PGCert IT: Programming for Industry

Challenge 03

This challenge is worth 5% of the total grade. Each exercise is worth 1% of the total grade.

A Bulls and Cows Game

Introduction

The bulls and cows game is a code-breaking game designed for two or more players. Each player chooses a secret code of 4 digits from 0 – 9. The digits must be all different. The goal of the game is for each player to guess the other player's secret code.

The players in turn present their guesses to the opponents. The opponents respond by telling the players:

a. The number of bulls, i.e. the number of matching digits in their right positions, and
b. The number of cows, i.e. the number of matching digits but in different positions.

For example, the secret code for the computer is 4281, the match responses for the following guesses are shown below:

```
Please enter your secret code:
4568
---
You guess: 1234
Result: 1 bull and 2 cows

Computer guess: 5940
Result: 0 bull and 2 cows
---
You guess: 1345
Result: 0 bull and 2 cows

Computer guess: 1279
Result: 0 bull and 0 cow
---
You guess: 4271
Result: 3 bulls and 0 cow

Computer guess: 4890
```

```
Result: 1 bull and 1 cow
---
You guess: 4281
Result: 4 bulls and 0 cow
You win! :)
```

More information about the game itself can be found here.

In this exercise, you need to design and implement a simple console program to play bulls and cows interactively against the computer. Both the player and the computer will have secret codes, and each will be trying to guess the secret code. Both the player and the computer only have seven attempts for guessing the secret codes. Before starting the game, the program should let the player enter the secret code for the computer to guess. You can use a simple random strategy (generating four unique digits randomly) for the computer to make its guesses. If the user entered an invalid secret code, the program should ask the user to try again. That is, you should use a try-catch block to recover from incorrect inputs made by the user.

Exercise One: Design the Game

Design your classes and methods (i.e. create UML class and sequence diagrams) before implementing the game. Don't have everything in one class, and try to promote code reuse as much as possible. You **must** have at least one use of inheritance in your project.

You also need to be careful when designing the parts for secret codes. We may change our mind about replacing the four digits to four-letter words in future!

Discuss with the tutors or lecturers about your design before moving on to the next exercise.

Exercise Two: Implement the game - player's turn

Implement the first part of the game allowing the player to guess the computer's secret code. The computer randomly generates the secret code at the beginning of the game, which it then lets the player guess. Remember that when generating the computer's secret code, each of the four digits must be different. Note that the player only has seven attempts to guess the secret code. The prompt for player input, results for each guess and the final outcome (i.e. whether the player has won the game or not) should be displayed appropriately to the console.

Exercise Three: Implement the game - computer's turn

Show the tutors your implementation so far and compare with your initial design. Note down anything that you like to change or you have changed from the initial design. Now, modify your code so that the player can now also enter a secret code when the game begins, which the computer must guess. Remember to verify that the player has chosen a valid secret code. The player and computer each take turns guessing the other's code. The game ends when either side successfully guesses the other's code (resulting in a win for that side), or when each side has made seven incorrect guesses (resulting in a draw).

Exercise Four: Making the computer smarter

Modify your code so that the player can choose to play against a smarter opponent. The computer should be more intelligent when guessing, instead of choosing any number at random. One possible strategy for implementing a smart opponent is given below:

Possible Strategy

This strategy involves keeping a *list* of all possible guesses, and then intelligently pruning that list based on the result of each guess made by the AI. In this strategy, the first guess by the computer will be chosen randomly. After this guess, all subsequent guesses will be carefully planned. The computer keeps a track of precisely which codes remain consistent with all the information it has received so far. The computer will only choose a guess that has a chance of being the correct one.

For example, let's assume the computer's first guess scored 1 bull and 1 cow. Then the only codes that still have a chance to be the correct one, are those which match up 1 bull and 1 cow with the first guess. All other codes should be eliminated. The computer will go through its list of all possible codes and test each against its first guess. If a code matches 1 bull and 1 cow with the first guess, then it will remember that the code is still a possible candidate for the secret code. If a code does not match 1 bull and 1 cow with the first guess, the code will be eliminated. After this is completed, the computer randomly chooses any of the possible candidates for the second guess.

If the computer's second guess then scored 2 bulls and 1 cow, the computer checks all the remaining candidates to see which codes match up 2 bulls and 1 cow with the second guess. Those codes that do not match are eliminated. In this manner, each guess is consistent with all the information obtained up until that point in the game.

To illustrate the process, consider the scenario shown below, in which a simpler version of the game is being played. In this demonstration, secret codes are only two digits in length, and may only contain the characters 1 through 4. The player has chosen "2 1" as their secret

code, which the AI is trying to guess. Using this process of elimination, the AI is able to quickly guess the player's code.

Your secret code: **2 1**

AI Move One:

Possible codes:
1 2
1 3
1 4
2 1
2 3
2 4 ← Randomly guesses 2 4
3 1
3 2
3 4
4 1
4 2
4 3

This scores 1 bull 0 cows.

Analysis:

Code	Score against guess 2 4	Result
1 2	0 bull 1 cow	Discard
1 3	0 bull 0 cow	Discard
1 4	**1 bull 0 cow**	**Keep**
2 1	**1 bull 0 cow**	**Keep**
2 3	**1 bull 0 cow**	**Keep**
2 4	2 bull 0 cow	Discard
3 1	0 bull 0 cow	Discard
3 2	0 bull 1 cow	Discard
3 4	**1 bull 0 cow**	**Keep**
4 1	0 bull 1 cow	Discard
4 2	0 bull 2 cow	Discard
4 3	0 bull 1 cow	Discard

AI Move Two:

Possible codes:
1 4 ← Randomly guesses 1 4
2 1
2 3
3 4

This scores 0 bulls 1 cow.

Analysis:

Code	Score against guess 1 4	Result
1 4	2 bull 0 cow	Discard
2 1	**0 bull 1 cow**	**Keep**
2 3	0 bull 0 cow	Discard
3 4	1 bull 0 cow	Discard

AI Move Three:

Possible codes:
2 1 ← Randomly guesses 2 1

This scores 2 bulls 0 cow.
Success!

Analysis:

None required - we won!

Exercise Five: Extending your game

For this exercise, choose **one** of the following three options to extend the game.

a. Using different set of codes

Sometimes the players change their minds about the type of secret codes they want to use. Modify your code so that the player and the computer can use four-letter words, from A - F, as the secret code to play the game.

b. Reading guesses from a file

Modify your code so that before the game begins, the player is asked whether they wish to enter their guesses manually, or to automatically guess based on pre-supplied guesses in a file.

If the first option is chosen, then the game should progress in the same fashion as in Task Five above. If the second option is chosen, then the following actions should be taken:

Firstly, the player should be asked to enter a filename. If the player enters an invalid filename, they should be re-prompted until they enter the name of a file that actually exists. This file should then be read and interpreted as a text file, where each line contains a separate guess. For example, a sample file `input.txt` may contain the following text:

```
1234
4321
5830
8437
1489
3271
2530
```

You may assume that each line of the file contains a valid guess.

Once the file has been read, then the game should proceed as normal. However, when the player would be prompted to enter a guess, the next guess in the list of pre-supplied guesses should automatically be chosen instead. If there are no more pre-supplied guesses (for example, if the player needs to enter their fifth guess but the file only contained four guesses), then the player should be prompted as normal.

c. Save to a File

Once you have modified your game to read from a text file, the next thing you might want to do is to save the output of the game into a text file. Modify your game so that the user can save the result to a text file when the game finishes. You should let the user specify the text file name, and the location where the user likes to store the file.