.Count(x => x is SolidBall);

/// <summary>
/// Gets the count of striped balls in the scene.
/// </summary>
0 references
public int StripedBallCount => _gameObjectsSnapshot.Count(x => x is StripedBall);
```

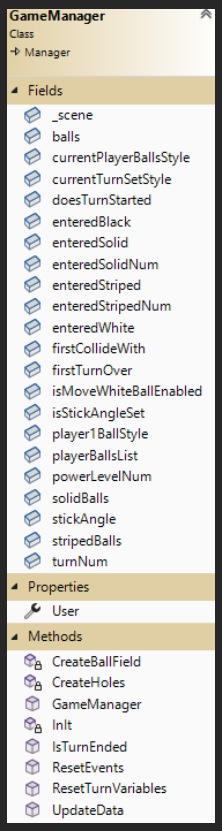
### Class Methods:

```
/// <summary>
/// Initializes a new instance of the GameScene class.
/// </summary>
0 references
public GameScene() ...

/// <summary>
/// Event handler for when the mouse wheel is changed within the scene.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">PointerRoutedEventArgs containing event data.</param>
1 reference
private void Scene_PointerWheelChanged(object sender, PointerRoutedEventArgs e)...
```

```
/// <summary>
/// Event handler for when the pointer is pressed within the scene.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">PointerRoutedEventArgs containing event data.</param>
1 reference
public void Scene_PointerPressed(object sender, PointerRoutedEventArgs e)...


/// <summary>
/// Adds a stick line to represent the angle of the ball movement.
/// </summary>
/// <param name="mousePosX">The X-coordinate of the mouse position.</param>
/// <param name="mousePosY">The Y-coordinate of the mouse position.</param>
2 references
public void AddStick(double mousePosX, double mousePosY)...


/// <summary>
/// Removes the stick angle line from the scene.
/// </summary>
2 references
public void RemoveLines()...
/// <summary>
/// Checks if the specified position is valid for moving the white ball.
/// </summary>
/// <param name="x">The X-coordinate of the position.</param>
/// <param name="y">The Y-coordinate of the position.</param>
/// <returns>True if the position is valid; otherwise, false.</returns>
1 reference
private bool ValidPos(double x, double y)...

/// <summary>
/// Gets the position of the pointer when left button pressed.
/// </summary>
/// <returns>The position of the pointer.</returns>
0 references
public GamePoint GetPointerPos()...
```

## 3.1.9 GameManager Class

The GameManager class is the central control unit of the billiard game, responsible for managing game state, turn progression, object placement, and user interactions. It maintains variables for player information, turn status, and ball styles. It handles event management, initializes the game scene with balls and holes, and ensures proper object placement. Additionally, it includes methods for updating game data and handling turn-specific actions.



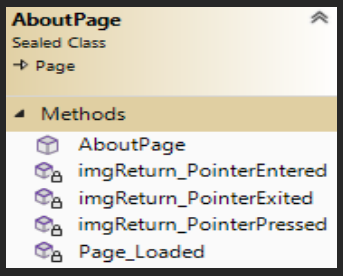

## Class Fields & Properties:

```
19 references
public static GameUser User { get; set; } = new GameUser(); // Gets or sets the current user of the game.

// Game variables:
public static string currentPlayerBallsStyle = "None"; // Player Ball style, gets chosen after a
                                                       // player enters balls to one of the holes after turn
                                                       // number 2 included. before that it set to None.

public static string player1BallStyle = "None"; // Player1 ball style
public static int turnNum = 1; // Turn number | 1,3,5... -> player1 | 2,4,6... -> player 2 |
public static bool firstTurnOver = false; // If first turn is over
public static bool currentTurnSetStyle = false; // What is the ball style at current turn
public static int enteredSolidNum = 0; // Number of solid balls entered the holes
public static int enteredStripedNum = 0; // Number of striped balls entered the holes

// Turn variables:
public static bool doesTurnStarted = false; // If turn started
public static bool isMoveWhiteBallEnabled = true; // If white ball can be moved by the current player
public static string firstCollideWith = "None"; // The name of the ball style the white ball first collided with
public static bool enteredSolid = false; // If player entered solid balls at his turn.
public static bool enteredStriped = false; // If player entered striped balls at his turn.
public static bool enteredWhite = false; // If player entered the white ball at his turn.
public static bool enteredBlack = false; // If player entered black ball at his turn.
public static double stickAngle = 180; // Sets the stick angle to 180 at start
public static int powerLevelNum = 1; // Sets the power level of the power of the blow on the white ball with the stick.
public static bool isStickAngleSet = false; //If player set an angle for the stick(cue) to hit the white ball.

// Game objects and related lists:
public static Ball[] balls; // All balls on table list
public static SolidBall[] solidBalls; // All solid balls on table list
public static StripedBall[] stripedBalls; // All striped balls on table list
public static List<string> playerBallsList; // All balls file names
```

## Class Methods:

```
/// <summary>
/// Initializes a new instance of the <see cref="GameManager"/> class.
/// </summary>
/// <param name="scene">The game scene associated with this manager.</param>
1 reference
public GameManager(GameScene scene) ...

/// <summary>
/// Checks if the current turn has ended.
/// (Check if any ball is still moving)
/// </summary>
/// <returns>True if the turn has ended; otherwise, false.</returns>
1 reference
public bool IsTurnEnded()...

/// <summary>
/// Resets all turn-related variables.
/// </summary>
1 reference
public void ResetTurnVariables()...

/// <summary>
/// Resets all events.
/// </summary>
2 references
public void ResetEvents()...

/// <summary>
/// Initializes the game scene by creating balls and holes.
/// </summary>
1 reference
private void InIt()...

/// <summary>
/// Updates game data.
/// </summary>
5 references
public static void UpdateData()...

/// <summary>
/// Adding all holes to the scene(table).
/// </summary>
1 reference
private void CreateHoles()...

/// <summary>
/// Creates the initial layout of the ball field by placing balls.
/// This method generates positions for the white ball, black ball, solid balls,
/// and striped balls on the game scene. It ensures that each ball is placed at a
/// valid position without overlapping with other balls or going out of bounds.
/// The white and black balls are positioned statically, while the solid and striped
/// balls are randomly distributed across predefined positions on the game field.
/// Once the balls are created and positioned, they are added to the scene for
/// rendering and interaction.
/// </summary>
1 reference
private void CreateBallField()...
```

## 3.1.10 AboutPage.xaml.cs Class

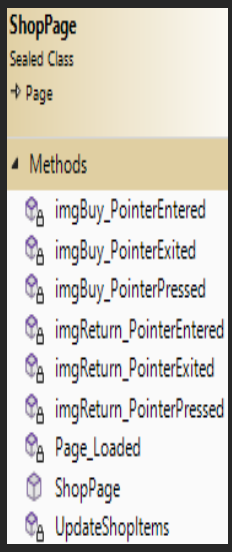



Represents the about/help page, with this page players can read instructions and information about the game.

## Class Methods:

```
/// <summary>
/// Initializing the page
/// </summary>
0 references
public AboutPage()...

