University of
South Australia

UniSA STEM

COMP 3023– Design Patterns with C++

Assignment 1: Werdle.

Weighting: 20%,
Due Date: See course website

Version	Date	Notes
1.0	14 Mar 21	Initial release

Introduction

In this individual assignment, you have been tasked by your client to create a command-line based game called Werdle in C++. While your client says Werdle is a *completely original game that has never existed before*, it bears a striking resemblance to Wordle, the web-based word game purchased by the New York Times from software engineer Josh Wardle for a seven-figure sum.

In Werdle, as in Wordle, players have six attempts to guess a hidden five-letter word. Players enter their five letter word guesses into the game. After entering their guess, each letter of the guess is marked as one of the following:

1) Not in the word,
2) In the word but in the wrong position, and
3) In the word and in the correct position.

If the player guesses the word correctly, they win.

Learning outcomes

After completing this assignment, you will have learnt to:

- Create a basic system in C++ using an object-oriented design.
- Write C++ code using modern C++ memory management.
- Write safe C++ code.
- Use C++ collections.

Assignment and submission requirements

The assignment has the following requirements. Failing to address any of these requirements will result in deducted marks.

1. The assignment must be submitted via LearnOnline.
2. You must submit three files: 1) a zipped GIT repository, 2) a functioning executable (.EXE), and 3) the system design document.
3. These files must be submitted as three separate files to LearnOnline.
4. The GIT repository must:
    a. Include the full GIT history;
    b. Contain your Visual Studio 2022 Solution and project files.
    c. The repository must be compressed using the common ZIP compression.
5. The functioning executable must:
    a. Run from command line under Windows 10 or higher.
6. The system design document must:
    a. Be submitted as PDF format.
    b. Note: it will also exist as Word document or similar in your repository.

User interface screens

The following subsections demonstrate the expected user interface for the system. Your user interface must match this output *exactly*. Further examples of output are provided in Appendix 1.

Main Menu

```
Welcome to Werdle.
Select an option :

 1. Play a game.
 2. View statistic.
 3. View help.
>
```

The main menu shall:

- Ignore invalid input and show the main menu again.

Play a game

Here is an example of a player trying to guess the word “speed”:

```
guess >apple
 a [p] p  l |e|
guess >start
 a [p] p  l |e|
[s] t  a  r  t
guess >speed
 a [p] p  l |e|
[s] t  a  r  t
[s][p][e][e][d]
Outstanding!
```

Play a game shall:

- Create a new session.

- Treat capital letters as lowercase letters.
- Ignore words > 5 characters and < 5characters.
- Present a letter from the word in the correct position wrapped in []. E.g. `[a]`
- Present a letter from the word in an incorrect position wrapped in ||. E.g. `|e|`
    - This should consume letters. For example, if the word was “apple”, entering “speed” would result in “` s [p]|e| e  d `” — the second “e” is not highlighted because the first “e” already consumed the single “e” in apple.
- Present the correct word if the player loses:

```
Correct answer: speed
```

- Upon winning, present a different message based on the number of guesses that have been made:

1 guess	Impossible!
2 guesses	Amazing!
3 guesses	Outstanding!
4 guesses	Great!
5 guesses	Nice one!
6 guesses	You got there!

- Return to the main menu after winning or losing.

View stastistics

```
Played: 1  Win%: 100  Current streak: 1  Max streak: 1

GUESS DISTRIBUTION
1: 0
2: 0
3: 1
4: 0
5: 0
6: 0
```

The view statistics shall:

- Show the number of sessions for 1..6 guesses (the guess distribution). In the screenshot, there was one session that had 3 guesses.
- Show the number of played sessions.
- Show the win percentage as a full number.
- Show the current win streak (i.e. the most recent number of consecutive games the player has won).
- Shows the max streak (i.e. the longest number of consecutive games the player has won).
- Return to the main menu.

View help

```
Guess the WERDLE in six tries.

Each guess must be a five-letter word. Hit the enter button to
submit.

Examples
[A] P  P  L  E
The letter A is in the correct position.
 D |E| A  L  T
The letter E is in the word but in the wrong position.
```

The view help shall:

- Return to the main menu.

