High school of Informatics, Postal and Financial Services Brno

STŘEDNÍ ŠKOLA **INFORMATIKY, POŠTOVNICTVÍ** A **FINANČNICTVÍ** BRNO

Software development documentation

Brno 2020

Project C4PE

High school of Informatics, Postal and Financial Services Brno

Picture guessing game

Software development documentation

Supervisor:

Mgr. František Skalka

Written by:

- Jakub Cetl
- Jaroslav Opluštil
- Lukáš Pařil

Field of study: Information Technology

Field number: 18-20-M/01

Class: IT4

Brno 2020

Project C4PE

High school of Informatics, Postal and Financial Services Brno

Abstract

Create a form desktop application for learning, entertaining, or gaining or practicing knowledge. Use the XAML and C# programming languages to program this software. The application must allow the storage of game data on the server (this functionality will be provided by the other party of the project). The application should include an introductory information screen, logging in to the program, setting up the program interface, the purpose focus on the field and the evaluation part. The appearance of the application should be user-friendly and intuitive in English. The application should be accompanied by documentation on the development and description of the program, also in English.

Brno 2020 Project C4PE

Table of Content

1 Introduction

This documentation deals with the development of an application that was part of the Concepts for Professional Education in Border Regions (C4PE) project. This international project is based on the INTERREG European Territorial Cooperation Program and aims to create cross-border vocational education in the framework of cooperation between schools.

This project was started in September 2019 and was finished in July 2020.

Picture 1: Logos

Our full team consisted of three Czech students from Střední škola informatiky, poštovnictví a finančnictví Brno and four Austrian students from HTL Krems.

We were tasked to create a functional application that would use the server as a database and main computing unit.

Our half of the team was tasked with developing an application front end which meant creating a user interface, algorithms written in C# language, and a common interface with our application and server's database.

At our first meeting with Austrian students, we presented our two ideas for this project. Our first idea was a library system that would clearly show in which phase the books are and all of this information would be stored on the server so that anyone can see if the book they need is in the library or is borrowed by someone. Our second idea was a multiplayer game that would be based on the Unity engine but this idea was quickly discarded as it did not fully correspond to the assignment of the project.

We settled on a Picture Guessing game presented by Austrian students. This application is part education and part entertaining game. The goal of the game is to correctly guess the picture that is gradually unveiling. The player is able to choose the topic of the picture, difficulty that will set the board size, rounds per game, and length of rounds, or the player can also go with custom difficulty and choose all of the settings himself.

The whole project was under management of Mgr. Jaroslav Mašek, Mgr. František Skalka, Mgr. Josef Smékal and Ing. Alena Hájková.

2 Hardware specifications

2.1 Minimum specifications

- OS: Windows 7
- Processor: Intel Atom Z3740
- Memory: 2 GB RAM
- Graphics: Integrated graphics card
- Storage: 25 MB of available space

2.2 Recommended specifications

- OS: Windows 10
- Processor: Intel i5 2400
- Memory: 4 GB RAM
- Graphics: Dedicated graphics card
- Storage: 25 MB of available space

3 Development

3.1 Application goals

Our goal for the application was to create a stable and bug-free game that was ready for easy expansion of any other content.

The most important goal was to have a fully functional and reliable stream of data between our client and the server.

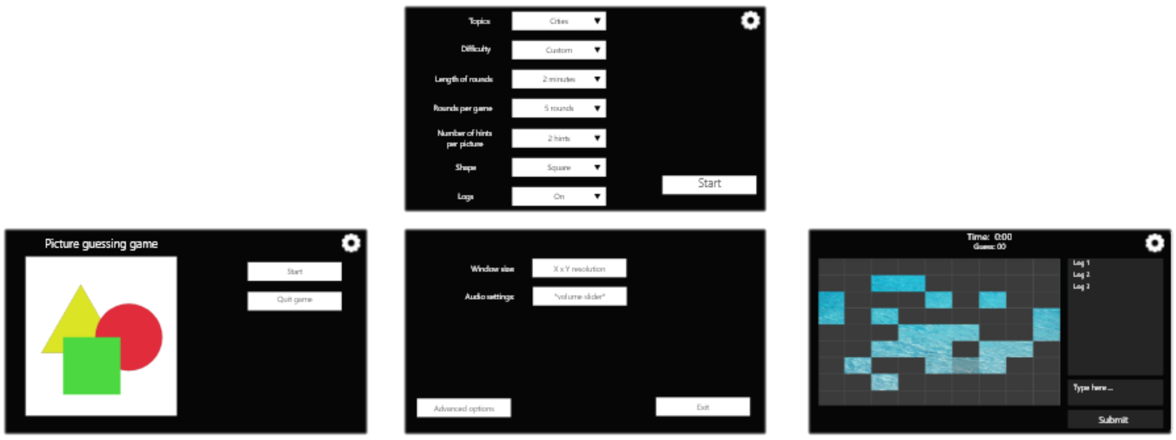
Our other target was a functional main game window with all the pictures uncovering and the game's continuity in case of more rounds.

3.2 Tools used

For the development of the game, we used a handful of tools. All of the tools we used, except for Trello, were completely new for us. Because of that we took our time and learn the basics of all of the tools so we could start using them very early in the project which helped us in the long run.

3.2.1 Adobe XD

For our mockup of the user interface, we used Adobe XD. This tool is very straightforward to use as it has a very simple UI and is made specifically for making these kinds of mockups.


Topics: Cities
Difficulty: Custom
Length of rounds: 2 minutes
Rounds per game: 5 rounds
Number of hints per picture: 2 hints
Shape: Square
Logs: On

Window size: X x Y resolution
Audio settings: "volume slider"

Time: 0:00
Guess: 00
Log 1
Log 2
Log 3

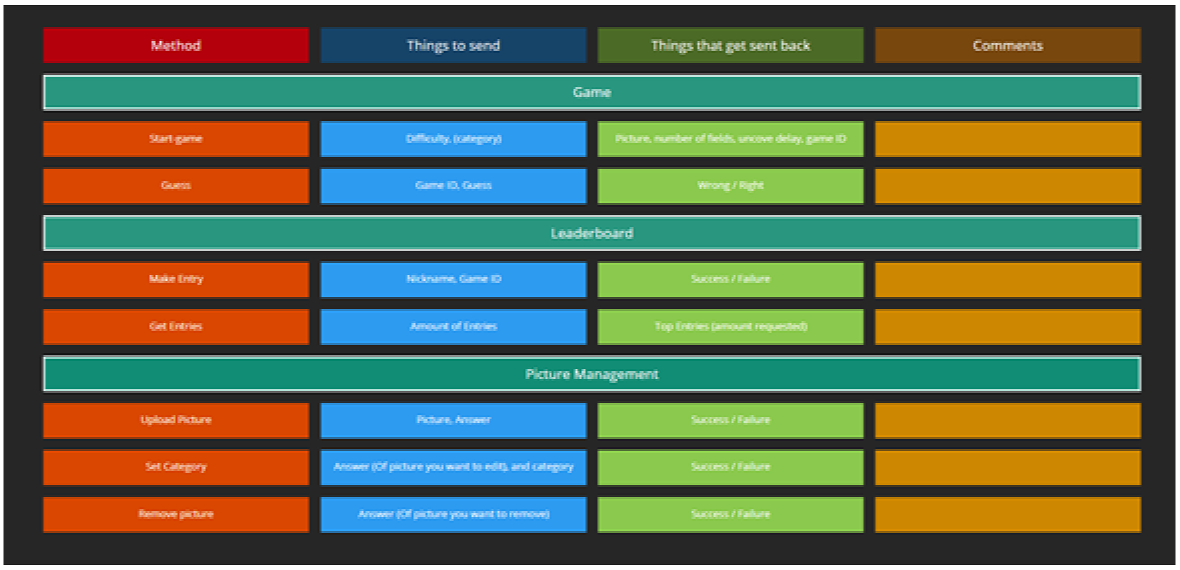
Picture 2: Adobe XD – Mockup

We went with the dark mode for the application as it is easier on the eyes and we can choose more vivid colors that will highlight important elements of the user interface.

This mockup helped us with a clear and simple representation of the final shape of the game.

3.2.2 Miro

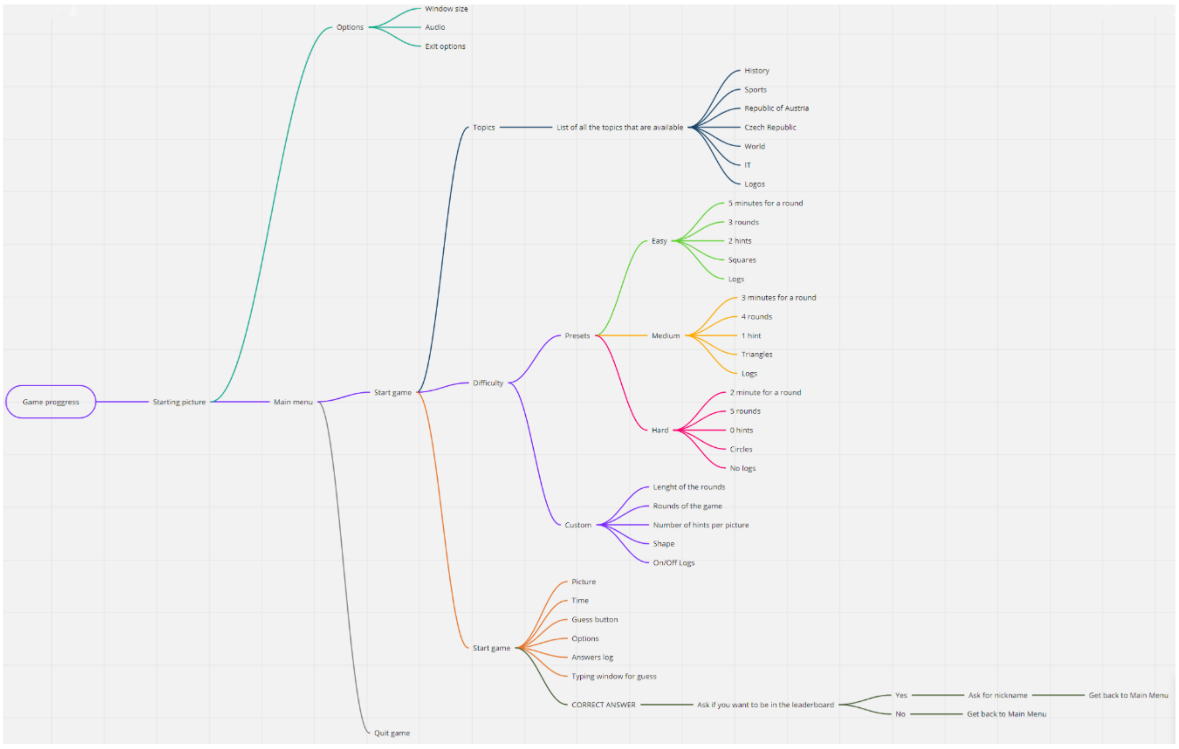
Miro was our second most used tool while developing this application as it can be used for a wide range of tasks. We used Miro for all of our visual concepts which included things like server and application communication, flowcharts, and mind maps.



Method	Things to send	Things that get sent back	Comments
Game			
Start game	Difficulty, (category)	Picture, number of fields, uncove delay, game ID	
Guess	Game ID, Guess	Wrong / Right	
Leaderboard			
Make Entry	Nickname, Game ID	Success / Failure	
Get Entries	Amount of Entries	Top Entries (amount requested)	
Picture Management			
Upload Picture	Picture, Answer	Success / Failure	
Set Category	Answer (Of picture you want to edit), and category	Success / Failure	
Remove picture	Answer (Of picture you want to remove)	Success / Failure	

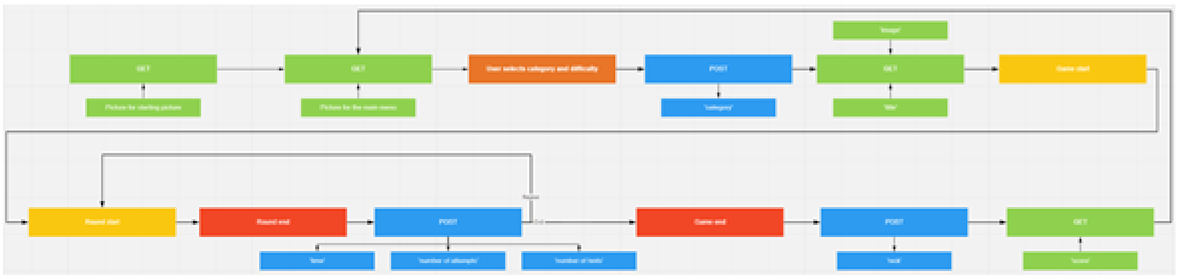
Picture 3: Miro – Communication between the server and the client

The first use of Miro was when we, together with the Austrian team, sat down and designed the first draft of the communication between the server and the client. It was a very elementary draft but it served its purpose as it helped us determine what our goal is and was our foundation for the more complex communication later in the project.

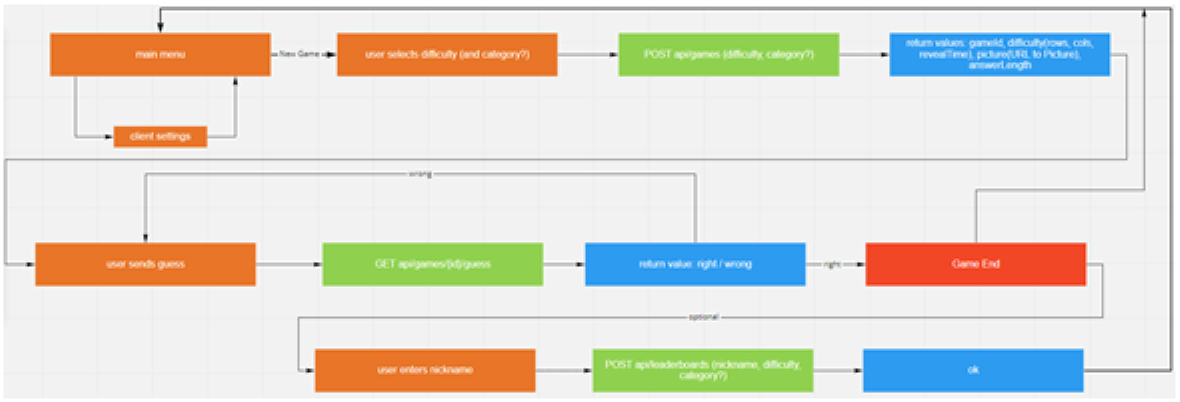


Picture 4: Miro – Mind map

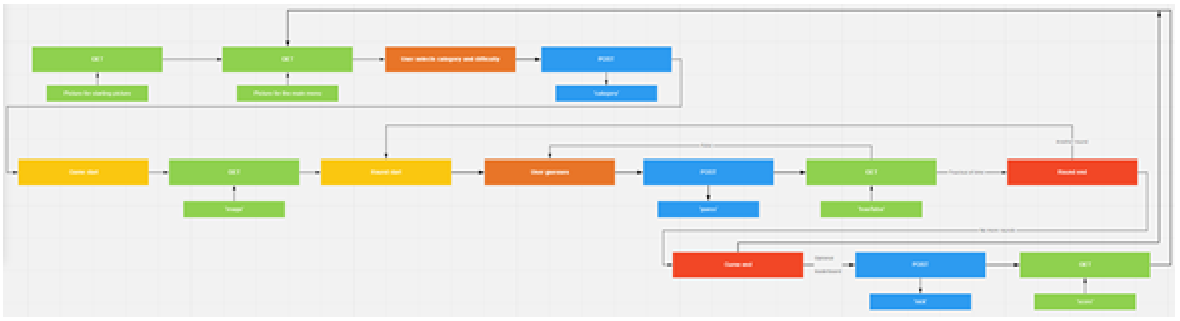
Miro also served as a great way to express each team's ideas mainly about the final form of communication between the server and the client.



Picture 5: Miro – Czech draft of communication between the server and the client


main menu
New Game
user selects difficulty (and category?)
POST api/games (difficulty, category?)
return values: gameId, difficulty(rows, cols, revealTime), picture(URL to Picture), answerLength
client settings
wrong
user sends guess
GET api/games/{id}/guess
return value: right / wrong
right
Game End
optional
user enters nickname
POST api/leaderboards (nickname, difficulty, category?)
ok

Picture 6: Miro – Austrian draft of communication between the server and the client



Picture 7: Miro – The final draft of communication between the server and the client

We settled on comparing the player's answer and the correct answer on the server. We chose this because of the security and also that we will need to use the server on something different than just a database.

3.2.3 Trello

For our development, we choose the agile method of working. Trello is a perfect fit for this kind of work as it lets us see what tasks are being worked on at the moment. It is also very easy to scale up the board as the development continues.

