# Practical Project: Guess A Number

This is additional practical project and **it is not mandatory and it is not included in the final score**. The main purpose is to use gained knowledge in different type of problems and to improve your portfolio and GitHub skills.

Logo

Description automatically generated with low confidence

Today we will make the console game "Guess A Number". "Guess A Number" is a game, in which your opponent "**the computer**" chooses a **random** number between 1 and 100 and your task is to **guess** this number. After each number you enter, the computer will give you a **hint** of whether the number is **greater** or **less** than the number you selected until you guess the **correct** number:

Text

Description automatically generated Text

Description automatically generated with medium confidence

## Create GitHub Repository

We already have a GitHub account created, so we're moving directly to creating a new repository.

Create a **new repository** from: <https://github.com/new>. Choose a **meaningful name**, e. g. "GuessANumberByUsername" add a **short description** and make your repo **public**:

Graphical user interface, text, application

Description automatically generated

|  |  |
| --- | --- |
| Icon  Description automatically generated | Please choose **your own original and unique name** for your project!  Your GitHub profile should be **unique**, not the same as your colleagues'.  You can follow this tutorial, but you can also **make changes** and **implement your project different** from your colleagues. |

Also, **add a** README.md file and .gitignore **for Visual Studio**, as shown below:

Graphical user interface, text, application, email

Description automatically generated

In Git projects the .gitignore **file** specifies which files from your repo are not part of the source code and should be ignored (not uploaded in the GitHub repo). Typically in GitHub, we upload in the repo **only the source code** and we don't upload the compiled binaries and temp files.

Finally, **change the license** to "MIT", which is the most widely used open-source license, and click on the [Create] **button** to **create your repository**:

Graphical user interface, text, application, email

Description automatically generated

Now your **repository is created** and looks like this:

Graphical user interface, text, email, website

Description automatically generated

Now let's see how to **write the code** of our game.

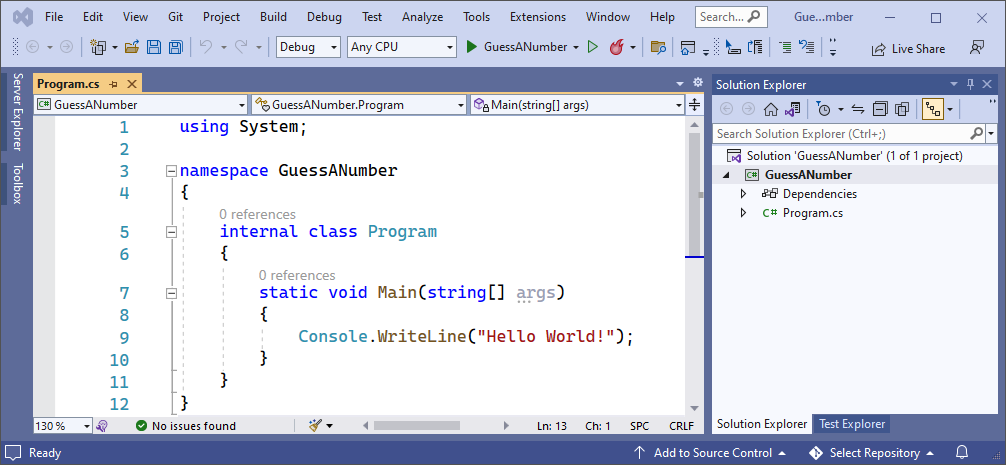
## Write the Game's Code

Let's create the game and play with it.

### Create a Visual Studio Project

First, we should **start Visual Studio** and **create a new C# console application**. Then, **choose an appropriate name** and a **place to save the project**. On the next screen, choose [.NET 3.1 (Long-term support)] and create the project.

Our project should be created and should look like this:



Before we continue, let's change the name of our main class – Program.cs to something more **meaningful**. You already know how to do this:

Graphical user interface, text, application

Description automatically generated

### Implement the Game Logic

Now let's start working on our project.

#### Read Player's Move

Create a variable from type "Random", which will help us **choose** a number **randomly** by using the method "Next()". We will use this **method** so each time the computer can choose a number **between** "**1 and 100**" **randomly**.