/// <summary>
/// The method is called when the page is loaded, sets the Height and Width for the main grid.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void Page_Loaded(object sender, RoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes a sound and navigate to a different page.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerPressed(object sender, PointerRoutedEventArgs e)...
```

## 3.1.11 ShopPage.xaml.cs Class



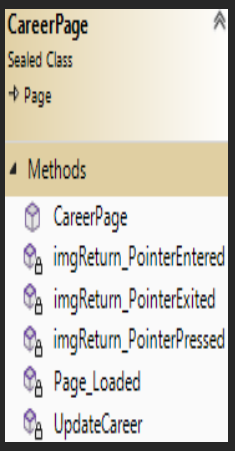

The ShopPage class manages the interface for browsing and purchasing items like table designs. Key features include initializing page dimensions and updating shop items based on user inventory. It handles events like pointer interactions with images, triggering actions such as changing images, playing sounds, and navigating pages. Purchasing items deducts costs from the user's balance if affordable or informs about insufficient funds.

### Class Methods:

```
/// <summary>
/// Initializing the page.
/// </summary>
0 references
public ShopPage()...

/// <summary>
/// The method is called when the page is loaded, sets the Height and Width
/// for the main grid, and updates the shop items list.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void Page_Loaded(object sender, RoutedEventArgs e)...

/// <summary>
/// Updates the list of available items in the shop based on the user's inventory.
/// For each table owned by the user, this method creates a visual representation
/// consisting of an image and a text block displaying the table's name and price.
/// The method iterates through each table in the user's inventory, creates a stack
/// panel to contain the visual elements, sets the orientation, width, and margin
/// properties of the stack panel, creates an image element for the table, sets its
/// dimensions and source to the corresponding table image, creates a text block to
/// display the table's name and price, sets its font, size, color, and margin properties,
/// adds the image and text block to the stack panel, and finally adds the stack panel to the
/// shop list for display.
/// </summary>
1 reference
private void UpdateShopItems()...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerExited(object sender, PointerRoutedEventArgs e)...
```

```
/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes a sound and navigates to a MenuPage.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgBuy_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgBuy_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes a sound and equips or buys (if player doesn't own yet the item and has enough money) the selected table.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private async void imgBuy_PointerPressed(object sender, PointerRoutedEventArgs e)...
```

## 3.1.12 CareerPage.xaml.cs Class

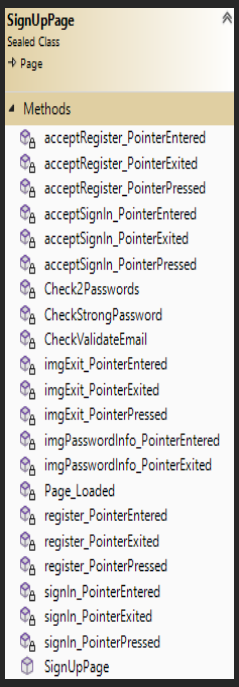



The CareerPage manages the interface for displaying player career information. It initializes the page layout and updates the career list with players' information and their money. The list is sorted based on players' money in descending order. Event handlers are implemented for pointer interactions with images, triggering actions like changing images, playing sounds, and navigating pages.

### Class Methods:

```
/// <summary>
/// The method is called when the page is loaded, sets the Height and Width for the
/// main grid, and updates the Career list.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void Page_Loaded(object sender, RoutedEventArgs e)...

/// <summary>
/// Updates the Career list with the players' information and their money.
/// Player information includes their position, username, and money.
/// The list is sorted based on the players' money in descending order.
/// </summary>
1 reference
private void UpdateCareer()...


/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes a sound and navigates to a MenuPage.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerPressed(object sender, PointerRoutedEventArgs e)...
```

## 3.1.13 SignUpPage.xaml.cs Class




The SignUpPage class manages the interface for user sign-up and sign-in functionality. It allows users to register or login by providing their username, password, and email. The page includes event handlers for pointer interactions with various UI elements, such as buttons and images, triggering actions like changing images, playing sounds, and validating user input. The class also contains methods for checking password strength, email validity, and matching passwords during registration. If registration or login is successful, the user is added to the database (if he register), and he receive a confirmation message before being navigated to the menu page. If registration fails due to missing data, weak passwords, invalid emails, or existing usernames/emails, appropriate error messages are displayed to the user.

### Class Methods:

```
/// <summary>
/// Initializing the page.
/// </summary>
0 references
public SignUpPage()...

/// <summary>
/// The method is called when the page is loaded, sets the Height and Width for the main grid.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void Page_Loaded(object sender, RoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void register_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void register_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound, closing SignInWindow, opening RegisterWindow, and
/// removing all data in SignInWindow boxes.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void register_PointerPressed(object sender, PointerRoutedEventArgs e)...
```

