Advanced Programming Techniques
COSC1076 | Semester 1 2022
Assignment 2 | Implementing Scrabble

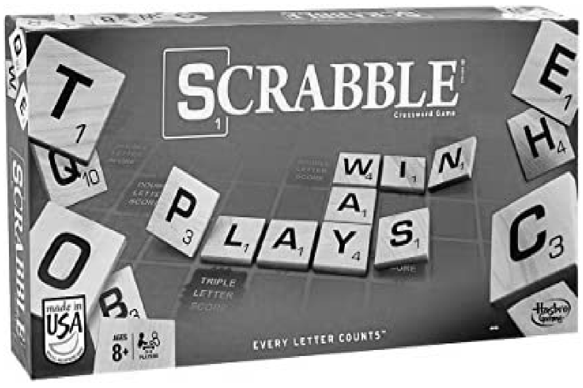
Assessment Type	Assessment contain both group (Milestones 1 & 2) and individual (Milestone 3 & 4) components. Clarifications/updates may be made via announcements/relevant discussion forums.
Due Date Milestones 1 & 2	11.59pm, Friday 13 May 2022 (Week 10)
Due Date Milestones 3 & 4	11.59pm, Friday 27 May 2022 (Week 12)
Silence Policy	From 5.00pm, Thursday 26 May 2022
Weight	45% of the final course mark
Submission	Online via Canvas. Submission instructions are provided on Canvas.

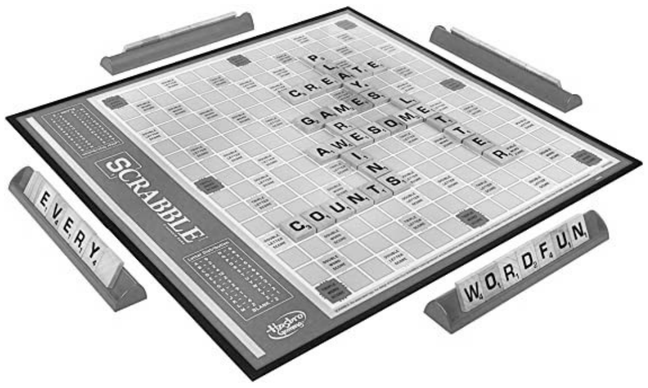
Contents

1 Introduction

1.1 Overview

In this assignment you will implement a ***2-player*** text-based version of the Board Game `Scrabble`.


(a) Scrabble box and pieces


(b) Example game state

For an explanation of the rules and gameplay:

- Rules explanation of full game: https://youtu.be/lG1QAyWvKlQ
- Rules: Available at (https://scrabble.hasbro.com/en-us/rules).

However, this assignment will use a modified version of the rules, detailed in Section 2.5.

In this assignment you will:

- Practice the programming skills covered throughout this course, such as:
  - ADTs
  - Linked Lists
  - Pointers
  - Dynamic Memory Management
  - File Processing
  - Program State Management
  - Exception Handling
- Practice the use of testing
- Implement a medium size C++ program:
  - Use features of C++14
  - Use elements of the C++ STL
- Work as a team
  - Use group collaboration tools

This assignment is divided into four Milestones:

- **Milestone 1 (Group work):** Test Cases, to be developed to ensure your `Scrabble` implementation is correct.
- **Milestone 2 (Group work):** A fully functioning implementation of the base `Scrabble` game play, which pass Milestone 1 tests. The group work (milestone 1 & 2) is worth 30% of the course mark. The group component (M1 & M2) is due 11.59pm, Friday 13 May 2022 (Week 10).
- **Milestone 3 (Individual work):** You will *individually* extend upon your group's implementation with additional functionality (called enhancements). The individual work is worth 15% of the course mark. The individual component (M3) is due 11.59pm, Friday 27 May 2022 (Week 12).
- **Milestone 4:** Written report & demostration. The report analysing the design and implementation of your software, and the use of your test cases. You will demonstrate your group and individual work. This is where your final work will be graded. The report is due 11.59pm, Friday 27 May 2022 (Week 12).

Group Progress Update (Group work): Your group will provide regular updates on your progress in this assignment to your tutor during your weekly lab classes. This will require your group to have completed a list of activities. Group Progress Update will not be marked directly, however, this will influence the final grade.

1.2 Group Work

The group work must be completed in groups of 4.

1. You may form groups with any student in the course.
2. We **strongly recommend** that you form groups from within your labs, because:
   - Your tutor will help you form groups,but only within your lab.
   - You will have plenty of opportunity to discuss your group's progress and get help from your tutor during the rest of the course. It will be extremely helpful for your whole group to be present, but this can't happen if you have group members outside the lab.

Groups for Assignment 2 must be **registered with your tutor by week 8 lab**[1]. Your tutor "register" your group on Canvas. If you are unable to find a group, discuss this with your tutor as soon as possible.