##### A little more information about **Random:**

.NET Core provides thousands of ready-to-use classes that are packaged into namespaces like the already known **System**. The **System** namespace contains fundamental classes, one of which is the **Random** class. It provides functionality to generate random numbers in C#. We will learn more about the **Random** class in the [Objects and Classes lesson](https://softuni.bg/trainings/3835/programming-fundamentals-september-2022#lesson-44422), but let's take a quick overall view of this class.

The line with code below creates a new object, which is an instance of the Random class. In this object will store the randomly generated number that we have to guess.

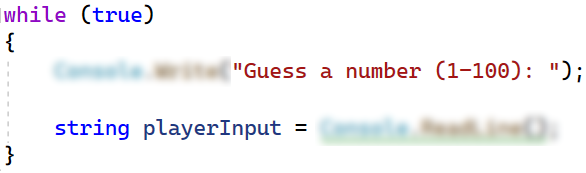


The following code returns a random number, using the **Next()** method. This is a method, provided by the **Random** class. By writing "**1, 101**" in the brackets, we indicate to the method that we want our randomly generated number to be in the range between **1** and **100**. You should note that the lower bound is inclusive and the upper bound is exclusive, that's why we have **101** as the second parameter of the **Next()** method.



You can read more about the **Random** class here <https://docs.microsoft.com/en-us/dotnet/api/system.random.next?view=net-6.0>.

Now write a while-loop to **iterate,** until the player guesses the computer's **random** number. Write on the console what the player should do and **read** his **input** **data**. You already know how to do that.



Now let's run the **app** in the console and check whether our current code **works** properly:

Graphical user interface

Description automatically generatedText

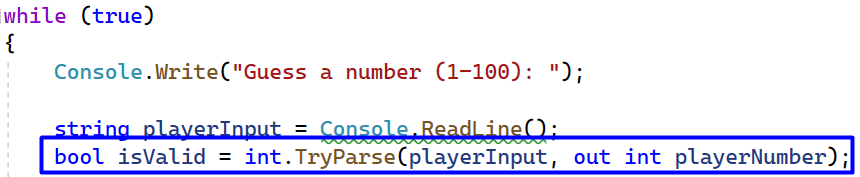
Description automatically generated

We can see that we have our text **written** on the console and we should be able to **read** the player’s input **repeatedly** because of our while-loop.

#### Check the Player's Input

Now **check** the player's input using the int.TryParse() method. It will review the input data and return us "true" or "false" depending on the data **submitted** by the player. If It's a **number** (**what we expect**) the method will return "true" otherwise "false".

As a **second** **parameter** of the method, we give the type and name of the **variable** in which the player's data will be **recorded** if he has submitted **correct** data. Do it as follow:



A little more information about int.TryParse(): <https://docs.microsoft.com/en-us/dotnet/api/system.int32.tryparse?view=net-6.0> .

Now we have created our Boolean variable "isValid" and we can write our if-else statements. First, we should check if the player's input data "**isValid**":



If data is valid write a nested if-else statement in which we will check all **three** possible cases.

First, if the player's number is **equal** to the computer's number that means the player **guessed** the computer's number, so you should **write** a message, and **stop** the application by using the keyword "break". Do it like this:

Graphical user interface, text, application, email, website

Description automatically generated

The other **two** cases are if the player's number is **higher** than the computer's number and the player’s number is **less** than the computer's number. Write the rest of the else-if statement by yourself:

Graphical user interface, text, application

Description automatically generated

Now let's run the **app** in the console and check whether our current code **works** properly:

Graphical user interface

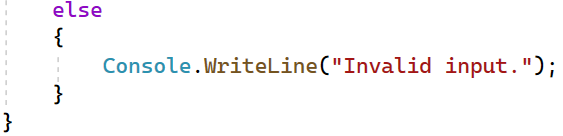
Description automatically generatedText

Description automatically generated with medium confidence

We can see that our application work's properly, but we are not finished yet.

#### Check for Invalid Input

Now what is left is to write else statement for the **final** case where the player's input is **invalid.** That's all it takes for the **game to work**.