```
/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void acceptRegister_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void acceptRegister_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler triggered when the registration acceptance button is pressed.
/// Plays a clicking sound effect. Checks if the username, password, verify password,
/// and email fields are empty. If any field is empty, displays an error message
/// indicating missing data. If the passwords do not match, displays an error message
/// indicating non-identical passwords. If the password is not strong enough, displays
/// an error message indicating that a stronger password is required. If the email is
/// not valid, displays an error message indicating an invalid. If all fields are filled
/// correctly and the user does not already exist in the database, adds the user to the
/// database, clears any previous error messages, displays a success message indicating
/// successful sign-up, and navigates to the MenuPage. If the username or email is already
/// used, displays an error message indicating that the username or email is already in use.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event data.</param>
1 reference
private async void acceptRegister_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Comparing the 2 passwords value to check if they equal.
/// </summary>
/// <param name="password1">Password from passwordBox1</param>
/// <param name="password2">Password from passwordBox2</param>
/// <returns>Returns "true" if they equal; else returns "false";</returns>
1 reference
private bool Check2Passwords(string password1, string password2)...
```

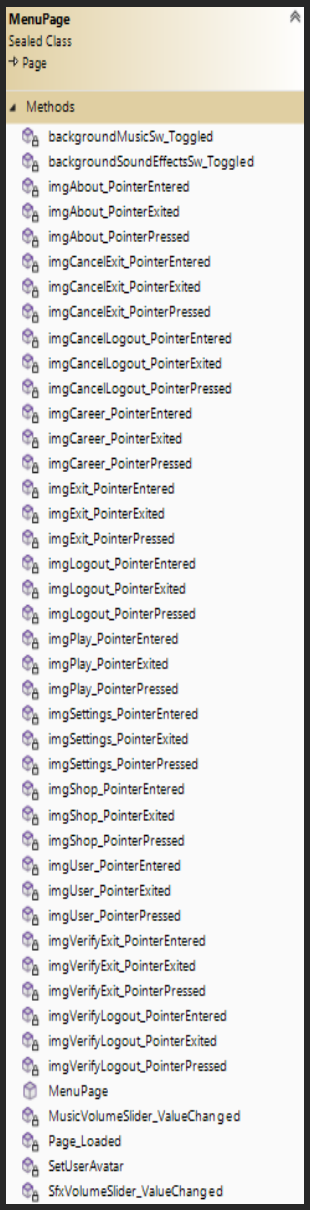
```
/// <summary>
/// Checking if the email player used is validate.
/// </summary>
/// <param name="email"></param>
/// <returns>if its validate, returns "true"; else returns "false"</returns>
1 reference
private bool CheckValidateEmail(string email)...

/// <summary>
/// Method checks if password is strong enough.
/// </summary>
/// <param name="password"></param>
/// <returns>Method returns "true" if password is strong enough if its meets the conditions. If not, returns "false".</returns>
1 reference
private bool CheckStrongPassword(string password)...


/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void acceptSignIn_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void acceptSignIn_PointerExited(object sender, PointerRoutedEventArgs e)...
```

```
/// <summary>
/// Event handler triggered when the sign-in acceptance button is pressed.
/// Plays a clicking sound effect. Checks if the username or email and password
/// fields are empty. If either field is empty, displays an error message indicating missing data.
/// If both fields are filled, attempts to validate if the user's login is valid.
/// If the user exists, retrieves the user's statistics and information, clears any previous error
/// messages, displays a success message welcoming the user, and navigates to the MenuPage.
/// If the user does not exist, displays an error message indicating that the user does not exist.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event data.</param>
1 reference
private async void acceptSignIn_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void signIn_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void signIn_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound, closing RegisterWindow, opening SignInWindow, and
/// removing all data in register boxes.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void signIn_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgPasswordInfo_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgPasswordInfo_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgExit_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgExit_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound, and exits the game.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgExit_PointerPressed(object sender, PointerRoutedEventArgs e)...
```

## 3.1.14 MenuPage.xaml.cs Class

The MenuPage class manages the main menu of the game, offering features like initialization, button interactions, sound effects, avatar settings management, dynamic welcome messages, and exit the game option. It provides users with an interactive and personalized experience while navigating through the game's menu options.

Class Methods:




```
/// <summary>
/// Initializing the page.
/// </summary>
0 references
public MenuPage()...

/// <summary>
/// Event handler for when page is loaded.
/// Makes sound and sets defualt settings.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void Page_Loaded(object sender, RoutedEventArgs e)...

/// <summary>
/// Function set the user welcome quote, colors and avatar for the user in the main menu.
/// </summary>
1 reference
private void SetUserAvatar()...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgPlay_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgPlay_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound and navigates to GamePage.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgPlay_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgAbout_PointerEntered(object sender, PointerRoutedEventArgs e)...
```

```
/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgAbout_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound and navigates to AboutPage.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgAbout_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCareer_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCareer_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound and navigates to CarrierPage.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCareer_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgShop_PointerEntered(object sender, PointerRoutedEventArgs e)...
```

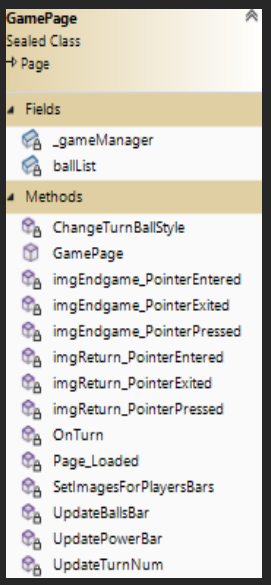
```
/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgShop_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound and navigates to ShopPage.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgShop_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgSettings_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgSettings_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound, opens SettingsGrid and collapses every other grid.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgSettings_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgExit_PointerEntered(object sender, PointerRoutedEventArgs e)...
```

```
/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgExit_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound, opens exitGrid and collapses every other grid.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgExit_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgVerifyExit_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgVerifyExit_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound and update user data, and exits the game.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgVerifyExit_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCancelExit_PointerEntered(object sender, PointerRoutedEventArgs e)...
```

```
/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCancelExit_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound and collapse exitGrid
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCancelExit_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the slider is toggled.
/// Makes sound and turn on\off the music.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void backgroundMusicSw_Toggled(object sender, RoutedEventArgs e)...

/// <summary>
/// Event handler for when the slider is toggled.
/// Makes sound and turn on\off the sound effects.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void backgroundSoundEffectsSw_Toggled(object sender, RoutedEventArgs e)...

/// <summary>
/// Event handler for when the slider is moved.
/// Changes the volume value for sound effects.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void SfxVolumeSlider_ValueChanged(object sender, Windows.UI.Xaml.Controls.Primitives.RangeBaseValueChangedEventArgs e)...

