

Julian Milligan ACCB5011

Assignment 1

by Julian Milligan

Submission date: 09-Dec-2021 05:52PM (UTC+0000)

Submission ID: 165609650

File name: 16671_Julian_Milligan_Julian_Milligan_ACCB5011_Assignment_1_382345_3442224.docx (104.8K)

Word count: 2413

Character count: 10910

Name – Julian Milligan

Student number – 1904804

Course – Bachelor of Science (Part-time): Computing (Software Engineering)

Module – L5 - Advanced Software Development

Module code - ACCA5011

Assignment title - OOP Programming (C++)

Date – 22/11/2021

Table of Contents

Introduction	3
Game	3
Idea.....	3
Rules.....	3
Code	4
Problem solving.....	4
Development.....	4
Testing.....	6
OOP Report	8
Reflection	12
Bibliography	14
Extra reading	15

Introduction

This report will cover the design and build of a simple game. It will also cover the testing of the code and a section on what is Object Oriented Program and why it is used.

Links to on-line video

Stream [Watch 'New channel meeting' | Microsoft Stream](#)

YouTube <https://youtu.be/cT7Lt0Kr32w>

Game

Idea

The game will be based around the premise of 'Guess the number'. Each player will take it in turns to guess the number and will be told if it's too high or too low. When the number is guessed correctly the winners name will be displayed.

Rules

There will be 3 levels.

Easy (range 1 -10) 3 tries.

Medium (range 1 – 50) 6 tries.

Hard (range 1 – 100) 10 tries.

Each player will take it in turns to guess.

The winner will be the one that guess the number first.

Code

Problem solving

Menu

- Ask the players details (min 2)
 - Name of player
 - While loop to ask name and create player object in Vector
- Ask the game level.
 - Number of rounds 1 , 3 or 5.
- Intro to the game and the rules.

Game play

- Keep score of each for each round.
- Keep track of turns , P1 – P2 - etc

Development

Classes

Player

- Public
 - Constructors
 - Inheritance for Game
 - Winner message
 - Get name
 - Get Score
- Private
 - Name
 - Score , default 0.

Game

- Public
 - Constructors
 - Virtual Winner message = 0
 - Get Winner
 - Set Level
 - Set number of rounds
 - Set number of players
- Private
 - Level
 - Number of rounds
 - Winner
 - Players
 - Generate Number

Turn

- Public
 - Constructor
 - Inheritance for Game
 - Get turn
 - Get Guess Result
- Private
 - Who's Turn?
 - Higher or lower than Generated number

Order of development

Task	Completed
1. Display intro , rules, and levels	Yes
2. Menu display	Yes
3. Capture game level	Yes
4. Capture number of rounds	Yes
5. Capture player name and create players	Yes
6. Generate random number based on level	Yes
7. Ask for player guess (by name) and check	Yes
8. If winner display winners name and increment score	Yes
9. If no winner loop through the number of tries	Yes
10. Repeat from number 6 on wards depending on the number of rounds.	Yes

Testing

Each time a new feature was added a line was added to the testing section so it could be tested to confirm it was doing as expected. Testing this way meant there would be no big surprises half or the end of the program.

```
//testing
cout << endl;
cout << "**** For testing only ****" << endl;
cout << "number of players - ";
newGame->getNumberOfPlayers();
cout << endl;
cout << "number of rounds - ";
newGame->getGameRound();
cout << endl;
cout << "game level - ";
newGame->getGameLevel();
cout << endl;
cout << "Player 1 info ";
player1->getPlayerName();
cout << endl;
cout << "Player 2 info ";
player2->getPlayerName();
cout << endl;
cout << "test vector " << endl;
playerList->getAllPlayers();
cout << endl;
cout << "test random number gen" << endl;
newTurn->setRandomNumber(1);
cout << "level 1 - " << newTurn->getRandomNumber();
cout << endl;
newTurn->setRandomNumber(2);
cout << "level 2 - " << newTurn->getRandomNumber();
cout << endl;
newTurn->setRandomNumber(3);
cout << "level 3 - " << newTurn->getRandomNumber() ;
cout << endl;
```

```
*** For testing only ***  
number of players - 2  
  
number of rounds - 1  
  
game level - 2  
  
Player 1 info Julian  
  
Player 2 info Kirsty  
  
test vector  
Julian  
Kirsty  
  
test random number gen  
level 1 - 0  
level 2 - 30  
level 3 - 47
```

In the main source.ccp file there are a few lines used to test various features of the program. These include checking the number of Players set , number of rounds , the names for both players , the vector that contains the players and the random number depending on which level was selected.

```
loop testing  
Object value6  
try 0
```

There is also some code to test the loop is working as expected.

Object value is the random number

Try is the number of loop iterations that have been.

OOP Report

What is Object Oriented Programming ?