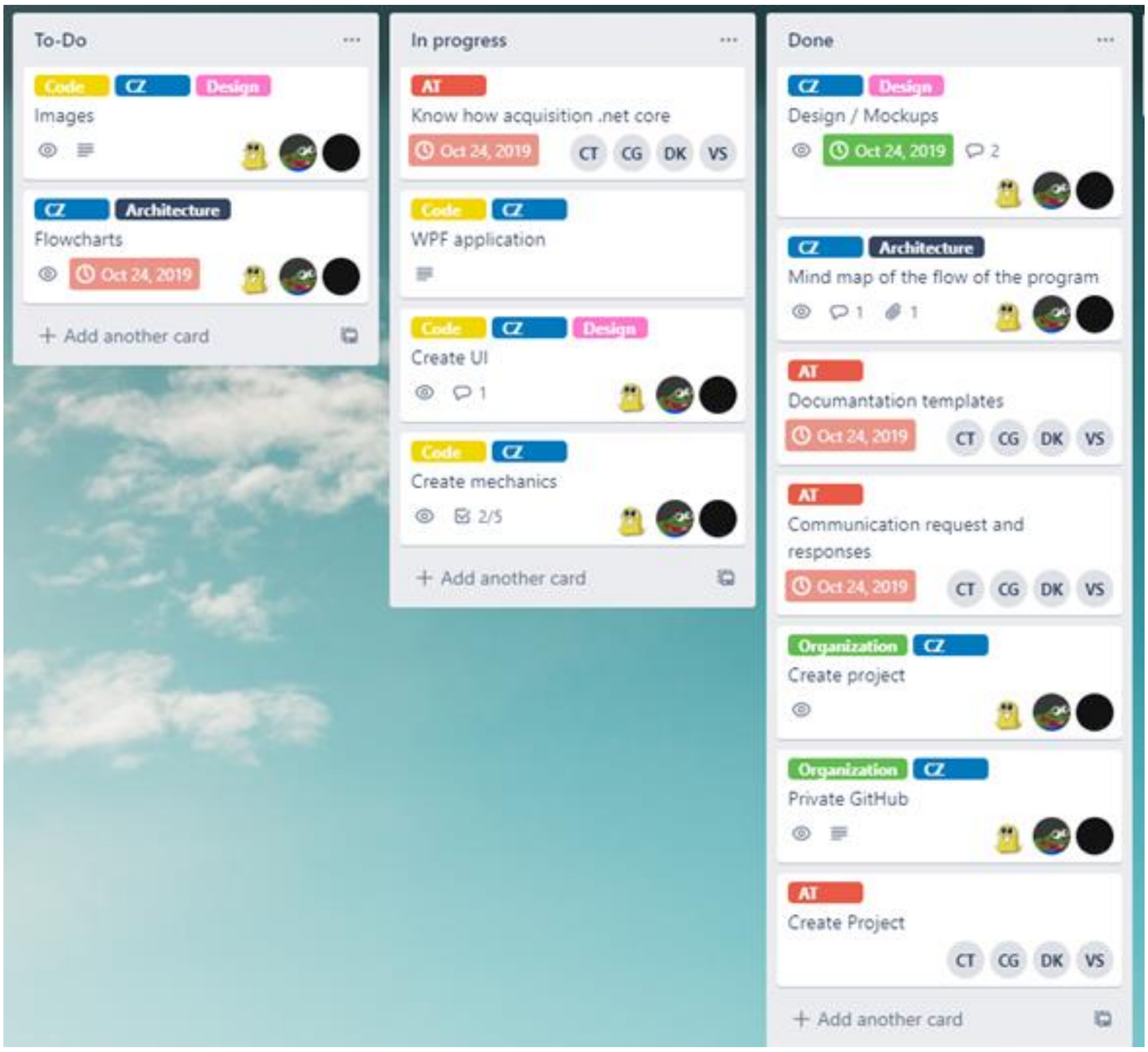
We had four main columns which all served to store tasks in different parts of completion.

The most left column was our backlog where we at the start of the project put all of our tasks.

Tasks that were planned to be started in the near future were moved into the To-Do column.

When we started working on a task we moved it into the In progress column where it stayed until finished. We tried to have as little as possible tasks in progress as it is easier to see what is being worked on and also that we do not glut ourselves with too many unfinished tasks.

After the task is done and review by other team members it is moved into the Done column so that everybody knows that this task is completed and other tasks can be moved into the To-Do or In progress columns.



To-Do ...

Code CZ Design
Images

CZ Architecture
Flowcharts
Oct 24, 2019

+ Add another card

In progress ...

AT
Know how acquisition .net core
Oct 24, 2019 CT CG DK VS

Code CZ
WPF application

Code CZ Design
Create UI
1

Code CZ
Create mechanics
2/5

+ Add another card

Done ...

CZ Design
Design / Mockups
Oct 24, 2019 2

CZ Architecture
Mind map of the flow of the program
1 1

AT
Documantation templates
Oct 24, 2019 CT CG DK VS

AT
Communication request and responses
Oct 24, 2019 CT CG DK VS

Organization CZ
Create project

Organization CZ
Private GitHub

AT
Create Project
CT CG DK VS

+ Add another card

Picture 8: Trello – Project board

3.2.4 Discord

As a communication channel, we used Discord which helped us to connect via text or video with Austrian students or within our team. We used Discord rarely as most of the communication was done within Trello.

3.2.5 Git

For the versioning system, we went with Git. This was our first time using any versioning system so we had to learn a lot. We went with a standard approach and used a terminal. As we did not use GUI we had to learn every common command. This took some time but in the end, it was very easy to work with Git through the terminal.

3.2.6 Visual Studio

For our development environment, we chose Visual Studio 2019 Community Edition as we were familiar with it and also for its wide range of support material. Its Git implementation was not very complex but for our needs, it was more than sufficient.

Because of the server connection, we had to implement a Newtonsoft.Json package from NuGet which provided us with the needed commands to ensure a simple link between the database and front end.

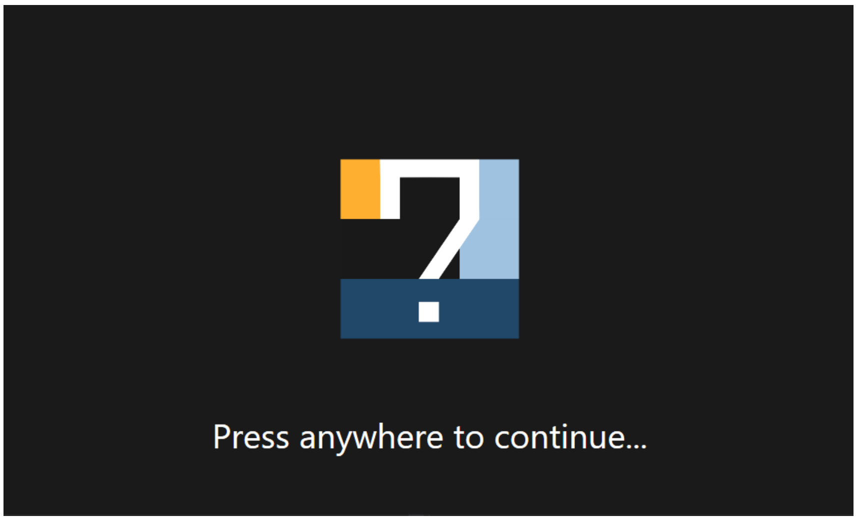
4 User interface

We went with a clean and minimalistic user interface that would be easy to maneuver but still be mostly invisible for the player when not needed.

As we decided in the mockup we went with dark mode. We omitted the vivid colors because they were distracting and did not bring anything to the player. Because of that, we went with monochromatic minimalist buttons with white borders which give the interface a clean and modern feel.

4.1 Title page

The first window that the player sees is TitlePage. This window serves only to greet the player with the game's logo and requires the player to click to continue.

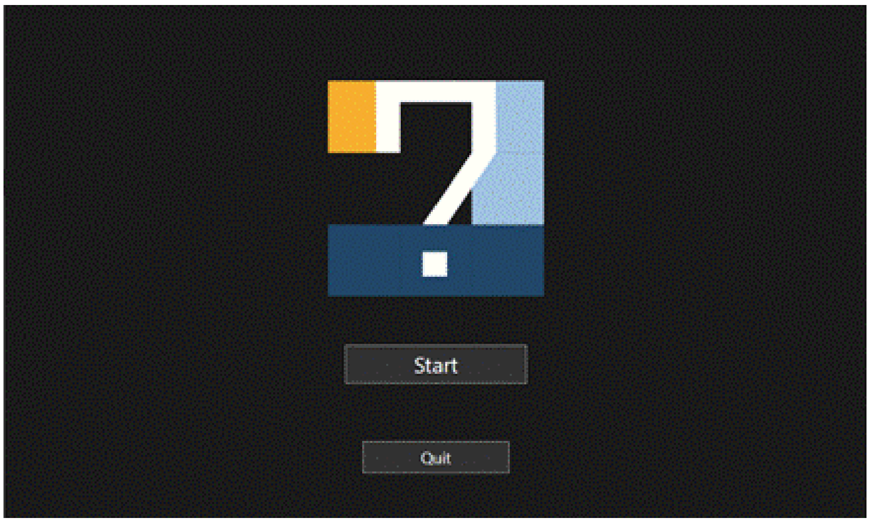


Picture 9: Application – Title page

4.2 Main menu

In the main menu, we have two buttons. We were considering the third button for options but after trying to implement it we found out we have no use for options so we scraped it and stuck with only Start and Quit buttons.

Start button takes the player to the Game setup and Quit button terminates the game.

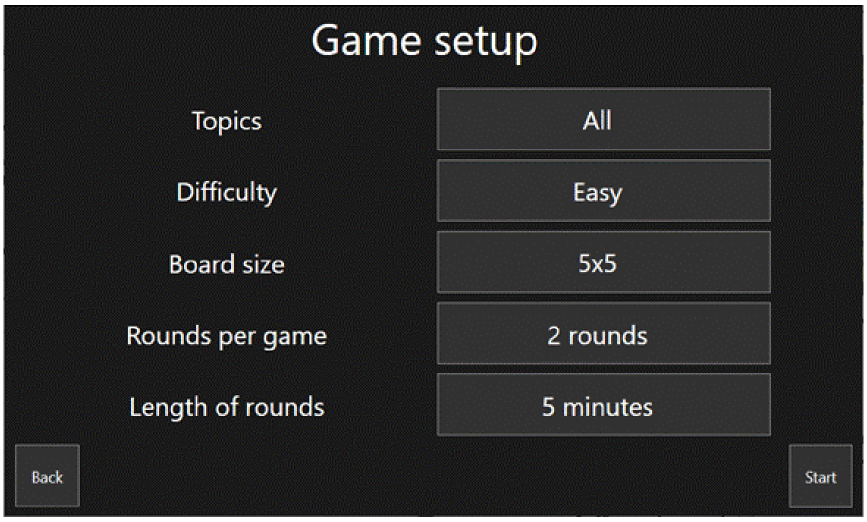


Picture 10: Application – Main menu

4.3 Game setup

In game setup player can choose his own settings for the game.

- The back button gets the player back to the main menu.
- The start button starts the game with the player's defined options.



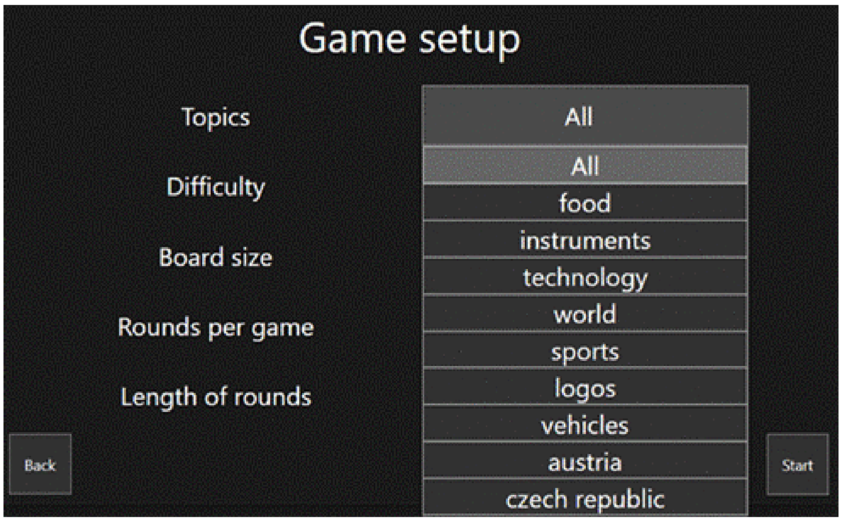
Game setup

Topics	All
Difficulty	Easy
Board size	5x5
Rounds per game	2 rounds
Length of rounds	5 minutes

Picture 11: Application – Game setup

4.3.1 Topics

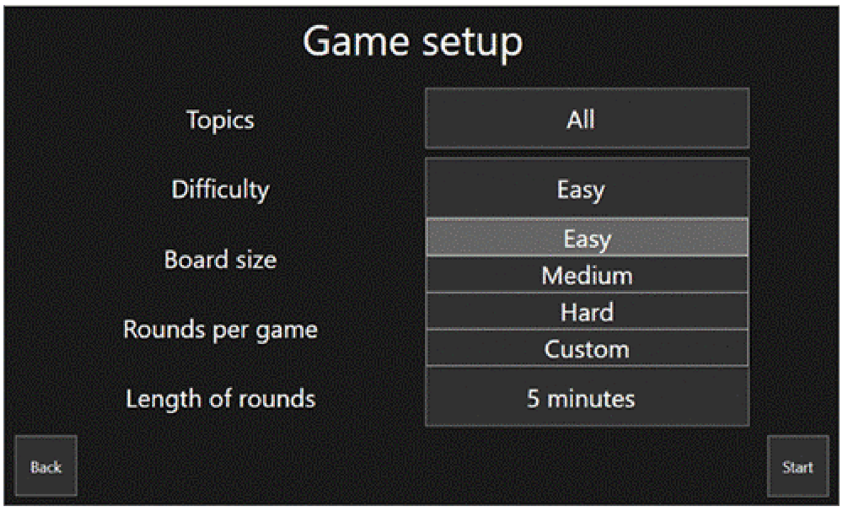
In topics, the player can choose from which category the pictures will be.



Picture 12: Application – Game setup (Topics)

4.3.2 Difficulty

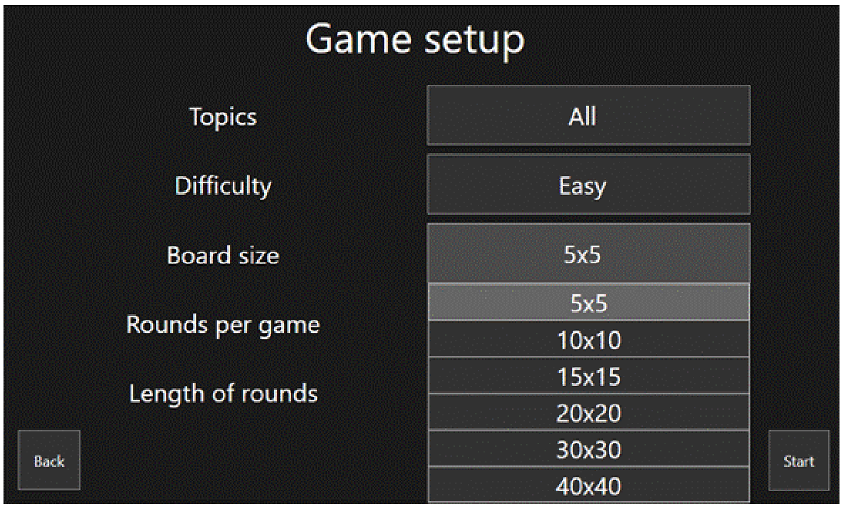
Difficulty sets board size, rounds per game, and length of rounds for the player. Players can also choose custom difficulty which will not be standardized and can be more challenging or easier as the player can choose different scenarios for the game.



Picture 13: Application – Game setup (Difficulty)

4.3.3 Board size

Board size affects the number of individual squares that hide the picture. The bigger the board size the more time player needs to guess correctly as the revealed squares are much smaller.


Game setup

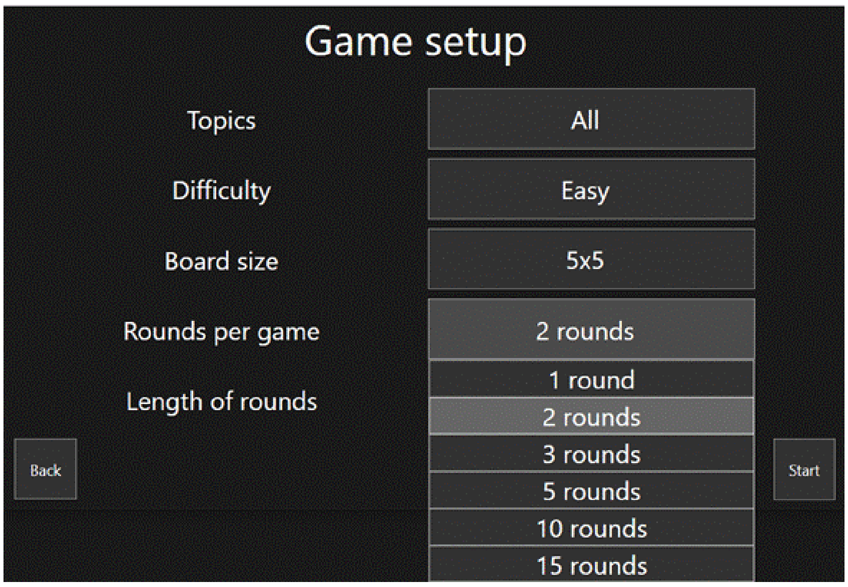
Topics: All
Difficulty: Easy
Board size: 5x5
Rounds per game
Length of rounds

5x5
10x10
15x15
20x20
30x30
40x40

Picture 14: Application – Game setup (Board size)

4.3.4 Rounds per game

Rounds per game is the number of rounds in one playthrough.


Game setup

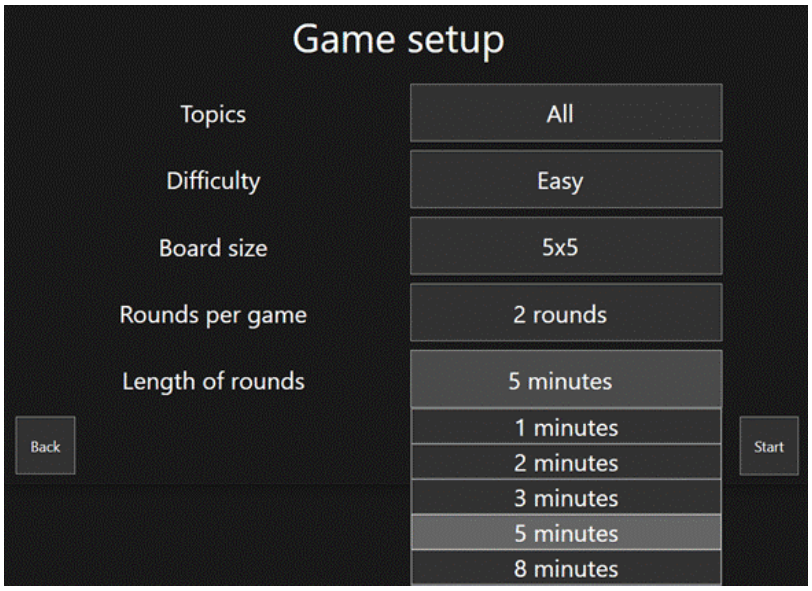
Topics: All
Difficulty: Easy
Board size: 5x5
Rounds per game: 2 rounds
Length of rounds

1 round
2 rounds
3 rounds
5 rounds
10 rounds
15 rounds

Picture 15: Application – Game setup (Rounds per game)

4.3.5 Length of rounds

The length of rounds is the time in minutes for one round. After the time is depleted the round ends and the game continues to another round or if it was the last round the game ends.