If at any point you have problems working with your group, inform your tutor immediately, so that issues may be resolved. This is especially important with the online delivery of the course. We will do our best to help manage group issues, so that everybody receives a fair grade for their contributions. To help with managing your group work we will be requiring your group to use particular tools. These are detailed in Section 5.

There are important requirements about keeping your tutor informed if you have been unwell or other wise unable to contribute to your group. Remember your actions affect everybody in your group.

1.3 Learning Outcomes

This assessment relates to all of the learning outcomes of the course which are:

- Analyse and Solve computing problems; Design and Develop suitable algorithmic solutions using software concepts and skills both (a) introduced in this course, and (b) taught in pre-requisite courses; Implement and Code the algorithmic solutions in the C++ programming language.
- Discuss and Analyse software design and development strategies; Make and Justify choices in software design and development; Explore underpinning concepts as related to both theoretical and practical applications of software design and development using advanced programming techniques.
- Discuss, Analyse, and Use appropriate strategies to develop error-free software including static code analysis, modern debugging skills and practices, and C++ debugging tools.
- Implement small to medium software programs of varying complexity; Demonstrate and Adhere to good programming style, and modern standards and practices; Appropriately Use typical features of the C++ language include basic language constructs, abstract data types, encapsulation and polymorphism, dynamic memory management, dynamic data structures, file management, and managing large projects containing multiple source files; Adhere to the C++14 ISO language features.
- Demonstrate and Adhere to the standards and practice of Professionalism and Ethics, such as described in the ACS Core Body of Knowledge (CBOK) for ICT Professionals.

2 Base Program Game play & Functionality

The base `Scrabble` program implements a 2-player text-based version of `Scrabble`, using a reduced rule-set. In the base game, the players take turns placing tiles from their hand onto the board. The rule changes for the base `Scrabble` game are described in Section 2.5.

This section details the behaviour of the base `Scrabble` program. What is presented in this spec is a description of the main functionality of your `Scrabble` program. Some parts are left open for you to decide the best course of action.

This spec does not give the rules of `Scrabble`. Canvas contains a link to the rules.

> ! Aspects of this specification are flexible and open to your interpretation. In general, where there is flexibility, it is up to you to determine the best course of action. You may ask questions on the forum for clarity. Make sure that your tests are written to ensure your program works correctly based on any decisions you make.

2.1 Launch

Your base `Scrabble` program will be run using the following terminal command:

[1] If your group spans multiple labs, have one of your tutors register the group.

```
$ ./scrabble
```

On launch, the program should display a welcome message:

```
Welcome to Scrabble!
-------------------
```

Following the welcome message, the program should continue to the **main menu**.

2.2 Main Menu

The main menu shows the options of your `Scrabble` program. By default there should be 4 options. The menu is followed by the *user prompt*.

```
Menu
----
1. New Game
2. Load Game
3. Credits (Show student information)
4. Quit

>
```

The user selects an option by typing a number, and pressing enter. Each menu option is described below. The user prompt is described in Section 2.4, including what to do for invalid input.

2.2.1 New Game

The program should:

1. Print a message for starting a new game
2. Ask for the player names
3. Create a new game of `Scrabble`,
4. Proceed with normal gameplay.

As an overview, this process may look like:

```
> 1

Starting a New Game

Enter a name for player 1 (uppercase characters only)
> <user enters name>

Enter a name for player 2 (uppercase characters only)
> <user enters name>

Let's Play!
<normal gameplay continues from here>
```

The players should only consist of letters (no numbers or symbols). Your program should validate (check) that the player name is valid.

The `Scrabble` gameplay is described in Section 2.3. Make sure you take note of the requirements for starting a new game, described in Section 2.3.11.

2.2.2 Load Game

The program should first ask the user for a filename from which to load a game.

```
> 2

Enter the filename from which load a game
>
```

The user enters the *relative path* to the saved game file, and presses enter.

After the filename is provided, the program must then conduct two validation checks:

1. Check that the file exists.
2. Check that the format of the file is correct. The format for saved games is described in Section 2.3.8.

If the filename passes both checks, the program should print a message, then load the game as described in Section 2.3.13, and continue with normal gameplay as described in Section 2.3.

```
> <filename>

Scrabble game successfully loaded
<game play continues from here>
```

2.2.3 Credits (Show student information)

The program should print the name, student number, and email address of each student in the group (one set if individual project), separated by new lines. Note that you should replace `<full name>`, `<student number>` and `<email address>` sections with your full name, student number and student email address.

After printing the student details, the program should return to the main menu.

```
> 3

----------------------------------
Name: <full name>
Student ID: <student number>
Email: <email address>

Name: <full name>
Student ID: <student number>
Email: <email address>

<Student 3, etc.>
----------------------------------

<main menu>
```

2.2.4 Quit

The program should print a goodbye message, and safely terminate *without crashing*.

```
> 4

Goodbye
```

2.3 Base Gameplay

During base `Scrabble` gameplay, the 2 players take turns placing tiles from their hand onto the board.