Your entire code should be similar to the following:

Graphical user interface, text, application, email

Description automatically generated

#### Test the Application

After you **run it**, the game should look like this:

Graphical user interface, text

Description automatically generatedGraphical user interface, text

Description automatically generated

Text

Description automatically generatedText

Description automatically generated

Text

Description automatically generated

## Upload Your Project to Github

We already know how to clone our repository by using **Git** **Bash** or **TortoiseGit**.

### Use TortoiseGit (Option 1)

Use Git **clone** for cloning with TortoiseGit. Go to the desired directory, **right-click** on a blank space **anywhere** in the folder and click [Git Clone].

Graphical user interface, application

Description automatically generated

Now go to our newly created **repository** and copy the **repository's** **URL**, you should already know how to do this. The last thing that we should do is to open TortoiseGit and paste the **URL** and click [OK].

Graphical user interface, application

Description automatically generated

Your files from your GitHub repo will be downloaded to a **sub-folder** called as your project in GitHub, "**GuessANumberByPeter**" in our case.

Graphical user interface, text, application, email

Description automatically generated

When we open the cloned **repository sub-folder**, it should look like this:

Graphical user interface

Description automatically generated with low confidence

The next thing to do is to **add** the **project** to our **cloned** **repository**. You can move your C# source code files from your **old project folder** to your **new repo folder**. You can use “**Cut & Paste**”. It should look like this:

Graphical user interface, text, application

Description automatically generated

Now to **upload** our changes from our working project folder to GitHub.

We can use TortoiseGit's [GitCommit…]. Go to your project's folder, **right-click** on blank space anywhere in the folder and click [Git Commit -> "main"…].

Graphical user interface, text, application

Description automatically generated

Add an **appropriate** message and click [Add] so you don't miss any files, finally click [Commit].

Graphical user interface, application

Description automatically generated

After that click [Push]and then [OK]:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

This is all you need to **upload your project source code** to your **GitHub repository** using TortoiseGit.

### Use Git Bash (Option 2)

As **alternative to the previous step**, if you don't have "**TortoiseGit**", you could use the "**Git Bash**" command line tool to upload your project to your GitHub repo.

First, if you don't have **Git** on your **computer**, you should **install it** from <https://git-scm.com/downloads>.

Go to the desired **directory**, right click on blank space **anywhere** in the folder, select "**Git Bash Here**" to open the Git command line console. If the "**Git Bash Here**" menu is missing, you should first install Git.

Graphical user interface, application

Description automatically generated

Type **"**gitclone**"** command followed by the link of your **repository**:

|  |
| --- |
| **git clone** |

This command is for cloning with **Git Bash**, paste your **repository** **URL** after the command.

Text

Description automatically generated

Your files from your GitHub repo will be downloaded to a **sub-folder** called as your project in GitHub, "**GuessANumberByPeter**" in our case.

Graphical user interface, text, application, email

Description automatically generated

When we open the cloned **repository sub-folder**, it should look like this:

Graphical user interface

Description automatically generated with low confidence

Next thing to do is to **add** your **project files** into your **cloned** **repository folder**. It should look like this:

Graphical user interface, text, application

Description automatically generated

Now we are ready to upload our changes from "**Git Bash clone**". Go to the desired **folder**, right click on blank space anywhere in the folder, select "GitBashHere" and run the following **commands**.

Type the following command:

|  |
| --- |
| git status |

The **git status** command displays the state of the working directory and the **staging area**.

A screenshot of a computer

Description automatically generated

Now type:

|  |
| --- |
| git add . |

This command **adds** all modified files.

Next type:

|  |
| --- |
| git commit -m "Your message here" |

This command**commits** your changes. We also should **add** an appropriate **message**.

Second to the last type.

|  |
| --- |
| git pull |

This command **updates** your local **repository**.

Now the last thing that we should do is to **push** our changes by using the command:

|  |
| --- |
| git push |

This command **pushes** your changes to our local **repository**.

Text, chat or text message

Description automatically generated

This is all you need to **update** your **repository** with **Git Bash**.

A little more information about it here: <https://git-scm.com/about>.