Game setup

Topics	All
Difficulty	Easy
Board size	5x5
Rounds per game	2 rounds
Length of rounds	5 minutes

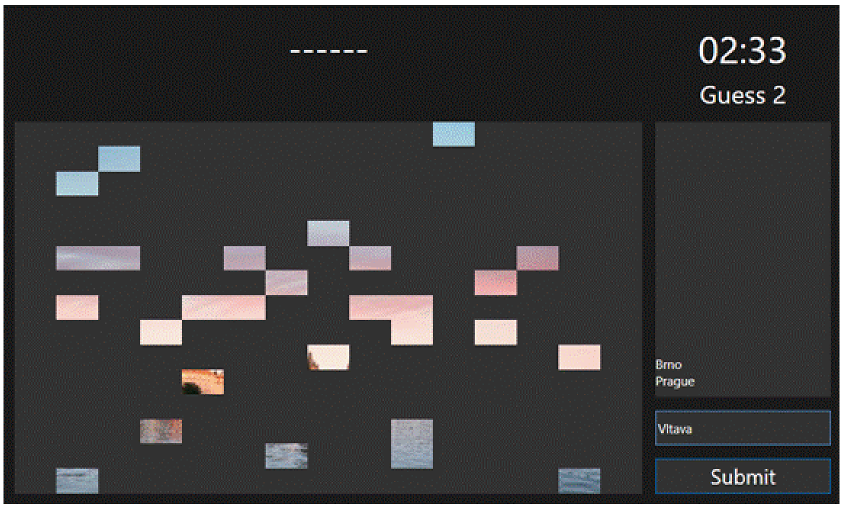
Picture 16: Application – Game setup (Length of rounds)

4.4 Game page

When the player chooses his difficulty and presses the start button they are put into the game page where the game itself takes place.

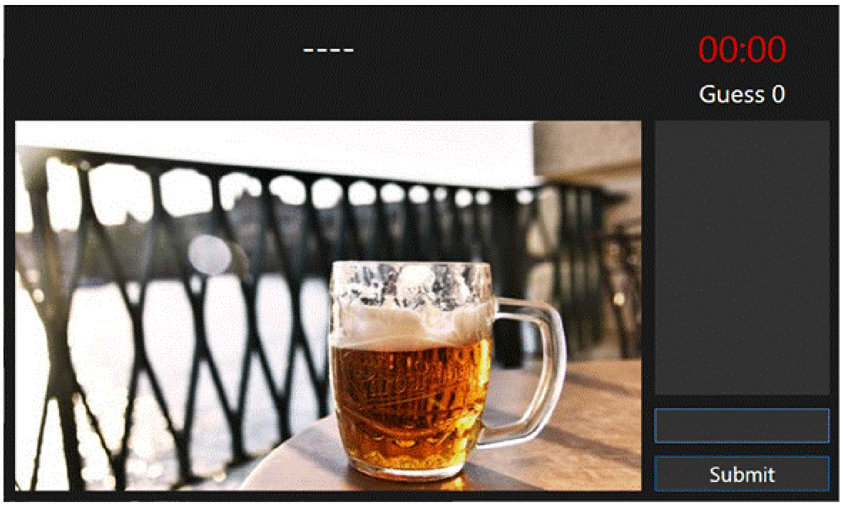
After the timer starts every second new piece of the picture reveals.

When the player guesses incorrectly their guess will log above the text box so that the player knows what they already guessed.



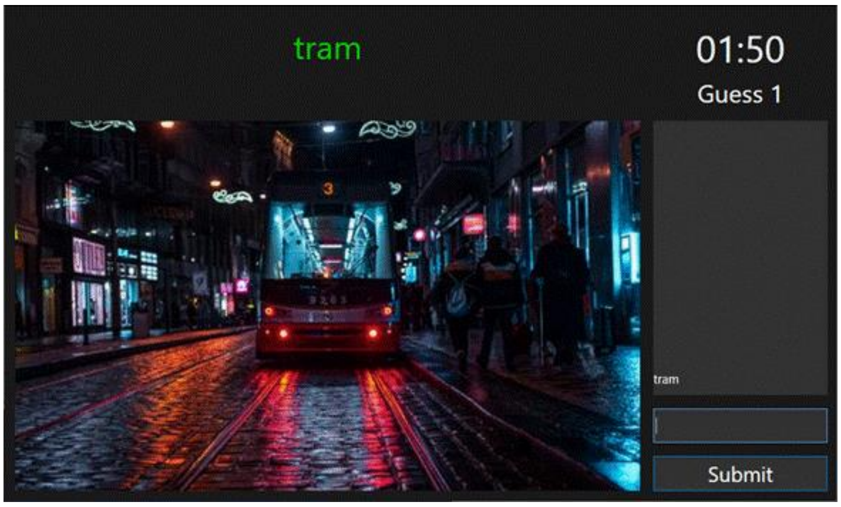
Picture 17: Application – Game page

If the time runs out the round will end and continue to the next round if there is any.



Picture 18: Application – Game lost

If the player guesses the hidden picture correctly all of the hidden boxes will reveal, the answer will be shown in green and the timer will be stoped. After a delay, the player is thrown into the next round if possible or the game ends.



Picture 19: Application – Game win

The game page consists of 7 parts.

In the middle of the top part, there is a hint in terms of how many letters the answer has.

On the right side from the top we have: timer, a number of guesses, log box, guess box, submit button.

In the middle, the biggest part of the page is the picture itself. The picture is divided into small boxes based on the difficulty that the player chooses that uncover every second.

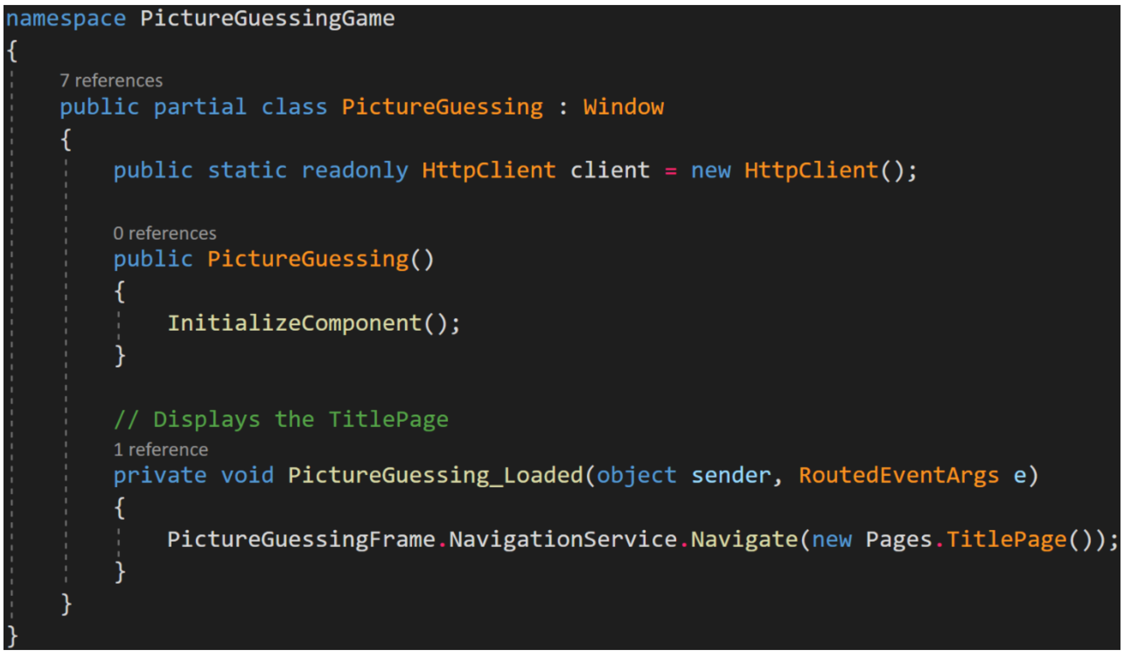
5 Code

5.1 Picture guessing

This is the window used for the whole application and it is also the only window. It contains only frame element that is used to display pages (e.g. main menu, game setup, ...).

It also creates an instance of HttpClient that is later used for requests to the server.

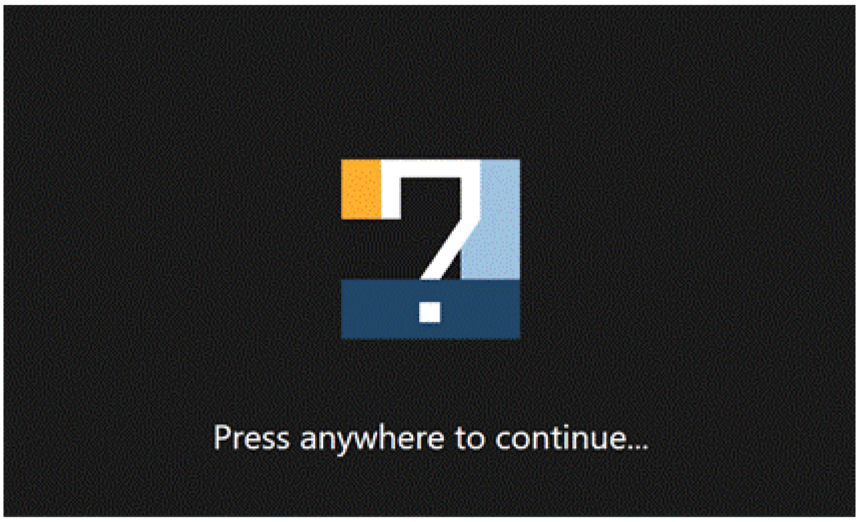
5.1.1 PictureGuessing_Loaded



```
namespace PictureGuessingGame
{
    7 references
    public partial class PictureGuessing : Window
    {
        public static readonly HttpClient client = new HttpClient();

        0 references
        public PictureGuessing()
        {
            InitializeComponent();
        }

        // Displays the TitlePage
        1 reference
        private void PictureGuessing_Loaded(object sender, RoutedEventArgs e)
        {
            PictureGuessingFrame.NavigationService.Navigate(new Pages.TitlePage());
        }
    }
}
```

This method will be called when the window is loaded and it will use the frame element to display the title page.

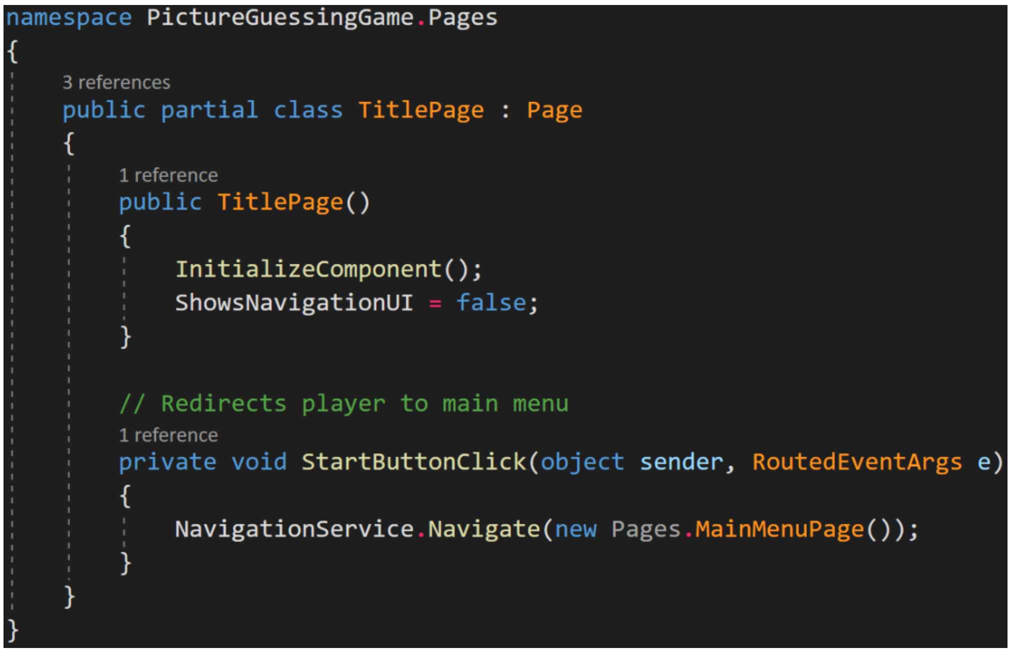
5.2 Title screen



Picture 20: Application – Title page 2

When you start the application the first thing you will see is the title screen. The only function of this screen is to greet the player and redirect him to the main menu. We achieve this by having an invisible button over the whole screen.

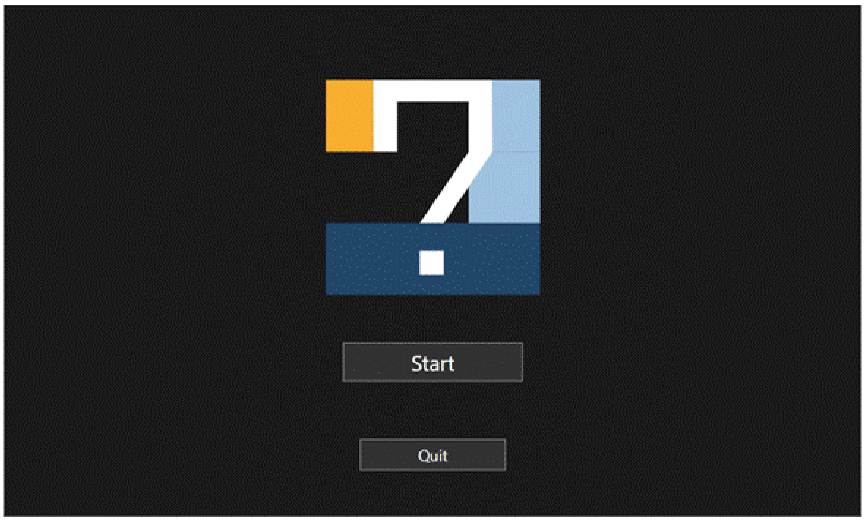
5.2.1 StartButtonClick



```
namespace PictureGuessingGame.Pages
{
    3 references
    public partial class TitlePage : Page
    {
        1 reference
        public TitlePage()
        {
            InitializeComponent();
            ShowsNavigationUI = false;
        }

        // Redirects player to main menu
        1 reference
        private void StartButtonClick(object sender, RoutedEventArgs e)
        {
            NavigationService.Navigate(new Pages.MainMenuPage());
        }
    }
}
```

This method is binded to the invisible button and it redirects the player to the main menu.

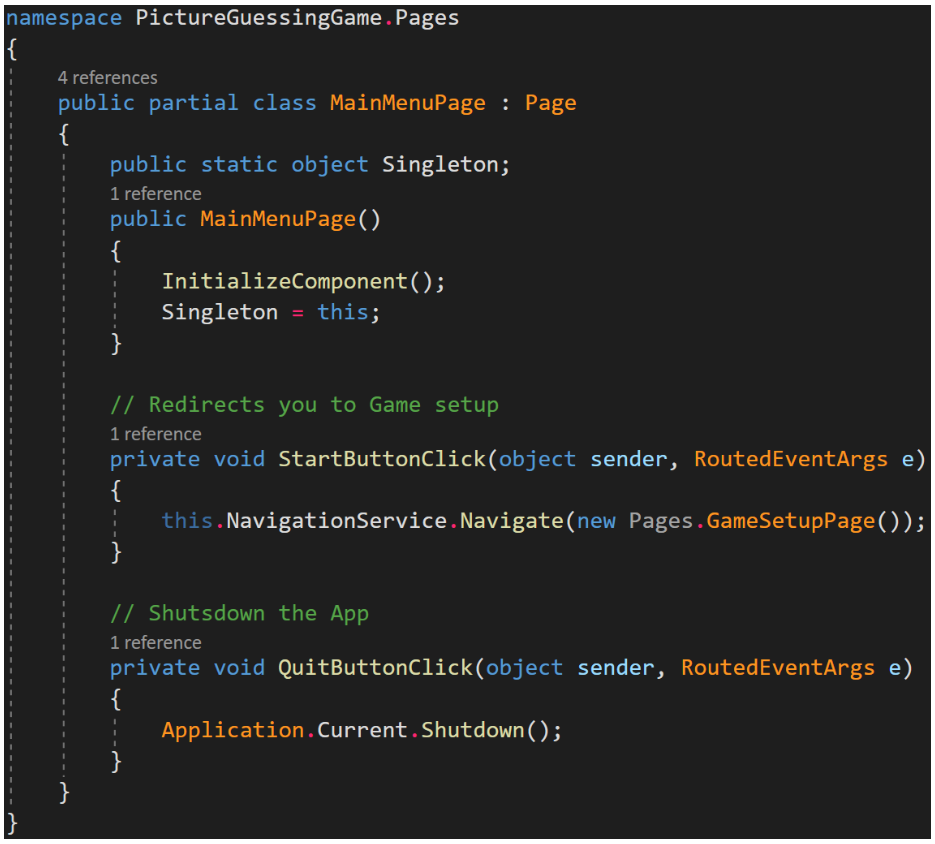
5.3 Main menu



Picture 21: Application – Main menu 2

Main menu currently has 2 buttons Start and Quit. Start button redirects you to game setup while Quit shuts down the application. While this menu doesn't have many options and therefore could be easily skipped it is easily scalable which is good for later additions (e.g. settings) and also it is less overwhelming.

5.3.1 MainMenuPage



```
namespace PictureGuessingGame.Pages
{
    4 references
    public partial class MainMenuPage : Page
    {
        public static object Singleton;
        1 reference
        public MainMenuPage()
        {
            InitializeComponent();
            Singleton = this;
        }

        // Redirects you to Game setup
        1 reference
        private void StartButtonClick(object sender, RoutedEventArgs e)
        {
            this.NavigationService.Navigate(new Pages.GameSetupPage());
        }

        // Shutsdown the App
        1 reference
        private void QuitButtonClick(object sender, RoutedEventArgs e)
        {
            Application.Current.Shutdown();
        }
    }
}
```

In the constructor of this page, we set the Singleton variable. We use the singleton pattern to be able to get to the menu from anywhere without having to create a new menu. (Singleton pattern = restricting the instantiation of the class only one instance that will be stored in a public static variable)

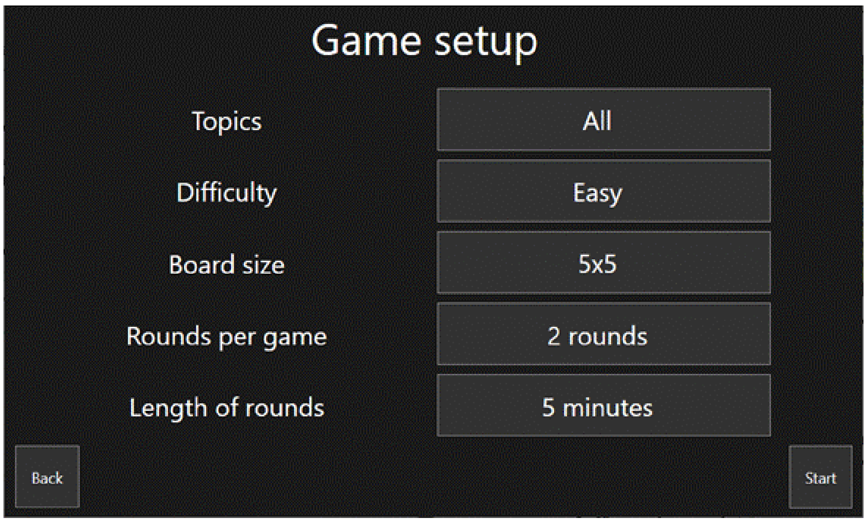
5.3.1.1 StartButtonClick

This binded to the start button. It redirects you to the Game Setup.

