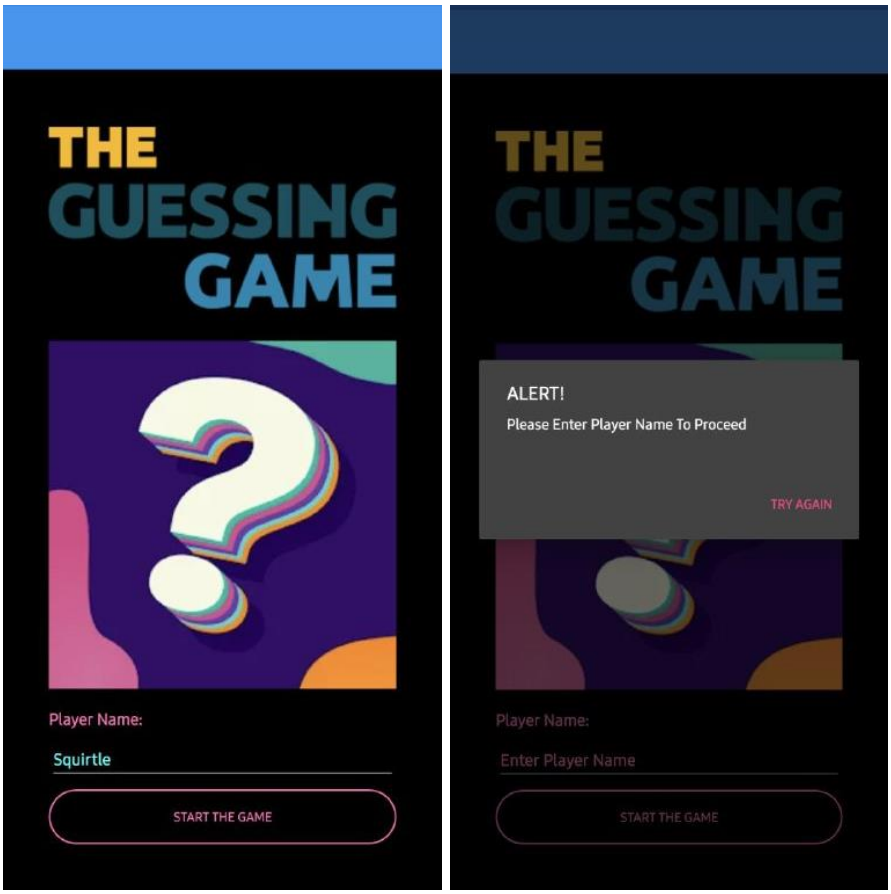
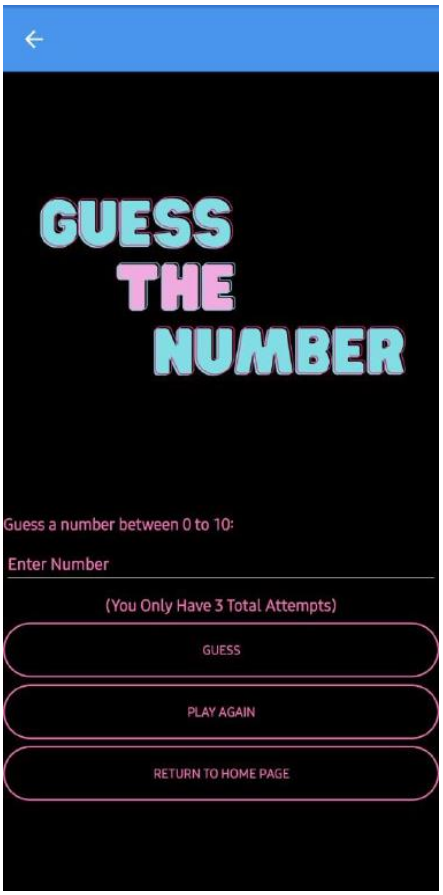
Table of Content


THE
GUESSING
GAME
Player Name:
Squirtle
START THE GAME
THE
GUESSING
GAME
ALERT!
Please Enter Player Name To Proceed
TRY AGAIN
Player Name:
Enter Player Name
START THE GAME


GUESS
THE
NUMBER
Guess a number between 0 to 10:
Enter Number
(You Only Have 3 Total Attempts)
GUESS
PLAY AGAIN
RETURN TO HOME PAGE

## 1.1 Introduction

In this assignment, we are tasked with creating an application called the "Number Guessing Game" on Visual Studio using Xamarin Form. This application can be applicable on Android or IOS. In this application, the program will first prompt users to enter their name. After users have entered their name and click on the “START THE GAME” button, users will be redirected to another page which is where the game starts.

On this page, users will be able to see a wording that displays “Guess a number between 0 - 10:”. Users then will be able to enter a number of their choice from 0 to 10. If the number entered is lower than the generated number, there will be a popup that displays “Guess Again But This Time Guess Higher ” and if the number entered is higher than the generated number, the popup will display “Guess Again But This Time Guess Lower”. In a case where the user entered the same number as the generated number, the user will be able to see a popup that displays an alert popup that is titled as “HOORAY!!!” . Lastly, users will have the choice to return to the home page or play again.

## 2.1 Background of problem

The problems we faced in completing this assignment was mainly not having sufficient knowledge to code it all at once and not familiar with the commands and language used in Xamarin Form and C#. This is because we are still new to this language . To resolve this problem, we had to self learn by watching numerous tutorials and videos to fully understand how to use Xamarin Forms and the functions available in it as well as reading through websites such as geeksforgeeks.com .

Secondly, some of our group member's Xamarin platform is not functioning properly due to some unforeseen circumstances. Which took him a few weeks of going through a couple of reinstallations to resolve this problem.

Lastly, learning how to make the buttons functionable for example navigating to a different page using buttons. In our program we have a total of 4 buttons which we took quite some time to learn how to validate the buttons on the c# page to be able to display the correct output.

## 3.1 Research question

- How to capture user input and save it to a variable?

- How to redirect to a new page?

- How to generate a random number to compare with the number user entered?

- How to display the appropriate message when the user inputs the correct and wrong numbers?
- How to let the user enter the number again if the user guessed the wrong number?

- How to make the UI neat and pleasing to increase User Experience (UX) ?

## 4.1 Objective

- To create a functional game application that works without any bugs and runs smoothly on Android and IOS devices using Xamarin Forms.

- Aim to provide the player with the best gaming experience by making the game interface UI neat and appealing.

- Make sure application can compare user input and computer generated number and display appropriate message accordingly

- Make sure application can be run successfully on Android and IOS devices

- Aims to capture user input successfully and display the appropriate messages

- The application can prompt the correct output

## 5.1 Scope & Limitation

### 5.1.1 Scope

- The application is able to compare the number that user inputs with the generated number, and display appropriate messages depending on the whether the user input is correct or not
- The application can prompt for user inputs and save them as variables
- The application can successfully redirect users to a new page upon clicking the specific buttons
- The application allows users to replay the game upon clicking the “Play Again” Button
- The application allows users to return to the homepage from the game page, to allow users to edit their Player Name.

### 5.1.2 Limitation

- A platform such as Microsoft Visual Studio must be installed in order for the code to run on it successfully for the application to load on the emulator.
- When the user submits an empty entry form in the main page, proper alert will be displayed but in the game page, users won't be able to submit empty entry forms as well as forms with input other than numbers. The application exits automatically whenever invalid inputs are entered.

## 6.1 Methodology

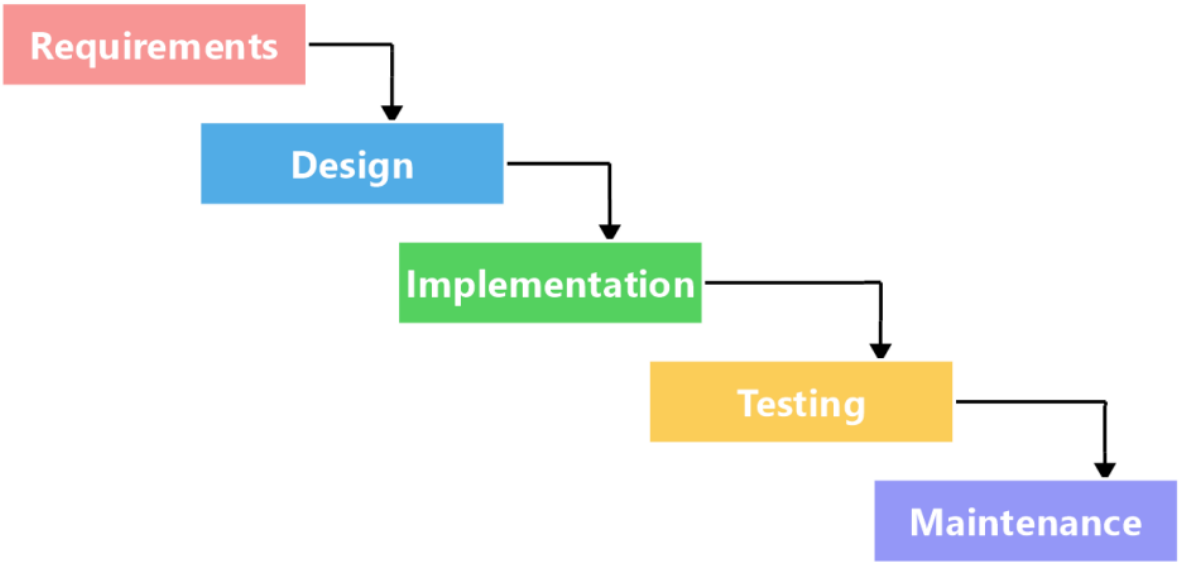



*Figure 1: Waterfall Methodology*

For this assignment, we chose to follow the waterfall methodology in order to complete the assignment. The Waterfall methodology, also known as the Waterfall model, is a sequential development process that flows like a waterfall through all phases of a project (for example, analysis, design, development, and testing), with each phase completely wrapping up before moving on to the next.

