PGCert IT: Programming for Industry

Assignment 03: Bulls & Cows

Due: Sunday 10th May 2020 at 11:59pm

This assignment requires you to develop a simple game called Bulls & Cows. There are **eight** compulsory tasks in this assignment to complete.

Before starting the assignment, read through and gain an understanding of the requirements.

Notes:

- The assignment is out of **60 marks** and is worth **10%** of your final Programming for Industry grade.
- You should have the following source files for the assignment:
  - `Keyboard.java`
- After completing the assignment, you should have a functional Java program with some number of classes (depending on your own design), two UML class diagrams, and one page of reflective text.
- Zip your IntelliJ project for the assignment, including all the source file, diagrams, and text, and submit the single Zip file before the due date.
- **IMPORTANT**: Read the instructions carefully before attempting each task.

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, if the computer's secret code 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 bulls and 2 cows
---
You guess: 1345
Result: 0 bulls and 2 cows

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

Computer guess: 4890
Result: 1 bull and 1 cow
---
You guess: 4281
Result: 4 bulls and 0 cows
You win! :)
```

Figure One: Example output from a Bulls & Cows game.

More information about the game itself can be found here.

Requirements

The main goal of this assignment is to develop the bulls and cows game that allows a single player to play interactively against the computer. The game stores two secret codes, one from the player and one from the computer. The player and the computer will try to guess each other's secret code. Both the player and the computer only have seven attempts for guessing the secret code. If the player enters an invalid input, the game should ask the player to try again. The game also lets the player choose the difficulty level to play against the computer. There are three levels: easy, medium and hard. The details of the difficulty levels will be described in later sections. In addition, the game can read from a text file that contains multiple guesses from the player, and save the results to another text file.

For this assignment, you'll complete a series of tasks as you work your way towards a fully functional implementation. As you complete each task, it might be a good idea to save a working version of the program at that point.

Task One: Design

For this assignment, there is very little which is already given to you. Through this assignment, you'll gain experience in designing and building a complex program from scratch. *Before starting to code*, don't forget to *design* your classes and methods (i.e. create UML class and sequence diagrams)! You should apply the concepts you have learned so far in the course. 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.

Using either pen & paper or the diagramming tool of your choice, prepare a UML class diagram which shows all the classes and important methods of your Bulls & Cows implementation, along with the appropriate relationships showing how these classes fit together. Include either a PDF, JPEG, or PNG image of your class diagram in the zip file with your final submission.

IMPORTANT: You must show your initial design to one of your lecturers / tutors for feedback on your design by 20th April. You may include this feedback as part of your reflection.

Note: It's OK if your implementation doesn't match your initial design 100% - things change! You'll document this process in Task Eight.

Task Two: The Beginning

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.

Task Three: Easy AI

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).

For this task, have the player play against an easy AI. When the AI makes a guess, it will simply generate a random (valid) guess.

Task Four: Medium AI

Modify your code so that at the beginning of the game (before the player enters their own secret code), they will be asked to select either an easy or medium AI opponent to play against.

If the player chooses to play against an easy AI, the game should proceed in exactly the same manner as in Task Three. However, if a medium AI is selected, the AI should keep track of guesses it has already made. The AI will not make the same guess twice.

Task Five: Hard AI

Modify your code so that the placer can additionally choose to play against a hard AI opponent.

If the player chooses to play against an easy or medium AI, the game should proceed in exactly the same manner as in Task Three or Four. However, if a hard AI is selected, the computer should be much more intelligent when guessing, rather than just choosing at random. One possible strategy for implementing the hard AI is given below.

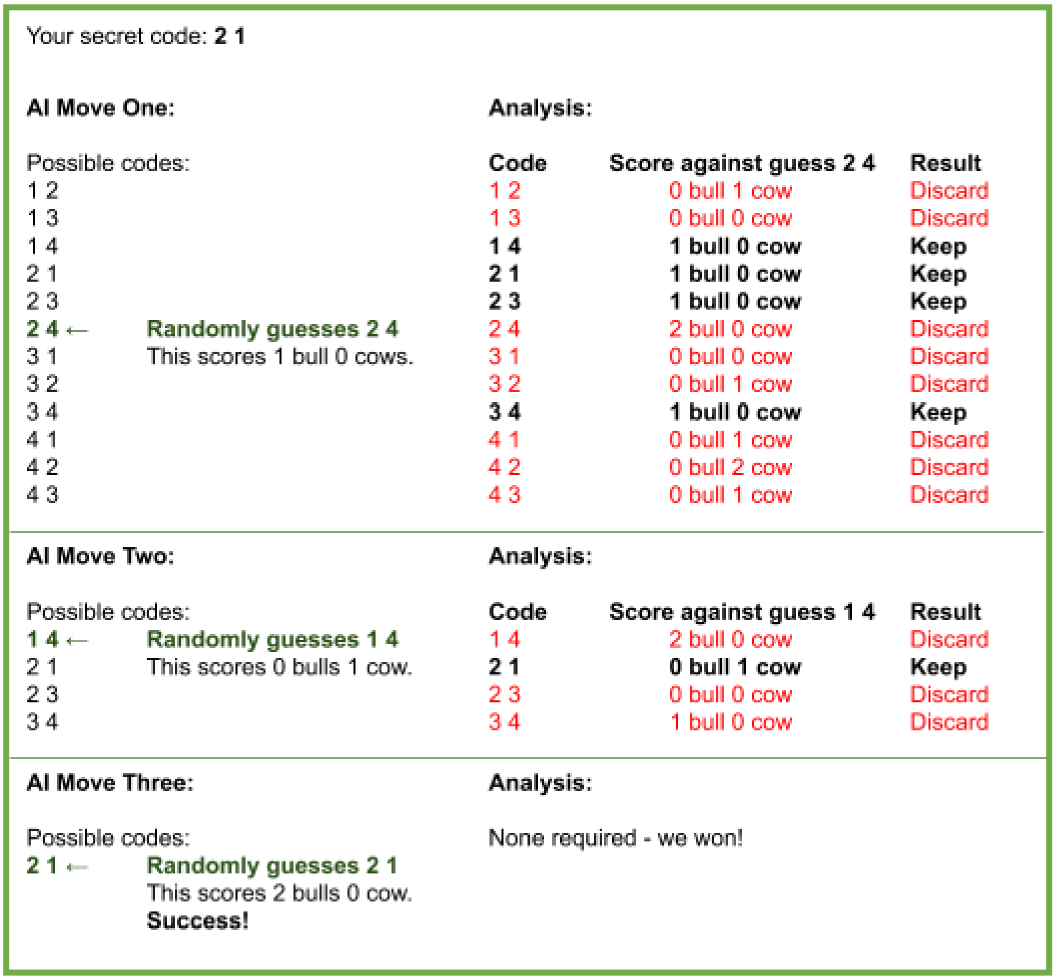
Possible Strategy

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 in figure two 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 This scores 1 bull 0 cows.
3 2
3 4
4 1
4 2
4 3

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 This scores 0 bulls 1 cow.
2 3
3 4

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!

Figure Two: Demonstration of “hard-mode” Bulls & Cows AI.

Task Six: 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
```