At the start of the player's turn, the program should show (in order):

1. The name of the current player
2. The scores of both players
3. The state of the board
4. The tiles in the current player's hand

5. The user prompt

The current player may then take one of two actions:

1. Place tile(s) onto the board to form a word
2. Replace one tile in their hand
3. Pass. Miss the turn and proceed to the other player.

Once the player successfully takes their action, their turn ends, and the other player's turn starts.

Alternatively, the player may perform one of two game functions:

1. Save the game to a file
2. Quit the game

Below is an example of a sequence of actions and game functions for two players, named `A` and `B`. The next sub-sections, describe the individual aspects of the base gameplay. *The board size is made small in the example to make the visualization clear. However you will implement a board that is 15x15 in size.*

```
< previous output >

A, it's your turn
Score for A: 6
Score for B: 6
    0   1   2   3   4   5
  -------------------------
A |   |   |   |   |   |   |
B |   |   |   | W | I | N |
C |   |   |   | A |   |   |
D | P | L | A | Y | S |   |
E |   |   |   |   |   |   |
F |   |   |   |   |   |   |

Your hand is
Y-4, C-3, A-1, B-3, R-1, S-1, E-1

> place C at C1
> place A at E1
> place Y at F1
> place Done

B, it's your turn
Score for A: 14
Score for B: 6
    0   1   2   3   4   5
  -------------------------
A |   |   |   |   |   |   |
B |   |   |   | W | I | N |
C |   | C |   | A |   |   |
D | P | L | A | Y | S |   |
E |   | A |   |   |   |   |
F |   | Y |   |   |   |   |


Your hand is
E-1, A-1, H-4, J-8, O-1, G-2, W-4

> replace A

< gameplay continues >
```

2.3.1 Tile Codes

Tiles are represented by a code:

```
<letter>-<value>
```

The value of all the tiles are given in the table below.

Value	Letter
1	A, E, I, O, U, L, N, S, T, R
2	D, G
3	B, C, M, P
4	F, H, V, W, Y
5	K
8	J, X
10	Q, Z

For example, the letter `M` tile has a value of: `3`

2.3.2 The Board

The display of the game board consists of two features:

1. Tile Display
2. Grid Co-ordinates

The board is a 2D grid, with size of 15x15. However, some locations of the 2D grid may be empty, meaning that no tile has been placed there. When the board is displayed, all locations that contain a tile are filled with the tile letter (letter only no value) and two spaces on either side, and empty locations filled with three-spaces.

The grid co-ordinates use:

- Uppercase Letters for rows
- Integers for columns

Board locations are always referenced in row-column fashion:

```
<row><column>
```

For example, the `W` is at grid co-ordinate `B3`, and the `I` is at grid co-ordinate `B4`.

```
    0   1   2   3   4   5
  -------------------------
A |   |   |   |   |   |   |
B |   |   |   | W | I | N |
C |   |   |   | A |   |   |
D | P | L | A | Y | S |   |
E |   |   |   |   |   |   |
F |   |   |   |   |   |   |
```

You could store the board as A vector of vectors of Tiles

In your implementation you ***must*** use a ***vector*** to store the board.

2.3.3 The Player's Hand

The player's hand is an *ordered* ***linked list*** *of tiles.*

In your implementation you ***must*** use a ***linked list*** to store the tiles in the player's hand. You must implement ***your own*** version of a Linked List.

The player's hand is displayed as a comma separated list of tiles. When displaying the hand, the value of each letter is also displayed.

```
Your hand is
Y-4, C-3, A-1, B-3, R-1, S-1, E-1
```

The order of tiles in the player's hand is important for testing purposes!

- When *adding* a tile, it is always added at the *end* of the list.
- When removing a tile, the *remaining* tiles stay in the *same order*.

2.3.4 The Tile Bag

The tile bag, contains the rest of the tiles that are not on the board or in player's hands. The tile bag must be stored as an *ordered* ***linked list***. The contents of the tile bag is *never* displayed to the users. However, the contents of the tile bag *is stored* in the saved game file.

! In your implementation you ***must*** use a ***linked list*** to store the tiles in the tile bag. You must implement ***your own*** version of a Linked List.

The order of the tile bag is determined when generating a new game. When a tile is drawn from the bag, it is taken from the *front of the linked list*. If tiles are added to the bag, they are added to the *end of the linked list*.

At the start of the game, the tile bag consists of 98 tiles. The number of tiles from each letter in the initial tile bag is given below (e.g. `Ax9` means that there are 9 tiles with letter `A`):

Ax9, Bx2, Cx2, Dx4, Ex12, Fx2, Gx3, Hx2, Ix9, Jx1, Kx1, Lx4, Mx2, Nx6, Ox8, Px2, Qx1, Rx6, Sx4, Tx6, Ux4, Vx2, Wx2, Xx1, Yx2, Zx1.