The Waterfall method's success is determined by the amount and quality of work done on the front end, including the user interface, user stories, and all feature variations and outcomes. With the majority of the research completed ahead of time, estimates of the time required for each requirement are more accurate, allowing for a more predictable release date.

### 6.1.1 Requirements

The first phase of the waterfall model is Requirements. In this stage, we read through the assignment question paper given and jot down the vital points that need to be included in the application. Below are the requirements of the Number Guessing Game Application :

- Entry form with proper placeholder to prompt users for their name
- Entry form with proper placeholder to prompt users to enter a number from 0 to 10
- Button that navigates users from Homepage to the Game page
- Button that validates user's guessed answer and display proper message
- Button that allows user to start over the game

After we review all the requirements needed for the application, we sit together and discuss how we want our User Interface (UI) to look like, which will be further explained on the next stage.

### 6.1.2 Design

Moving on to the designing phase, we begin by sketching our initial sketch on a piece of paper. We then go over the details and functions we would like to include in our application.

Next, we created a flow chart of our application. The flowchart visually displays how the application's algorithm works. This further helps with our understanding of how the application will turn out.

Then, we use Figma to create a low-fidelity prototype of how our application should look. The purpose of developing this prototype is to ensure that each and every member of the group is fully aware of how the application's output should look like.

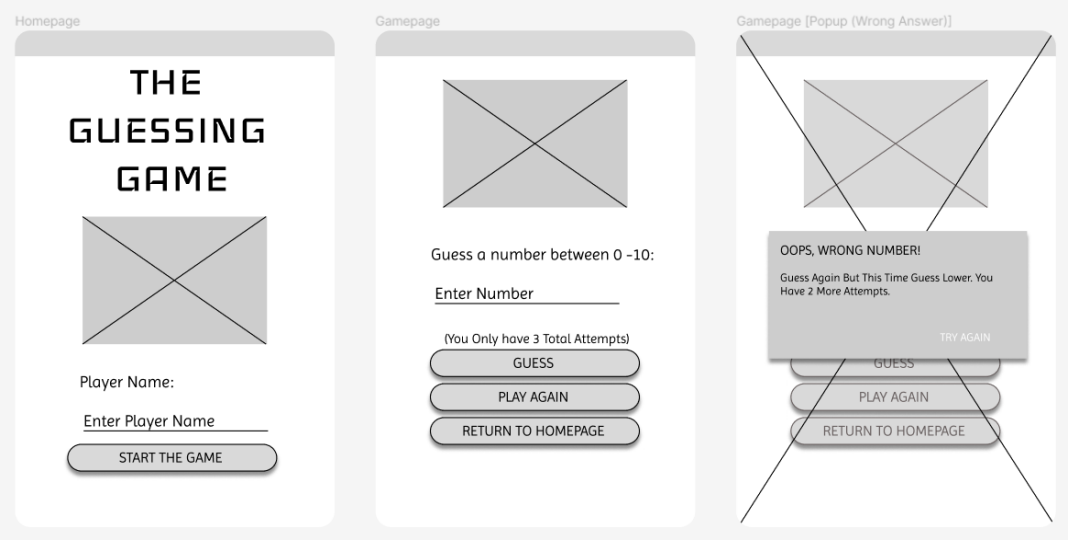



*Figure 2: Low Fidelity Protoype for Guessing Game Application*

After thorough discussion among our group members, we came to a conclusion that we would be sticking to a set of colors which are hot pink, black and aqua blue to apply to the minimalist theme we are going for which does not require that much color.

### 6.1.3 Implementation

In the third phase, which is the implementation phase, we divided the coding part among four of our members. We made sure that everyone understood the part they needed to do. This implementation phase lasted around 3 weeks as we had to do a lot of self learning by watching tutorials and understanding codes from websites such as geeksforgeeks.com to fully finish the assigned code.

### 6.1.4 Testing

After one of our group members compiled the code all together, we tested it out by using our own emulator and phones separately. By doing all the testing, we manage to capture some bugs in our application.

### 6.1.5 Maintenance

Lastly, in the maintenance phase, we manage to fix bugs together as if any issue arises, we would notify each other and find solutions to fix the bug by finding solutions online. We manage to fix some of our major bugs by adding validations in our code to lessen the error produced.

## 7.1 Implementation

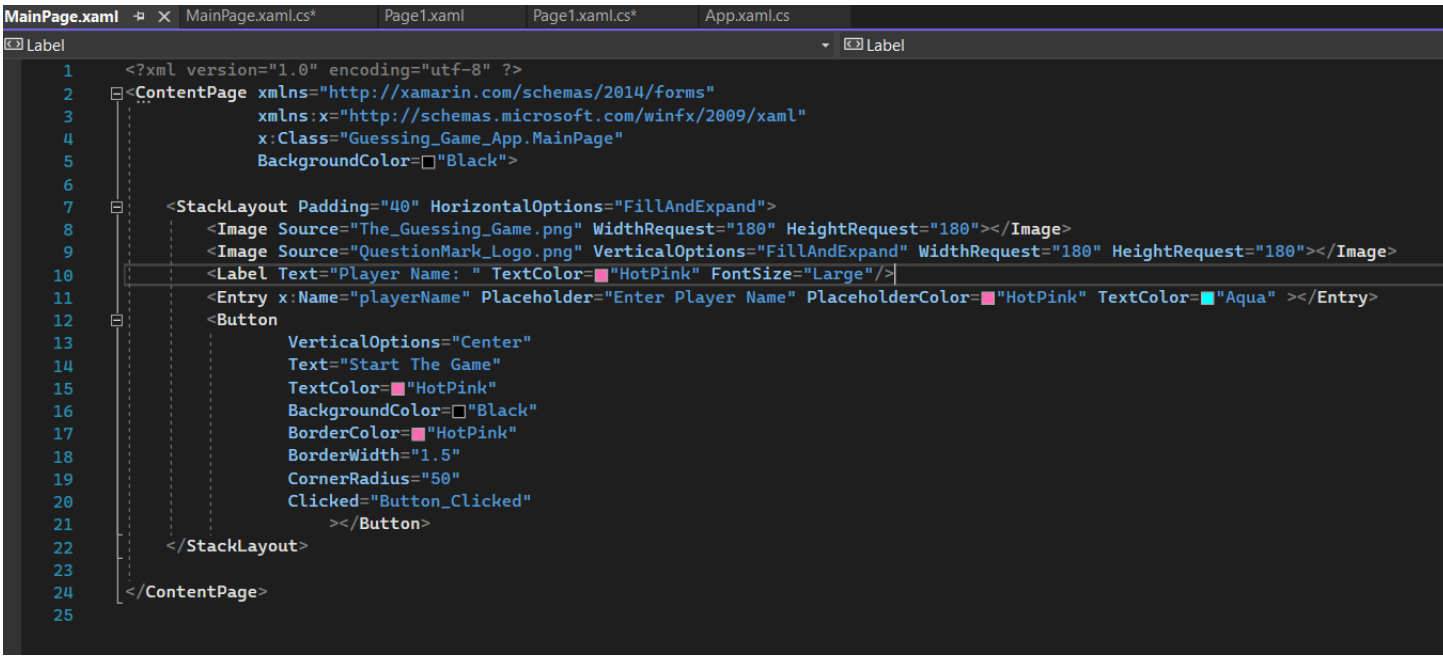



*Figure 7.1: MainPage.xaml*

As we can see in the figure above, this is the code for the layout of the first page of this application. 2 images were attached on top, whereas below the images we have proper labeling for the entry form and lastly we have a button that navigates users to “Page1”. This navigation part could be observed in Figure 7.2 and Figure 7.3.