5.3.1.2 QuitButtonClick

This binded to the quit button. It shuts down the application.

5.4 Game setup page

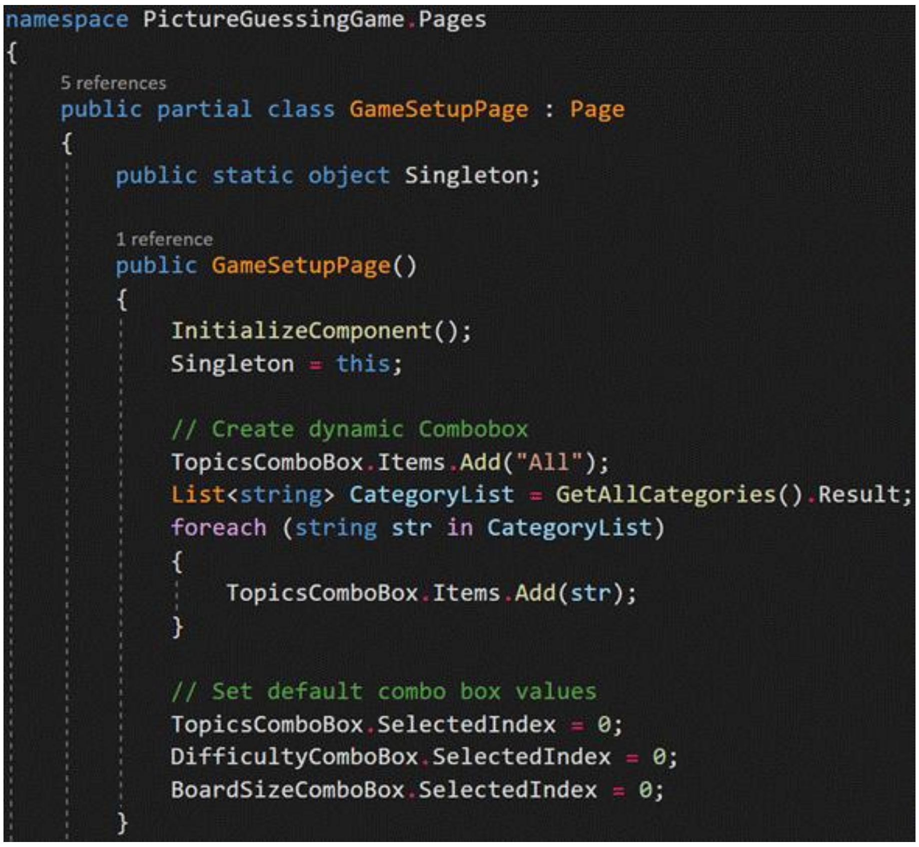

Game setup

Topics	All
Difficulty	Easy
Board size	5x5
Rounds per game	2 rounds
Length of rounds	5 minutes

Picture 22: Application – Game setup 2

Game setup is used to tweak all the settings of the game we want to play. You can change a total of 5 combo boxes with settings that will then affect the game. Accompanied by Start and Back buttons.

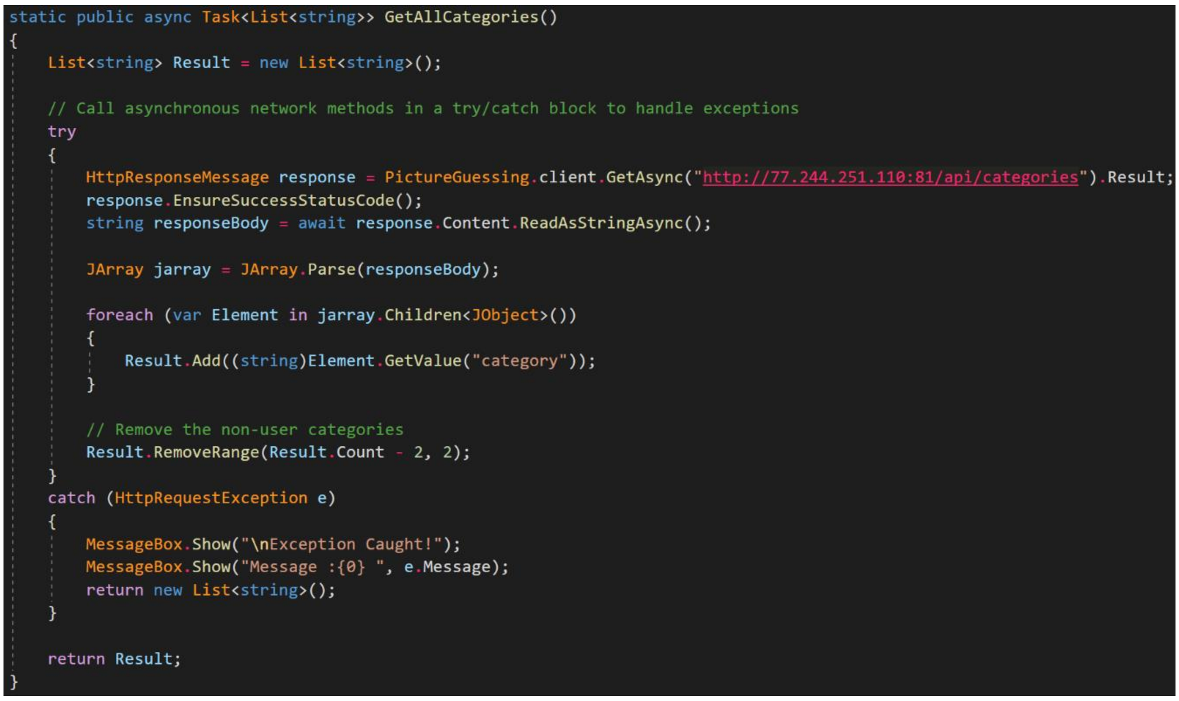
5.4.1 GameSetupPage



```
namespace PictureGuessingGame.Pages
{
    5 references
    public partial class GameSetupPage : Page
    {
        public static object Singleton;

        1 reference
        public GameSetupPage()
        {
            InitializeComponent();
            Singleton = this;

            // Create dynamic Combobox
            TopicsComboBox.Items.Add("All");
            List<string> CategoryList = GetAllCategories().Result;
            foreach (string str in CategoryList)
            {
                TopicsComboBox.Items.Add(str);
            }

            // Set default combo box values
            TopicsComboBox.SelectedIndex = 0;
            DifficultyComboBox.SelectedIndex = 0;
            BoardSizeComboBox.SelectedIndex = 0;
        }
```

In this constructor, we again first set the singleton variable. Then we proceed to create the category combo box. In the beginning, these were like all the others created inside XAML → predefined by us (frontend), so we asked our Austrian colleagues to implement a Get for all the categories. That means that adding new categories is handled on the server-side which is ideal because that is where the pictures are added. That means that any new category added to the server will be automatically updated on the client. And then we select default values for our combo boxes.

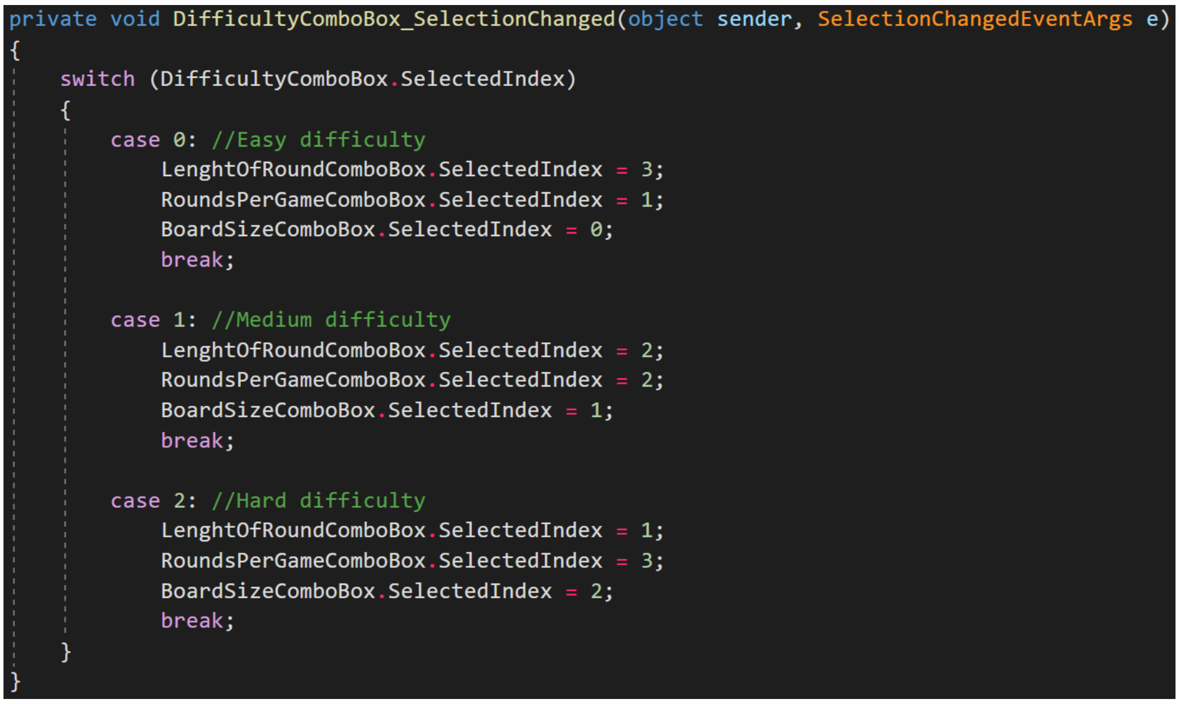
5.4.2 GetAllCategories



```
static public async Task<List<string>> GetAllCategories()
{
    List<string> Result = new List<string>();

    // Call asynchronous network methods in a try/catch block to handle exceptions
    try
    {
        HttpResponseMessage response = PictureGuessing.client.GetAsync("http://77.244.251.110:81/api/categories").Result;
        response.EnsureSuccessStatusCode();
        string responseBody = await response.Content.ReadAsStringAsync();

        JArray jarray = JArray.Parse(responseBody);

        foreach (var Element in jarray.Children<JObject>())
        {
            Result.Add((string)Element.GetValue("category"));
        }

        // Remove the non-user categories
        Result.RemoveRange(Result.Count - 2, 2);
    }
    catch (HttpRequestException e)
    {
        MessageBox.Show("\nException Caught!");
        MessageBox.Show("Message :{0} ", e.Message);
        return new List<string>();
    }

    return Result;
}
```

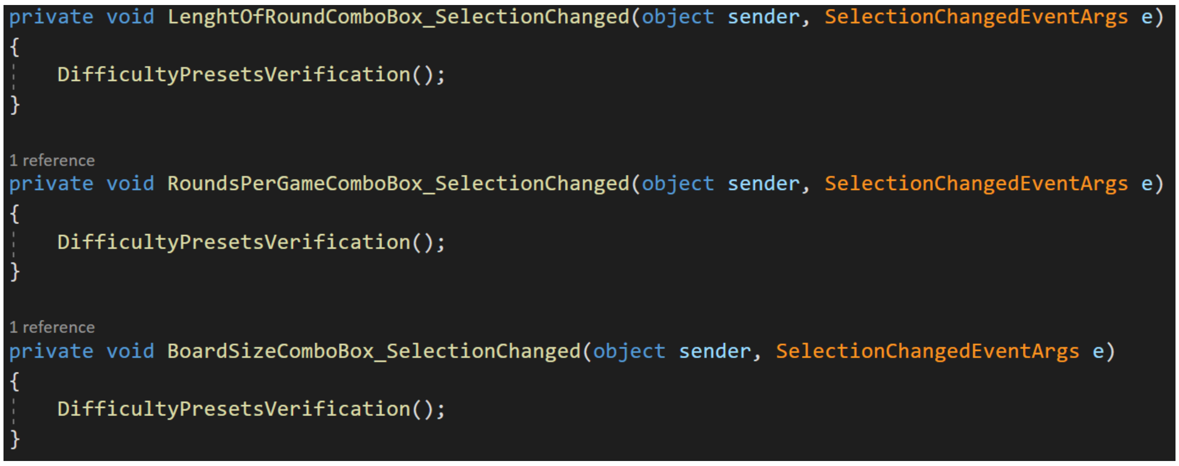
This function calls Get on the server. The server returns names of all the categories that we have to convert to string and add to the Result List of strings that we will return at the end as our message. This whole function is also encapsulated in try and catch block to catch exceptions.

5.4.3 DifficultyComboBox_SelectionChanged



```
private void DifficultyComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    switch (DifficultyComboBox.SelectedIndex)
    {
        case 0: //Easy difficulty
            LenghtOfRoundComboBox.SelectedIndex = 3;
            RoundsPerGameComboBox.SelectedIndex = 1;
            BoardSizeComboBox.SelectedIndex = 0;
            break;

        case 1: //Medium difficulty
            LenghtOfRoundComboBox.SelectedIndex = 2;
            RoundsPerGameComboBox.SelectedIndex = 2;
            BoardSizeComboBox.SelectedIndex = 1;
            break;

        case 2: //Hard difficulty
            LenghtOfRoundComboBox.SelectedIndex = 1;
            RoundsPerGameComboBox.SelectedIndex = 3;
            BoardSizeComboBox.SelectedIndex = 2;
            break;
    }
}
```

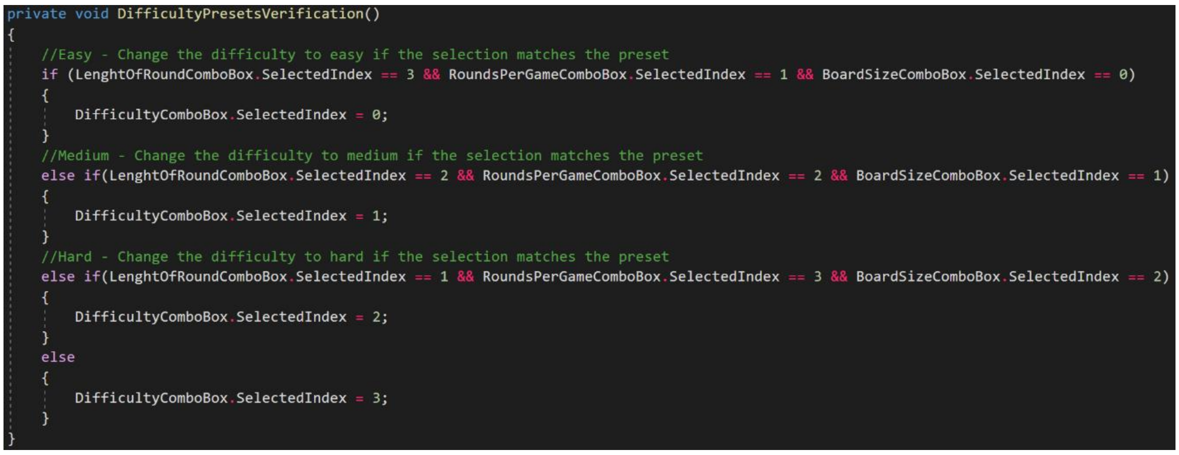
This method is binded to the difficulty combo box. Its purpose is to change all the other options on change of difficulty. We use difficulty as a preset (e.g. easy = board size: 5x5, rounds per game: 2, length of round: 5 minutes).



```
private void LenghtOfRoundComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    DifficultyPresetsVerification();
}

1 reference
private void RoundsPerGameComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    DifficultyPresetsVerification();
}

1 reference
private void BoardSizeComboBox_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    DifficultyPresetsVerification();
}
```

All the other combo boxes have binded call on method DifficultyPresetVerification.

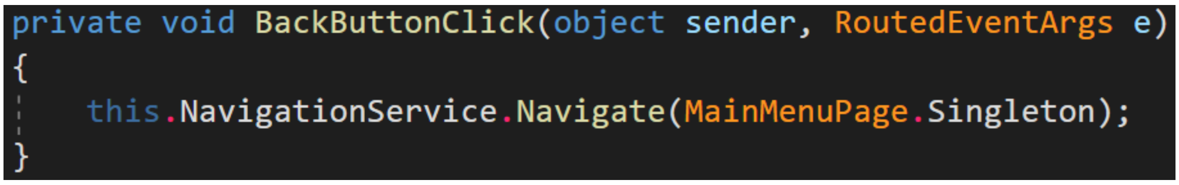
5.4.4 DifficultyPresetVerification



```
private void DifficultyPresetsVerification()
{
    //Easy - Change the difficulty to easy if the selection matches the preset
    if (LenghtOfRoundComboBox.SelectedIndex == 3 && RoundsPerGameComboBox.SelectedIndex == 1 && BoardSizeComboBox.SelectedIndex == 0)
    {
        DifficultyComboBox.SelectedIndex = 0;
    }
    //Medium - Change the difficulty to medium if the selection matches the preset
    else if(LenghtOfRoundComboBox.SelectedIndex == 2 && RoundsPerGameComboBox.SelectedIndex == 2 && BoardSizeComboBox.SelectedIndex == 1)
    {
        DifficultyComboBox.SelectedIndex = 1;
    }
    //Hard - Change the difficulty to hard if the selection matches the preset
    else if(LenghtOfRoundComboBox.SelectedIndex == 1 && RoundsPerGameComboBox.SelectedIndex == 3 && BoardSizeComboBox.SelectedIndex == 2)
    {
        DifficultyComboBox.SelectedIndex = 2;
    }
    else
    {
        DifficultyComboBox.SelectedIndex = 3;
    }
}
```

This method is used to check if one of the presets is selected. If you have selected all the properties that easy difficulty defines (board size, rounds per game, length of round) then the easy difficulty will be selected. When one of these parameters doesn't match a preset the custom difficulty will be selected.