/// <summary>
/// Event handler for when the slider is moved.
/// Changes the volume value for music.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void MusicVolumeSlider_ValueChanged(object sender, Windows.UI.Xaml.Controls.Primitives.RangeBaseValueChangedEventArgs e)...
```

```
/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgUser_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgUser_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound and opens EditAvatarGrid and closes every other grid.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgUser_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgLogout_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgLogout_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes sound and opens LogOutGrid and closes every other grid.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgLogout_PointerPressed(object sender, PointerRoutedEventArgs e)...
```

```
/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgVerifyLogout_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgVerifyLogout_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Changes to SignUpPage Frame.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgVerifyLogout_PointerPressed(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// Changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCancelLogout_PointerEntered(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCancelLogout_PointerExited(object sender, PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// collapsing the logOutGrid.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgCancelLogout_PointerPressed(object sender, PointerRoutedEventArgs e)...
```




## 3.1.15 GamePage.xaml.cs Class

The GamePage class manages game logic, player turns, ball movements, win/lose conditions, and UI updates. The code includes event handlers for user interactions and navigation between game pages. Overall, the GamePage class ensures smooth gameplay, handling game state and providing an immersive user experience.

### Class Fields:

```
private GameManager _gameManager; // The game manager that manages the entire game.
List<string> ballList = new List<string>(); //Ball list for all the balls file names.
```

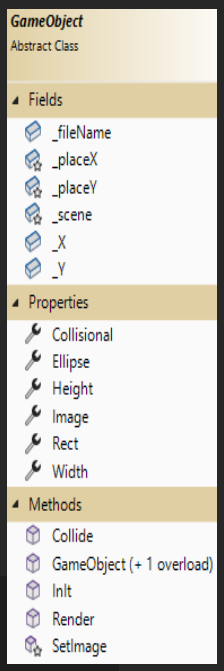
### Class Methods:

```
/// <summary>
/// Initializing the page.
/// </summary>
0 references
public GamePage()...

/// <summary>
/// The method is called when the page is loaded, sets the Height and Width for the main grid, sets the game state
/// to start and adds OnTurn event.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void Page_Loaded(object sender, RoutedEventArgs e)...

/// <summary>
/// This method handles various game scenarios and conditions, including player interactions,
/// ball movements, turn endings, and win/lose conditions. It manages the flow of gameplay by
/// evaluating the current state of the game, such as whether the turn has started,
/// whether the player has completed their turn, and what actions need to be taken based
/// on the player's input and the state of the game board.
/// Additionally, it updates relevant UI elements to reflect changes in the game state, such
/// as displaying the current turn number and updating the stick power bar.
/// This method plays a crucial role in maintaining the game's logic and ensuring a smooth and coherent
/// player experience throughout the gameplay session.
/// </summary>
1 reference
private void OnTurn()...

/// <summary>
/// Updates the visual settings for stick power bar.
/// </summary>
1 reference
private void UpdatePowerBar()...

/// <summary>
/// Updates the visual settings for players turn.
/// </summary>
1 reference
private void UpdateTurnNum()...

/// <summary>
/// Updates the visual for balls bar.
/// </summary>
2 references
private void UpdateBallsBar()...
```

```
/// <summary>
/// Sets the images for both players balls bar to their style,
/// in the turn which the style is set for them.
/// </summary>
1 reference
private void SetImagesForPlayersBars()...


/// <summary>
/// Swaps the current player ball style each turn to the opposite one.
/// </summary>
2 references
private void ChangeTurnBallStyle()...

/// <summary>
/// Event handler for when the pointer enters the image.
/// If game over grid is visible, changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerEntered(object sender, Windows.UI.Xaml.Input.PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer exits the image.
/// If game over grid is visible, changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerExited(object sender, Windows.UI.Xaml.Input.PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// If GameOverPop is visible, makes a sound, changing game state to pause, resets events and navigates to a MenuPage.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgReturn_PointerPressed(object sender, Windows.UI.Xaml.Input.PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer enters the image.
/// If game over grid is visible, changes image, plays sound, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgEndgame_PointerEntered(object sender, Windows.UI.Xaml.Input.PointerRoutedEventArgs e)...
/// <summary>
/// Event handler for when the pointer exits the image.
/// Changes image, and changes cursor type.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgEndgame_PointerExited(object sender, Windows.UI.Xaml.Input.PointerRoutedEventArgs e)...

/// <summary>
/// Event handler for when the pointer presses the image.
/// Makes a sound, adds user money, updates data in database, changing game state to pause, resets events and navigates to a MenuPage.
/// </summary>
/// <param name="sender">The object that raised the event.</param>
/// <param name="e">The event arguments.</param>
1 reference
private void imgEndgame_PointerPressed(object sender, Windows.UI.Xaml.Input.PointerRoutedEventArgs e)...
```

## 3.2 GameEngine

### 3.2.1 Project Diagram




## 3.2.2 GameObject Class

GameObject is a class for representing basic game objects in a game engine. It includes properties for position, appearance, and collision, along with methods for rendering and collision handling. The class can be inherited to create specific game objects with customized behavior and appearance.

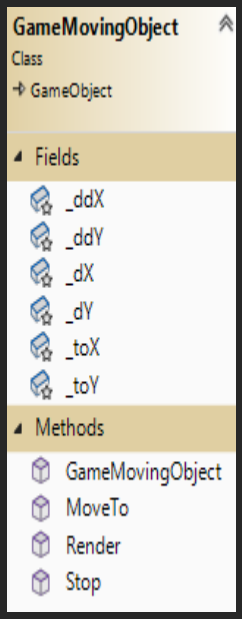
### Class Fields & Properties:

```
public double _X; // Current horizontal position
public double _Y; // Current vertical position

protected double _placeX; // Initial horizontal position
protected double _placeY; // Initial vertical position

14 references
public Image Image { get; set; } // Appearance of the object
public string _fileName; // File name of the image

15 references
public double Width => Image.ActualWidth; // Shortcut property for width
7 references
public double Height => Image.ActualHeight; // Shortcut property for height
2 references
public virtual Rect Rect => new Rect(_X, _Y, Width, Height); // Rectangle surrounding the object
0 references
public virtual Ellipse Ellipse => new Ellipse(); // Ellipse surrounding the object

6 references
public bool Collisional { get; set; } = true; // Indicates if the object is collisional

protected Scene _scene; // The game scene
```

## Class Methods:

```
/// <summary>
/// Constructs a basic game object.
/// </summary>
/// <param name="scene">The game scene.</param>
/// <param name="fileName">The file name of the image.</param>
/// <param name="width">The width of the object.</param>
/// <param name="height">The height of the object.</param>
/// <param name="placeX">The initial horizontal position.</param>
/// <param name="placeY">The initial vertical position.</param>
1 reference
public GameObject(Scene scene, string fileName, double width, double height, double placeX, double placeY)...

/// <summary>
/// Constructs a basic game object.
/// </summary>
/// <param name="scene">The game scene.</param>
/// <param name="fileName">The file name of the image.</param>
/// <param name="placeX">The initial horizontal position.</param>
/// <param name="placeY">The initial vertical position.</param>
1 reference
public GameObject(Scene scene, string fileName, double placeX, double placeY)...

/// <summary>
/// Resets the object to its initial position.
/// </summary>
1 reference
public virtual void InIt()...