The `ScrabbleTiles.txt` file in the starter code contains the list of tiles (letter and value) that can be loaded at the start of the game.

2.3.5 Player Action: Place Tile(s)

The current player will place multiple tiles (up to a maximum of 7 tiles in one turn) by repeating the command.

`place <tile1> at <grid location>`

The command contains two elements (should accommodate any number of white spaces in between):

1. A tile to place
2. The grid location to place the tile

When he is done placing tiles for the current turn, the current player should type:

`place Done`

For example, using the above hand and board, if the player performs:

```
> place C at C1
> place A at E1
> place Y at F1
> place Done
```

This results in the board:

```
    0   1   2   3   4   5
  -------------------------
A |   |   |   |   |   |   |
B |   |   |   | W | I | N |
C |   | C |   | A |   |   |
D | P | L | A | Y | S |   |
E |   | A |   |   |   |   |
F |   | Y |   |   |   |   |
```

After the command is given, the program must:

1. Check that the command is correctly formatted.
2. Check that the placement of tile is legal according to the rules of `Scrabble`. (No requirement to check if the word is a valid scrabble word - can be done if you are interested.)

If the player's action is *legal*, the program should:

1. Place the tile(s) onto the board

2. Update the player's score
3. Draw a replacement tile from the tile bag and add it to the player's hand, if there are available tiles
4. Continue with the other player's turn

2.3.6 Player Action: Replace a Tile

The current player may replace one tile in their hand using the command:

```
replace <tile>
```

For example, using the above hand the player may take the following replace action:

```
> replace A
```

After the command is given, the program must:

1. Check that the command is correctly formatted.
2. Check that the tile is in the player's hand.

If the player's action is legal, the program should:

1. Remove the tile from the players hand and place it in the tile bag. (If the player has two tiles with the same code, the first tile in the list should be replaced)
2. Draw a new tile from the tile bag and add it to the player's hand
3. Continue with the other player's turn

2.3.7 Player Action: Pass

A player can pass his turn by typing the command:

```
> pass
```

The turn is handed to the next player. No changes to the points.

2.3.8 Function: Saving the Game

The current player may save the game to a file using the command:

```
save <filename>
```

The program should save the current state of the game to the provided filename (overwriting the file if it already exists). Then the program should display a message and continue with the gameplay. The current player does not change, so that a player may save the game and then take a turn.

```
> save savedGame

Game successfully saved

>
```

If the program has problems saving the file, it should display a message, and continue with normal gameplay *without crashing.*

The format of the saved file is as given below. Each item is saved on a new line.

```
<player 1 name>
<player 1 score>
<player 1 hand>
<player 2 name>
<player 2 score>
<player 2 hand>
<Board State>
<tile bag contents>
<current player name>
```

The format for each of the items is:

- Name: ASCII text
- Score: Integer
- Player hand and tile bag: comma separated ordered list
- Current board shape: Height, width
- Board State: All tiles currently placed on the board should appear as a list of tile@position.

For example, if the game in Section 2.3 was saved the saved game file will look like:

```
A
14
B-3, R-1, S-1, E-1, H-4,G-2, O-1
B
6
E-1, H-4, J-8, O-1, G-2, W-4, R-1
    0   1   2   3   4   5
  -------------------------
A |   |   |   |   |   |   |
B |   |   |   | W | I | N |
C |   | C |   | A |   |   |
D | P | L | A | Y | S |   |
E |   | A |   |   |   |   |
F |   | Y |   |   |   |   |
P-3, B-3, F-4, .....
A
```

2.3.9 Function: Quit

The program should quit *without crashing*, as per the instructions in Section 2.2.4.

2.3.10 Special Operation: BINGO!

If the player places all seven tiles in the hand to form a word on their turn, then the program should print out an additional message, before displaying the game information. Remember to update the player's score accordingly. A bonus will have additional 50 points.

```
> place .. at ..

BINGO!!!

<display game>

>
```

2.3.11 Special Operation: Starting a New Game

When a new game is started, a special sequence of operations must be conducted:

1. Create the ordering for the tile bag
2. Set up the initial player hands
3. Start with an empty board, with player 1 as the starting player

You will need to devise *your own algorithm* to "shuffle" the bag of tiles to create a "random" initial order. This is left up to your own invention. The lectures will talk about randomness is C++ programs.

Then the initial tiles are added to the player's hands. 7 tiles are drawn from the tile bag and placed in the 1st player's hand. Then 7 tiles are drawn from the tile bag and placed in the 2nd player's hand.

Finally, the board starts with no tiles placed, so that when displayed, it should be empty.

2.3.12 Special Operation: Ending a Game

The game ends when:

1. The tile bag is empty, and
2. One player has no more tiles in their hand or passes his turn twice.

If the game ends, the program should:

- Display the end game message
- Display the scores
- Display the name of the winning player
- Then quit, according to Section 2.2.4.

