Final C# Project

Spaceman

Overview

This project is slightly different than others you have encountered thus far on Codecademy. Instead of a step-by-step tutorial, this project contains a series of open-ended requirements which describe the project you'll be building. There are many possible ways to correctly fulfill all of these requirements, and you should expect to use the internet, Codecademy, and other resources when you encounter a problem that you cannot easily solve.

Project Goals

Invaders from outer space have arrived and are abducting humans using tractor beams. Players must crack the codeword to stop the abduction!

Ok, we'll admit it's quite a bit like that classic game, "Hangman", but with a better premise. Plus, building this command-line game was the Codecademy 2019 Software Engineer Internship Backend Programming Challenge!

Setup Instructions

Download the starter code. You'll define the methods in **Game.cs** and run the code to see your game in action.

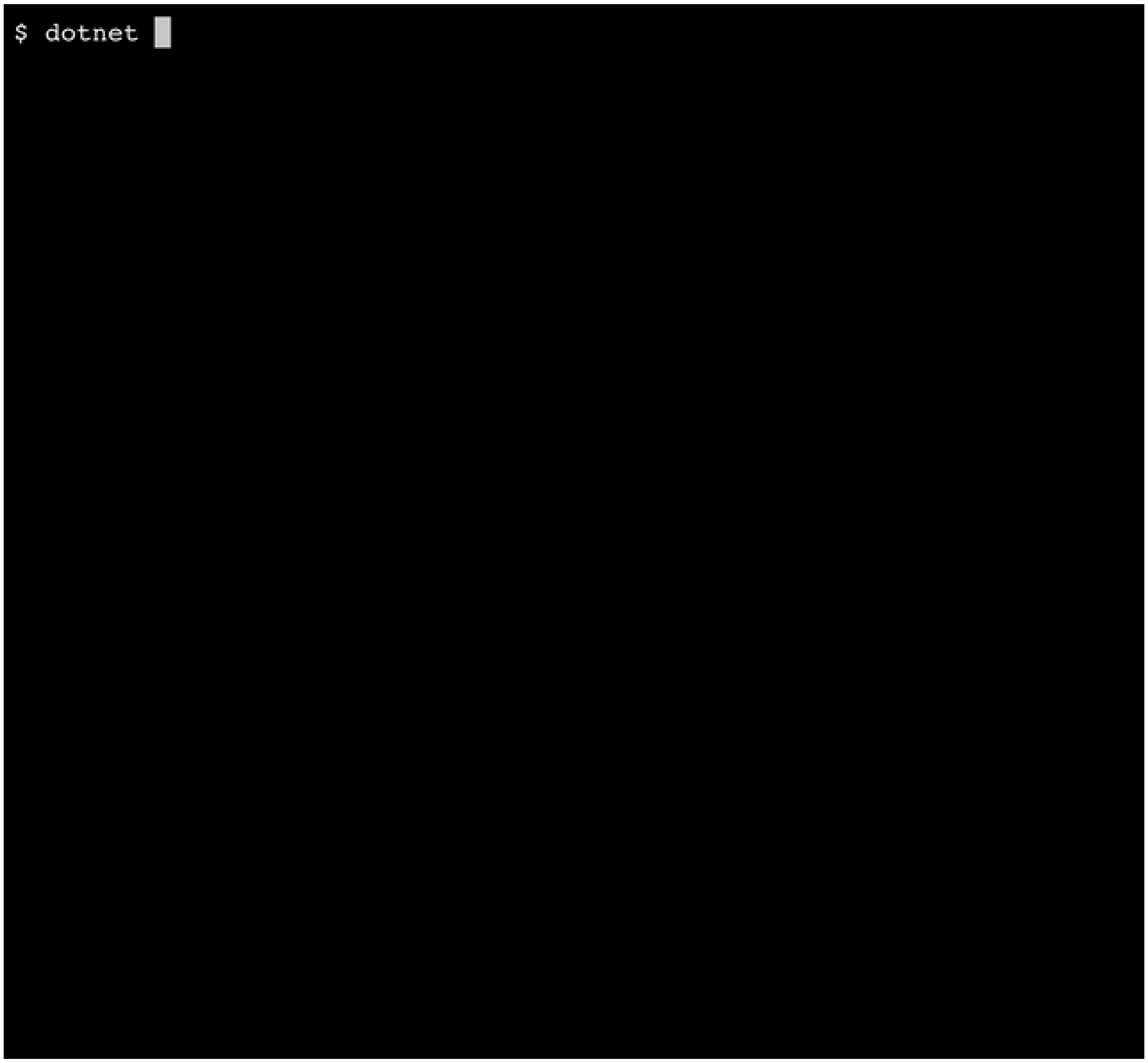
Tasks

Prerequisites

1. In order to complete this project, you should have completed the first 6 sections of Learn C# (through Learn C#: Classes).