Figure Three: Example contents of a text file containing guesses to be used instead of the player entering their guess using a keyboard.

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.

Task Seven: Saving to a File

Modify your code so that, when the game ends (win, lose, or draw), the player is asked if they wish to save the results to a text file. If they do, then they'll be prompted to enter a filename. The game should then save the following information to the given file:

- The player and computer's secret number
- Each guess that was made, in the same order as occurred during the game, along with the result of that guess (i.e. how many bulls & cows it got)
- The identity of the winner (or a message stating that the game was a draw)

The data must be readable when opened in a standard text editor. An example results file is given in figure four overleaf, though exactly how your file is organized is up to you.

```
Bulls & Cows game result.
Your code: 4568
Computer’s code: 4281
---
Turn 1:
You guessed 1234, scoring 1 bull and 2 cows
Computer guessed 5940, scoring 0 bulls and 2 cows
---
Turn 2:
You guessed 1345, scoring 0 bulls and 2 cows
Computer guessed 1279, scoring 0 bulls and 0 cows
---
Turn 3:
You guessed 4271, scoring 3 bulls and 0 cows
Computer guessed 4890, scoring 1 bull and 1 cow
---
Turn 4:
You guessed 4281, scoring 4 bulls and 0 cows
You win! :)
```

Figure Four: Example results text file showing the secret codes, moves, and results of a Bulls & Cows game.

Task Eight: Reflection

Now that you’re complete, it’s time to reflect upon your design.

To begin, create another UML class diagram modelling your source code for Bulls & Cows (again, include this diagram in your Zip file for submission). You might find that this is different than the initial design you did in Task One. In a text file named `reflection.txt`, write two to four paragraphs including the following:

- Reflect back on the first day against now, what have you been able to achieve and what skills have you learned from this course?
- Why you believe your design is a good design for this assignment?
- If your final implementation is different to initial design, explain why you have made such changes.

Marking Guide

This assignment will be marked as follows:

A UML class diagram is given as evidence of initial design work.	*5 marks*
The computer chooses a valid secret code for the player to guess.	*3 marks*
The player can guess the computer's secret code. If the player enters an invalid code, they'll be prompted to try again.	*4 marks*
The player can enter a secret code for the computer to guess. If the player enters an invalid code, they'll be prompted to try again.	*2 marks*
The computer can guess the player's code in "easy mode". The "AI" will guess a random *valid* code each move.	*3 marks*
The game ends and one side wins if that side correctly guesses the other's code. A message stating who won is displayed.	*3 marks*
The game ends in a draw if each side has made seven incorrect guesses. A message stating the game was a draw is displayed.	*2 marks*
The computer can guess the player's code in "medium mode". The AI is guaranteed not to make the same guess twice during a particular game.	*4 marks*
The computer can guess the player's code in "hard mode". The AI uses an intelligent algorithm, such as the one given in the example, to make guesses.	*10 marks*
The player can supply a list of predefined guesses in a text file. These are used instead of prompting the player to enter guesses each round.	*5 marks*
Game results can be saved to a text file.	*7 marks*
A detailed UML class diagram and reflective paragraph is given to document the final implementation.	*5 marks*
Good software design (multiple appropriate classes, code reuse, good structure, variable names, comments, etc).	*7 marks*