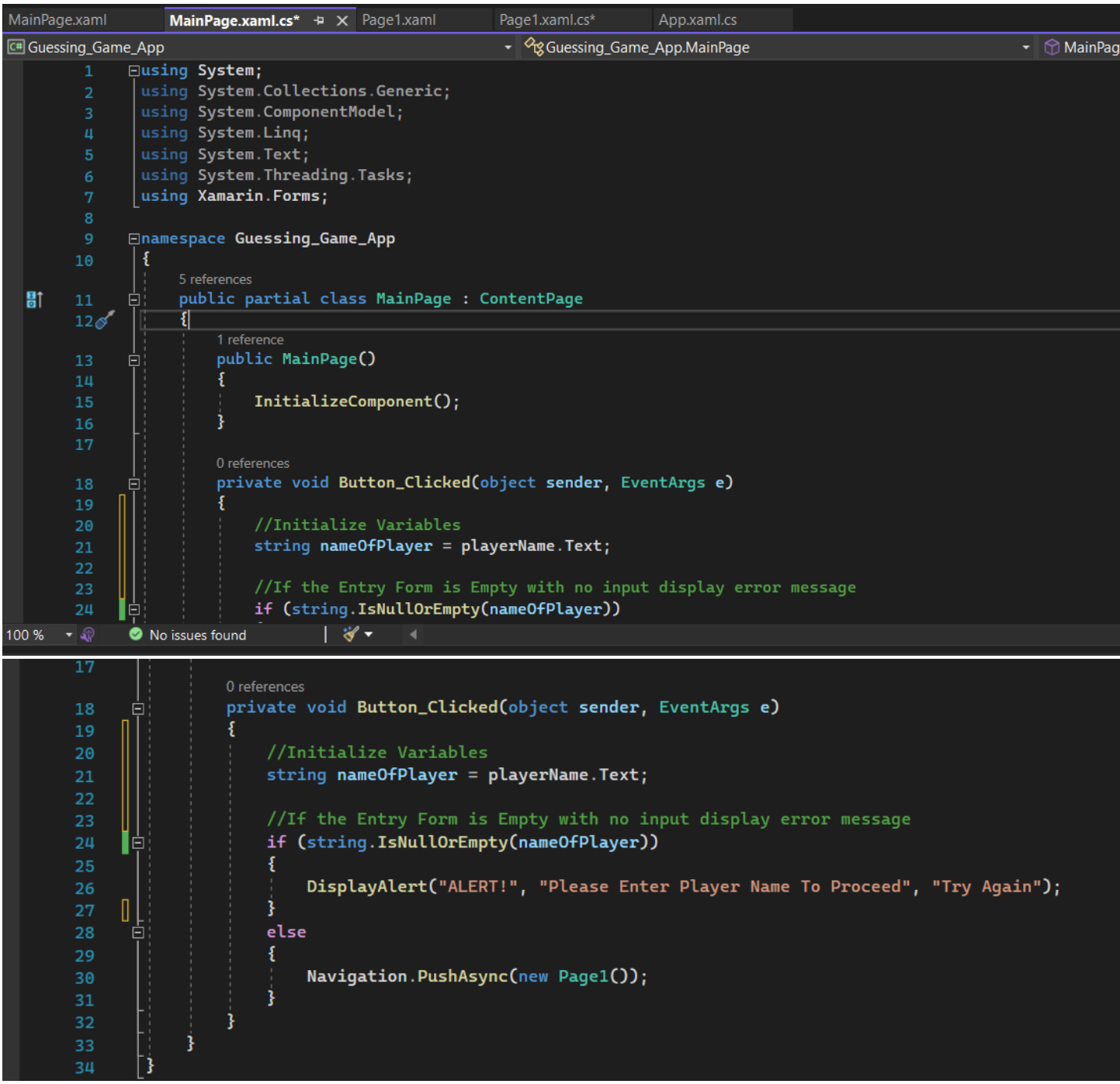



*Figure 7.2 & Figure 7.3: MainPage.xaml.cs*

This is the C# code that includes all the validation as well as navigation for the “MainPage” of our application. In the “Button_Clicked” function, variables were initialized and the if-else conditions were coded as well. The user will only be navigated to “Page1” if the entry form is filled up. If the entry form is left empty, an alert message will be displayed, asking the user to enter the entry form.