For example:

```
Game over
Score for <player 1 name>: 000
Score for <player 2 name>: 000
Player <winniner player name> won!

Goodbye
```

2.3.13 Special Operation: Loading a Game

To load a game from a saved game file, the program should read the contents of the saved game file, and update all data structures in the program using the information in the saved game file. See Section 2.3.8 for the format of the saved game file. Specifically, the program should take note of:

- The player's name and scores
- The tiles in each players hand
- The state of the board
- The order of the tiles in the tile bag
- The current player - the next player to take a turn

Once the game has been loaded, gameplay continues resumes with the current player.

2.4 User Prompt

The user prompt is displayed whenever input is required from the user. It is a greater-than symbol (>), followed by a space. It is assumed that all user inputs are provided as a single line of input.

When shown the prompt, the user should see in their terminal window:

```
> █
```

If at any point the user enters *invalid input*, or the validation checks of the input fail (see each section) then the program should print `Invalid Input` and re-show the prompt.

```
> dome
Invalid Input
> █
```

2.4.1 EOF Character

If an any time the user enters the EOF (end-of-file) character, the the program should Quit, following the procedure in Section 2.2.4. That is:

```
> ^D

Goodbye
```

This behaviour with the EOF character is necessary to ensure your program terminates at the end of every test case.

2.5 Rule Changes

For the base `Scrabble` implementation, the following rules have been modified:

- A New game always begins with Player 1 and an empty board
- Players can only replace one tile at a time.
- There are no blank tiles. (The original game has two bank tiles in the tile bag).
- There are no "Premium Squares" on the board. Each board position is equal.
- Only the letters placed on the board contribute towards points. The tiles left in each player at the end of the game is not used in computing the points.

3 Deliverables

3.1 Mandatory Requirements

As part of your implementation, you ***must***:

- Implement your own Linked List
- Use your Linked List implementation to store the player's hands and the tile bag.
- You should use vectors to store the current board state.
- You may only use the C++14 STL. You may not incorporate any additional libraries.

If you fail to comply with these mandatory requirements, marks will be deducted.

3.2 Milestone 1: Test Cases (Group Component)

For Milestone 1, you must develop test cases for your `Scrabble` implementation, including your enhancements. These test cases will help ensure that your `Scrabble` implementation is correct.

A single test case consists of 4 files, 2 mandatory and 2 optional.

1. `<testname>.input` - Input to provide to the `Scrabble` program via `stdin`
2. `<testname>.output` - Expected output from the `Scrabble` program on `stdout`
3. (Optional) `<testname>.save` - Input save file to provide to the `Scrabble` program if required by the test.
4. (Optional) `<testname>.expsave` - Expected output saved game file.

A test is run using the following sequence of commands.

```
./scrabble < <testname>.input > <testname>.gameout
diff -w <testname>.output <testname>.gameout
if [-e <testname>.expsave] diff -w -y <testname>.expsave <actual_gamesave>
```

If this command displays any output, then the test has failed. **Testing uses the `diff` command. This command checks to see if two files have any differences. The `-w` options ignores any white-space.**

To make testing reliable, you should note if the test evaluates the saved game output, then ensure the test uses a suitable filename in place of `<actual_gamesave>`.

! The `Scrabble` program has some degree of randomness due to the random ordering of the tile bag. To overcome this issue when testing, when appropriate, you may start your test with loading a saved game (will contain the ordering of the tile bag, hence the execution from that point is deterministic).

3.3 Milestone 2: Basic `Scrabble` Implementation (Group Component)

For Milestone 2, your group must implemented the base `Scrabble` program as described in Section 2.

Section 2 lists the components that your group must implement. Generally, it is up to your group to decide the best way to implement these components. However, there are some important requirements that your group must satisfy.

Aspects of this specification are flexible and open to your interpretation. It is up to your group to determine the best course of action. You will need to analyse and justify your choices in the report.

3.3.1 Requirements

Your group will implement a game of `Scrabble` that:

- Is a **2-player** game.
- Both players are "human users" that play the game by interacting with the terminal, that is, through standard input/output.
- Using the `Scrabble` board of fixed size.

You will implement a simplified version of `Scrabble`. These changes are listed in section 2.5.

Your implementation should provide the following functionality:

- A main menu, that allows users to perform actions such as setting up a new game, loading an existing game, showing "credits" (details of the people who wrote the software), and quitting the program.
- Save a game to a file.
- Load a previously saved game from a file, and resume game play from the saved state.
- A way to represent and display the `Scrabble` board and player hands the user.
- A User prompt for entering all commands from standard input.
- Base gameplay functionality as described in Section 2.3.
- The program should terminate without crashing if the EOF character is given on standard input.
- Completely error free. Your program must not crash, segfault or otherwise contain logic errors.

Your group will also need to consider the **data structures** that are used to represent aspects of `Scrabble`, and functionality. It is up to your group to make this decision provided that you meet the **requirements** in section 3.1. In your report your group will be marked on your **analysis** of the above choices.