/// <summary>
/// Defines the response to a collision with another game object.
/// </summary>
/// <param name="gameObject">The game object with which this object collides.</param>
3 references
public virtual void Collide(GameObject gameObject)...

/// <summary>
/// Renders the object on the screen.
/// </summary>
9 references
public virtual void Render()...

/// <summary>
/// Sets the Image source to the image that is in filename.
/// </summary>
/// <param name="fileName">File name of the image</param>
2 references
protected void SetImage(string fileName)...
```

### 3.2.3 GameMovingObject Class




GameMovingObject, inherits from the GameObject class. It represents moving objects in a game and adds properties and methods for movement control. The class includes fields for velocity, acceleration, and target position, along with methods for updating the object's position, stopping its movement, and moving it to a specified location with given speed and acceleration. The `Render` method ensures that the object stays within the boundaries of the game scene.

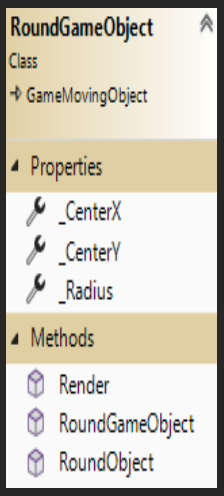
#### Class Fields:

```
protected double _dX;  // Horizontal velocity
protected double _dY;  // Vertical velocity
protected double _ddX; // Horizontal acceleration
protected double _ddY; // Vertical acceleration
protected double _toX; // Target position on the horizontal axis
protected double _toY; // Target position on the vertical axis
```

#### Class Methods:

```
/// <summary>
/// Initializes a new instance of the GameMovingObject class with the specified parameters.
/// </summary>
/// <param name="scene">The scene where the object will be rendered.</param>
/// <param name="fileName">The file name of the object's image.</param>
/// <param name="placeX">The initial horizontal position of the object.</param>
/// <param name="placeY">The initial vertical position of the object.</param>
/// <param name="maxSpeed">The maximum speed of the object. Default is 4.</param>
1 reference
public GameMovingObject(Scene scene, string fileName, double placeX, double placeY, double maxSpeed = 4) ...

/// <summary>
/// Updates the position of the object and renders it on the scene.
/// Checks all sides to make sure object is in border and not out of it.
/// </summary>
9 references
public override void Render()...

/// <summary>
/// Stops the movement of the object.
/// </summary>
1 reference
public virtual void Stop()...

/// <summary>
/// Moves the round game object to the specified coordinates with the given speed and acceleration.
/// </summary>
/// <param name="toX">The X-coordinate of the destination position.</param>
/// <param name="toY">The Y-coordinate of the destination position.</param>
/// <param name="speed">The speed of movement.</param>
/// <param name="acceleration">The acceleration of movement.</param>
0 references
public void MoveTo(double toX, double toY, double speed = 1, double acceleration = 0)...
```

## 3.2.4 RoundGameObject Class




RoundGameObject, represents a round game object in the game. It inherits from the `GameMovingObject` class and adds properties and methods specific to round objects.

The class includes properties for the radius of the round object and methods for creating and rendering the object as an Ellipse. The `Render` method overrides the base class method to set the position of the round game object based on its center coordinates and radius.

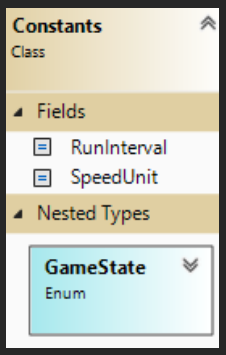
### Class Properties:

```
9 references
public double _Radius { get; set; } // Gets or sets the radius of the round game object.
3 references
public double _CenterX => _X + _Radius; // Gets the X-coordinate of the center of the round game object.
3 references
public double _CenterY => _Y + _Radius; // Gets the Y-coordinate of the center of the round game object.
```

### Class Methods:

```
/// <summary>
/// Initializes a new instance of the RoundGameObject class with the specified parameters.
/// </summary>
/// <param name="scene">The scene where the game object exists.</param>
/// <param name="fileName">The file name of the image representing the game object.</param>
/// <param name="radius">The radius of the round game object.</param>
/// <param name="_X">The X-coordinate of the round game object.</param>
/// <param name="_Y">The Y-coordinate of the round game object.</param>
1 reference
public RoundGameObject(Scene scene, string fileName, double radius, double _X, double _Y)...

/// <summary>
/// Creates and returns an Ellipse object representing the round game object.
/// </summary>
/// <returns>An Ellipse object representing the round game object.</returns>
0 references
public Ellipse RoundObject()...

/// <summary>
/// Renders the round game object on the screen by setting its position.
/// </summary>
9 references
public override void Render()...
```

## 3.2.5 Constants Class

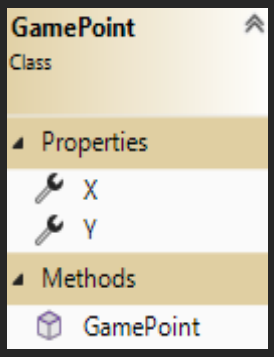



The `Constants` class encapsulates crucial numerical values and game state definitions for a game engine. It establishes the speed of the project, the global speed unit for moving objects, and defines the possible states the game can be in. This centralization of constants promotes code readability, maintainability, and consistency throughout the game development process.

### Class Fields & Nested Types:

```
public const double RunInterval = 5; //the speed the project will run at // pc = 5, laptop = 0.001
public const int SpeedUnit = 10; //global speed unit for gameMovingObject

7 references
public enum GameState //game state setting
{
    Loaded,
    Started,
    Paused,
    GameOver
}
```

## 3.2.6 GamePoint Class




GamePoint, defines a two-dimensional point with X and Y coordinates. It features properties to access and modify these coordinates, along with a constructor to initialize a new instance of a point with specified coordinates.

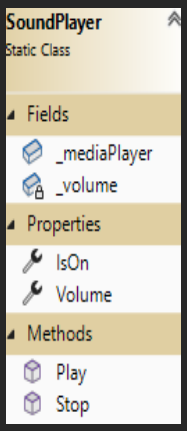
### Class Fields & Properties:

```
9 references
public double X { get; set; } // Gets or sets the X-coordinate of the point.