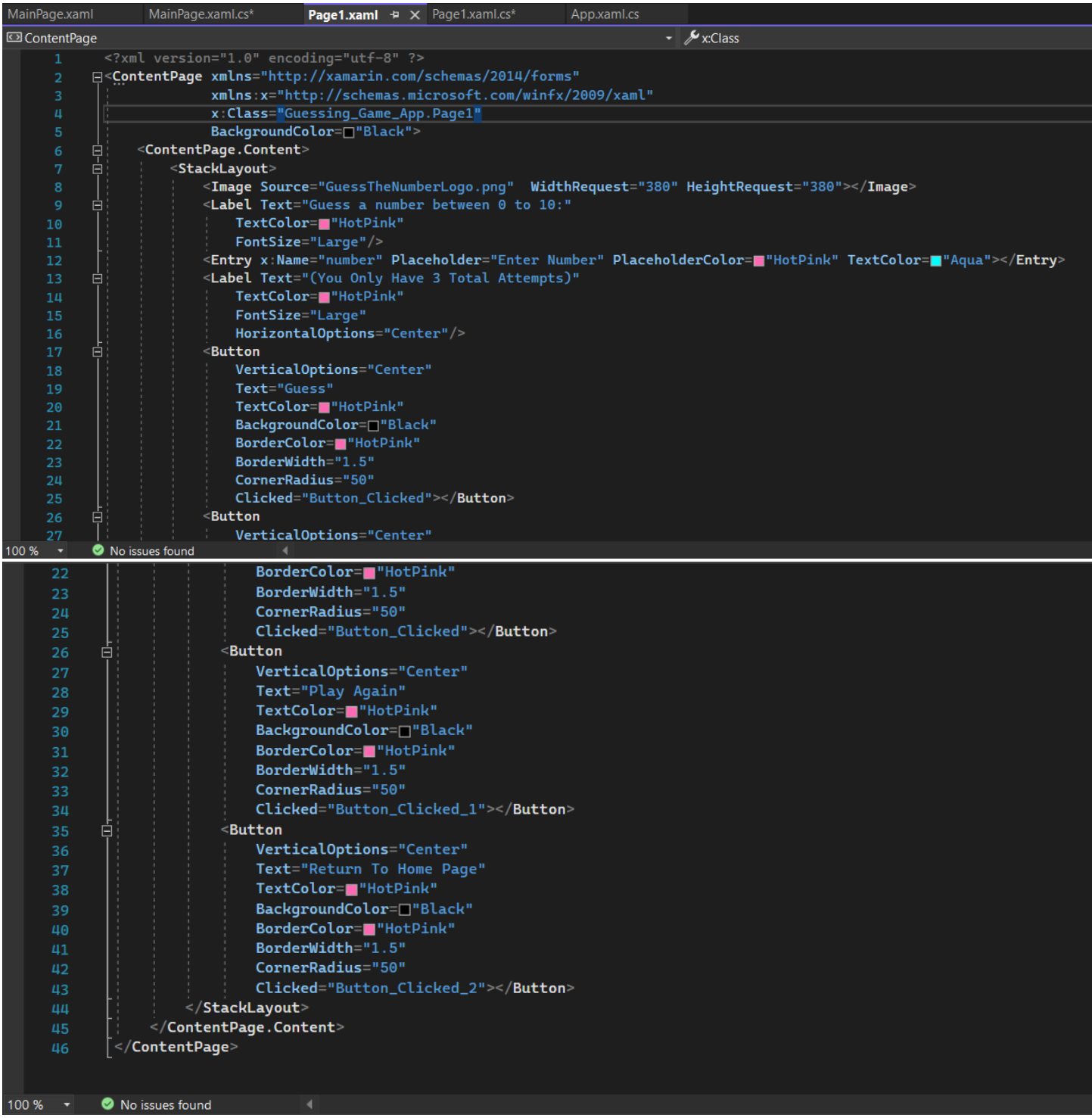
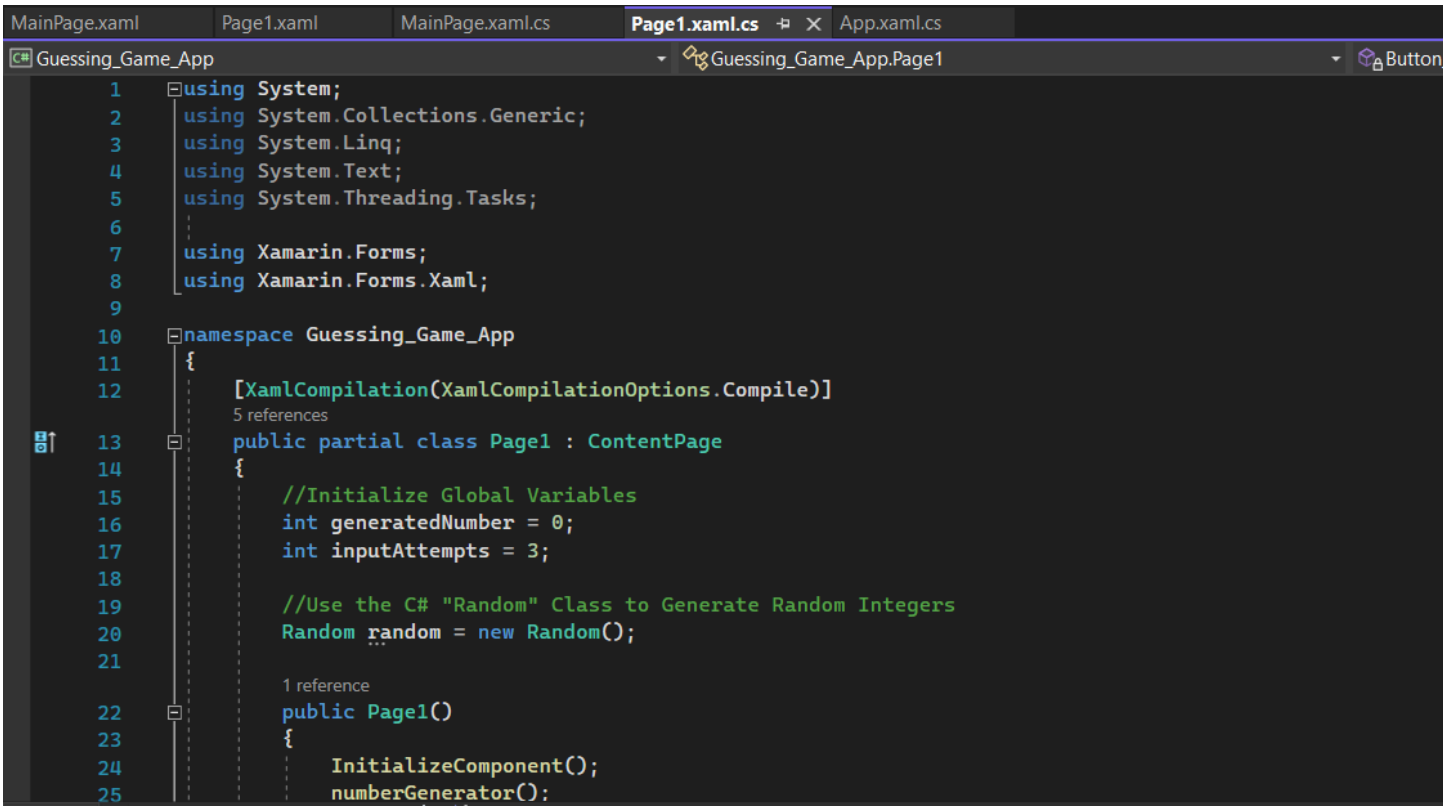
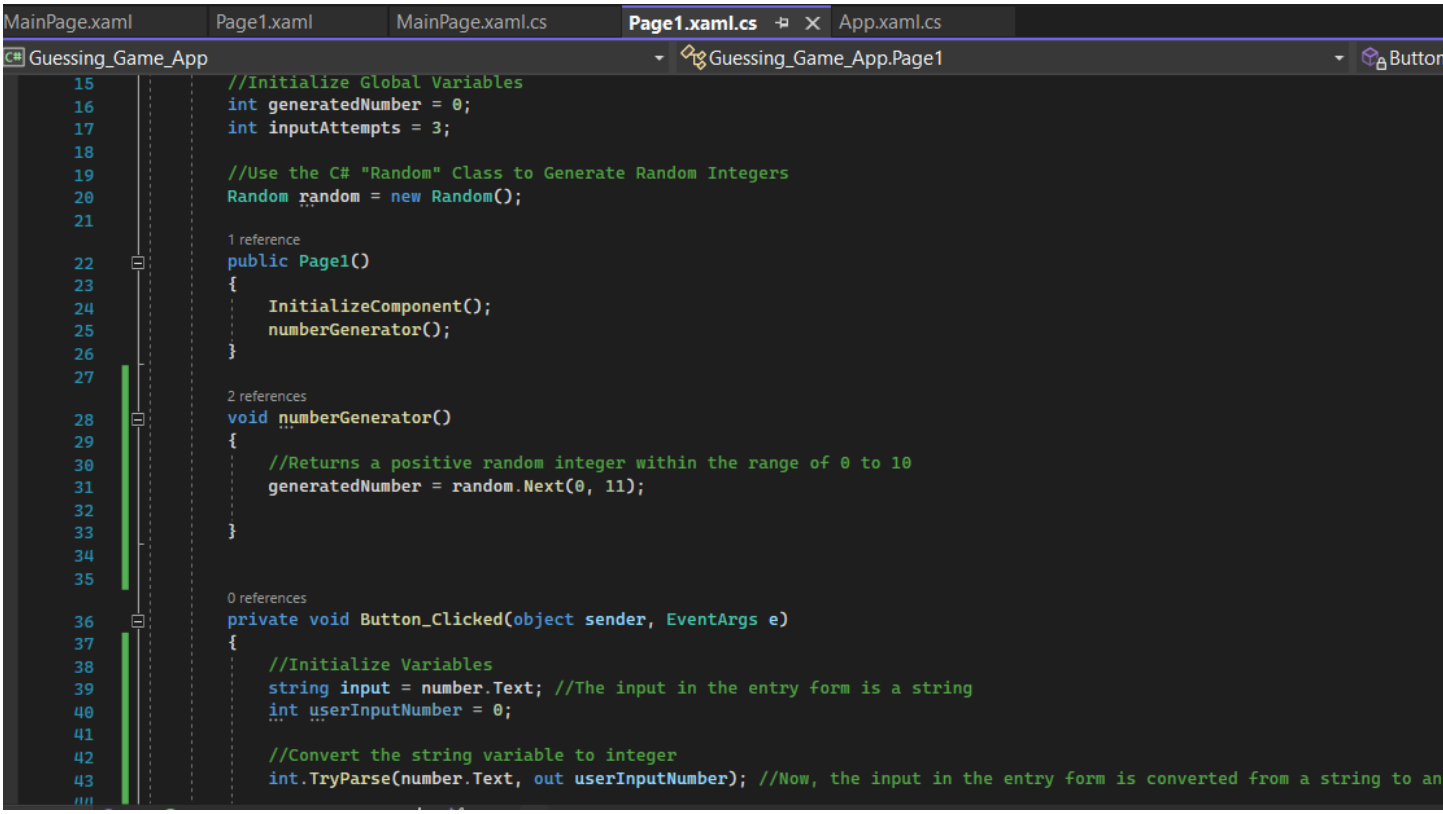
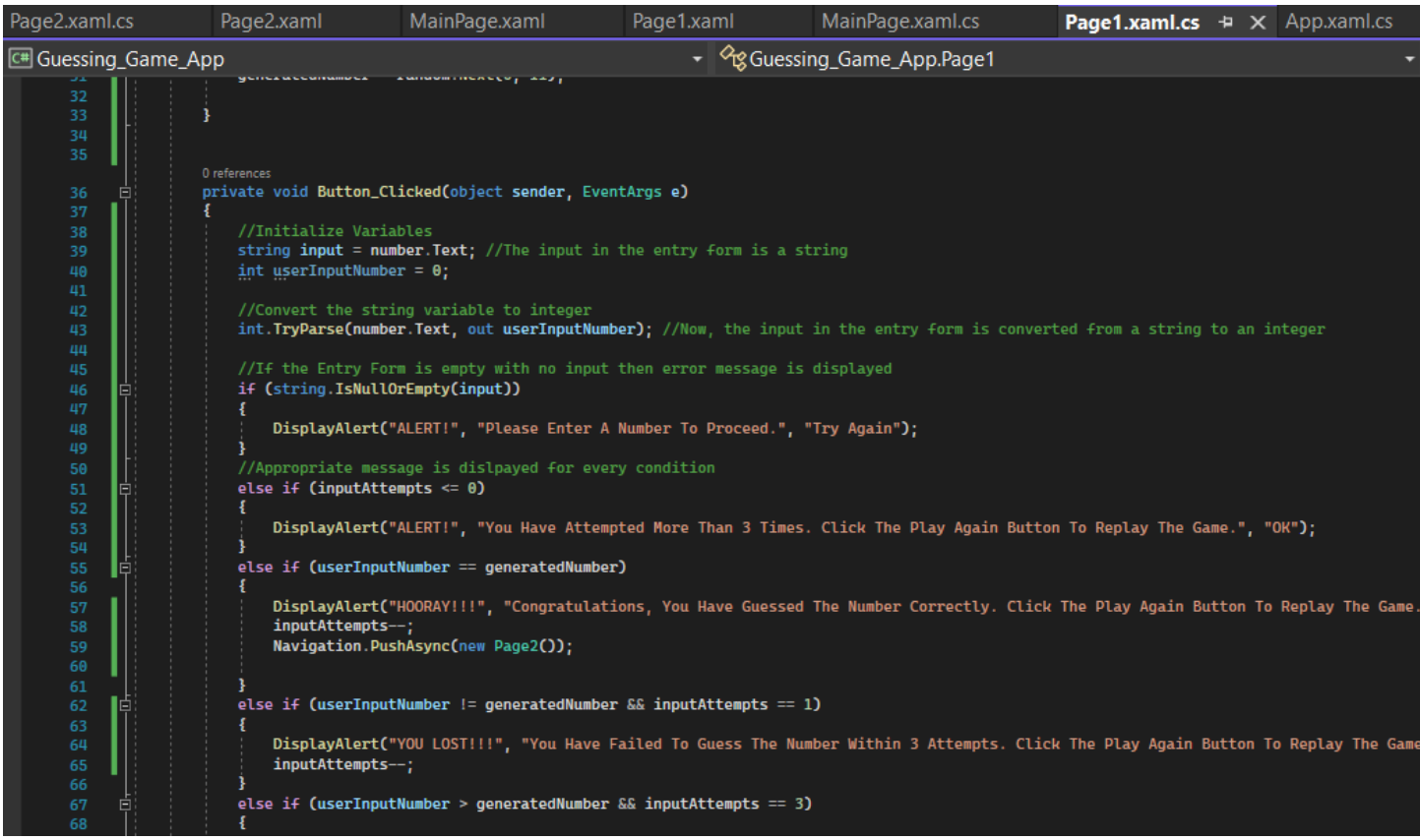