Project Requirements

2. You'll design a `Game` class in **Game.cs**, which will be used in the `Main()` method in **Program.cs**. Your code will primarily focus on the game logic — we've already provided a `Ufo` class that you can use to draw the spaceship with each wrong guess.
3. Check the GIF below to get a sense of what you'll be building. The gif shows the game be started with `dotnet run`, but you'll run it in Visual Studio using Start without Debugging.
4. In the game, the user guesses one character at a time. The game ends once the word is complete or the use makes five wrong guesses. The game can be shut down early by pressing CTRL + C.



```
$ dotnet
```

3. In **Game.cs**, make a `Game` class within the `Spaceman` namespace.

Make a `Greet()` method that will be called at the beginning of each. It should print out a greeting to the player.

Hint
The signature of `Greet()` will be:

```
public void Greet()
```

In the GIF we used a greeting like this:

```
=============
UFO: The Game
=============
Instructions: save your friend from alien abduction by guessing the
letters in the codeword.
```

4. In the `Game` class we'll need to keep track of these six items:

- **Codeword** — the word players will guess
- **Current word** — to be filled in as the player guesses correctly
- **Maximum number of guesses** — the number of wrong guesses before the abduction is finished
- **Current number of wrong guesses** — increases with each guess that is not in the codeword
- An array containing a few options for the codeword — so that the codeword isn't the same every game
- An instance of the `Ufo` class — to print the UFO images

  Make a property/field for each.

  Hint
  Use the `Ufo()` constructor to make a new instance.

  You can review fields, properties, and constructors in the Basic Classes lesson.

5. **Create a constructor** that:

- Sets the codeword to a random value in the array of strings
- Sets the maximum guesses to `5` — there are only 5 wrong guesses before the abduction is complete
- Sets the wrong guesses to `0`
- Sets the current word to a string of underscores (_) that is the same length as the codeword

Hint
You'll need to [use the `Random` class]() to get a random value from `wordbank`.

You can build the current word using a `for` loop that iterates from `0` to the length of the codeword. The loop should add an underscore (`_`) to the current word each time.

6. We need a way to check if the player won. Create a `DidWin()` method that will return `true` when the current word matches the codeword.
   To compare strings, use the `Equals()` method:

   ```
   CodeWord.Equals(CurrentWord);
   ```
   Hint
   If you want to try another approach, return `true` if the current word no longer contains underscores (`_`).

7. We also need to check if the player lost. Create a `DidLose()` method that returns `true` when the number of wrong guesses is greater than or equal to the max number of guesses.

8. Next, **create a `Display()` method** that will print all the necessary game information to the screen. That includes:

- The UFO
- Current word
- Number guesses remaining

The `string` version of a `Ufo` can be accessed with its own `Stringify()` method.

Hint
This returns a string which, when printed to the console, looks like a spaceship!

```
Ufo spaceship = new Ufo();
Console.WriteLine(spaceship.Stringify());
```

9. With everything else in place, it's time to write the code for the interaction. Define an `Ask()` method that:
   - Asks the user to guess a letter and captures their input
   - Checks that the input is length of `1`. If it is not, tell the user to input one letter at a time and use `return` to break out of the method

- Checks if the codeword `Contains()` the guess
- If it does, tell the user, find all instances of the guess in the codeword, and replace it with the guess, e.g. if the guess is `g` and it appears in the first and last position, then replace the first and last letters with `g`
- If the codeword does not contain the guess, tell the user, increase the number of wrong guesses by `1`, and call the `AddPart()` method on the `Ufo` object

To replace a letter in a `string` use the `Remove()` and `Insert()` pattern:

```
someWord = someWord.Remove(index, 1).Insert(index, "z");
```

Hint
Remember that `Console.ReadLine()` returns a `string`, but each element within a `string` is a `char`. Convert one to the other using `String.ToCharArray()`.

Here's the documentation for:

- The `String.ToCharArray()` method
- The `String.Remove()` method
- The `String.Insert()` method

10. Your `Game` class is complete! Let's use it in **Program.cs**:

- Create a new `Game` object and have it `Greet()` the player
- Create a loop that calls the object's `Display()` and `Ask()` methods. The loop should run until `DidWin()` or `DidLose()` returns `true`
- It should print a message if the user wins or loses

Hint
You can create the loop in a few ways:
- Make the loop run infinitely by using `while(true)`. `break` out of the loop if either end-game condition is true
- Make the loop run until either end-game condition is `true`

Extensions

Great work! If you'd like to extend your project on your own, you could consider the following:
- The game doesn't record past guesses and doesn't check if a user has already guessed a letter. Add that feature.

- Create your own spaceship design! Go into **Ufo.cs** and read through it. Edit the code as needed.