On the site w3schools.com they say

‘Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.’ [1]

In simple terms it is grouping several attributes together to allow quicker development , these could be data or functions that will act on the data. The idea is providing a clear structure for the program.

The main idea behind OOP is stop the developer from repeating themselves by using the DRY principle , DRY stands for DON'T REPEAT YOURSELF [2]

A good example could be a garage. In the garage we have cars all cars have some items in common. Wheels , engine , windows , a colour , body type and seats . Here we have to classes.

Class Name – Garage.
It Links to Cars

Class Name – Car
Attribute – Wheels
Attribute – Engine
Attribute – Windows
Attribute – Colour
Attribute – Body Type
Attribute - Seats

Another class could be types of cars such as Off Road , sports , electric or custom. Each of these will inherit all the attributes from the class Car but extras can then be added for each of the type of cars. As an example, Electric could have Battery type , Auto drive or Charge type.

Class Name – Electric
Attribute – Wheels - inherited from Car
Attribute – Engine - inherited from Car
Attribute – Windows - inherited from Car
Attribute – Colour - inherited from Car
Attribute – Body Type - inherited from Car
Attribute – Seats - inherited from Car
Attribute – Battery Type
Attribute – Auto Drive
Attribute – Charge Type

The class Garage could also have some attributes such as repair time , hourly rate etc. As well having Attributes the class would also have methods. These will be functions that can alter , set, or read the data in an attribute. When a car is created a constructor will be called and values entered for various Attributes in the class. With in the constructor default values could also be set . If there was an attribute for date booked in this could be todays date.

As each car is created it could then be stored in an array or vector within the Garage class. Further classes could be written for staff members and then expanded to mechanics and office staff. The same principles would be in affect staff would have some common attributes like name , employee number , pay scale , Date of birth , etc. The mechanic would have some more detailed items.

So how does this help in the development of a program. If this was written using Arrays to hold all the data or another data structure. If a new type of car was to be repaired at our garage, then a large amount of code would need to be altered to add it. With OOP we would just need to add a new class and any extra attributes it would need. Also, if the garage ever needed to keep track of Co2 emissions then this could be added to the Car class with the methods created to set and get the data. This would then be available to all the class that would inherit from the Car Class.

In the code that has been written for the game 'Guess the number' 3 classes have been created.

1. Game
2. Player
3. Turn

These 3 were chosen as the developer felt these were 3 unique parts of the problem that need to kept separate but in turn inherit from the main class Game.

Game contains the basic setup information for the game such as number of players , number of rounds to be played and the level to which the players choose. Each of the attributes has a method to get the data stored in them. They also have a method to set the data within them too. A main constructor that takes no overrides has also been created to set the defaults values on

```
_gameLevel = 0;  
_gameRound = 0;  
_numberOfPlayers = 0;
```

It also calls another method called `setMessages()` which is used to set a few messages for in game use. These include a brief introduction to the game , the rules and how the levels are made up.

```
1  #pragma once
2
3  #include <iostream>
4  #include <string>
5
6
7  using namespace std;
8
9  class Game
10 {
11 private:
12
13
14
15     string _welcomeMessage;
16     string _gameRules;
17     string _gameLevelText;
18     int _gameLevel ;
19     int _gameRound ;
20     int _numberOfPlayers;
21
22
23 public:
24
25     Game();
26
27     void setMessages();
28
29
30     void getWelcomeMessage();
31     void getGameRules();
32     void getGameLevelText();
33     void getGameLevel();
34     void getGameRound();
35     void getNumberOfPlayers();
36
37
38     void setGameLevel(int level);
39     void setGameRound(int round);
40     void setNumberOfPlayers(int players);
41
42
43
44
45 };
46
47
```

Player is the next class, and this is set to inherit from Game currently thought nothing is being inherited from the Game class. In this class the following is created

String _playerName;

Int _score ;

Vector <Player*> playerList;

Getters and Setters have been created for playerName and score. Also, two constructors have been created one that's accepts no overrides and one that accepts and string to set the player's name. The constructor with no overrides does not sets the score to 0.

addPlayer is a method to add a new player to the vector playerList. It does not add the full object to the vector only a pointer to the memory location of the object. This way we do not have to duplicate all the data in the vector and can access the object by the vector.

```
1  #pragma once
2  #include <vector>
3  #include "Game.h"
4  class Player : public Game
5  {
6
7  private:
8
9      string _playerName;
10     vector <Player*> playerList;
11     int _score;
12
13 public:
14
15     Player();
16     Player(string name);
17
18     void setPlayerName(string name);
19     void setScore();
20
21     void getPlayerName();
22     void getAllPlayers();
23
24     void addPlayer(Player* player);
25
26     virtual string winningMessage() = 0;
27 };
28
29
```

On line 26 a fully virtual string is created; this will be used to display a personalized winning message to the winner.