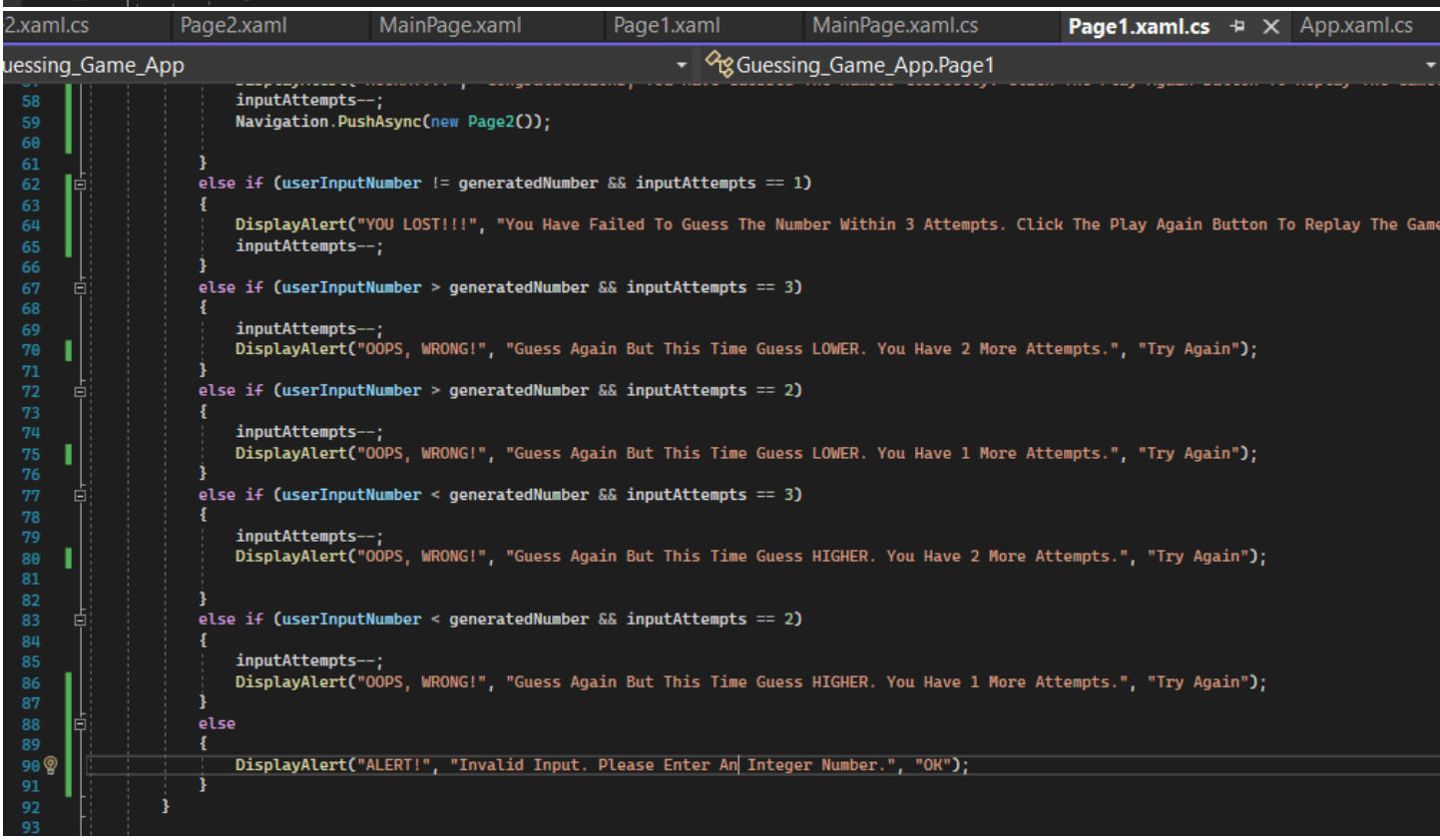

*Figure 7.3 & Figure 7.4: Page1.xaml*

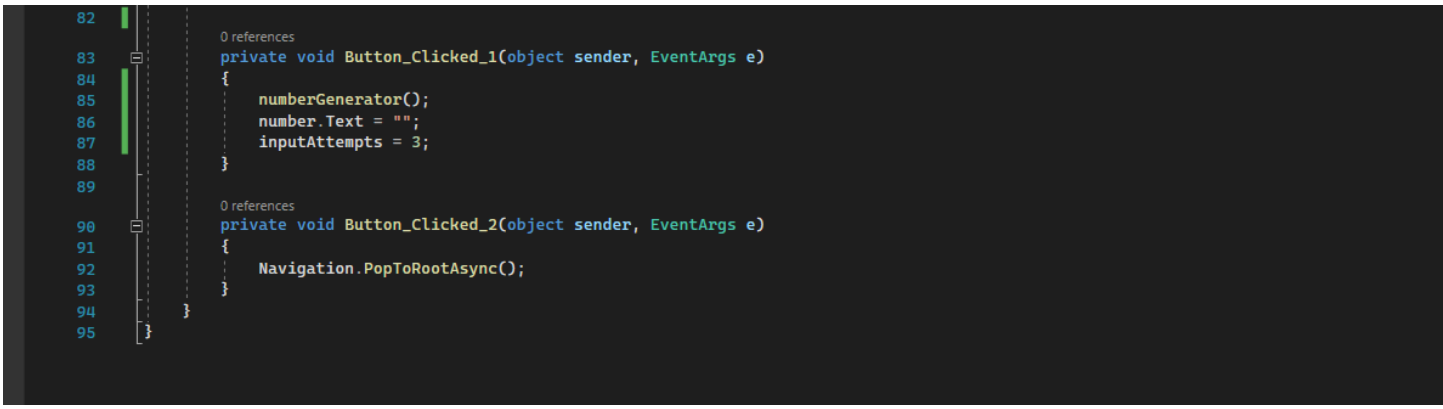
The code for the layout page of “Page1” could be seen from Figure 7.3 and Figure 7.4. An image, an entry form with proper labels and three buttons with different functions are included in it. The functions as well as the validations part of the code could be seen from Figure 7.5, Figure 7.6, Figure 7.7 and Figure 7.8.


