Te Hoe Whakairi Kōrero
Department of Creative Arts and Digital Information

Bachelor of Information and Communication Technologies

Graduate Diploma in Information and Communication Technologies

Assessment Two
Practical Assessment Two

Best Programming Practices (C# .NET) BCDE222

Semester Two 2023

Due date: Friday 15 September 2023

Time: 5:00 pm

Instructions:
See page 2.
This is an Individual Assignment.

TOTAL MARKS: **25**

Student Name/ID .....

If a learner needs to apply for an extension, they can do so by completing the extension request form (app505m-extension-of-time-application.pdf). Extension requests must be submitted to the lecturer prior to the assessment due date.

If an assessment is handed in late without an approved extension, a penalty of 10% per day will apply, up to a maximum of 50%. If an assessment is received more than five days after the due date without an approved extension, it will not be marked. Should a learner wish to appeal any decisions, they may do so in writing to the Head of Department within ten days of receiving the decision.

This assessment is worth 25% of the total marks for this course. To pass this course, students must gain an average of at least 50% across all assessments, and gain at least 50% in Assessment 3.

This paper has four (4) pages including the cover sheet.

INSTRUCTIONS:

This assessment relates to the following Learning Outcome:

Learning Outcome Two: Demonstrate ability to implement a prototype system.

Overview

For an overview of Chess Maze game, see http://www.mazelog.com/show?9

You need to design, implement and test a <<Model>> code related to Chess Maze game for **one** of the following topics.

1 GAME PLAYER: an application allows a single user to play Chess Maze game.

2 LEVEL DESIGNER: an application allows a single user to lay out and edit Chess Maze levels.

3 FILE HANDLER: an application allows the loading and saving of Chess Maze levels and games in assorted file formats.

It means that you need to:

1 Identify the features that your <<Model>> is able to do

2 Design your <<Model>> using a class diagram, which should include the suitable interfaces and Enums provided.

3 Implement your <<Model>> designed.

4 Write unit tests to test the correctness of your <<Model>> implementation.

The <<Model>> will be able to be used later in association with a <<View>> in Assessment 3. It needs to be written in C#. Naturally, you will be expected to use good design, programming and testing practices.

- Hard code and stub the functionality required from the other applications.
- Includes unit tests (which should be automated).

You **MAY** work in groups for creating the code for this assessment but must make **INDIVIDUAL** presentations. Clearly comment the source of all code that is not your own in your source code files and presentation slides. In your presentation you may use code provided by others in the class but **will only be awarded with marks for the code you have written**. See the marking rubric for details of how the amount of code you use from others will impact on your mark.

Students should show the progress of their assessment work to the course tutor every week and get formative feedback.

Outside the Scope

- Creating the game levels programmatically
- Solving a game level programmatically
- Checking if a given state of a game level can be solved
- Creating a <<View>>

- Creating a <<Controller>>

Required Structures

- The number of the row and the column starts from 0.
- The coordinate of the most top-left position is (0, 0).

Provided Files

You will be provided with a set of interfaces and Enums for the major components of Chess Maze game. You MUST use them and can extend them if needed.

Submission

Students must zip and submit their work including the items listed below into the corresponding dropbox on the course Moodle site by the deadline indicated.

1 A 10-minute MAXIMUM (optionally narrated) PowerPoint slide presentation explaining how much of the <<Model>> you have implemented and tested.

2 A digital copy of the project code including unit testing code.

3 A digital copy of the class diagram created.

The presentation should cover the following:

1 The expected (i.e., self-marking) marks for each task you expect to get marks for.

2 How much of the analysis and the design of the <<Model>> classes have you implemented?

- A list of features of the <<Model>> prioritized using MoSCoW (MUST-HAVE, SHOULD-HAVE, COULD-HAVE, and WON'T-HAVE).
- UML Class diagram.

3 How many of the MUST-HAVE features of the <<Model>> has been implemented?

- Use snapshots to clearly indicate which block of code is behind which MUST-HAVE feature of the <<Model>> you have implemented.

4 How many of the unit tests which prove the correctness of the <<Model>> your code can pass?

- Use snapshots to clearly indicate which unit test is to test which MUST-HAVE feature of your <<Model>>.
- Use snapshots to clearly indicate which unit test can pass against your <<Model>> implemented.

Marking Rubric

Element	0	1	2	3	4	5	Mark
Documentation / Define scope of first iteration using MoSCoW	Not attempted	1-4 scoped MUST-HAVE features	5-8 scoped MUST-HAVE features	9-12 scoped MUST-HAVE features	13-16 scoped MUST-HAVE features	> 16 scoped MUST-HAVE features	0 NOTE: %IMPLEMENTED will be reduced for incomplete MUST-HAVE **feature** coverage
Class diagram	Not attempted	Correctly identify all required classes	Correctly identify all required classes and relationships	Correctly identify all required classes, attributes and relationships	Correctly identify all required classes, attributes, methods, and relationships	Correctly identify all required classes, attributes, methods, relationships, interfaces, and Enums	0
% IMPLEMENTED requirements	Not attempted	All MUST-HAVE features implemented, plus correct corresponding diagram and documents	All MUST-HAVE features implemented, plus correct corresponding diagram and documents	All MUST-HAVE features implemented, plus correct corresponding diagram and documents	All MUST-HAVE features implemented, plus correct corresponding diagram and documents	All MUST-HAVE features implemented, plus correct corresponding diagram and documents	* 3
Unit tests	Not attempted	1-8 tests partial test coverage of functionality	9-16 tests partial test coverage of functionality	17-24 tests partial test coverage of functionality	25-32 tests partial test coverage of functionality	> 32 tests full test coverage of functionality	0 NOTE: %TESTS will be reduced for incomplete test coverage
% TESTS passed	No tests pass	All tests pass	All tests pass	All tests pass	All tests pass	All tests pass	* 2

Final marks = 3 * min { marks(Class diagram), marks(MoSCoW features), marks(Implemented requirements) }
+ 2 * min { marks(Number of unit tests), marks(Number of tests passed) }