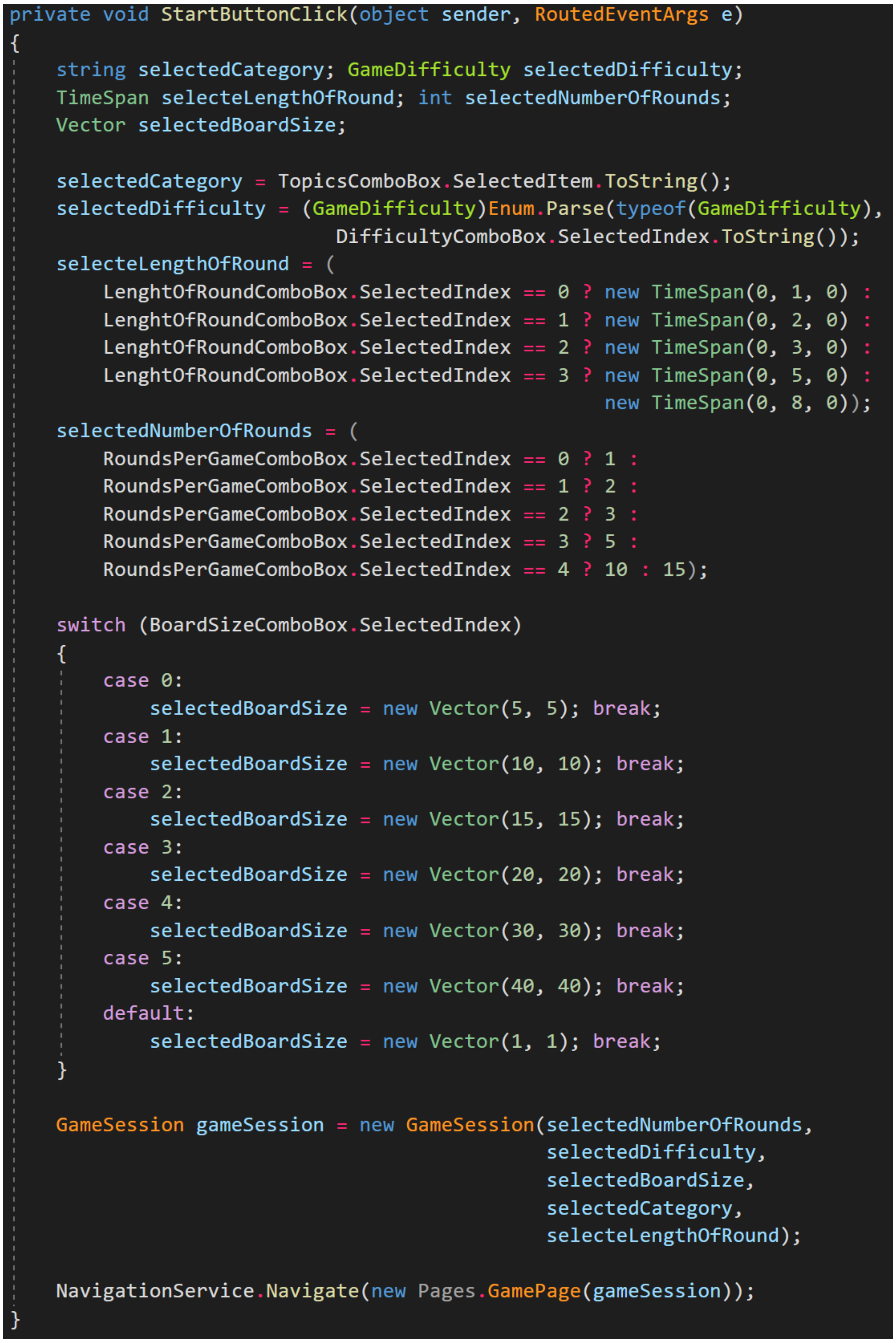
5.4.5 BackButtonClick



```
private void BackButtonClick(object sender, RoutedEventArgs e)
{
    this.NavigationService.Navigate(MainMenuPage.Singleton);
}
```

Back button simply redirects you to the main menu. It uses the Singleton variable.

5.4.6 StartButtonClick



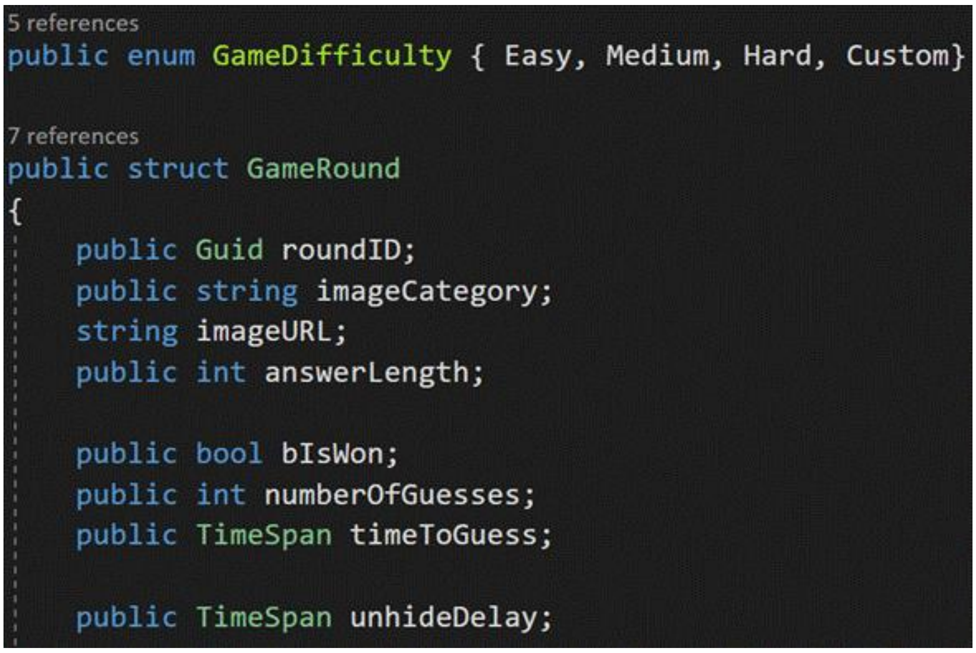
```
private void StartButtonClick(object sender, RoutedEventArgs e)
{
    string selectedCategory; GameDifficulty selectedDifficulty;
    TimeSpan selecteLengthOfRound; int selectedNumberOfRounds;
    Vector selectedBoardSize;

    selectedCategory = TopicsComboBox.SelectedItem.ToString();
    selectedDifficulty = (GameDifficulty)Enum.Parse(typeof(GameDifficulty),
                            DifficultyComboBox.SelectedIndex.ToString());
    selecteLengthOfRound = (
        LenghtOfRoundComboBox.SelectedIndex == 0 ? new TimeSpan(0, 1, 0) :
        LenghtOfRoundComboBox.SelectedIndex == 1 ? new TimeSpan(0, 2, 0) :
        LenghtOfRoundComboBox.SelectedIndex == 2 ? new TimeSpan(0, 3, 0) :
        LenghtOfRoundComboBox.SelectedIndex == 3 ? new TimeSpan(0, 5, 0) :
                                                   new TimeSpan(0, 8, 0));
    selectedNumberOfRounds = (
        RoundsPerGameComboBox.SelectedIndex == 0 ? 1 :
        RoundsPerGameComboBox.SelectedIndex == 1 ? 2 :
        RoundsPerGameComboBox.SelectedIndex == 2 ? 3 :
        RoundsPerGameComboBox.SelectedIndex == 3 ? 5 :
        RoundsPerGameComboBox.SelectedIndex == 4 ? 10 : 15);

    switch (BoardSizeComboBox.SelectedIndex)
    {
        case 0:
            selectedBoardSize = new Vector(5, 5); break;
        case 1:
            selectedBoardSize = new Vector(10, 10); break;
        case 2:
            selectedBoardSize = new Vector(15, 15); break;
        case 3:
            selectedBoardSize = new Vector(20, 20); break;
        case 4:
            selectedBoardSize = new Vector(30, 30); break;
        case 5:
            selectedBoardSize = new Vector(40, 40); break;
        default:
            selectedBoardSize = new Vector(1, 1); break;
    }

    GameSession gameSession = new GameSession(selectedNumberOfRounds,
                                              selectedDifficulty,
                                              selectedBoardSize,
                                              selectedCategory,
                                              selecteLengthOfRound);

    NavigationService.Navigate(new Pages.GamePage(gameSession));
}
```

This method is binded to the start button. It takes all the input that the user gave us and uses it to create a new GameSession. Then we give this new GameSession to the GamePage constructor which is where we navigate next.

5.5 Game session

Before we dive into the GamePage we need to take a look at the GameSession code. This is the only script that doesn't have a page (UI). It is used to drive the game logic. It contains code that is responsible for getting images and creating games. It also contains the definition of other classes.

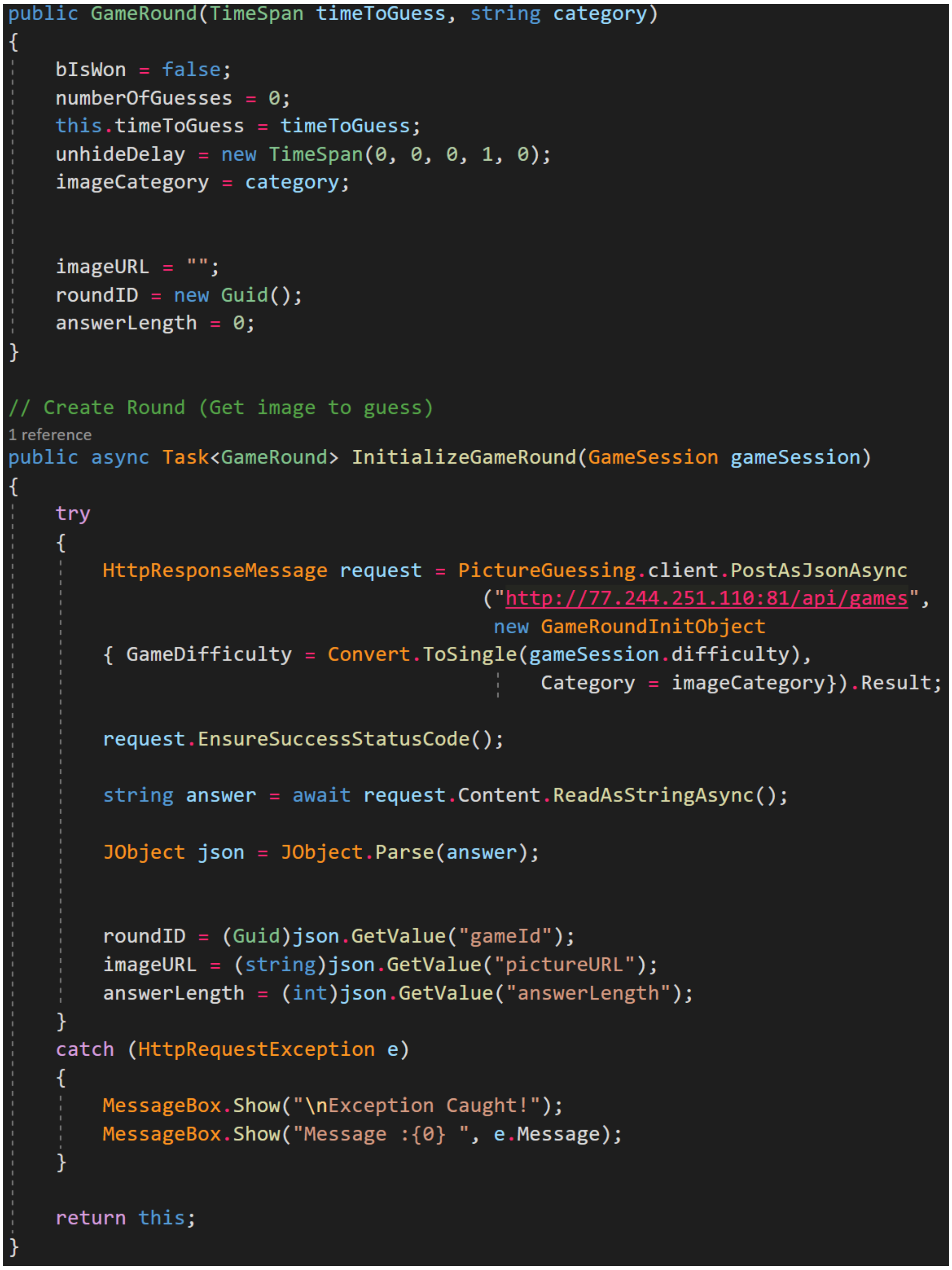
5.5.1 GameRound



```
5 references
public enum GameDifficulty { Easy, Medium, Hard, Custom}

7 references
public struct GameRound
{
    public Guid roundID;
    public string imageCategory;
    string imageURL;
    public int answerLength;

    public bool bIsWon;
    public int numberOfGuesses;
    public TimeSpan timeToGuess;

    public TimeSpan unhideDelay;
```

One game round represents one image.

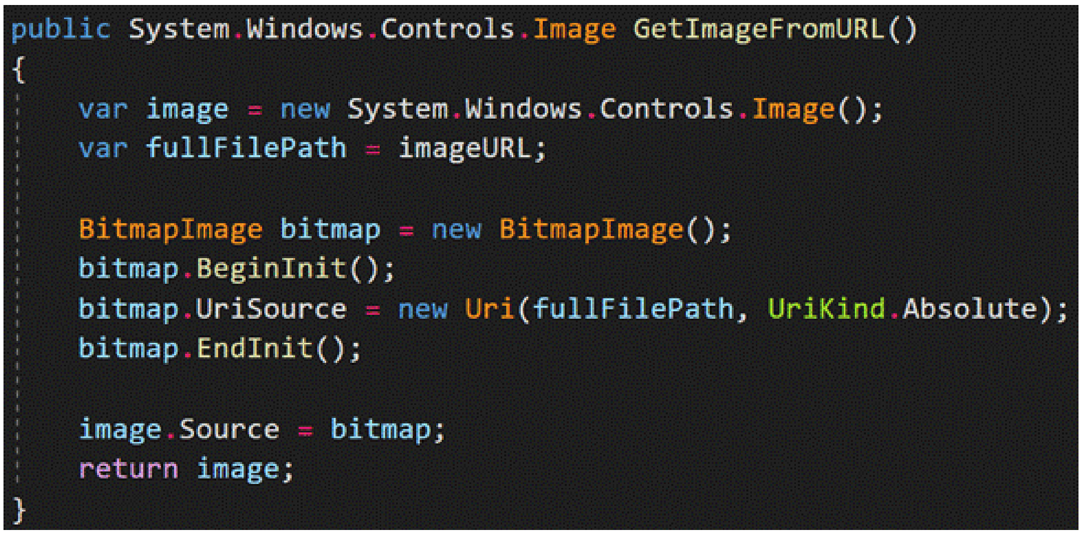
5.5.1.1 InitializeGameRound



```
public GameRound(TimeSpan timeToGuess, string category)
{
    bIsWon = false;
    numberOfGuesses = 0;
    this.timeToGuess = timeToGuess;
    unhideDelay = new TimeSpan(0, 0, 0, 1, 0);
    imageCategory = category;


    imageURL = "";
    roundID = new Guid();
    answerLength = 0;
}

// Create Round (Get image to guess)
1 reference
public async Task<GameRound> InitializeGameRound(GameSession gameSession)
{
    try
    {
        HttpResponseMessage request = PictureGuessing.client.PostAsJsonAsync
                                        ("http://77.244.251.110:81/api/games",
                                         new GameRoundInitObject
        { GameDifficulty = Convert.ToSingle(gameSession.difficulty),
                                            Category = imageCategory}).Result;

        request.EnsureSuccessStatusCode();

        string answer = await request.Content.ReadAsStringAsync();

        JObject json = JObject.Parse(answer);


        roundID = (Guid)json.GetValue("gameId");
        imageURL = (string)json.GetValue("pictureURL");
        answerLength = (int)json.GetValue("answerLength");
    }
    catch (HttpRequestException e)
    {
        MessageBox.Show("\nException Caught!");
        MessageBox.Show("Message :{0} ", e.Message);
    }

    return this;
}
```

Gets the image to guess from the server together with the roundID and answerLength.

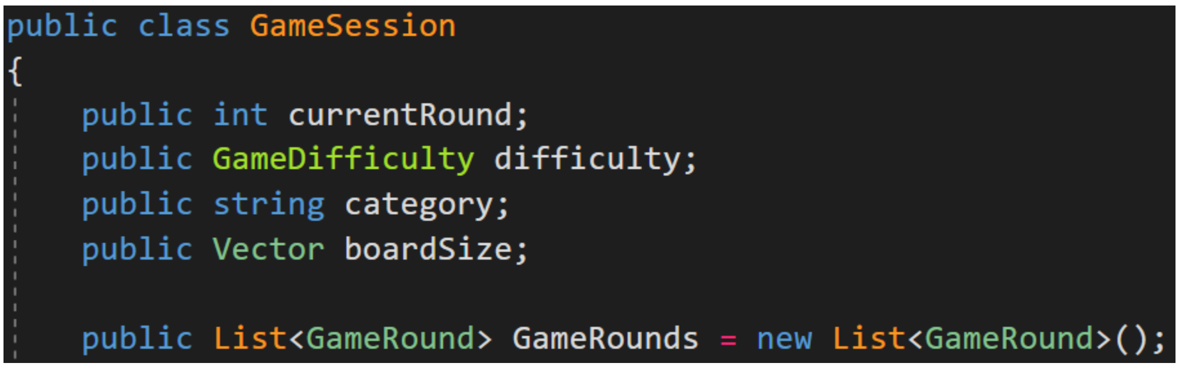
5.5.1.2 GetImageFromURL



```
public System.Windows.Controls.Image GetImageFromURL()
{
    var image = new System.Windows.Controls.Image();
    var fullFilePath = imageURL;

    BitmapImage bitmap = new BitmapImage();
    bitmap.BeginInit();
    bitmap.UriSource = new Uri(fullFilePath, UriKind.Absolute);
    bitmap.EndInit();

    image.Source = bitmap;
    return image;
}
```

This method converts the imageURL to BitmapImage so we can display it.