MainPage.xaml
Page1.xaml
MainPage.xaml.cs
Page1.xaml.cs
App.xaml.cs
Guessing_Game_App
Guessing_Game_App.Page1
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace Guessing_Game_App
{
[XamlCompilation(XamlCompilationOptions.Compile)]
5 references
public partial class Page1 : ContentPage
{
//Initialize Global Variables
int generatedNumber = 0;
int inputAttempts = 3;
//Use the C# "Random" Class to Generate Random Integers
Random random = new Random();
1 reference
public Page1()
{
InitializeComponent();
numberGenerator();


MainPage.xaml
Page1.xaml
MainPage.xaml.cs
Page1.xaml.cs
App.xaml.cs
Guessing_Game_App
Guessing_Game_App.Page1
//Initialize Global Variables
int generatedNumber = 0;
int inputAttempts = 3;
//Use the C# "Random" Class to Generate Random Integers
Random random = new Random();
1 reference
public Page1()
{
InitializeComponent();
numberGenerator();
}
2 references
void numberGenerator()
{
//Returns a positive random integer within the range of 0 to 10
generatedNumber = random.Next(0, 11);
}
0 references
private void Button_Clicked(object sender, EventArgs e)
{
//Initialize Variables
string input = number.Text; //The input in the entry form is a string
int userInputNumber = 0;
//Convert the string variable to integer
int.TryParse(number.Text, out userInputNumber); //Now, the input in the entry form is converted from a string to an


Page2.xaml.cs
Page2.xaml
MainPage.xaml
Page1.xaml
MainPage.xaml.cs
Page1.xaml.cs
App.xaml.cs
Guessing_Game_App
Guessing_Game_App.Page1
0 references
private void Button_Clicked(object sender, EventArgs e)
{
//Initialize Variables
string input = number.Text; //The input in the entry form is a string
int userInputNumber = 0;
//Convert the string variable to integer
int.TryParse(number.Text, out userInputNumber); //Now, the input in the entry form is converted from a string to an integer
//If the Entry Form is empty with no input then error message is displayed
if (string.IsNullOrEmpty(input))
{
DisplayAlert("ALERT!", "Please Enter A Number To Proceed.", "Try Again");
}
//Appropriate message is dislpayed for every condition
else if (inputAttempts <= 0)
{
DisplayAlert("ALERT!", "You Have Attempted More Than 3 Times. Click The Play Again Button To Replay The Game.", "OK");
}
else if (userInputNumber == generatedNumber)
{
DisplayAlert("HOORAY!!!", "Congratulations, You Have Guessed The Number Correctly. Click The Play Again Button To Replay The Game.
inputAttempts--;
Navigation.PushAsync(new Page2());
}
else if (userInputNumber != generatedNumber && inputAttempts == 1)
{
DisplayAlert("YOU LOST!!!", "You Have Failed To Guess The Number Within 3 Attempts. Click The Play Again Button To Replay The Game
inputAttempts--;
}
else if (userInputNumber > generatedNumber && inputAttempts == 3)
{


2.xaml.cs
Page2.xaml
MainPage.xaml
Page1.xaml
MainPage.xaml.cs
Page1.xaml.cs
App.xaml.cs
uessing_Game_App
Guessing_Game_App.Page1
inputAttempts--;
Navigation.PushAsync(new Page2());
}
else if (userInputNumber != generatedNumber && inputAttempts == 1)
{
DisplayAlert("YOU LOST!!!", "You Have Failed To Guess The Number Within 3 Attempts. Click The Play Again Button To Replay The Game
inputAttempts--;
}
else if (userInputNumber > generatedNumber && inputAttempts == 3)
{
inputAttempts--;
DisplayAlert("OOPS, WRONG!", "Guess Again But This Time Guess LOWER. You Have 2 More Attempts.", "Try Again");
}
else if (userInputNumber > generatedNumber && inputAttempts == 2)
{
inputAttempts--;
DisplayAlert("OOPS, WRONG!", "Guess Again But This Time Guess LOWER. You Have 1 More Attempts.", "Try Again");
}
else if (userInputNumber < generatedNumber && inputAttempts == 3)
{
inputAttempts--;
DisplayAlert("OOPS, WRONG!", "Guess Again But This Time Guess HIGHER. You Have 2 More Attempts.", "Try Again");
}
else if (userInputNumber < generatedNumber && inputAttempts == 2)
{
inputAttempts--;
DisplayAlert("OOPS, WRONG!", "Guess Again But This Time Guess HIGHER. You Have 1 More Attempts.", "Try Again");
}
else
{
DisplayAlert("ALERT!", "Invalid Input. Please Enter An Integer Number.", "OK");
}
}



```
82
        0 references
83      private void Button_Clicked_1(object sender, EventArgs e)
84      {
85          numberGenerator();
86          number.Text = "";
87          inputAttempts = 3;
88      }
89
        0 references
90      private void Button_Clicked_2(object sender, EventArgs e)
91      {
92          Navigation.PopToRootAsync();
93      }
94  }
95  }
```

*Figure 7.5 - Figure 7.9: Page1.xaml.cs*

The backend code for the “Page1” file is displayed from Figure 7.5 to Figure 7.9. In Figure 7.5 and Figure 7.6, the global variables were initialized and we have used the C# in built “Random” Class to generate our random numbers for the game. In the function named “numberGenerator” as seen in Figure 7.6, we have set the range of the random numbers to be from 0 to 10 as per the assignment requirements.

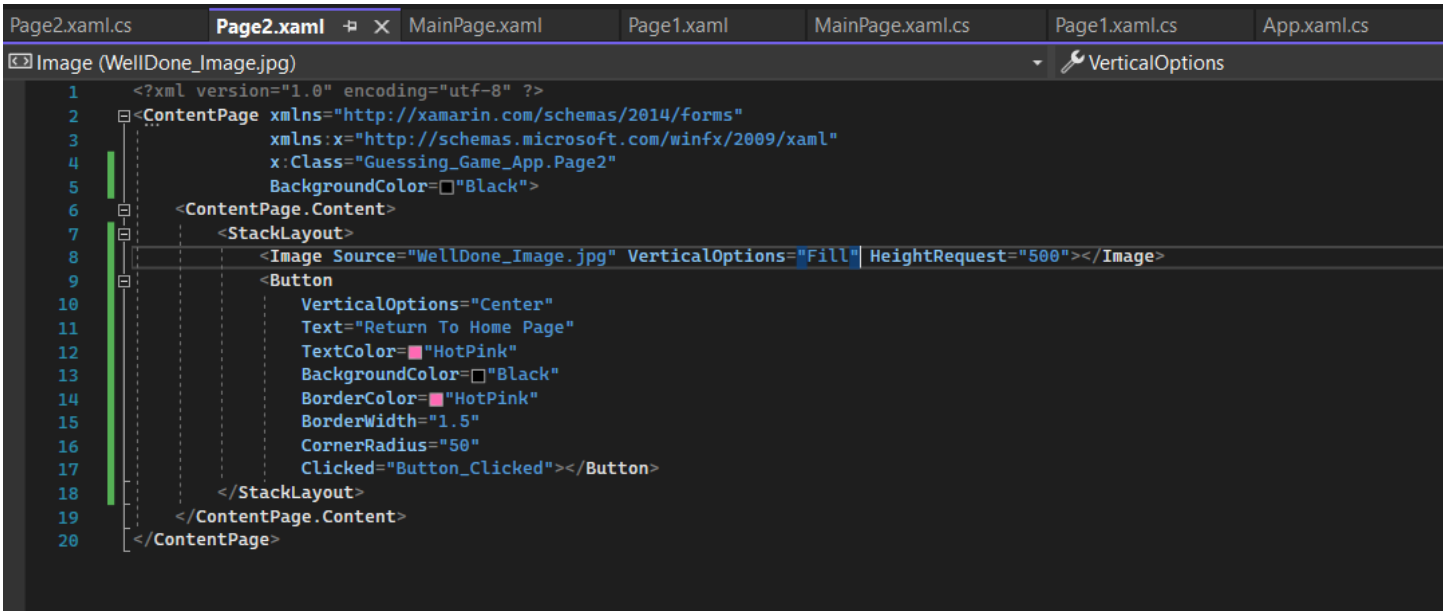
The functions of each button were coded out in the functions “Button_Clicked”, “Button_Clicked_1” and “Button_Clicked_2”, as seen in Figure 7.7, Figure 7.8 and Figure 7.9. The function “Button_Clicked” is the backend code that carries out the functions of the “GUESS” button, where as “Button_Clicked_1” and “Button_Clicked_2” are the backend code that carries out the functions of the “PLAY AGAIN” and “RETURN TO HOME PAGE” buttons.

In the “Button_Clicked” function, that can be seen in Figure 7.7 and Figure 7.8, we have initialized the necessary variables as well as validated every single outcome possible, making sure there is

an appropriate message displayed for each situation. This would be further discussed in the Discussion part below with attached pictures of the output.

Moreover, in the function “Button_Clicked_1”, we have made sure users could replay the game, having to guess a new number between 0 to 10, within 3 attempts. The moment users click on the “PLAY AGAIN” button, the “numberGenerator” function will generate a new number for the user to guess and the empty form will be emptied out for the user to enter their new number.

Lastly for the “Button_Clicked_2” function, the moment the user clicks on the “RETURN TO HOME PAGE” button, they will be redirected to the homepage, enabling them to edit their player name and start over the game.