Finally, remember to do the the group contribution spreadsheet described in 5.3.

3.4 Milestone 3: Enhancements (Individual Component)

In Milestone 3, as an individual you will make signification expansions(s) to the functionality of your group's `Scrabble` program. This milestone is your opportunity to showcase to us your skills, capabilities and knowledge! **You will select your enhancements from the provided options in Section 4**. Additionally, if you group's Milestone 2 solution has significant errors or is significantly incomplete, please read Section 4.3.

Milestone 3 is a very open-ended. You are given some directives, however, there is a lot of room for you to make considered choices. However, this showcase of your skills is not just about "making the code work". A major focus is on how you choose to implement an enhancement, and the justifications of the reasons why you chose a given data structure, class hierarchy, language feature, or algorithm to name a few examples. Enhancements are classified as minor or major. Enhancements must be substantially functional to get marks.

To get a higher grade you will need to implement one or more minor or major enhancements. This is described in the marking rubric.

3.4.1 Configurable Enhancements

Where reasonably possible, your enhancements should be configurable. That is, it should be possible to enable/disable each enhancement at run-time (not through compilation). In particular, this means your enhancements can be "turned-off" so that your program runs the same as Milestone 2 (verbatim).

3.4.2 Changes to Saved Game file

Some enhancements may require you to modify the format of the saved-game file. However, it is recommended that your program with enhancements *still supports* loading games from the default saved-game format used in Milestone 2.

To distinguish your new saved-game format from the "default" milestone 2 format, we recommend adding a special code at the start of *your new* saved-game format, such as:

```
#myformat
<initial tile bag>
...
```

3.5 Milestone 4: Written report & Demonstration (Group + individual Component)

3.5.1 Written report

Your group must write a report, that *analyses* what your group has done in this assignment. Additionally, each **individual** must write a short report that *analyses* their individual enhancement(s). The group should combine the group report and the individual report and one group member should submit it via canvas. The combined report is due at the same time as the individual submission (11.59pm, Friday 27 May 2022 (Week 12)).

- The report should be A4, 11pt font, 2cm margins and single-space.
- The section of the report describing the group's work must be no more than **4 pages**.
- Each individual must add **1 additional page** about their enhancements.
- Thus the final report **must not exceed 8 pages** for a group of four.
- Only the first 4 pages (group), and 1 page (individual) will be marked.
- Modifying fonts and spacing will count as over length.
- Figures, Tables and References count towards these page limits.

In this assignment, you are marked on the analysis and justification of the choices you have made. Your report will be used (in conjunction with the demonstration) to determine the quality of your decisions and analysis.

Good analysis provides factual statements with *evidence* and *explanations.*
Statements such as:

> *"We did <xyz> because we felt that it was good"* or *"Feature <xyz> is more efficient"*

do not have any analysis. These are unjustified opinions. Instead, you should aim for:

> *"We did <xyz> because it is more efficient. It is more efficient because ... "*

We are asking for a combined report as it keeps the context of each individual's enhancements with the whole group's original implementation.

3.5.2 Group Component of the Report

In the **group** section of your report, you should discuss aspects such as:

- Your group's use of *at least one* linked list, and the reasons for where the linked list is used.
- Your group's choices of ADTs, and how these leads to a "well designed" program.
- The efficiency of your implementation, such as the efficiency of your use of data structures.
- The reason for each of your tests.
- Your group co-ordination and management.

3.5.3 Individual Component of the Report

In the **individual** section of your report, you should discuss aspects such as:

- The design of your enhancements, including any changes (and additions) to the data structures and ADTs you had to make when enhancing your group's implementation.
- The efficiency of your enhancements, such as the efficiency of your use of data structures.
- Limitations and issues that you encountered with your group's implementation.

3.5.4 Demonstration

The demonstration will be conducted virtually. More information on the demonstration and how to conduct it will be provided in week 10.

4 Suggested Enhancements for Milestone 3

4.1 Minor enhancements

Minor enhancements are smaller in scope, and require only a small modification to your software design.

4.1.1 Help!

Whenever there is a user prompt, the user may type "`help`" and the program should display some text to help the user determine what they commands they may execute.

4.1.2 Better Invalid Input

Whenever the user enters invalid input at the user prompt, the program should show a useful error message to explain why the input was invalid.

4.1.3 Colour

Use colour to display tiles on the board and in the player's hand. The Linux, Mac (and most similar) terminals support the use of colour through the use of *escape codes*. A simple internet search will show you tutorials for working with escape codes.

4.1.4 Simple Hints

Provide a hint to the user of where a given tile could be placed on the board, and how many points the user will score. You will need to invent your own command so the user can ask for a hint.

4.1.5 Expandable Board

The board should only display enough rows and columns to fully show the board, plus one additional empty row and column around the edge of the board. Any additional empty rows and columns should not be displayed.