9 references
public double Y { get; set; } // Gets or sets the Y-coordinate of the point.
```

### Class Methods:

```
/// <summary>
/// Initializes a new instance of the GamePoint class with the specified coordinates.
/// </summary>
/// <param name="X">The X-coordinate of the point.</param>
/// <param name="Y">The Y-coordinate of the point.</param>
16 references
public GamePoint(double X, double Y)
{
    this.X = X;
    this.Y = Y;
}
```

## 3.2.7 SoundPlayer Class




The SoundPlayer class is a static class used for playing sound effects in a game. It utilizes the `Windows.Media.Playback` namespace to manage media playback. Key features include a static `MediaPlayer` instance, volume control, a method to play sound effects, and a method to stop playback. Overall, it provides a convenient way to incorporate sound effects into a game.

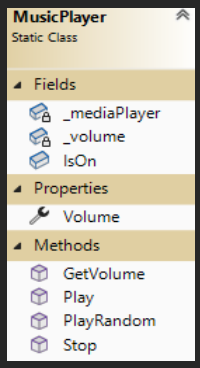
### Class Fields & Properties:

```
public static MediaPlayer _mediaPlayer = new MediaPlayer(); // The media player
4 references
public static bool IsOn { get; set; } = true; // Gets or sets whether the sound player is on.

private static double _volume = 0.5; // The volume level
```

### Class Methods:

```
/// <summary>
/// Gets or sets the volume level of the sound player.
/// </summary>
2 references
public static double Volume...


/// <summary>
/// Plays the specified sound effect file.
/// </summary>
/// <param name="fileName">The name of the sound effect file to play.</param>
49 references
public static void Play(string fileName)...

/// <summary>
/// Stops playing the current sound effect.
/// </summary>
0 references
public static void Stop()...
```

## 3.2.8 MusicPlayer Class




The MusicPlayer class is a static class in the game engine responsible for playing music. Key features include playing a specified music file, playing a random music file from an array, stopping music playback, and adjusting the volume level. This class provides functionality to incorporate background music into a game.

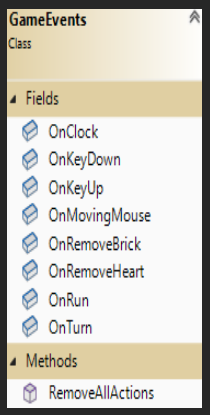
### Class Fields & Properties:

```
private static MediaPlayer _mediaPlayer = new MediaPlayer(); // The media player
public static bool IsOn = false; // Indicates if the music player is on
private static double _volume = 0.5; // The volume level
```

### Class Methods:

```
/// <summary>
/// Plays the specified music file.
/// </summary>
/// <param name="fileName">The name of the music file to play.</param>
1 reference
public static void Play(string fileName)...

/// <summary>
/// Plays a random music file from the specified list.
/// </summary>
/// <param name="fileNames">The array of music file names.</param>
2 references
public static void PlayRandom(string[] fileNames)...

/// <summary>
/// Stops playing the music.
/// </summary>
1 reference
public static void Stop()...

/// <summary>
/// Gets or sets the volume level of the music player.
/// </summary>
3 references
public static double Volume...

/// <summary>
/// Gets the current volume level of the music player.
/// </summary>
/// <returns>The current volume level.</returns>
0 references
public static double GetVolume()...
```

### 3.2.9 GameEvents Class




The `GameEvents` class defines various actions or events that can occur during the game, such as running, clock ticking, key presses, mouse movement, and more. Each event is represented by an `Action` delegate with specific parameters or return types. The class also provides a method `RemoveAllActions()` to reset all events to null, used for cleanup or resetting game state.

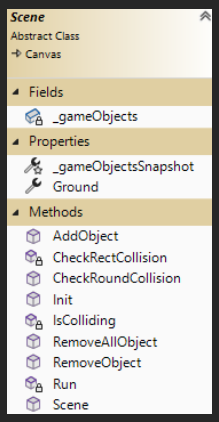
#### Class Fields:

```
public Action OnRun;                //The event where everyone who registers will move
public Action OnClock;              //The event that will trigger the clock
public Action<VirtualKey> OnKeyUp;  //The event that those who sign up for will be able to respond to the release of a key
public Action<VirtualKey> OnKeyDown; //The event that whoever registers for will be able to respond to the press of a key
public Action<int> OnRemoveHeart;   //The event that whenever the ball reaching the floor will cause the game page to lose heart
public Action OnRemoveBrick;        //The event that whenever the ball hits a brick it will cause the brick to change level\color or disapper.
public Action OnMovingMouse;        //whenever a player moves the mouse, the event accuers.
public Action OnTurn;               //whenever a turn is ended the event accuers.
```

#### Class Methods:

```
/// <summary>
/// resets all the events to null.
/// </summary>
1 reference
public void RemoveAllActions()...
```

## 3.2.10 Scene Class




Scene class encapsulates essential functionalities for managing game elements, handling collisions, and rendering objects within a game environment, making it a crucial component for building games in the game engine.

### Class Fields & Properties:

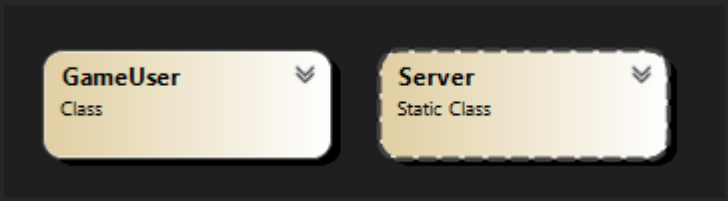
```
private List<GameObject> _gameObjects = new List<GameObject>(); // Collection of all game objects
12 references
protected List<GameObject> _gameObjectsSnapshot => _gameObjects.ToList(); // Copy of game objects list
1 reference
public double Ground { get; set; } // Ground level
```

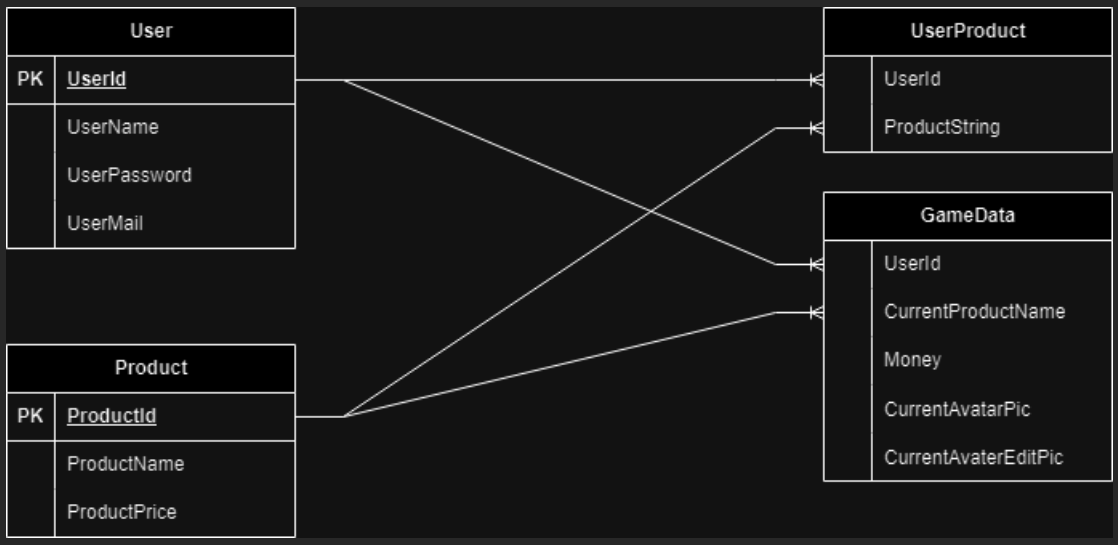
### Class Methods:

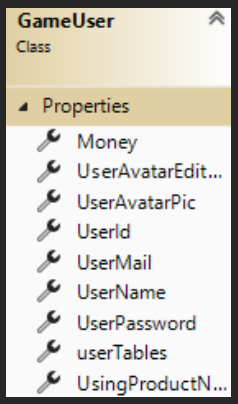
```
/// <summary>
/// Constructor for the Scene class.
/// </summary>
1 reference
public Scene()...