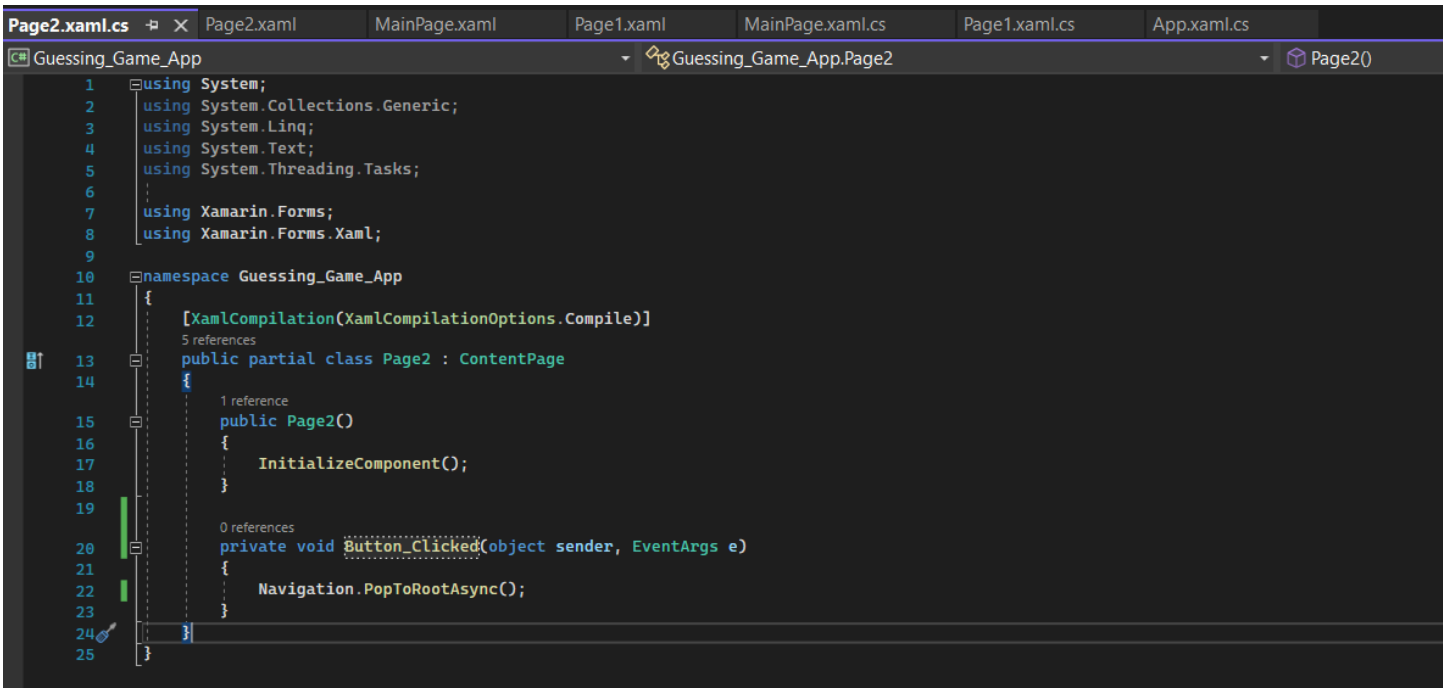



*Figure 7.10: Page2.xaml & Figure 7.11: Page2.xaml.cs*

Figure 7.10 displays the code of the Xamarin file fof “Page2” whereas Figure 7.11 displays the code of the C# file. If the user guessed the correct answer, then the user will be redirected to “Page2” from “Page1”. There will be an image attached on the page, as well as a button below the image that says “RETURN TO HOME PAGE”, which redirects the user to the homepage upon clicking.

## 8.1 Discussion

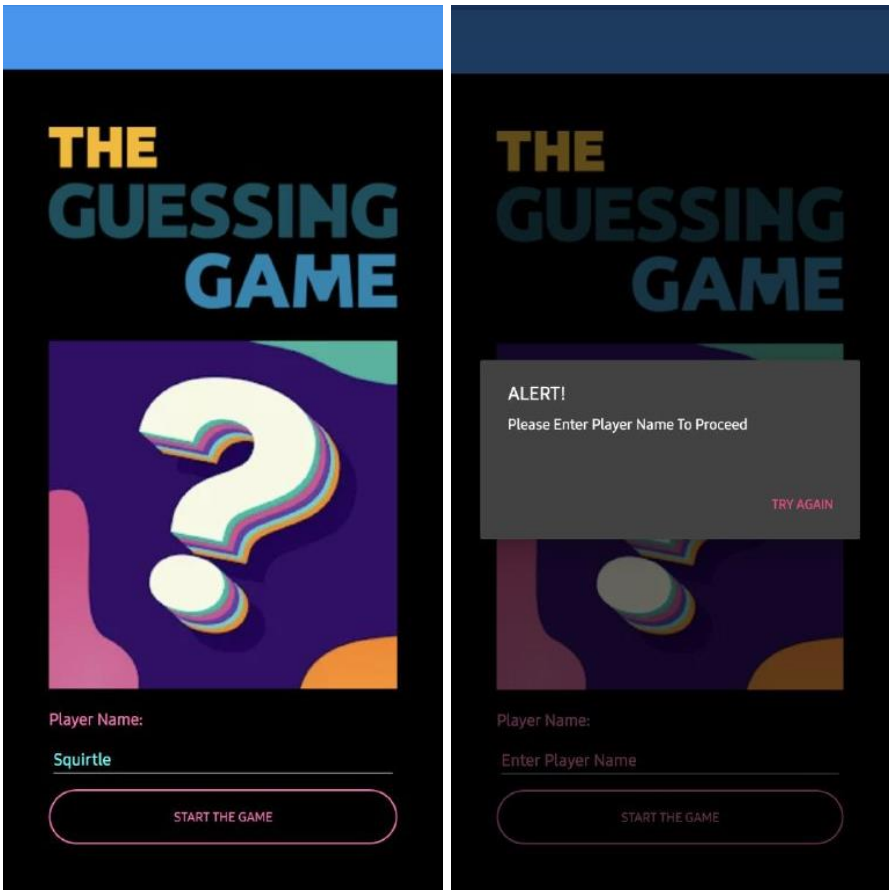



*Figure 5.1 : Main Page &*

*Figure 5.2 : Alert Popup*

Figure 5.1 shown above is the main page of the Guessing Game application. In this page, the program will prompt users to enter their name. Users will be able to interact with the application by entering their name at the given space. After users have input their name, they will be able to press the “START THE GAME” button. This button will navigate users to the Game Page where they will be able to start playing the game.

In Figure 5.2, in a case where users did not input their name and directly press the “START THE GAME” button, an Alert Popup will appear. This popup will appear to remind users to input their name. This entry form accepts alphanumeric characters as well as symbols, and is not limited to only alphabets.

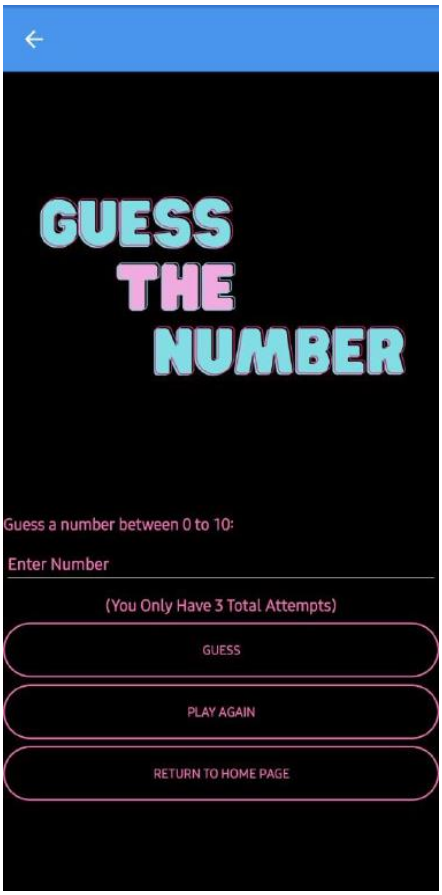
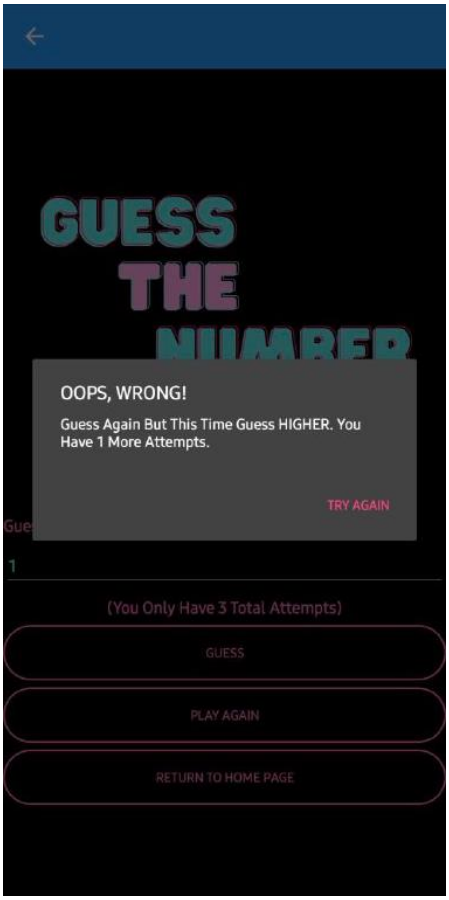



*Figure 5.3 : Game Page*

On this Game Page, users will be able to see a big picture which displays “GUESS THE NUMBER”. This picture adds color to the page which makes it look way more attractive. Below it, the game will prompt users to enter a number of their choice. Each user only has a total of 3 attempts. This entry form also has alphanumeric characters as well as symbols. As long as the wrong input is entered, whether it is a number, symbol or an alphabet, the users attempts will reduce and an appropriate message will be displayed.

