Department of Computer Science

CS1701 Level 1 Group Project: CS1810 Software Implementation

Assignment 3 – Software Implementation

Table of Contents

Assessment Title	Software Implementation
Module Leader	Lela Koulouri
Distribution Date	6 December 2020
Submission Deadlines	12 March 2021, 11:00 (Submission) 16 or 17 March 2021 (VIVA)
Feedback by	25 April 2021
Contribution to overall module assessment	100%
Indicative student time working on assessment	120 Hours
Page Limit	8 pages
Assessment Type (individual or group)	Individual

Main Objective of the Assessment

This assignment is used to assess your abilities to implement a computer program as a solution to a problem, and your ability to communicate your solution.

This assignment contributes 100% to the grade in CS1810 Software Implementation. It is assessed by a member of the teaching team through a 'VIVA' event. In the VIVA, you will demonstrate and explain the computer program that you have implemented.

Description of the Assessment

Each member of the group is expected to implement a Java program for **one** of the game variations described below. Each member of the group must choose a different game variation.

The game descriptions include the required, core functionalities of the program. However, marks will also be awarded for *additional* functionalities that you have come up with. The additional functionalities should be useful and ingenuous and showcase your software development skills. **The additional functionalities should not replace or conflict with the required functionalities of the game descriptions.**

The game variations can be implemented using a Graphical User Interface (for example, windows, buttons, textboxes, drop-down menu, etc.) or a console user interface (input and output printed in the console).
You are allowed to use tools, such as WindowBuilder, to build the GUI (i.e., autogenerate code). However, if you choose to do so, you will receive 0 marks for the GUI.

Game 1: Hangman

Main task:

The program must allow the user to play Hangman against the computer. The computer selects a word at random and the player tries to guess letters in the secret word. The player is given a certain number of 'guesses' at the beginning. The player inputs their guessed letter and the computer either reveals the letter, if it exists in the secret word, or penalises the player by reducing the number of guesses remaining. The game ends when either the player guesses the secret word, or the player runs out of guesses.

At the start of the game, the computer lets the player know how many letters the secret word contains and how many guesses they have. The computer keeps track of all letters the player has not used. After each guess, the computer should tell the player whether the letter is in the word and also display the word with guessed letters shown and unguessed letters replaced with an underscore separated by spaces. For example, if the secret word is 'firkin', and the player has only guessed 'i', the program should display:
_ i_ _ i_

The computer should also try to discourage user errors, by issuing warnings. If the player enters an input that is not a letter (e.g., it is a number) or a letter that they have entered earlier, they should get a warning. Three warnings will reduce the number of remaining guesses by one (this is repeated for every three subsequent warnings that are given). The computer should also tell the player how many warnings they have received up to that point and how many guesses they have left. The game ends if the player correctly guesses all the letters in the secret word or runs out of guesses. If the player runs out of guesses before completing the word, the computer should tell them that they have lost and reveal the word. If the player wins, the computer should print a congratulatory message. Irrespective of whether the player wins or loses, the program should ask the player if they want to play again or quit. If the user wants to play again, the program should make sure previously-used word(s) are not selected again.

Variations:

- **Student A:** The program should select a word at random from a list of at least 100 words stored in a text file. The player can select the difficulty level; the difficulty level can be determined by parameters such as setting the number of guesses initially allocated to the player; and assigning higher penalties (-2 guesses) when the player inputs incorrect vowels.
- **Student B:** After a win, the program should display the score. The score is the number of guesses remaining multiplied by the number of unique letters in the secret word. So, for example, if the player has 2 guesses left and the word had 5 unique letters, their score is 10. The scores are stored in a file, and a leaderboard (ranked record of the highest scores) is kept and displayed.

Ideas for additional functionalities for either variation (optional):

➢ The player is able to guess the full word in one go. A wrong guess of a full word costs two guesses.
➢ The program keeps the player vs computer score of the number of games won; for example, if the player has won a game, the score becomes 1-0. If the player continues playing, the score should be updated after each game accordingly.
➢ Your own ideas.

****

Game 2: Snakes and Ladders

Main task:
The program should allow the user to play Snakes and Ladders against 1-3 other players.

The board has 100 squares, starting from 1 and going up to 100. The players start at square 1. The first player to reach square 100 wins the game. Each player rolls the dice when their turn comes. Based on the dice value, the players move. The dice has six sides.