/// <summary>
/// Checks for collisions between rectangular game objects.
/// </summary>
1 reference
private void CheckRectCollision()...

/// <summary>
/// Checks for collisions between round game objects.
/// </summary>
1 reference
public void CheckRoundCollision()...


/// <summary>
/// Checks if two round game objects are colliding.
/// </summary>
1 reference
private bool IsColliding(RoundGameObject obj1, RoundGameObject obj2)...

/// <summary>
/// Action executed continuously during the game loop.
/// </summary>
1 reference
private void Run()...

/// <summary>
/// Initializes all game objects to their initial positions.
/// </summary>
1 reference
public void Init()...

/// <summary>
/// Removes the specified game object from the scene.
/// </summary>
/// <param name="gameObject">The game object to remove.</param>
2 references
public void RemoveObject(GameObject gameObject)...

/// <summary>
/// Removes all game objects from the scene.
/// </summary>
1 reference
public void RemoveAllObject()...

/// <summary>
/// Adds a game object to the scene.
/// </summary>
/// <param name="obj">The game object to add.</param>
7 references
public void AddObject(GameObject obj)...
```

## 3.3 DataBase Project

### 3.3.1 Project Diagram





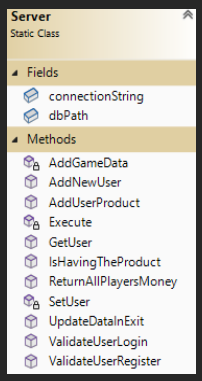

## 3.3.2 GameUser Class

This class encapsulates user-related data and provides default values for its attributes, ensuring smooth handling within the game database system.

### Properties:

```
5 references
public int UserId { get; set; } = 0; //user number
3 references
public string UserName { get; set; } = "Error Loading"; //user name
1 reference
public string UserMail { get; set; } = "Error Loading"; //user mail
0 references
public string UserPassword { get; set; } = "Error Loading"; //user password
6 references
public int Money { get; set; } = 0; //user amount of money
6 references
public string UsingProductName { get; set; } = "GreenTable"; //user current using product name
3 references
public string UserAvatarPic { get; set; } = "DefaultAvatar"; //user avatar picture
3 references
public string UserAvatarEditPic { get; set; } = "DefaultAvatarEdit"; //user avatar edit picture
2 references
public List<string> userTables { get; set; } = new List<string>() //all possible products user can buy/equip
{
    "GreenTable",
    "RedTable",
    "BlueTable",
    "OrangeTable",
    "PinkTable"
};
```

## 3.3.3 Server Class




The `Server` class handles user registration, login, and game data management in the database. It provides methods for validation, user addition, retrieval, and updating of user information, as well as product inventory management. These operations involve interacting with the SQLite database through SQL queries to ensure data integrity and user authentication.

### Class Fields:

```
public static string dbPath = ApplicationData.Current.LocalFolder.Path; //computer path
public static string connectionString = "Filename=" + dbPath + "\\DBGame.db"; //
```

### Class Methods:

```
/// <summary>
/// Validates the registration of a user by checking if the provided username or email already exists in the database
/// </summary>
/// <param name="userName">The username to be validated.</param>
/// <param name="userMail">The email address to be validated.</param>
/// <returns>
/// Returns the User ID if either the username or email already exists in the database; otherwise, returns null.
/// </returns>
3 references
public static int? ValidateUserRegister(string userName, string userMail)...


/// <summary>
/// Validates a user's login credentials by checking if the provided username/email
/// and password match an existing user record in the database.
/// </summary>
/// <param name="userNameORMail">The username or email address provided by the user for login.</param>
/// <param name="userPassword">The password provided by the user for login.</param>
/// <returns>
/// Returns the User ID if the provided username/email and password match an existing
/// user record in the database; otherwise, returns null.
/// </returns>
1 reference
public static int? ValidateUserLogin(string userNameORMail, string userPassword)...


/// <summary>
/// Adds a new user to the system if the provided username and
/// email are unique, and returns the user's information.
/// </summary>
/// <param name="name">The username of the new user.</param>
/// <param name="password">The password of the new user.</param>
/// <param name="mail">The email address of the new user.</param>
/// <returns>
/// Returns the GameUser object containing the information of the newly added user if the registration is successful;
/// otherwise, returns null if the provided username or email already exists in the database.
/// </returns>
1 reference
public static GameUser AddNewUser(string name, string password, string mail)...
```

```
/// <summary>
/// Sets the user information from the database based on the provided user ID.
/// </summary>
/// <param name="userId">The unique identifier of the user to retrieve.</param>
/// <returns>
/// Returns a GameUser object containing the information of the user with the specified ID if found; otherwise, returns null.
/// </returns>
2 references
public static GameUser GetUser(int userId)...

/// <summary>
/// Fills additional user data such as money, current product name, and avatar pictures based on the provided GameUser object.
/// </summary>
/// <param name="user">The GameUser object for which additional data needs to be retrieved and set.</param>
/// <remarks>
/// This method queries the database to retrieve additional user-specific data stored in the [GameData] table,
/// including the user's money balance, the name of the product they are currently using, and their avatar pictures.
/// If data is found, it sets the corresponding properties of the provided GameUser object.
/// </remarks>
1 reference
private static void SetUser(GameUser user)...