## \* Modify the Code, Write Your Own Features

Now, it's time to **play with the code** and **modify it**.

|  |  |
| --- | --- |
| Icon  Description automatically generated | This is your own project. **Be unique**. Don't be a copy-paster!   * Implement your **own features**. * **Implement the code yourself**, using your own coding style, code formatting, comments, etc. * Make the project **more interesting**. Learn by playing with the code and adding your own changes. |

Below are a few **ideas** of what you can implement or modify as an addition to your code.

### Add Difficulty

You can add logic for difficulty, so the player can have **only a few tries** to guess the number.

### Restart the Game

You can automatically **restart the game** after it is finished (or ask the player to play again).

### Additional Ideas

* You can **add levels** so every time when the player guesses the number, the range between the minimum and maximum number gets bigger e. g. **Level 1 (1 - 100)**, **Level 2 (1-200)** etc.
* You can add anything else to your code, based on your own ideas?

### Commit to GitHub

Now **commit and push your code changes** to your GitHub repo!

|  |  |
| --- | --- |
| Icon  Description automatically generated | It is very important to **commit frequently** your code to GitHub. This way you create a **rich commit history** for your project and your GitHub contribution graph is growing:  A picture containing chart  Description automatically generated |

## Create a README.md File

It's highly recommended to provide documentation as part of your project in GitHub to describe what the project is doing. So, let's make one for this **project**. Let's start by editing the README.md file from our repo at GitHub:

Graphical user interface, text, application, email

Description automatically generated

Add a project name. Use "#" in front of the text to indicate the **title**:

Graphical user interface, application

Description automatically generated

You can **view** the current progress by pressing the [Preview] button:

### Documentation Sections

Add **information** about your project in your README.md file: project goals, technologies used, screenshots, live demo, etc. Typically, you should have the following **sections**:

* **Project title** (should answer the question "What's inside this project?")
* **Project goals** (what problem we solve, e. g. we implement a certain game)
* **Solution** (should describe how we solve the problem 🡪 algorithms, technologies, libraries, frameworks, tools, etc.)
* **Source code link** (give a direct link to your source code)
* **Screenshots** (add screenshots from your project in different scenarios of its usage)
* **Live demo** (add a one-click live demo of your code)

### Use Markdown

Note that the GitHub README.md file is written in the **Markdown language**. Markdown combines text and special formatting tags to describe formatted text documents.

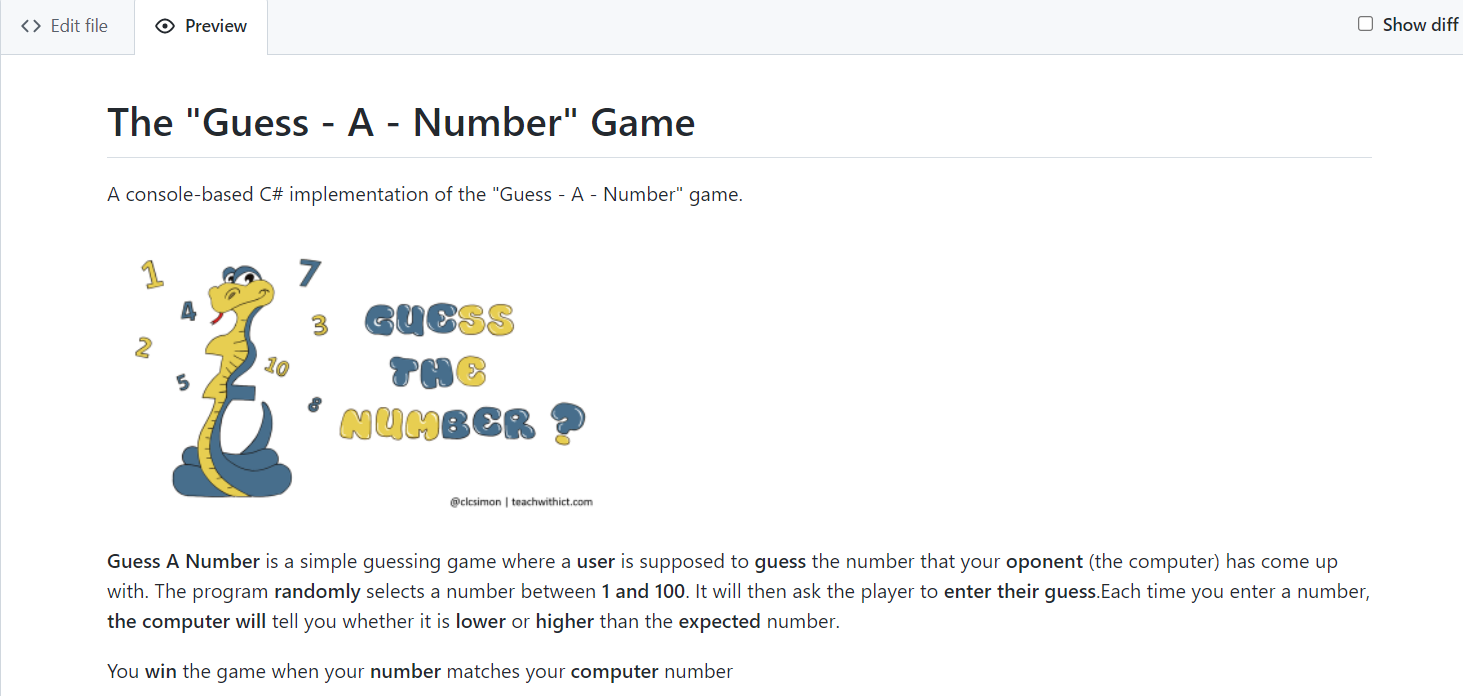
You can learn more about **Markdown** here: <https://docs.github.com/en/get-started/writing-on-github/getting-started-with-writing-and-formatting-on-github/basic-writing-and-formatting-syntax>.