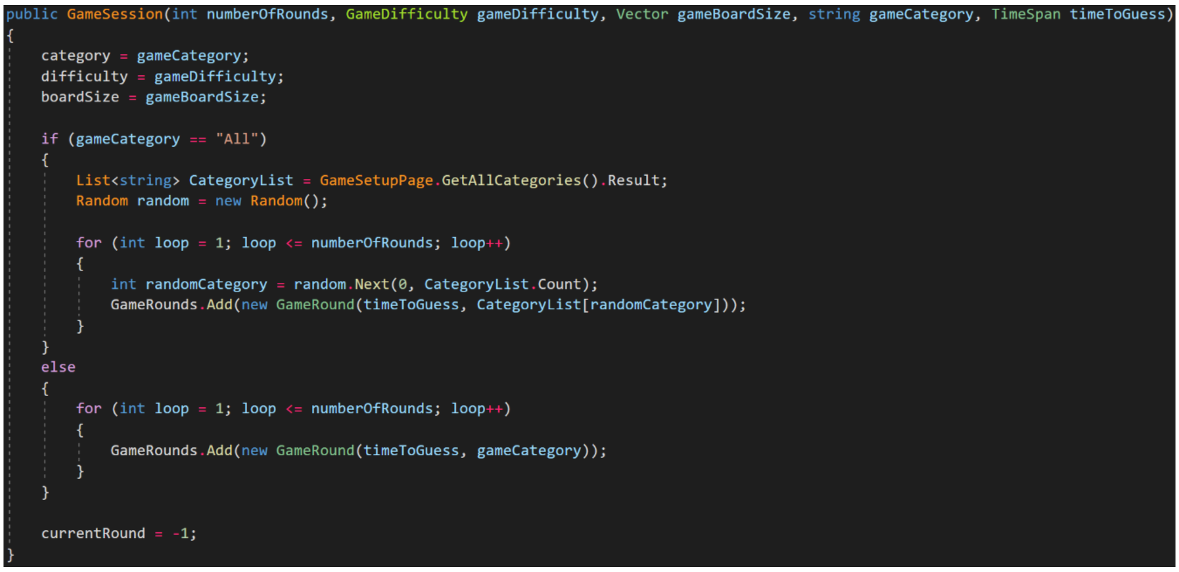
5.5.2 GameSession



```
public class GameSession
{
    public int currentRound;
    public GameDifficulty difficulty;
    public string category;
    public Vector boardSize;

    public List<GameRound> GameRounds = new List<GameRound>();
```

Game session is used to store all the information about a game. You can have multiple GameRounds but only one GameSession. It stores the information relevant to the whole game as well as the game rounds themselves.

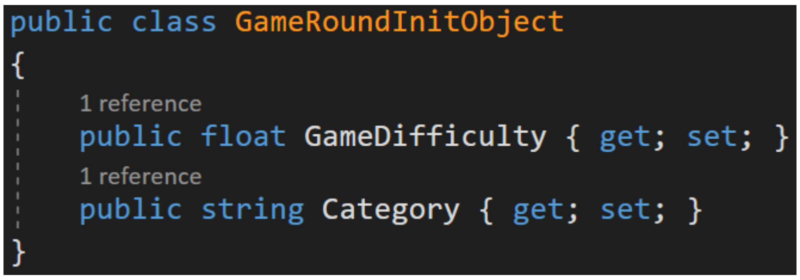
5.5.2.1 GameSession (Constuctor)



```
public GameSession(int numberOfRounds, GameDifficulty gameDifficulty, Vector gameBoardSize, string gameCategory, TimeSpan timeToGuess)
{
    category = gameCategory;
    difficulty = gameDifficulty;
    boardSize = gameBoardSize;

    if (gameCategory == "All")
    {
        List<string> CategoryList = GameSetupPage.GetAllCategories().Result;
        Random random = new Random();

        for (int loop = 1; loop <= numberOfRounds; loop++)
        {
            int randomCategory = random.Next(0, CategoryList.Count);
            GameRounds.Add(new GameRound(timeToGuess, CategoryList[randomCategory]));
        }
    }
    else
    {
        for (int loop = 1; loop <= numberOfRounds; loop++)
        {
            GameRounds.Add(new GameRound(timeToGuess, gameCategory));
        }
    }

    currentRound = -1;
}
```

This constructor is called from the GameSetup page upon clicking the start button. It sets the basic information passed through the constructor. It then proceeds to check if the passed topic is “All” if yes the GameRounds created will be passed random topics. else they will all be created with the topic passed. The All (Random) topic has to be handled on the frontend because there is no server command for it.

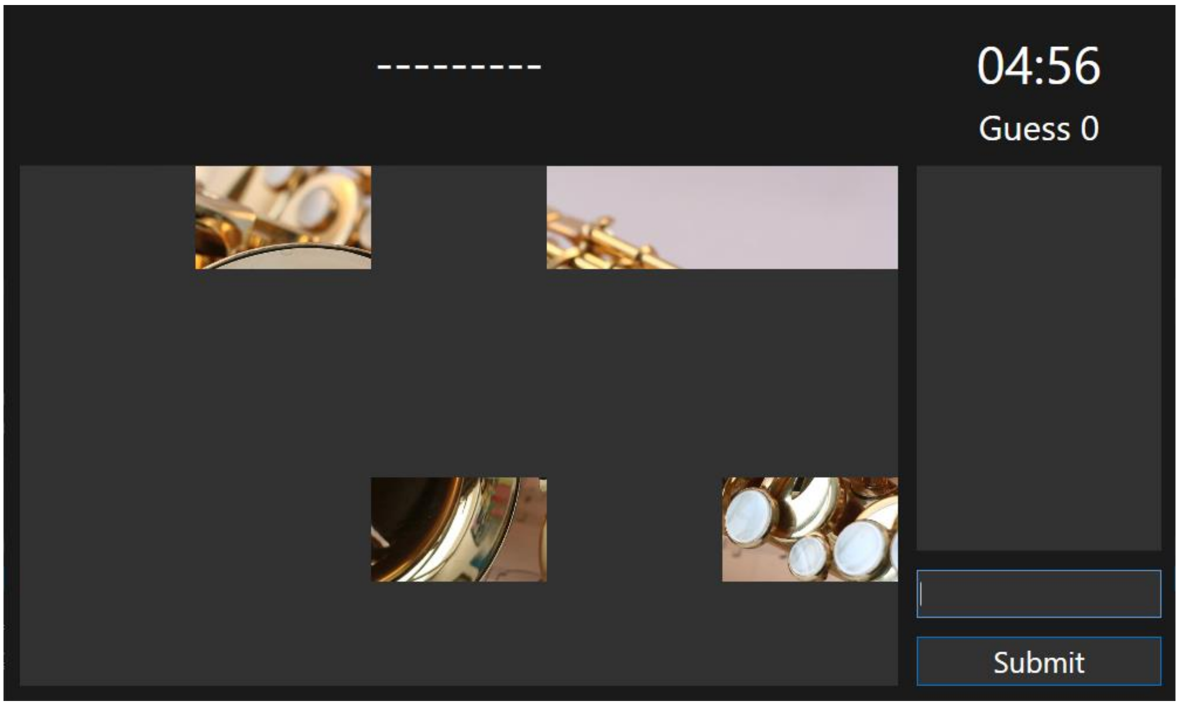
5.5.2.2 GameRoundInitObject



```
public class GameRoundInitObject
{
    1 reference
    public float GameDifficulty { get; set; }
    1 reference
    public string Category { get; set; }
}
```

This object is passed as the parameter when creating a game round in order to pass the difficulty and category (topic) to the server.

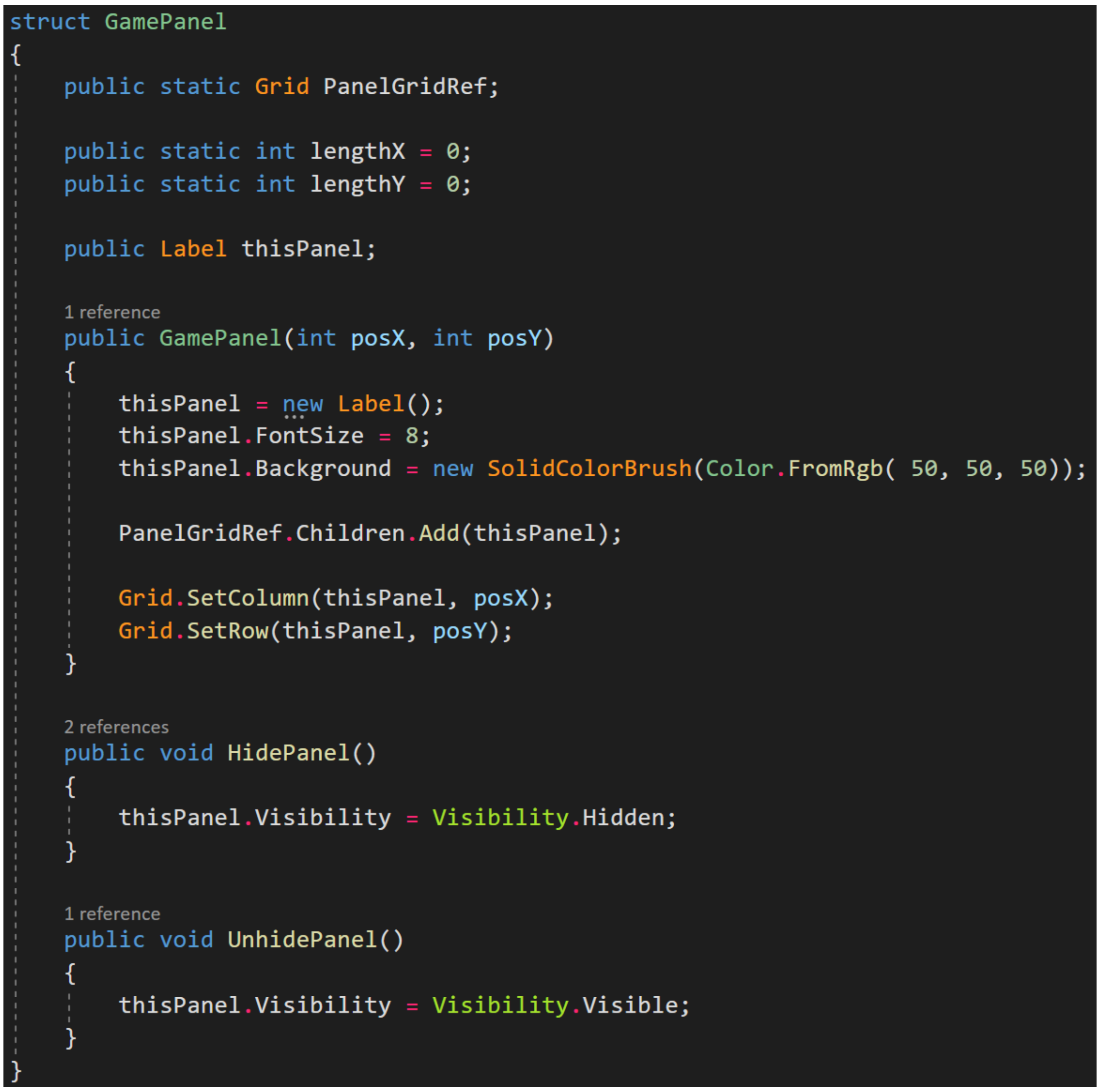
5.6 Game page



Picture 23: Application – Game Page 2

Game page is the window that displays the game and gets player input.

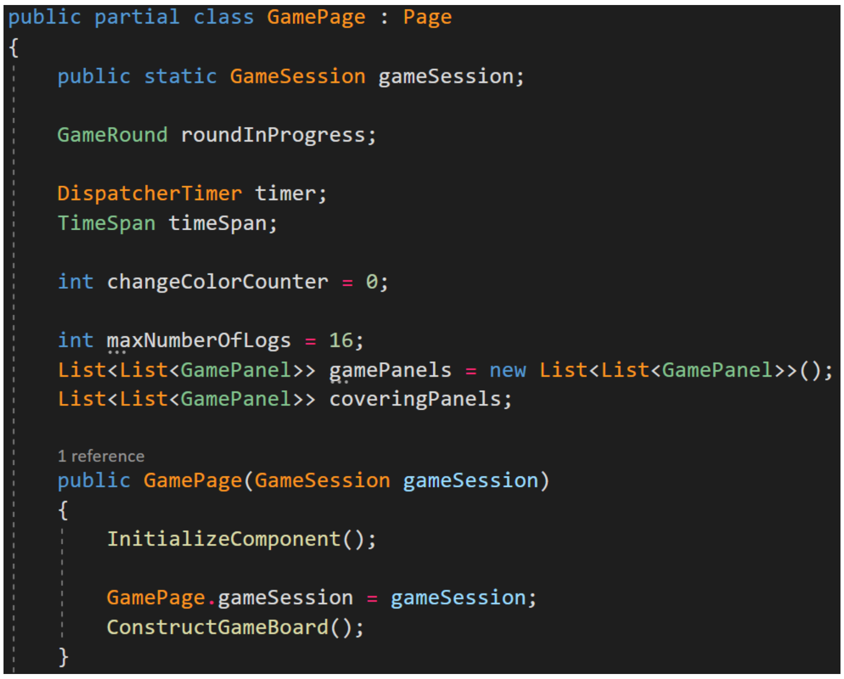
5.6.1 GamePanel



```
struct GamePanel
{
    public static Grid PanelGridRef;

    public static int lengthX = 0;
    public static int lengthY = 0;

    public Label thisPanel;

    1 reference
    public GamePanel(int posX, int posY)
    {
        thisPanel = new Label();
        thisPanel.FontSize = 8;
        thisPanel.Background = new SolidColorBrush(Color.FromRgb( 50, 50, 50));

        PanelGridRef.Children.Add(thisPanel);

        Grid.SetColumn(thisPanel, posX);
        Grid.SetRow(thisPanel, posY);
    }

    2 references
    public void HidePanel()
    {
        thisPanel.Visibility = Visibility.Hidden;
    }

    1 reference
    public void UnhidePanel()
    {
        thisPanel.Visibility = Visibility.Visible;
    }
}
```

Game panels are used to hide the image and slowly uncover it.

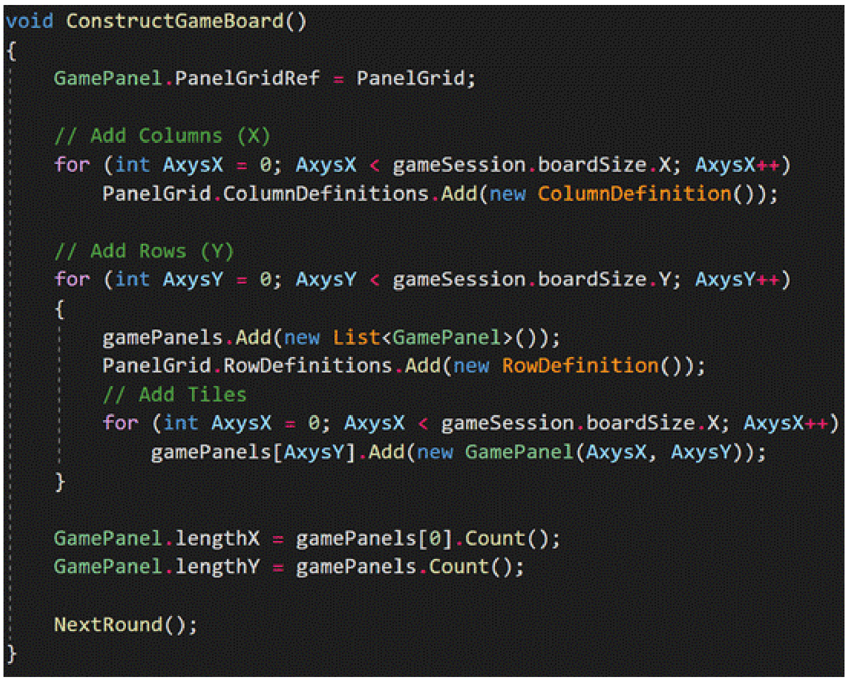
5.6.2 GamePage



```
public partial class GamePage : Page
{
    public static GameSession gameSession;

    GameRound roundInProgress;

    DispatcherTimer timer;
    TimeSpan timeSpan;

    int changeColorCounter = 0;

    int maxNumberOfLogs = 16;
    List<List<GamePanel>> gamePanels = new List<List<GamePanel>>();
    List<List<GamePanel>> coveringPanels;

    1 reference
    public GamePage(GameSession gameSession)
    {
        InitializeComponent();

        GamePage.gameSession = gameSession;
        ConstructGameBoard();
    }
```

Game page controls the game flow using the GameSession passed in its constructor. It also contains all the elements displayed to the user.