A total of 3 buttons are available on this page. The “RETURN TO HOMEPAGE” button will redirect users back to the Main Page. Next, the “PLAY AGAIN” button will allow the users to replay the game with a new number generated for them to guess. Lastly, upon entering a number and pressing the “GUESS” button, users might end up in 4 different scenarios which can be seen in Figure 5.4 , Figure 5.5, Figure 5.6 and Figure 5.7.



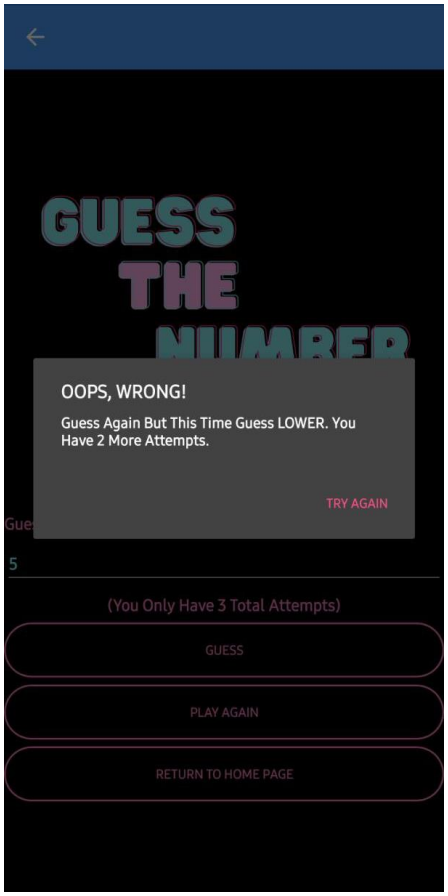



*Figure 5.4 : Popup (Guess Higher) & Figure 5.5 : Popup(Guess Lower)*

The popup shown above will appear if the number users entered is not similar to the number generated. The number entered might be lower or higher than the number generated. Users will receive a message depending on their own scenario. For example, if the number that users input is lower than the generated number, a popup such as shown in Figure 5.4 will be displayed and vice versa. Moreover, the number of attempts left will be also displayed in the popup, for example in Figure 5.5, the user still has 2 attempts left to guess the correct number.

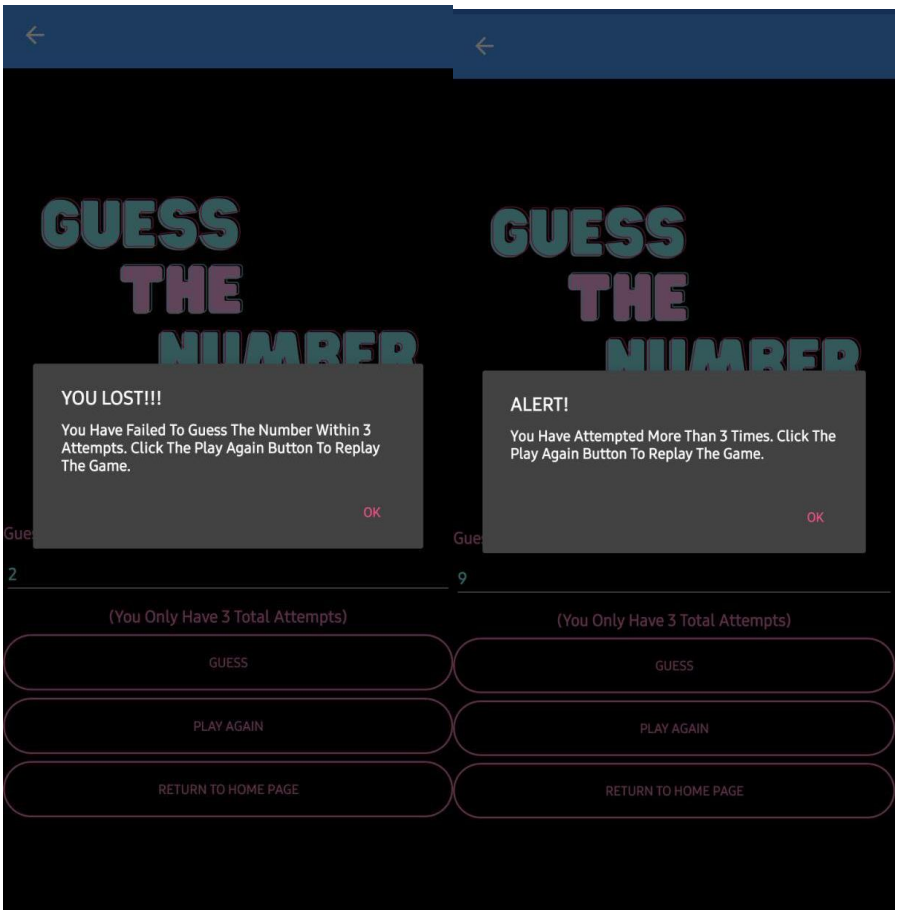



*Figure 5.6 & Figure 5.7 : Popup (Lost)*

This popup as shown on Figure 5.6 will appear if users are not able to guess the correct number within 3 tries. Upon pressing the “OK” button, users will be able to go back to the Game Page, enabling them to choose either from, replaying the game or returning to the home page. The popup in Figure 5.7 will be displayed whenever the user input attempts crosses 3 attempts. This is because the attempts have reached its limit and users cannot answer anymore as long as they have answered more than 3 times. If the user wants to replay the game, they should click either on the “PLAY AGAIN” button to replay the game without changing their Player Name, or the

“RETURN TO HOME PAGE” button where they could return to homepage and change their Player Name, then replay the game once again.

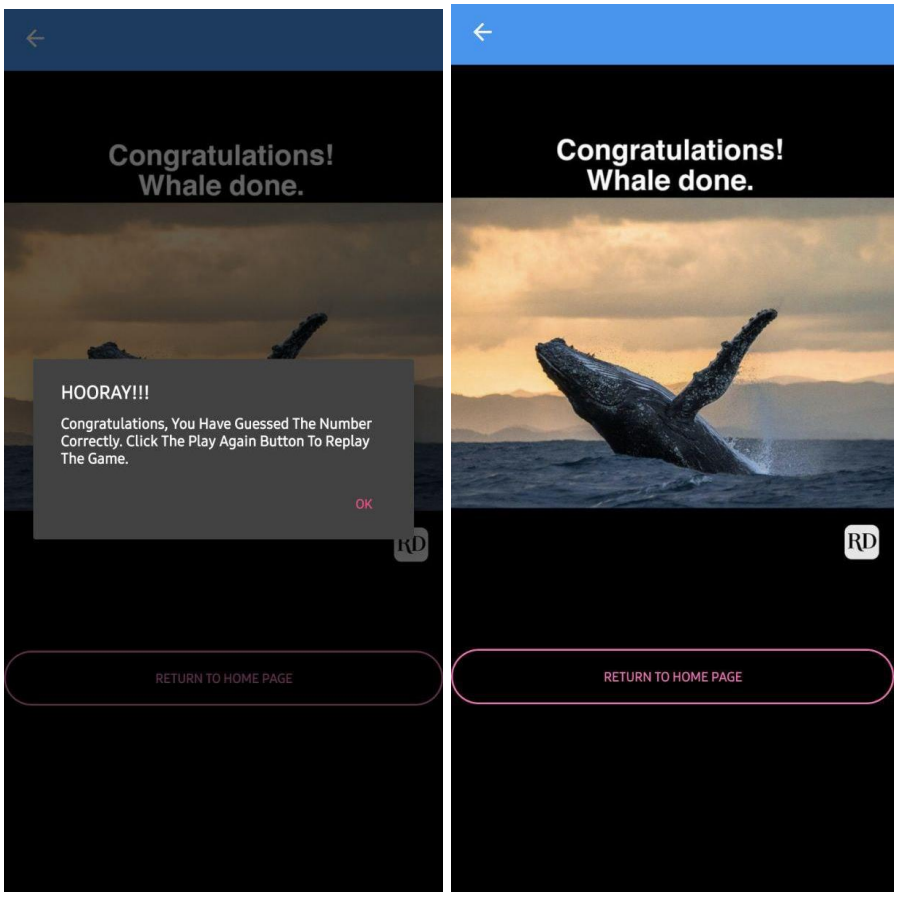



*Figure 5.8 & Figure 5.9 : Popup and Winners Page(Won)*

This congratulatory popup will appear when the user manages to successfully guess the number generated within 3 guesses. Furthermore, upon guessing the correct number, the users will be redirected to a new page (“Page2”) with an image containing congratulatory messages well as a button below the image that functions exactly as the “RETURN TO HOME PAGE” button in the Game Page (“Page1”).

- 