### Project Goals

Start your documentation by describing your **project goals**. What problem does your project solve?

### Sample Documentation

This is an **example** of how you can document your project. Don't copy and paste it!



|  |  |
| --- | --- |
| Icon  Description automatically generated | **Write the project documentation yourself**. Don't copy and paste it!  This is your **unique GitHub profile** and your own unique project. **Be different** from others. |

Find an **appropriate** **image** and add it. You can add **images** as follows:



You can add information about the **inputs** and **outputs** of the project:

Graphical user interface

Description automatically generated with medium confidence

### Your Solution

Describe how you **solve the problem**: algorithms, technologies, libraries, frameworks, tools, etc.

### Link to the Source Code

Add a **link** to your **source code** as follows:

Logo

Description automatically generated with low confidence

### Screenshots

Add **screenshots** of your project:

1. **Take a screenshot** with your favourite tool (e.g. the [Snipping Tool](https://support.microsoft.com/en-us/windows/open-snipping-tool-and-take-a-screenshot-a35ac9ff-4a58-24c9-3253-f12bac9f9d44) in Windows).
2. **Paste** the screenshot in the GitHub Markdown editor, using [Ctrl+V]:

Example screenshots for the "Guess a Number" game:

Text

Description automatically generated

## Upload Your App to Replit

You already should have a **Replit** profile. Now let's add our **project** there so we can share it with our **friends** and add it to our **GitHub** profile. You already should know how to do that.

Open the **menu** in the upper **left corner**. Click "**Create**", then select the **language** in which your project is **written**, select a name, and **create** the project. If your project is in C#, choose “**Mono C#**”. In Replit the C# projects work faster with Mono, than with .NET 6.

Graphical user interface, text, application, chat or text message

Description automatically generated

Add your code in "Main.cs" file.

Graphical user interface, text, application

Description automatically generated

**Click** [Run] and enjoy your console application directly in the Web browser:

Text

Description automatically generated

You can now **share** your app with your friends.

## Add Replit Link to Your README.md

Now add a "**one-click live demo**" of your project from your GitHub project documentation. You can do it as follows:

A picture containing text

Description automatically generated

You can take a **screenshot** from Replit.com and **paste it** into the GitHub documentation editor directly with **[Ctrl+V]**.

This is what it should look like after the changes in your README.md documentation:

Graphical user interface, text, application

Description automatically generated

Now we have completed our **second console game** and we have our second **project** in our GitHub portfolio.