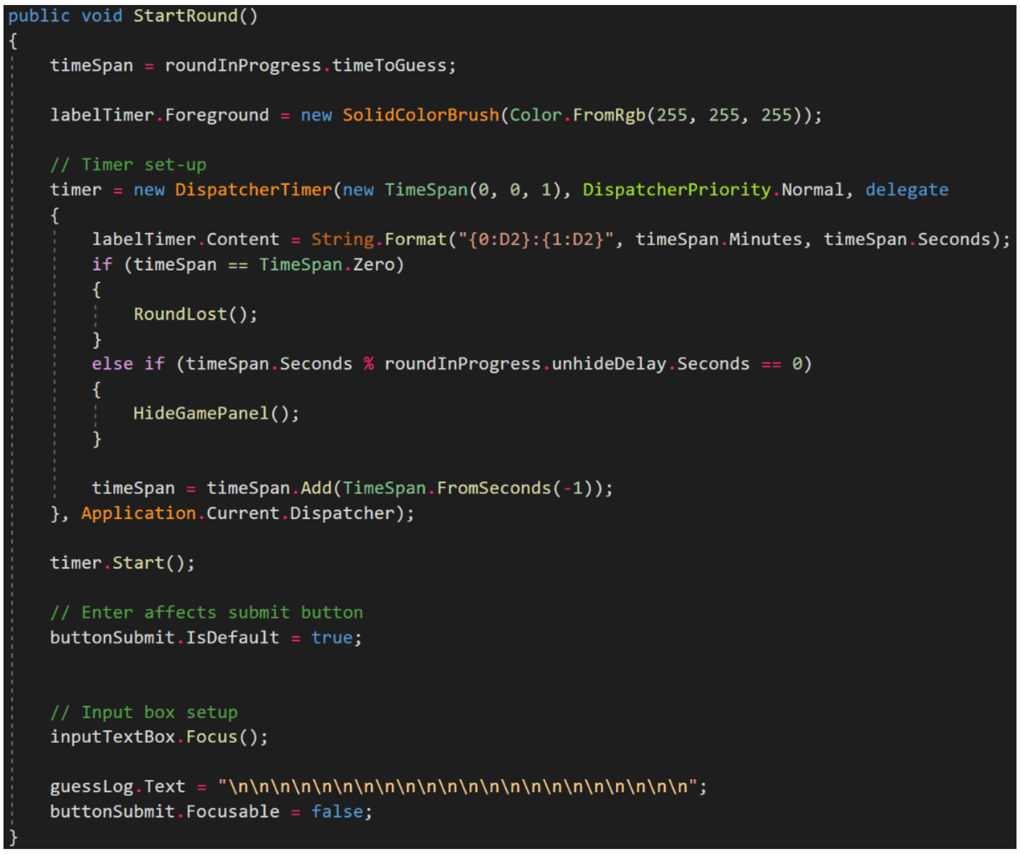
5.6.3 ConstructGameBoard



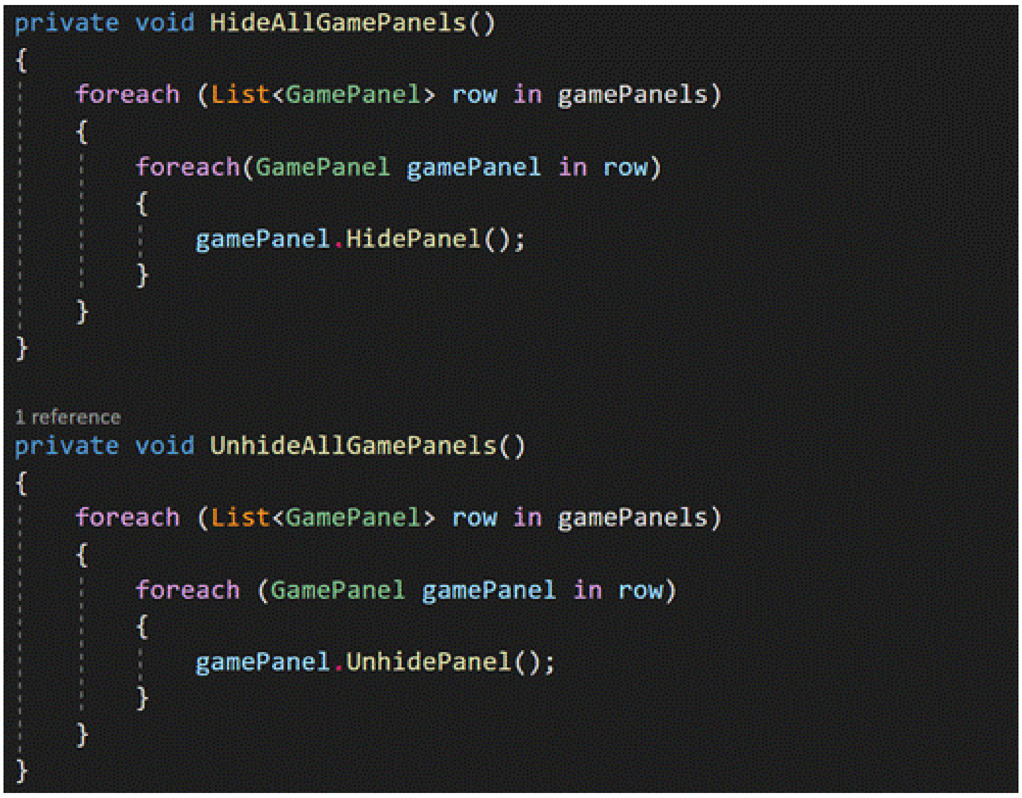
```
void ConstructGameBoard()
{
    GamePanel.PanelGridRef = PanelGrid;

    // Add Columns (X)
    for (int AxysX = 0; AxysX < gameSession.boardSize.X; AxysX++)
        PanelGrid.ColumnDefinitions.Add(new ColumnDefinition());

    // Add Rows (Y)
    for (int AxysY = 0; AxysY < gameSession.boardSize.Y; AxysY++)
    {
        gamePanels.Add(new List<GamePanel>());
        PanelGrid.RowDefinitions.Add(new RowDefinition());
        // Add Tiles
        for (int AxysX = 0; AxysX < gameSession.boardSize.X; AxysX++)
            gamePanels[AxysY].Add(new GamePanel(AxysX, AxysY));
    }

    GamePanel.lengthX = gamePanels[0].Count();
    GamePanel.lengthY = gamePanels.Count();

    NextRound();
}
```

Creates game panels and adds them to the grid.

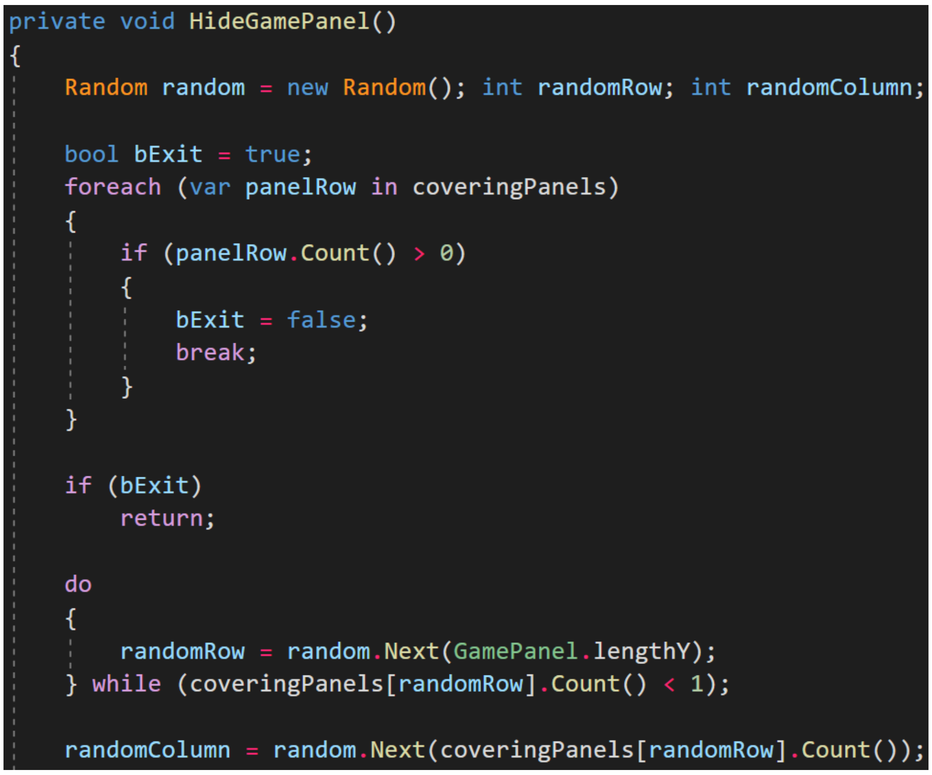
5.6.4 HideGamePanel



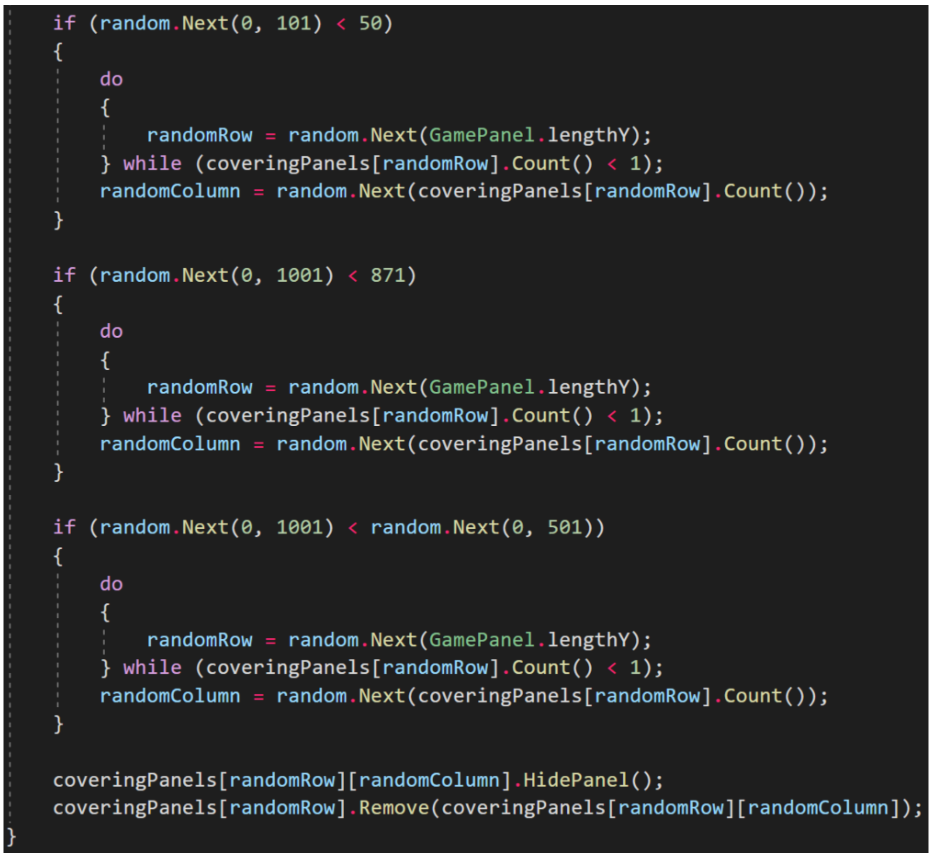
```
public void StartRound()
{
    timeSpan = roundInProgress.timeToGuess;

    labelTimer.Foreground = new SolidColorBrush(Color.FromRgb(255, 255, 255));

    // Timer set-up
    timer = new DispatcherTimer(new TimeSpan(0, 0, 1), DispatcherPriority.Normal, delegate
    {
        labelTimer.Content = String.Format("{0:D2}:{1:D2}", timeSpan.Minutes, timeSpan.Seconds);
        if (timeSpan == TimeSpan.Zero)
        {
            RoundLost();
        }
        else if (timeSpan.Seconds % roundInProgress.unhideDelay.Seconds == 0)
        {
            HideGamePanel();
        }

        timeSpan = timeSpan.Add(TimeSpan.FromSeconds(-1));
    }, Application.Current.Dispatcher);

    timer.Start();

    // Enter affects submit button
    buttonSubmit.IsDefault = true;


    // Input box setup
    inputTextBox.Focus();

    guessLog.Text = "\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n";
    buttonSubmit.Focusable = false;
}
```



```
private void HideAllGamePanels()
{
    foreach (List<GamePanel> row in gamePanels)
    {
        foreach(GamePanel gamePanel in row)
        {
            gamePanel.HidePanel();
        }
    }
}

1 reference
private void UnhideAllGamePanels()
{
    foreach (List<GamePanel> row in gamePanels)
    {
        foreach (GamePanel gamePanel in row)
        {
            gamePanel.UnhidePanel();
        }
    }
}
```



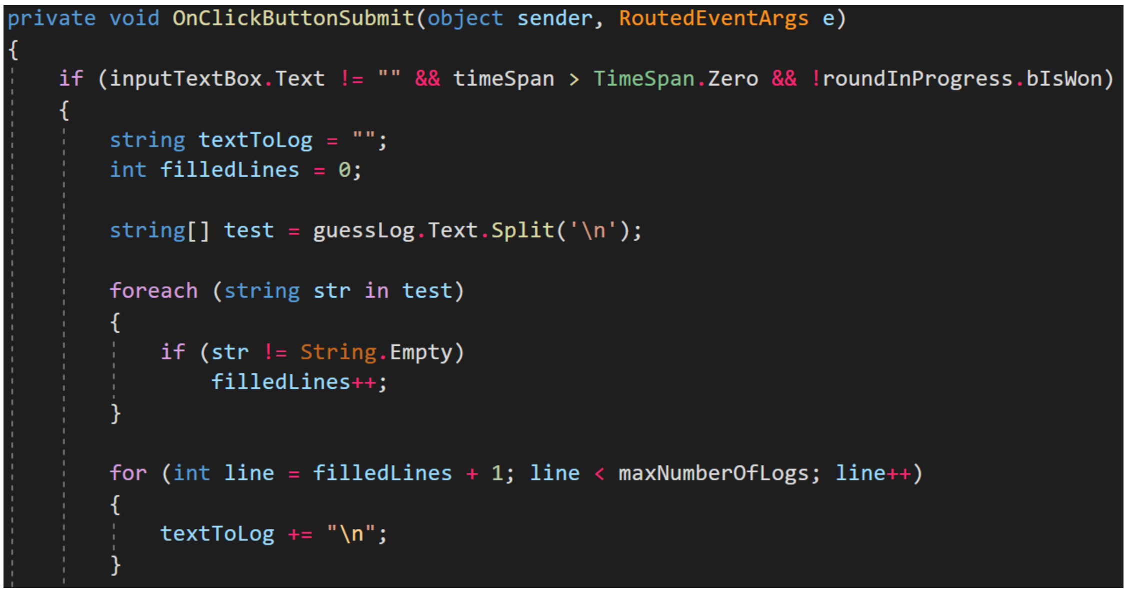
```
private void HideGamePanel()
{
    Random random = new Random(); int randomRow; int randomColumn;

    bool bExit = true;
    foreach (var panelRow in coveringPanels)
    {
        if (panelRow.Count() > 0)
        {
            bExit = false;
            break;
        }
    }

    if (bExit)
        return;

    do
    {
        randomRow = random.Next(GamePanel.lengthY);
    } while (coveringPanels[randomRow].Count() < 1);

    randomColumn = random.Next(coveringPanels[randomRow].Count());
```



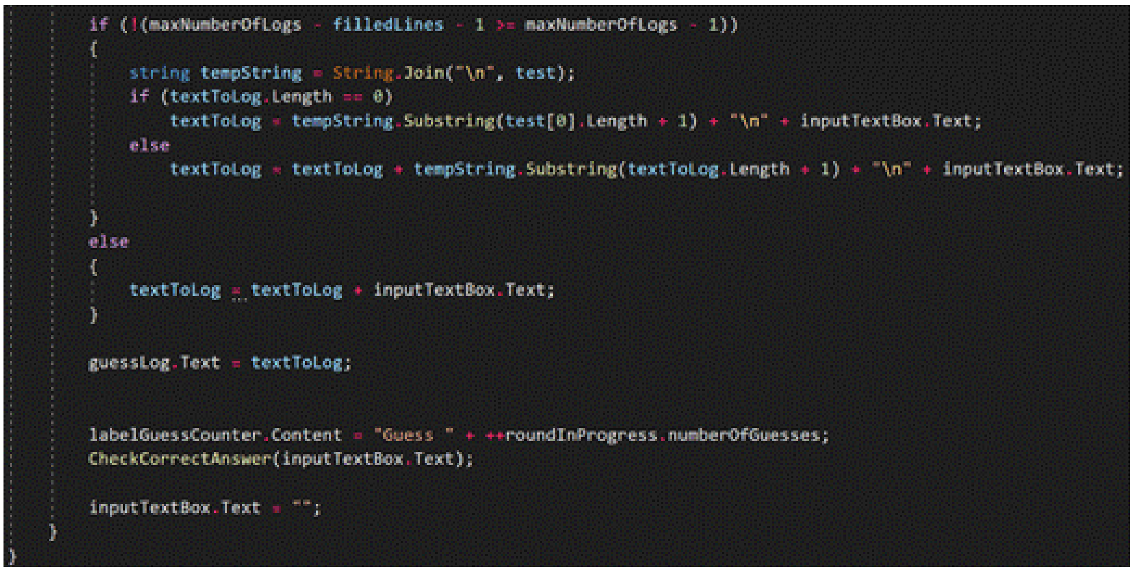
```
    if (random.Next(0, 101) < 50)
    {
        do
        {
            randomRow = random.Next(GamePanel.lengthY);
        } while (coveringPanels[randomRow].Count() < 1);
        randomColumn = random.Next(coveringPanels[randomRow].Count());
    }

    if (random.Next(0, 1001) < 871)
    {
        do
        {
            randomRow = random.Next(GamePanel.lengthY);
        } while (coveringPanels[randomRow].Count() < 1);
        randomColumn = random.Next(coveringPanels[randomRow].Count());
    }

    if (random.Next(0, 1001) < random.Next(0, 501))
    {
        do
        {
            randomRow = random.Next(GamePanel.lengthY);
        } while (coveringPanels[randomRow].Count() < 1);
        randomColumn = random.Next(coveringPanels[randomRow].Count());
    }

    coveringPanels[randomRow][randomColumn].HidePanel();
    coveringPanels[randomRow].Remove(coveringPanels[randomRow][randomColumn]);
}
```

Hides a random game panel.

5.6.5 OnClickButtonSubmit



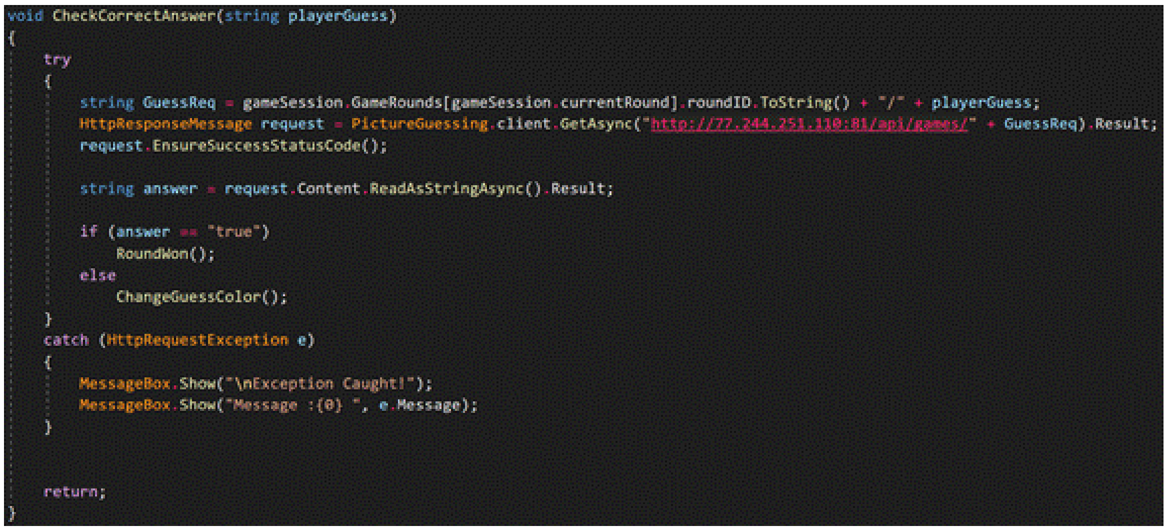
```
private void OnClickButtonSubmit(object sender, RoutedEventArgs e)
{
    if (inputTextBox.Text != "" && timeSpan > TimeSpan.Zero && !roundInProgress.bIsWon)
    {
        string textToLog = "";
        int filledLines = 0;

        string[] test = guessLog.Text.Split('\n');

        foreach (string str in test)
        {
            if (str != String.Empty)
                filledLines++;
        }

        for (int line = filledLines + 1; line < maxNumberOfLogs; line++)
        {
            textToLog += "\n";
        }
```



```
        if (!(maxNumberOfLogs - filledLines - 1 >= maxNumberOfLogs - 1))
        {
            string tempString = String.Join("\n", test);
            if (textToLog.Length == 0)
                textToLog = tempString.Substring(test[0].Length + 1) + "\n" + inputTextBox.Text;
            else
                textToLog = textToLog + tempString.Substring(textToLog.Length + 1) + "\n" + inputTextBox.Text;

        }
        else
        {
            textToLog = textToLog + inputTextBox.Text;
        }

        guessLog.Text = textToLog;


        labelGuessCounter.Content = "Guess " + ++roundInProgress.numberOfGuesses;
        CheckCorrectAnswer(inputTextBox.Text);

        inputTextBox.Text = "";
    }
}
```

Binded to button submit. Checks the answer passed into the textbox. Also controls the log system (Text starts at the bottom).