Required classes

Your assignment must design and implement at least the following classes:

Game — The Game object drives the game. Its behaviour changes depending on the state of the game. The Game object:

- Is responsible for initialising and starting a new session (i.e. the current session).
- Is responsible for presenting the previous games statistics.
- Is responsible for presenting the game help information.
- Knows about the current session.
- Knows about all past sessions.

There should only be a single instance of Game at any one time.

Session — Represents a playthrough of the game (e.g. up to six guesses of an unknown word). When the first session is created, it should be assigned **the first word** from the dictionary. Subsequent sessions should be assigned subsequent words in-order. This is for testing purposes. A session object:

- Knows the guesses that have been made for that session.
- Knows if any guess in the session was correct.
- Knows the final score of the session, where the score is calculated as the number of guesses in the session.

Guess — Represents a single guess made by the player. A guess object:

- Knows the guess that was made.
- Knows if the guess was correct.
- Is responsible for presenting the guess results back to the player.

You are provided the following classes:

Dictionary — contains an array of words that you must use in your game. The system must select words in-order from the dictionary so that session 1 selects word 1 from the dictionary, session 2 selects word 2, etc.

Task 1 – System design

Refer to your previous courses (e.g. OOP, Systems Design…) that have covered UML for the skills necessary to complete this task.

This task requires you to create UML class diagram of the system to be built. Use the description of the classes above and the domain model to inform the class diagram. There may be more classes in the system than those described above.

1.1. Create a document to describe your system design.
- The document must include a title, your full name, email address, and student ID.

1.2. Create a GIT repository and add your document to the repository.
1.3. Create the UML class diagram for the system.
1.4. Document any design decisions you have made as dot points (e.g. why certain classes have certain methods).
1.5. Add a PDF of your UML class diagram to your repository.
1.6. Commit your changes to your repository.

Make sure to:
- Carefully consider the direction of the associations between classes. Only have a class accessible by another class if it is necessary.
- Indicate multiplicity on the associations.

Task 2 – Implementation

Weeks 1–5 of the course focusses on the knowledge and skills necessary to complete this task.

2.1. Create a new C++ console application using Visual Studio 2022 in your repository.
2.2. Add an appropriate GIT ignore file.
2.3. Add the provided source code available on the course website.
2.4. Commit your changes to GIT.
2.5. As a sanity check, clone your repository somewhere else on your computer and ensure the entire Visual Studio solution is there as expected.
2.6. Implement the system based on the class diagram you design.
2.7. Commit your changes to GIT.

[Optional] Bonus Task – Algorithms

This task is optional for bonus marks. You can only get a maximum of 100% on the assignment, but some students may use this task to maximise their marks.

3.1. Appropriately use modern C++ standard algorithms from <algorithm> to replace manual for loops.

This bonus task will require research. There are many algorithms in <algorithm> that could be used in this system. Think about what your code is doing on your collections and search for an algorithm to replace any manual for loops. Appropriately using lambda would also be awarded bonus marks.

Design constraints

DC1. The required classes specified above must be implemented.
DC2. The Visual Studio 2022 solution you submit must compile and run.
DC3. C++ standard library collections must be used for storing collections of objects.
DC4. Dynamically allocated memory must be cleaned up correctly.
DC5. Collections of pointers must be cleaned up correctly.
DC6. Class member functions should be declared const unless they cannot be.
DC7. Initialiser lists must be used for constructors where initialisation is required.
DC8. The code style guide available on the course website[1] must be followed with the following exception: you can decide whether to place opening braces on the statement line or the following line.
DC9. Your system must not use external libraries (e.g. BOOST).

Workplan

The following workplan is suggestion. First, do not wait until the due date to start the assignment; under the "GIT" marking criteria, we are expecting to see consistent commits each week. You can progressively build the system as your understanding of C++ develops. Build the classes first, refactor the collections and memory management, add const declarations. *The design of C++ allows this progress refactoring of the system and is your best approach for a successful assignment*.

- If you have not played Wordle, begin by playing a game of Wordle online (https://www.nytimes.com/games/wordle/index.html).
- You have enough information to immediately begin drawing the system design diagram (Task 1).
- For the implementation task (Task 2):
    - During Week 3, you have enough information to build the basic class structures.
    - During Week 4, you will have enough information to implement the C++ collections.
    - During Week 5, you will have enough information to implement modern memory management.
    - Week 6 should then be spent testing and submission.