For example, if a player is on square 10 and they roll 5, they will move to square 15. A player wins if they arrive at/land on square 100, *exactly*. That is, if a player is on square 97, and they roll 4, they cannot move. They must keep playing until they roll exactly 3.

Each player should have a colour or a name assigned to them. So, after each turn, the program should display:

[Lela] rolled a [5] and moved from square [10] to [15].

Before the game begins, the program should place a number of snakes and ladders on the board.

Each snake has a head and a tail on the board. If the player lands on a square where a snake's head is, the player should move down to the square where the end of the snake's tail is. Similarly, if the player lands on a square where the bottom of the ladder is, the player should move up to the square where the top of the ladder is. That is, the head of a snake is always in a square of a higher number than the square where the end of its tail is, and the top of a ladder is always in a square of a higher number than the square where its bottom is.
If a player rolls 6, they move the number given (moving down a snake or up a ladder if moving six squares forward lands on one of them) and then they get an additional turn.
The game continues until all players, but one, reach square 100.

The board also contains two biscuits (one on each of two squares determined by the program before the game begins). If a player lands on a biscuit square they can keep the biscuit and feed it to the next snake that they encounter so that they do not have to move down to the tail square.

It is not a requirement that the board and its contents (snakes, ladders, etc.) are displayed to the players.

Variations:

- **Student A:** The board also contains two big sticks. If a player lands on a big stick square, they can keep the big stick and use it to create an extra step on the next ladder that they encounter, which will move them up 10 squares (one row) on the board from the square where the head of the ladder is. Note: a big stick cannot be used for any ladder the top of which is in the row which includes the 'Finish' square (the row of squares from 91 to 100).
- **Student B:** The board also contains two big sticks. If a player lands on a big stick square, they can keep the big stick and use it to scare away the next snake that they encounter which will result in the snake moving 10 squares (one row) up the board. Note: a big stick cannot be used for any snake the head of which is in the row which includes the 'Finish' square (the row of squares from 91 to 100).

Ideas for additional functionalities for either variation (optional):

- ➢ The snakes and ladders are randomly placed on the board, so they are in different positions every time the users play the game.
- ➢ The player is able to select the game's difficulty level by selecting the number of snakes and their 'length', the number of ladders and their 'height', and/or the number of biscuits/big sticks available on the board.
- ➢ Biscuits and big sticks are removed from the board once they have been picked up. They can be used at any point in the game not just the next time the player who holds the big stick/biscuit lands on the head of a snake/bottom of a ladder.
- ➢ Your own ideas.

****

Game 3: Mastermind

Main task:
The program should allow the user to play Mastermind against the computer. The computer should generate a random code of four colours from a range of six different colours (e.g., Red, Green, Blue, Yellow, Orange, Pink). There should not be any repeating colours in the generated code.

The player should be able to enter a guess, for example 'RGBY'. The program should print '+' if the colour occurs in the code and is in the right position and '-' if the guessed colour is in the code but is in the wrong position.

Any '+' symbols should be displayed first, and, the computer does not reveal which colour in the guess was right. So, for example, if the player's guess was 'GPBY' and the code was 'RGBY', the computer should display:

```
++-
```

The number of allowed guesses is 12.

If the player runs out of guesses, the computer should tell them that they have lost the game and reveal the code. If the player enters the correct code, they win the game, and the computer should print a congratulatory message. Irrespective of whether the player wins or loses, the program should ask the user if they want to play again or quit.

Variations:

- **Student A:** At the start of the game, the player should be able to choose the number of the colours in the range (6-10), the number of colours in the code (3 or 4), and the maximum number of guesses allowed. At the end of each game, a score should be displayed. The score is calculated by taking the number of guesses remaining, adding the number of colours in the range, then adding the number of colours in the code.
- **Student B:** The player should be able to choose whether they want to play with repeating colours in the code or not. If they choose to play with repeating colours, the game should correctly support this through the gameplay.

Ideas for additional functionalities for either variation (optional):

➢ The program keeps the player vs computer score; for example, if the player has won a game, the score is 1-0. If the player continues playing, the score should be updated after each game, accordingly.

➢ Your own ideas.

****

Game 4: Battleship

Main task:

The program should allow the user to play Battleship against the computer. The computer places the following five types of ship of different lengths on a 10 by 10 board. The types of ship and their lengths are shown below:

1. Aircraft carrier: 5 squares long
2. Battleship: 4 squares long
3. Submarine: 3 squares long
4. Destroyer: 3 squares long
5. Patrol Boat: 2 squares long

The computer should randomly place the ships on the board, vertically or horizontally, taking care that no ship overlaps with another ship or is out of the bounds of the 10 by 10 board. The player 'shoots' in order to hit and sink all the ships of the computer.