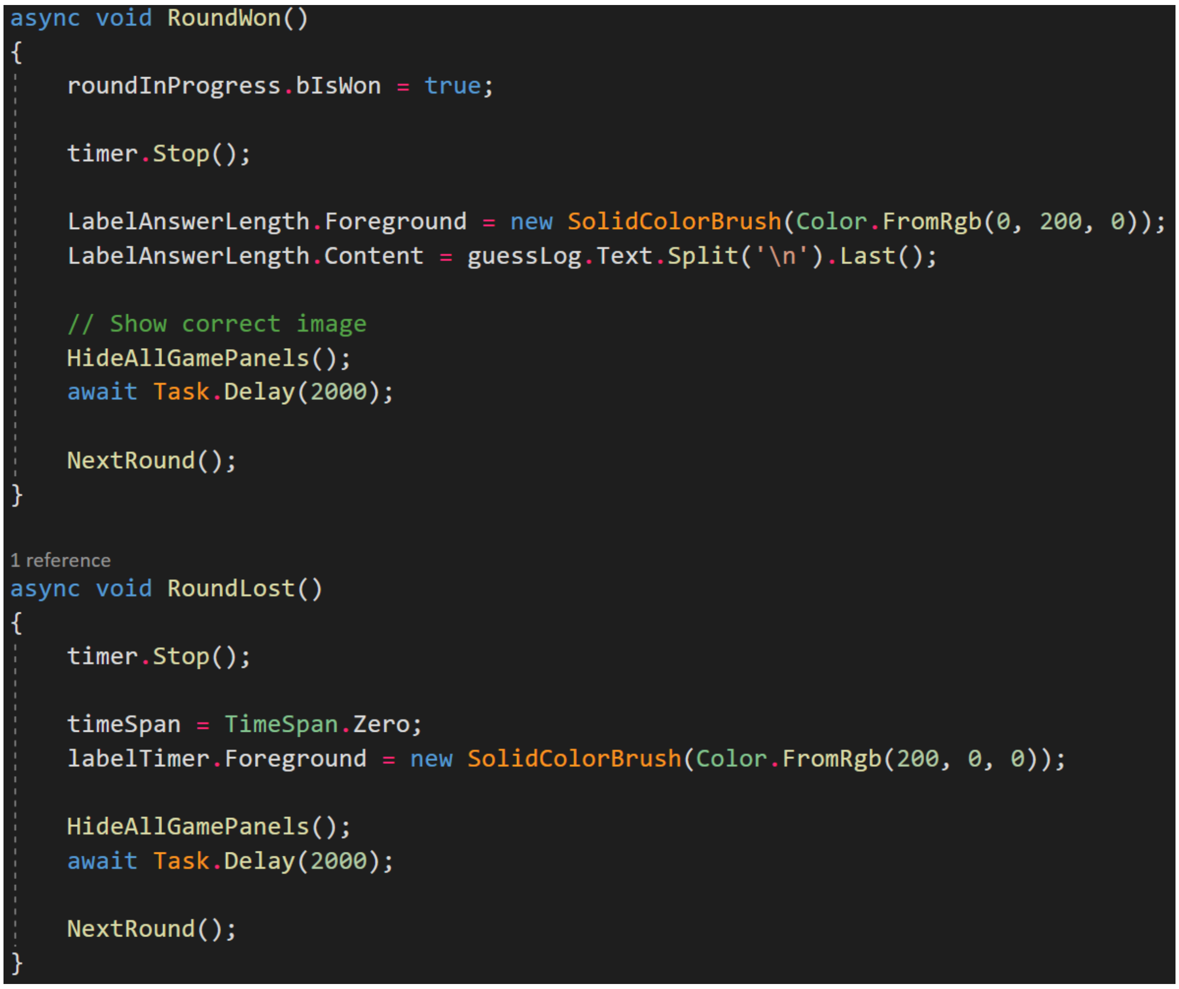
5.6.6 CheckCorrectAnswer



```
void CheckCorrectAnswer(string playerGuess)
{
    try
    {
        string GuessReq = gameSession.GameRounds[gameSession.currentRound].roundID.ToString() + "/" + playerGuess;
        HttpResponseMessage request = PictureGuessing.client.GetAsync("http://77.244.251.110:81/api/games/" + GuessReq).Result;
        request.EnsureSuccessStatusCode();

        string answer = request.Content.ReadAsStringAsync().Result;

        if (answer == "true")
            RoundWon();
        else
            ChangeGuessColor();
    }
    catch (HttpRequestException e)
    {
        MessageBox.Show("\nException Caught!");
        MessageBox.Show("Message :{0} ", e.Message);
    }


    return;
}
```

Sends the answer to the server that returns true or false.

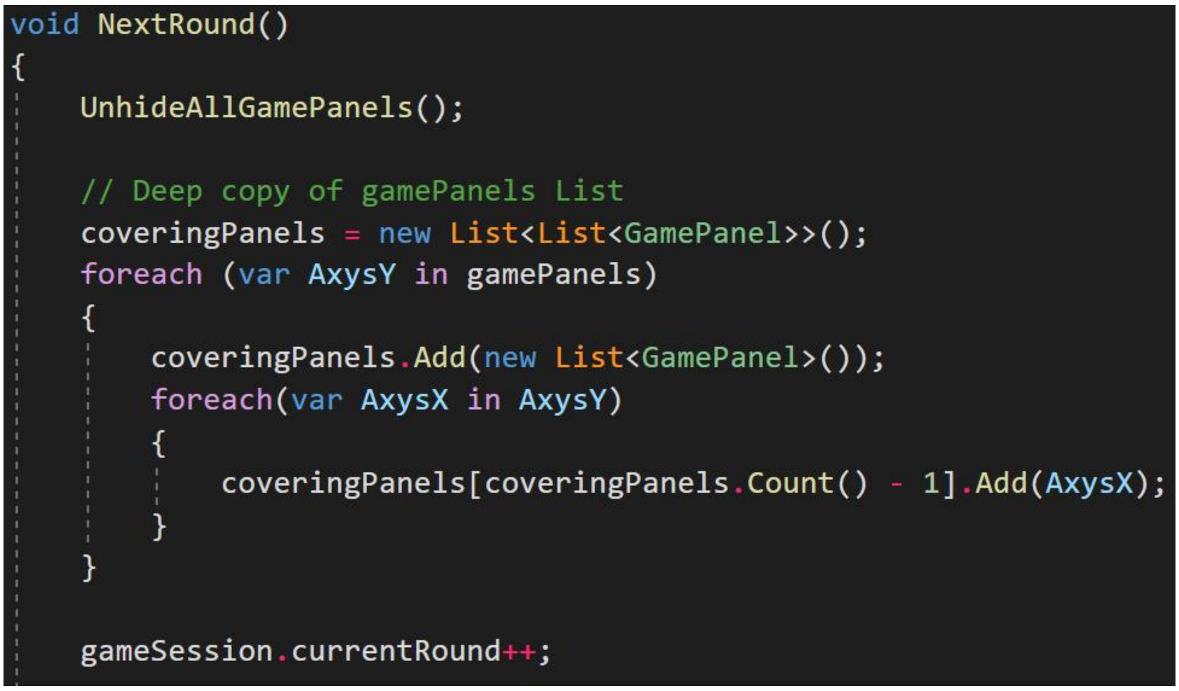
5.6.7 RoundWon/RoundLost



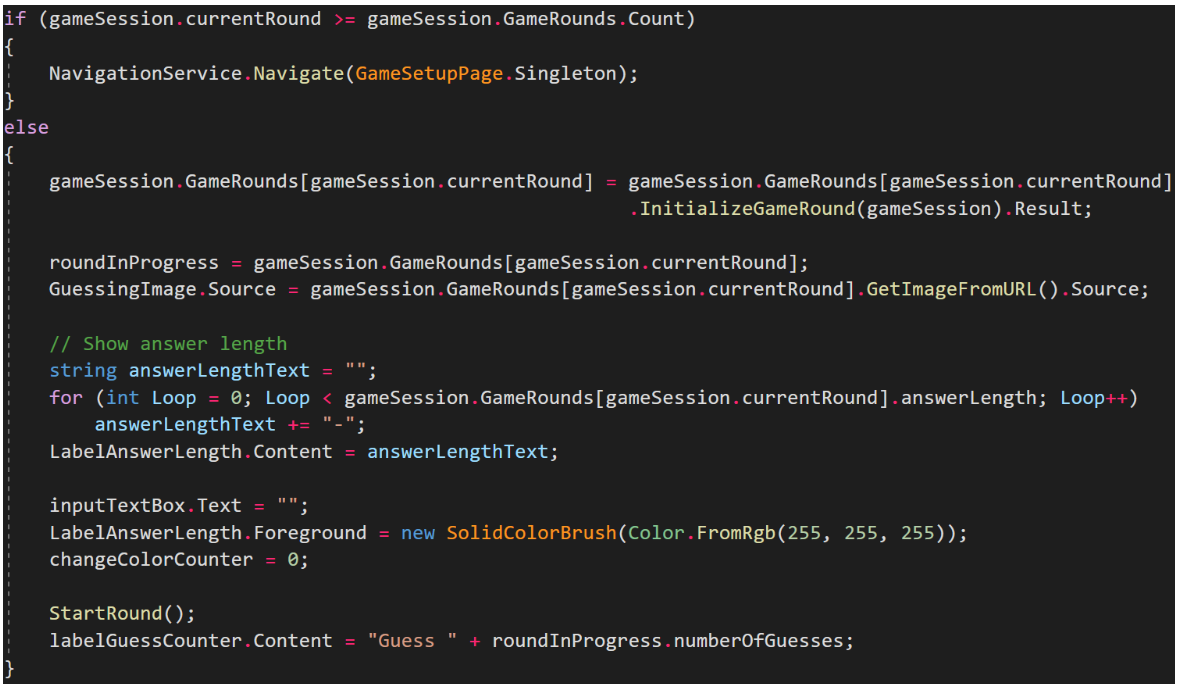
```
async void RoundWon()
{
    roundInProgress.bIsWon = true;

    timer.Stop();

    LabelAnswerLength.Foreground = new SolidColorBrush(Color.FromRgb(0, 200, 0));
    LabelAnswerLength.Content = guessLog.Text.Split('\n').Last();

    // Show correct image
    HideAllGamePanels();
    await Task.Delay(2000);

    NextRound();
}

1 reference
async void RoundLost()
{
    timer.Stop();

    timeSpan = TimeSpan.Zero;
    labelTimer.Foreground = new SolidColorBrush(Color.FromRgb(200, 0, 0));

    HideAllGamePanels();
    await Task.Delay(2000);

    NextRound();
}
```

These make sure that the rounds are ended correctly and show the player their result.

5.6.8 NextRound



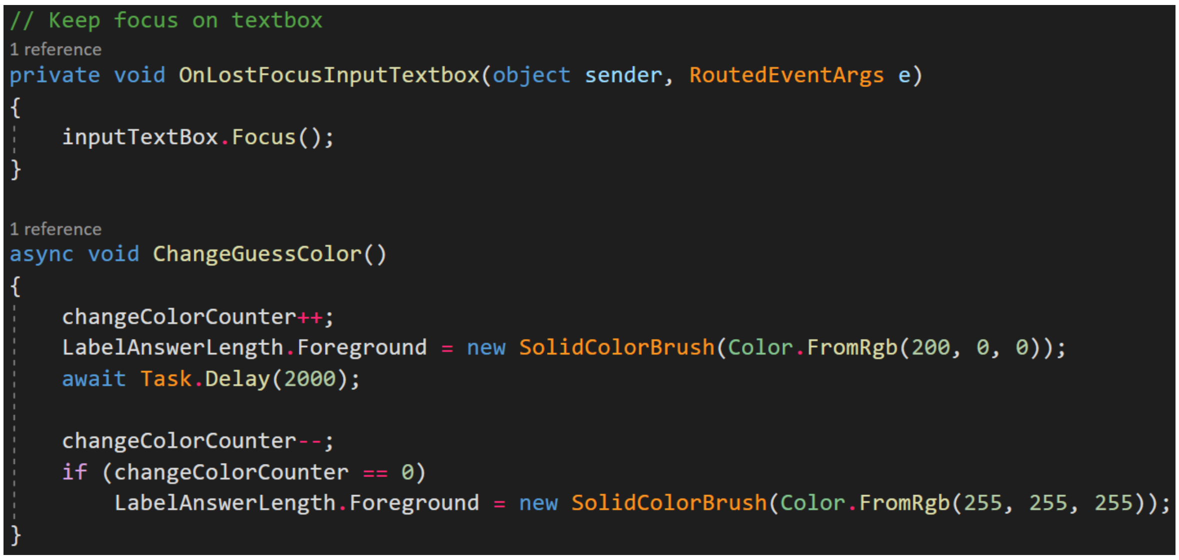
```
void NextRound()
{
    UnhideAllGamePanels();

    // Deep copy of gamePanels List
    coveringPanels = new List<List<GamePanel>>();
    foreach (var AxysY in gamePanels)
    {
        coveringPanels.Add(new List<GamePanel>());
        foreach(var AxysX in AxysY)
        {
            coveringPanels[coveringPanels.Count() - 1].Add(AxysX);
        }
    }

    gameSession.currentRound++;
```



```
if (gameSession.currentRound >= gameSession.GameRounds.Count)
{
    NavigationService.Navigate(GameSetupPage.Singleton);
}
else
{
    gameSession.GameRounds[gameSession.currentRound] = gameSession.GameRounds[gameSession.currentRound]
                                                        .InitializeGameRound(gameSession).Result;

    roundInProgress = gameSession.GameRounds[gameSession.currentRound];
    GuessingImage.Source = gameSession.GameRounds[gameSession.currentRound].GetImageFromURL().Source;

    // Show answer length
    string answerLengthText = "";
    for (int Loop = 0; Loop < gameSession.GameRounds[gameSession.currentRound].answerLength; Loop++)
        answerLengthText += "-";
    LabelAnswerLength.Content = answerLengthText;

    inputTextBox.Text = "";
    LabelAnswerLength.Foreground = new SolidColorBrush(Color.FromRgb(255, 255, 255));
    changeColorCounter = 0;

    StartRound();
    labelGuessCounter.Content = "Guess " + roundInProgress.numberOfGuesses;
}
```

Makes sure that the next round is started correctly and all of the statistics are stored from the previous round.

5.6.9 OnLostFocusInputTextBox



```
// Keep focus on textbox
1 reference
private void OnLostFocusInputTextbox(object sender, RoutedEventArgs e)
{
    inputTextBox.Focus();
}

1 reference
async void ChangeGuessColor()
{
    changeColorCounter++;
    LabelAnswerLength.Foreground = new SolidColorBrush(Color.FromRgb(200, 0, 0));
    await Task.Delay(2000);

    changeColorCounter--;
    if (changeColorCounter == 0)
        LabelAnswerLength.Foreground = new SolidColorBrush(Color.FromRgb(255, 255, 255));
}
```

Forces player to write into the text box.

5.6.10 ChangeGuessColor

If the player guesses incorrectly the color of the hint will change to red and then after the delay changes back to white.

6 Project evaluation

The project is in our eyes successful as it delivered what we have agreed on at the beginning.

Before connecting to the database the front-end took us roughly five months to make which included the whole working local application.

After connecting to the database the finishing touches took us another two months in which we mostly focused on optimizing the data stream between the client and the server. Also in this time we have decided on a color palette for our UI and changed to the dark theme.

We have been using Discord as our main communication channel between our team but after a few months into the project, the channel for communication between our two teams went into radio silence. As we did not think we need communication every week we have been working separately with the Austria team for over a month. This lead to an unfortunate situation where we had completely different takes on how the database should work. After some arguments, we have settled on a solution that suited both sides. This caused us probably a week or two to bring the database to a working state.

We were using Trello mainly for informing us and the other team of our advancements and issues we had encountered. The main problem with using Trello was that we have chosen big epics that were not very descriptive and sometimes did not even cover all of the work we were doing. For the next project, we would most likely divide these epics into smaller chunks that would be more descriptive and we would also assign only one person per task.

Miro was our most used tool (excluding Visual Studio) mainly because of its easy visual representation of our ideas. Miro was used mostly at the start of the project as we needed to agree on the main attributes of the game such as communication between client and server, the flow of the game, and the logical function of the game.

We had no significant issue using Visual Studio, Adobe XD nor Git.

7 Conclusion

As stated before we perceive this project as a success. As a team, we had learned a lot about cooperation inside a small team and also how to communicate our ideas with each other.

Inside our team, we had effortless communication as we already knew each other and knew what to expect from one and other. Communication with the other team was more difficult as we never had a project where we had to communicate with the other team that is on top speaking a different language. In the end, we think we did great as we spoke to the Austrian team frequently and clearly communicated our ideas and improvements.

We learned a lot of new skills using online tools for teamwork and organization of tasks.

Visual Studio was a great development environment for the Windows Presentation Platform. We learned advanced procedures in C# and also learned how to work with XAML mainly because of the custom buttons we made and the grid inside the Game page.

The teachers were very helpful throughout the project.

Mr. Skalka learned us the basics of C# and XAML in WPF. Mr. Mašek always tried to answer all of our questions and organized all of the online meetings with Austrian students. All around we were very pleased with the way our teachers approached the project.

In conclusion, we view this opportunity to create this project as a great experience that will help us in our journey through the next years of school or even in future jobs.

Attachments

Picture list

URLs

C4PE

App repository

Application repository (on GitHub): http://bit.ly/C4PE_GitHub

C4PE

Miro Board

Miro Board: http://bit.ly/C4PE_Miro