4.1.6 High Scores

The program maintains a list of high scores achieved by any player who has played your game of `Scrabble`. You will need to devise a way to save, load, and display the high scores.

4.2 Major enhancements

Major enhancements are large in scope, and require significant modifications to your software design.

4.2.1 3-4 Player modes

Implement the 3 or 4-player mode of `Scrabble`. You will need to change the number of tiles in each player's hand according to the rules of `Scrabble`. This will also affect the format of the saved game file.

4.2.2 Check correctness of player words

check the correctness of the word played by a player against a given scrabble dictionary. The dictionary file will be provided on canvas.

4.2.3 Write an AI

It would be nice to play `Scrabble` in a single-player mode against the computer. This enhancement requires you to develop an AI (Artificial Intelligence) so that your program can be used in single-player mode, and a person can play against the computer AI. When it's the AI's turn, it should make its move automatically without the user having to input a command.

An AI implies *intelligence*. An AI doesn't take random actions. It uses logic and *heuristics* to determine a "good" move, and hopefully the "best" action to take. Heuristics are ways to "guess" or "estimate" if an action is good or bad. Thus, your AI needs to have "intelligence" to figure out a good action to take on its turn. You may not be able to decide the "optimal" action, however, the choice must be better than a random action.

For this enhancement, you may wish to consider:

- You may need to *redesign and change* the data structures that are used to represent `Scrabble`. However, you must *still* meet the mandatory minimal use of each data structure as given in Milestone 2.
- The AI should not take a long time to calculate its move.
- You will need to modify the saved-game format to record if an AI is being used.

In your *individual* report you should justify why your AI has a "good" heuristic and is intelligent. You should also describe any *significant* changes to the choice of data structure(s) describe and justify any other necessary changes to your group's `Scrabble` program.

We recommend that the AI is enabled using a command-line argument, such as:

```
$ ./scrabble --ai
```

(Note two dashed are used in due to the Unix command-line argument styles for multi-letter arguments.)

4.3 Milestone 2 Code with Significant Errors

This section is a ***general*** *statement. This is provided as a* ***starting point*** *from which to approach your tutor. This option may* ***only*** *be used if you have discussed the matter with your tutor.*

It is possible that your group's `Scrabble` program may contain errors. If the errors in the your group's `Scrabble` program are small, while you should fix these in your individual work, they are not considered a *significant* change.

However, if your group's `Scrabble` program has significant errors or is missing significant functionality, then you **may be able to negotiate** with your tutor to fix these error as a **minor** enhancement. As each group's program is different, this will be determined on a case-by-case basis.

Please note that this does not excuse you from failing to make sufficient contributions to your group. While you may be able to "make-up" some functionality in this milestone, you should not use this as an excuse to make an insufficient contribution.

5 Managing Group Work

This group assignment can be conducted entirely online, without you ever meeting your group members face-to-face. This isn't a problem, with the available online tools. The challenge for you will be using these tools *effectively*. You will need to **make extra efforts** and be **very dedicated and diligent** in working with your team members. This will include setting up dedicated times for meetings and group programming sessions.

5.1 Group Work Tools

To help manage your group work, and demonstrate that you are consistently contributing to your group, we are going to require you to use a set of tools.

5.1.1 MS Teams

Each group will be required to create a team on the RMIT MS Teams platform. Your group must **add your tutor(s)** to your MS team[2].

Your MS team will be the *only official* communication platform for the assignment. This means you must:

- Only use the MS Team channels for group chats
- Hold all team meeting through MS Teams and record all team meetings (except for the conversations in the lab sessions).
- Store any group files (not in Git) in the MS Team

If there are disputes, we will use the record on MS Teams as the source of evidence. You may not use other platforms (including Discord) as we cannot verify the identify of the users.

[2]If your group is from multiple labs with different tutors, add *all* of your tutors to the MS Team.

5.1.2 Git Repository

Your group must have a **private Git repository** that hosts your group's code. This may be on BitBucket or Github. Your group must **add your tutor(s)** to this repository[3]. This git repository will be used as the *evidence of your individual contribution* to your group.

For the individual component you should copy/fork the repository before commencing your individual work.

5.2 Suggested Weekly Schedule

On Canvas, we have provided a spreadsheet that contains a schedule of the suggested tasks your group should finish each week. Additionally, in the labs your tutor will compare your group's progress to this schedule. The tasks are flexible if your group runs into issues, such as a student being sick. However, you **must inform your tutor** of any issues so that you can negotiate with your tutor to rearrange the tasks.

5.3 Group Contributions

For Milestone 1 and 2 you will need to provide a record of your contribution to your group. This record is also contained on the schedule spreadsheet which has a location where each student can record their contributions. You should also note any issues that your have encountered on this spreadsheet.

! Include the filled-in weekly group schedule spreadsheet with your Milestone 1 and Milestone 2 submissions.