A ship sinks when all of its squares have been hit.

The player should enter the X,Y coordinates of a shot. The computer should display the appropriate message according to the outcome of the shot:

- My ship was hit!
- You missed!
- You sank my [ship type]!

A list of the sunk and un-sunk ships up to that point should also be displayed.

The 10 by 10 board should be visible to the player, but not the ships. That is, at the start of the game, an empty 10 by 10 board should be displayed. After each shot, the board should be updated by adding an 'M' on the chosen square, if it was a miss, or an 'H', if it was a hit.

The program should keep a running score for the player during the game (the score is set to 0 at the start of the game). Each shot (hit or miss) deducts one point from the score; and each hit adds one point; when a ship is sunk, a number of points equal to the length of that ship multiplied by 2 is added to the score (for example, +10 for the aircraft carrier). The game ends when the player has sunk all of ships or if the player enters 'Quit'. At the end of a game, the computer should display the score of the player and ask the player if they want to play again.

Variations:

- **Student A:** The player has four radars which they can drop at an X,Y position on the board. The radars can detect a ship that is one square away from its position. The grid below shows an example in which the radar is dropped ('o' in the grid), the yellow squares show its detection radius, and that a destroyer (the red squares) ship is therefore detected by the radar.

			🟨	🟨	🟥	🟥	🟥		
			🟨	o	🟨				
			🟨	🟨	🟨				

  To drop a radar, the player has to enter an X,Y position. The computer should display whether a ship has been detected or not, but not in what direction. The computer should also display how many radars the player has left.
- **Student B:** The board also contains two sea monsters, Kraken and Cetus, which will be annoyed if hit. If Kraken is hit, it will consume all of the points in the player's score at the time at which it is hit. If Cetus is hit, it will cause all un-sunk ships on the board to move to different places on the board.
  Each sea monster should be placed on a random square (which is not occupied by a ship) immediately after the computer has placed the ships on the board.

Ideas for additional functionalities for either variation (optional):

- Display and store player statistics, such as hit-to-miss ratio and number of shots.
- Leaderboard.
- Your own ideas.

For this assignment, you must submit ***two*** files:

1. A ***Java program*** that implements the functionalities specified in the description of your chosen game variation.
   You must export the Eclipse project as ZIP and submit the ZIP file. Here is a link to a video that shows you how to export an Eclipse project as ZIP.

2. A PDF ***report*** that has the following contents:
   a) **Cover page** indicating the module and assessment block codes and titles; the assignment title; the number and title of the game, the variation that you have undertaken (A or B); your name; your ID (e.g., 2012345); the name of your tutor; and your group name (e.g., Green05).

   b) **Requirements Specification**
   State whether your program includes all required functionalities for your chosen game variation. If not, state which functionalities it omits.
   Make a list of any additional functionalities that you have included in the program.

   c) **Algorithm and User Interface Design**
   State whether your program follows the algorithm design and User Interface design ('prototype') that you have previously submitted.

If the program does not follow the design, indicate what changes you had to make to your design. Briefly explain why you had to change your design. For example, a reason could be that your tutor gave you feedback indicating that the original algorithm had an error, or that you had more time to think about and optimised the original algorithm, or that you were not able implement all the required or additional functionalities that your design included.

d) Testing
Use a table to show how you tested your program; that is, list the functionalities you tested; the input that you used to test each functionality with; the expected and observed output of the program (actual results or program behaviour); whether the test passed/failed; and the reasons in case of failure.

e) Source Code Listing
Copy-paste all your code in this section. Do not add screenshots.

The PDF report should not exceed eight pages (not including section 'e) Source Code Listing').

The deadline for the submission of this assignment (Java Program and Report) is 12 March 2021, 11:00.

- **VIVA**

The aim of the VIVA is to confirm that you wrote the program that you have submitted.
At the VIVA, you will be presented with the code that you have submitted. Then, you will be asked to demonstrate that the code runs and how it works. Then, you will be asked to explain what parts of the code do. You may also be asked to make some changes to the code.
In order to pass the VIVA:

i) You must attend the VIVA
ii) Your code must run at the VIVA
iii) You must provide adequate and convincing explanations on:
   - How the code works.
   - What randomly selected parts of the code do.