/// <summary>
/// Adds a new product to the user's inventory in the database.
/// </summary>
/// <param name="userId">The unique identifier of the user.</param>
/// <param name="productName">The name of the product to add to the user's inventory. Default is "GreenTable".</param>
/// <remarks>
/// This method inserts a new record into the [UserProduct] table, associating the specified user with the provided product name.
/// If no product name is provided, the default product name "GreenTable" is used.
/// </remarks>
2 references
public static void AddUserProduct(int userId, string productName = "GreenTable")...
```

```
/// <summary>
/// Adds default game data for a new user to the database.
/// </summary>
/// <param name="userId">The unique identifier of the user.</param>
/// <param name="currentProductName">The name of the product currently in use by the user. Default is "GreenTable".</param>
/// <param name="money">The amount of money associated with the user. Default is 0.</param>
/// <param name="currentAvatarPic">The file name of the user's avatar picture. Default is "DefaultAvatar".</param>
/// <param name="currentAvatarEditPic">The file name of the user's edited avatar picture. Default is "DefaultAvatarEdit".</param>
/// <remarks>
/// This method inserts a new record into the [GameData] table, initializing the game-related data for a new user.
/// By default, the user starts with the product "GreenTable", 0 money, and default avatar pictures.
/// The method allows specifying custom values for the current product name, money, and avatar pictures if desired.
/// </remarks>
1 reference
private static void AddGameData(int userId, string currentProductName = "GreenTable", int money = 0, string currentAvatarPic = "DefaultAvatar", string currentAvatarEditPic = "DefaultAvatarEdit")...


/// <summary>
/// Updates the game data for a user in the database upon exiting the game session.
/// </summary>
/// <param name="user">The GameUser object containing the updated game data to be saved.</param>
/// <remarks>
/// This method updates the user's game data stored in the [GameData] table in the database with the latest information upon exiting the game session.
/// It updates the user's money balance, the name of the currently used product, and the avatar pictures to reflect the changes made during the game session.
/// </remarks>
1 reference
public static void UpdateDataInExit(GameUser user)...


/// <summary>
/// Checks if the user possesses a specific product.
/// </summary>
/// <param name="user">The GameUser object representing the user whose inventory is to be checked.</param>
/// <param name="productName">The name of the product to check for in the user's inventory.</param>
/// <returns>
/// Returns true if the user possesses the specified product; otherwise, returns false.
/// </returns>
1 reference
public static bool IsHavingTheProduct(GameUser user, string productName)...


/// <summary>
/// Retrieves the money balances of all players from the database and returns
/// them in a dictionary with player names as keys and their corresponding money
/// balances as values.
/// </summary>
/// <returns>
/// Returns a dictionary containing player names as keys and their corresponding money balances as values.
/// </returns>
1 reference
public static Dictionary<string, int> ReturnAllPlayersMoney()...
```

```
/// <summary>
/// Executes a SQL query that doesn't return any data, such as INSERT, UPDATE, or DELETE statements.
/// </summary>
/// <param name="query">The SQL query to be executed.</param>
/// <remarks>
/// This method opens a connection to the SQLite database using the provided connection string, executes the given SQL query,
/// and performs the corresponding action on the database without returning any data. It is typically used for executing
/// INSERT, UPDATE, or DELETE statements to modify database records.
/// </remarks>
4 references
private static void Execute(string query)...
```

# 4. Project Quality

## 4.1 algorithms problems

While doing this project, I did not predicted I'm going to need to use so much physics and math in one "simple game", mostly because I made this project on UWP, when this work environment is not suited for this kind of game.
I did not expect that level of difficult in making ball collisions and movement, a lot of time on this project was spent to make the physics of collisions and the ball movement as real as possible, I tried getting the help of every person I know that could help, I even tried Chat GPT3 and got no result for my problem, but eventually, I did it, after almost 2 months of work on the physics side of the game, I sat none-stop from the morning to the next, almost every day, and managed to make it the best real like physics I've seen so far in my time of experience and in my researches, that someone made on any UWP project, and It felt like I made the impossible.

One of the biggest issues I had in making this physics, was the angles, I did not know Math.Sin() and Math.Cos() and Math.Tan() returns radians value and not degrees, and apparently no one else in my environment knew too, so after a lot of debugs, and some time researching, I finally found out that the reason my angles was miscalculated in my game, was because it wasn't degrees, and sadly the documentation of the Math class did not tell the results were calculated as radians.

Another little problem I ran into was adding sound effects to balls collision, because media player isn't making sound in parallel, and too many request to make sound effects, makes the game lag a lot, and results in delay of the sound effects or no sound at all, that's why I had to remove the sound effects, and I couldn't find any solution on the internet for that sadly.

## 4.2 improvement suggestions

If I had more time working on this project, I would try spending it to improve the physics more to make it more like the physics in real life.
I would also like to add ball rolling images so it would like it's a 3D game.

# 5. Reflection

When I started making this project I never expected such a long journey, with a lot of learning in the progress, taking advice from friends and family, sitting hours on days on weeks, solving bugs, issues, being independent and a self-learner.

I never thought for a second, that I might fail, give up, or won't succeed, I had a vision, a target, and I just sat for hours and hours until I make the game look like it.

The project didn't steal my life, I both enjoyed programming and doing other activities in the day, I even had time to win an Israel tournament in a video game in the time being.

From day one that I learnt about this world, I was sucked into it, I love everything about programming, networks, cyber-security, clouds, everything, I'm so interested in that, that I love working with it and studying it.

This 4 years of experience and 3 years of Cyber class was amazing to me, and I enjoyed every moment from it, but its not over for me, I'm just starting my programming career, from the army to high tech companies, to who knows what.

I knew this about me already because its not my first time, but this project just showed me my independent learning ability again and strengthened it.

I would like to thank to Margalit Golkarov and Oleg Katsnelson for teaching me how to be a successful programmer and giving me more interest in the computer world from what I already had.

# 6. Sources

The game is designed with original art most of which were created by me with the help of two the programs: Paint, Photopia.

The game sound effects and music were also edited by me in DaVinci resolve program and taken from YouTube.

Rest of the sources I used:

- A free font called “BurBank” that was used to create buttons.
- YouTube free music library for 7 different music options.
- Opengameart website for the table style.