Constantly commit to GIT. Use it as a safety net so that you can always revert to a working version of your project.

[1] http://codetips.dpwlabs.com/style-guide

Marking Scheme

Criteria	Mark
Design (15%)	
Design diagram – Marked on completeness of diagram and syntax. Required classes are present and used correctly.	15%
Functional requirements (50%)	
Main menu and View help— The main menu and view help is present and meets the requirements.	5%
Play game — The play game state meets the requirements.	30%
Statistics — The statistics screen meets the requirements. Statistics are calculated correctly.	15%
Non-functional requirements (35–45%)	
GIT – GIT has been used. Optimal marks are rewarded for 1) at least one commit each in Week 4, Week 5, Week 6 and 2) at least one commit per functionality in the system. Commit messages should use the imperative mood.	10%
Design constraints – The design constraints have been addressed in the code implementation.	25%
[Optional] Bonus task – The bonus task has been sufficiently addressed.	+10%

- The maximum marks achievable is 100%.
- You will be required to explain your design and code during a practical class. Not being able to explain any of these elements can result in losing marks.

Extensions

Late submissions will not be accepted for this course unless an extension has been approved by the course coordinator. Late submissions that have not been approved will receive a mark of zero. Refer to the course outline for further information regarding extensions.

Academic Misconduct

This is an individual assignment. Your submitted files will be checked against other student's submissions, and other sources, for instances of plagiarism. Students are reminded that they should be aware of the academic misconduct guidelines available from the University of South Australia website.

Deliberate academic misconduct such as plagiarism is subject to penalties. Information about Academic integrity can be found in Section 9 of the Assessment policies and procedures manual at: http://www.unisa.edu.au/policies/manual/

Appendix 1 — Example output

```
Welcome to Werdle.
Select an option :

 1. Play a game.
 2. View statistic.
 3. View help.
> 3
Guess the WERDLE in six tries.

Each guess must be a five-letter word. Hit the enter button to submit.

Examples
[A]  P  P  L  E
The letter A is in the correct position.
D  |E|  A  L  T
The letter E is in the word but in the wrong position.

Welcome to Werdle.
Select an option :

 1. Play a game.
 2. View statistic.
 3. View help.
> 1
guess >apple
 a [p] p  l |e|
guess >spams
 a [p] p  l |e|
[s][p] a  m  s
guess >spelt
 a [p] p  l |e|
[s][p] a  m  s
[s][p][e] l  t
guess >speed
 a [p] p  l |e|
[s][p] a  m  s
[s][p][e] l  t
[s][p][e][e][d]
Great!

Welcome to Werdle.
Select an option :

 1. Play a game.
 2. View statistic.
 3. View help.
> 2
Played: 1  Win%: 100  Current streak: 1  Max streak: 1

GUESS DISTRIBUTION
1: 0
2: 0
3: 0
4: 1
5: 0
6: 0
```

```
Welcome to Werdle.
Select an option :

 1. Play a game.
 2. View statistic.
 3. View help.
> 1
guess >spelt
 s  p  e  l  t
guess >svelt
 s  p  e  l  t
 s  v  e  l  t
guess >times
 s  p  e  l  t
 s  v  e  l  t
 t  i  m  e  s
guess >dealt
 s  p  e  l  t
 s  v  e  l  t
 t  i  m  e  s
|d| e  a  l  t
guess >dream
 s  p  e  l  t
 s  v  e  l  t
 t  i  m  e  s
|d| e  a  l  t
|d|[r] e  a  m
guess >dregs
 s  p  e  l  t
 s  v  e  l  t
 t  i  m  e  s
|d| e  a  l  t
|d|[r] e  a  m
|d|[r] e  g  s
Correct answer: crowd

Welcome to Werdle.
Select an option :

 1. Play a game.
 2. View statistic.
 3. View help.
> 2
Played: 2  Win%: 50  Current streak: 0  Max streak: 1

GUESS DISTRIBUTION
1: 0
2: 0
3: 0
4: 1
5: 0
6: 0
Welcome to Werdle.
Select an option :

 1. Play a game.
 2. View statistic.
 3. View help.
>
```