If you do not attend the VIVA, or your code does not run, or you do not provide adequate and convincing explanations at the VIVA, you will fail the VIVA and your grade in this assignment (CS1810 – Software Implementation) will be set to Fail.

University procedures and policies relating to extenuating circumstances apply.

The VIVA will take place on 16-17 March 2021 and attendance is compulsory.

- **Code Integration Task (Assignment 1 – Group Project Review)**

The Code Integration Task is part of Assignment 1 – Group Project Review.
For the Code Integration Task, you should try to integrate the code produced by all, or some, of your group members, such that it runs as a single program.

For example, you could make a Graphical User Interface with buttons corresponding to each of the Game variations implemented by your group members (i.e., 'Hangman A', 'Hangman B', 'Mastermind A', 'Mastermind B', etc.). When the user presses a button (for example, the 'Battleship A' button), the program should execute the code for variation A of the Battleship game.

Another solution could be that the program displays a message such as 'Choose a game' in the console, and the user types in the title of the game they would like to play.

The Code Integration Task is submitted and assessed as part of Assignment 1 – Group Project Review.

LEARNING OUTCOMES AND MARKING CRITERIA

Learning Outcomes
In order to get a pass grade (D- or above) in this assignment, you must meet the learning outcomes below, that is, you must demonstrate ability to:

LO1: Plan, manage and track a non-trivial activity.
LO2: Take an open-ended problem and define and refine the requirements.
LO3: Effectively present and communicate solutions to their peers.
LO4: Create and use technical documentation.
LO5: Produce a working computer program as a solution to a given non-trivial problem.
LO6: Develop skills in the use of high-level object-oriented languages.

The learning outcomes, marking criteria, and weighting for each element are shown in the table below.

Assignment Element	Learning Outcome	Marking Criteria	Weighting
Software Implementation (CS1810)	LO1, LO4, LO5, LO6	Your grade will be based on the following aspects: *1. Java Program (85%)* i) The program has all the required functionalities, and everything works as specified. ii) The program includes additional functionalities that are useful, ingenuous and complex. iii) Code quality (good programming practices, efficiency, appropriate structures and libraries, comments and format). iv) Appropriate use of object-oriented programming features. v) User interface quality (usability). *2. Report (15%)* i) The implementation is consistent with the algorithm and User Interface designs. If not, adequate explanation and justification were provided. ii) Testing is thorough and clearly presented.	100% (Pass/Fail subject to **VIVA** outcome)
Please refer to the rubric on WISEflow for more details on the marking criteria.			

FORMAT OF THE ASSESSMENT

This assignment is individually assessed. After submission is made, you will have to demonstrate and explain your Java program during a VIVA event. The VIVA will take place on 16-17 March 2021.
You will receive feedback on your work and a grade on WISEflow in week 30 (25 April 2021).

SUBMISSION INSTRUCTIONS

You must submit:

1. A ZIP file consisting of the Eclipse project with your **Java program**.

It must be uploaded via WISEflow by **11:00GMT, Friday 12th March 2021.**
Your student ID number must be used for the file names (e.g. 2012345.zip).
Here is a link to a video that shows you how to export an Eclipse project as ZIP.

2. Your **report** as a PDF file.

Your report should be a single PDF file no longer than 8 pages.
As indicated above, your report should contain the following:

a) Cover page indicating the module and assessment block codes and titles; the assignment title; the number and title of the game, the variation that you have undertaken (A or B); your name; your ID (e.g., 2012345); the name of your tutor; and your group name (e.g., Green05).
b) Requirements Specification
c) Algorithm and User Interface Design
d) Testing
e) Source Code Listing (please copy-paste the code, do not take screenshots).

It must be uploaded via WISEflow by **11:00GMT, Friday 12th March 2021.**
Your student ID number must be used for the file names (e.g. 2012345.pdf)

AVOIDING PLAGIARISM

Please ensure that you understand the meaning of plagiarism and the seriousness of the offence. Information on plagiarism can be found on the College's Student Handbook.

LATE COURSEWORK

The clear expectation is that you will submit your coursework by the submission deadline stated in the study guide. In line with the University's policy on the late submission of coursework (revised in July 2016), coursework submitted up to 48 hours late will be accepted, but capped at a threshold pass (D- for undergraduate or C- for postgraduate). Work submitted over 48 hours after the stated deadline will automatically be given a fail grade (F). Please refer to the Computer Science Student Handbook, available on Blackboard Learn, for information on submitting late work, penalties applied and procedures in the case of extenuating circumstances.