The turn class is the final of the 3 classes, and this controls the random number, the number of tries and the winningGuess which is used the main while loop in source.cpp.

As with the previous class there are a few methods to set values and get them and a constructor that sets the _winningGuess back to 0 when whenever it's called.

The methods in this class are a little more complex than the previous class. In this one the random number needs to be generated depending on the level selected at the start of the program. A switch is used and is passed the level as an integer from the override in _setRandomNumber. Depending on the value passed the switch assigns a value to the rangeMax which is then passed to line 35 where it's used to generate the random number. This can be seen working in the test section above. As part of the DRY principle the developer

also assigned a value to the `_numberOfTries` , even though there is another method created just for this it is not used.

```
void Turn::setRandomNumber(int level)
{
    int rangeMax = 0;

    switch (level) {
    case 1:
        rangeMax = 10;
        _numberOfTries = 3;
        break;
    case 2:
        rangeMax = 50;
        _numberOfTries = 6;
        break;
    case 3:
        rangeMax = 100;
        _numberOfTries = 10;
        break;
    default:
        rangeMax = 0;
        _numberOfTries = 0;
    }

    random_device dev;
    mt19937 rng(dev());
    uniform_int_distribution<mt19937::result_type> dist(0, rangeMax - 1);

    _randomNumber = dist(rng);
}
```

In the main source.ccp each class is included at the top of the file together with the std namespace.

3 function signatures are also included

Pause() is used to pause the program at the end.

assignmnetDetails() displays some text about the developer and course information.

askGuess () is used to ask the player to make a move , as this was repeated a function was used to keep it simple.

Line 30 – 36 create a new object for Game , Turn and Player (this player one is for the vector playerList) as mentioned in the reflections stage the developer does feel this was the wrong place to have this and that it would be better used in the Game class.

The number of players is set to 2 but because a class is used then in the future this could be changed to ask how many players and the value created dynamically.

Lines 42 – 54 are used to ask to the players name and create the player object and add it to the vector `playerList`. This code does not meet with the DRY principle but at the time the developer was not able to dynamically create an object name. The ideal solution would have been

1. Create for loop based on number of players
2. Loop through asking their name
3. Create object with name player + the number of the player . (One loop and it would be `player0`)
4. Add player to the vector `playerList`.
5. Go to line 2 for next player.

Further down in the number of rounds is asked and using the method in `Game` called `setGameRound(round)` the value is set and used in the while loop. The same is down for the level and passed to `setRandomNumber(levels)` to the number of tries and random number is created.

The main loop consisted of two while loops one for the number of rounds and an inner one for the number of tries. Each player name is displayed, and they are asked for their guess. Their guess is checked against the `randomNumber` and if it does match the message 'We have a winner is displayed on the screen and the method `setScore()` is called which increments the players score.

The main loop could be a lot smaller however the developer was not able to work out how to pass pointers to the function , even though he could get it to work in a smaller sample program and have a result returned it would not work in the code. If it had then the loops would be a lot smaller and most of it would be in a function.

Reflection

The vector `playerList` in the class `game` may have been better suited in the `Game` class, I feel this may have complicated the code by creating another player object just to use it.

I feel that making the game complex by adding in the number of rounds did not help in the design of the program. Looking back, I now feel that I should have kept it as basic as possible and that added features which would have been easier because I was using OOP principles. I felt with time given this was possible however I did not consider projects in work with urgent start dates.

Bibliography

- [1] w3schools, [Online]. Available: https://www.w3schools.com/cpp/cpp_oop.asp. [Accessed 12 2021].
- [2] A. S. Baghel, "c sharpcorner," 29 09 2017. [Online]. Available: <https://www.c-sharpcorner.com/article/software-design-principles-dry-kiss-yagni/>. [Accessed 12 2021].
- [3] D. Powers, PHP object-oriented solutions, friendsof , 2008.
- [4] B. Johnson, Professional Visual Studio 2017, wrox, 2018.

Extra reading

[c++ - Where should I put functions that are not related to a class? - Software Engineering Stack Exchange](#)

[c++ - How to put a break line in string? - Stack Overflow](#)

[Add a Tab to a String in C# | Delft Stack](#)

[C++ Object Oriented \(tutorialspoint.com\)](#)

Julian Milligan ACCB5011 Assignment 1

GRADEMARK REPORT

FINAL GRADE

56/100

GENERAL COMMENTS

Instructor

Classes (25): 18
Polymorphism (10): 0

Game (20): 10

Report (35): 22

Testing (5): 3.5

Demo (5): 2.5

Feedback:
=====

The report needed to provide more discussion of OOP concepts such as inheritance and polymorphism.

The code provides 3 classes but the game is a bit basic. Inheritance is also provided but it is used in a way that does not have much relevance.

Demo should have discussed the code. No audio.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15