5.4 Notifying of Issues

If there are **any issues** that affect your work, such as being sick, you **must keep your group informed** in a timely manner. Your final grade is determined by you (and your group's) work over the entire period of the assignment. We will treat everybody fairly, and account for times when issues arise when marking your group work and contributions. However, you must be upfront and communicate with us.

If you fail to inform us of issues in a **timely fashion** and we deem that your actions significantly harm your group's performance, we may award you a reduced grade. It is academic misconduct if you are *deliberately dishonest*, *lie to*, or *mislead* your group or teaching staff in a way that harms your group.

6 Getting Started

6.1 Designing your Software

This assignment requires you and your group to *design the ADTs and Software* to complete your implementation. It is up to your group to determine the "best" way to implementing the program. There isn't necessarily a single "right" way to go about the implementation. The challenge in this assignment is mostly about software design, not necessarily the actual game-play.

Trying to solve the whole program at once is too large and difficult. So to get started, the best thing to do is start small. You don't have to figure out the whole program at once. Instead, start with the smallest working program. Then add a small component, and make sure it is working. Then keep adding components to build up your final program.

You can get help about your ideas and progress through the lab updates. This is where you can bring your ideas to your tutor and ask them what they think. They will give some and ideas for your progress.

6.2 Starter Code

The start-up code is very limited. It contains the following files:

File	**Description**
`ScrabbleTiles.txt`	Initial content of the tile bag. Letter and value.
`Tile.h/cpp`	Skeleton definition of a tile
`Node.h/cpp`	Skeleton definition of a Linked List of tiles
`LinkedList.h/cpp`	Skeleton definition of a Linked List of tiles
`scrabble.cpp`	Empty Main function
`Makefile`	Simple Makefile for compiling

[3]If your group is from multiple labs with different tutors, add *all* of your tutors to the MS Team.

7 Submission

Follow the detailed instructions **on Canvas** to complete your submission for Assignment 1.

Assessment declaration: When you submit work electronically, you agree to the assessment declaration.

7.1 Silence Period

A silence policy will take effect from **5.00pm, Thursday 26 May 2022**. This means no questions about this assignment will be answered, whether they are asked on the discussion board, by email, or in person. Make sure you ask your questions with plenty of time for them to be answered.

7.2 Late Submissions, Extensions & Special Consideration

Late submission accrue a penalty of 10% per day up to 3 days, after which you will lose ALL the assignment marks. The late penalty for Milestone 1, 2 is applied to the group component of the assignment mark. The late penalty for Milestone 3, 4 is applied to the individual component of the assignment mark.

Extensions will not be given for this assignment.

Where special consideration is granted, generally it is awarded on an individual basis, not for the entire group. Extensions of time will not be granted, instead an equivalent assessment will be conducted which may take the form of a written and practical coding exercises. This equivalent assessment covers the non-group components of the rubric. The group-work component will be assessed on your group participation for the duration of Assignment 2 for which you were unaffected and able to contribute. This will take into consideration how well you kept your group informed of your ability to work and contribute to the group.

7.3 Group Work Penalties

In severe cases of poor group work:

- We may apply an individual mark to any or all categories of the rubric.
- We may file a charge of academic misconduct if we determine that a student has been deceitful or untruthful in a manner than has a severe adverse academic impact on the other students in the group.

8 Marking guidelines

The detailed breakdown of this marking guidelines is provided on the rubric linked on Canvas. The marks are divided into the following categories:

- Group Component (30/45):
  - `Scrabble` Implementation: 15/45
  - Test Cases: 5/45
  - Analysis & Design: 5/45
  - Group Work: 5/45
- Individual Component (15/45):
  - Individual Enhancements Implementation: 10/45
  - Individual Enhancements Analysis: 5/45

9 Academic integrity and plagiarism (standard warning)

CLO 6 for this course is: *Demonstrate and Adhere to the standards and practice of Professionalism and Ethics, such as described in the ACS Core Body of Knowledge (CBOK) for ICT Professionals.*

Academic integrity is about honest presentation of your academic work. It means acknowledging the work of others while developing your own insights, knowledge and ideas. You should take extreme care that you have:

- Acknowledged words, data, diagrams, models, frameworks and/or ideas of others you have quoted (i.e. directly copied), summarised, paraphrased, discussed or mentioned in your assessment through the appropriate referencing methods

- Provided a reference list of the publication details so your reader can locate the source if necessary. This includes material taken from Internet sites. If you do not acknowledge the sources of your material, you may be accused of plagiarism because you have passed off the work and ideas of another person without appropriate referencing, as if they were your own.

RMIT University treats plagiarism as a very serious offence constituting misconduct. Plagiarism covers a variety of inappropriate behaviours, including:

- Failure to properly document a source
- Copyright material from the internet or databases
- Collusion between students

For further information on our policies and procedures, please refer to the RMIT Academic Integrity Website.

The penalty for plagiarised assignments include zero marks for that assignment, or failure for this course. Please keep in mind that RMIT University uses plagiarism detection software.