Assessment Task Information

Key details:	
Assessment title:	**Practical Programming Assignment 2**
Module Name:	Object Oriented Programming
Module Code:	IY4101
Teacher's Name:	Andrew Bradley
Assessment will be set on:	Jan Cort: 1st May 2024
Feedback opportunities:	
Assessment is due on:	Jan Cort 30th May 2024
Assessment weighting:	30%

Assessment Instructions

What do you need to do for this assessment?

Task:

In this assignment, you are going to design and implement classes using Object-Oriented Programming (OOP) principles to represent various elements in a game called MysteryWorld. Marks will be awarded for correct solutions that demonstrate a good understanding of C++ version 14 concepts, including classes, data types, problem-solving, inheritance and polymorphism.

Game Specifications

- A level in MysteryWorld is a room made up of a 10 x 10 grid. Each cell in the room is referred to by its row and column number (coordinate). For example, the '@' in the room shown in Figure 1 is placed at cell (3,1).
- A room in MysteryWorld can be populated by two types of entities: inanimate and animate entities.
- There are two kinds of inanimate entities: stones, represented by 'S', and holes, represented by 'O'. Holes also have a depth in meters, represented by a number between 0 and 20. As expected, inanimate entities do not move.
- Each living entity has a measure of its health, represented by an integer value between 0 and 100, and all living entities can move. There are three types of living entities: dragons, represented by the symbol '#'; human beings, represented by '@'; and monsters, represented by the '*' character. Each human being has a name, and monsters have strength represented by a number between 0 and 5.
- All animate objects can move, and their health is usually reduced when they move.

Animate Entities Wandering Around the World

- **Humans**: Humans always move to the cell to their right, provided that this cell is not occupied and that they are not at the edge of the room. Otherwise, they move to an unoccupied, randomly chosen cell adjacent to their current position. Their health is reduced by one every time they move.
- **Dragons**: Dragons move to a cell that is randomly chosen from those cells that are unoccupied and adjacent to their current position. The health of the dragon is very volatile and is reduced by a random number between 0 and 5 every time it moves.
- **Monsters**: Monsters move as many cells as their strength in any direction (up, down, left, or right) as long as the destination is free and the monster remains within the room. For instance, the monster in position (9,3) in Figure 1 with strength 5 can move to position (9,8), but it cannot move to (4,3) because the cell is occupied. The monster cannot move to any other cell because it will reach the end of the room. If there are no possible movements, the monster will move to a random empty cell next to it. The health is reduced by as many points as the strength.

Note: In the unlikely event that all possible options are invalid, the entity will not move. Furthermore, the default movement of all living entities is to a random adjacent cell.

- A typical level for a game would look like the following:
  This shows 2 stones, 2 holes, 3 humans, 3 monsters and 2 dragons on this level. Note that a full stop is used to denote an unoccupied cell in the level.

```
  0 1 2 3 4 5 6 7 8 9
0 . . . . . . . . . .
1 . S . . . # . . . .
2 . . . O . . . . . .
3 . @ . . . . . * . .
4 . . . O . . . . . .
5 . . S . . . . . . .
6 . . . . . . . @ . .
7 # . . @ . . . . . .
8 . . . . . . . * . .
9 . . . * . . . . . .
```
Figure 1

You will implement your program as a console application. It is suggested that you have the following classes at a minimum:

1. *Entity class*: This is an abstract class which will act as the parent (base) class to all the different entities (Holes, Stones, Humans…) of the game.
2. *Room class*: This is the class that stores the positions of all the entities. It should ideally contain the Methods to add entities, display the room, display information etc.
3. *Main class*: This is the main class that handles the game. The menu will be implemented here and you will need to add the call to the specific actions.

Functional Requirements

1. The program should be a console application. The user interacts with the program by typing instructions into the standard input, and the program gives output by writing to the console.

   When the program starts, it should display the initial state of the game, and a menu. The initial state of the game should contain 2 holes, 2 stones, 3 monsters, 3 humans, 2 dragons, placed randomly in the room, with the constraint that two entities cannot occupy the same position. In the initial state, the depth of the holes is a random number between 0 and 20, monsters will be allocated a random strength between 0 and 5 and the three humans will be: Harold, David and Clare. All the animate entities are 100% healthy.

2. The initial menu should offer the following options:
   1. Display Room
   2. Move all the animated entities
   3. Display the entity state by coordinates
   4. Reset the room
   5. Finish

3. If the user selects '1' from the menu, the state of the game should be displayed as shown in Figure 1

4. If the user selects '2' from the menu, all the living entities should be moved according to the rules specified earlier.

5. If the user selects '3' from the menu, they should be prompted to specify a row, and a column, and then the properties of the entity at that position (if there is one) should be displayed, the output might be as follows:

   ```
   Entity properties:
   Type: human
   Name: Harold
   Health: 47
   ```

   If there is no entity in that cell, then your program should output a suitable message.

6. If the user selects '4' from the menu, then the state of the board should be reinitialised, in the manner described in functional requirements.

7. If the user selects '5' from the menu, then the program should terminate.

Structure:

- **Program Design Description**: Describe the design of your program detailing how the classes have been designed in a report. You can use a flowchart, pseudocode (not C++), or UML diagram to describe your design.
- **Inheritance Implementation**: Implement all the different types of entities using inheritance. All entities should inherit from the Entity class.
- **Naming Conventions**: Use standard C++ naming conventions. For example, variable names must be meaningful nouns or noun phrases and should be written using camel case.
- **Program Structure**: Create a well-structured C++ program that uses the concepts learned up to this point and makes sensible use of comments.
- **Method Explanation**: Use comments to explain each method that you have written.
- **Testing**: Develop a sensible way to test that the program works. Every option in the menu must be tested. Automated/Unit Testing is the preferred approach.
- **AI Utilisation Explanation**: Include a brief explanation of how you utilised AI tools, if any, in the development of your solution. This should cover what specific tasks that AI assisted with and how it influenced your approach to the assignment.
- **Version Control Management**: It is strongly recommended that you use version control management during the implementation of this program.

Theory and/or task resources required for the assessment:

- Lippman, S. B., Lajoie, J., & Moo, B. E. (2005). C++ Primer. 4th Edition. Addison Wesley Professional. ISBN 9780201721485
- Perry, Greg M.; Miller, Dean, C. (2013). C Programming Absolute Beginner's Guide. 3rd edition. ISBN 978-0789751980

Referencing style:
You **may** include a Harvard style reference list at the end of your report. A full bibliography is NOT required.

Expected word count:

You are expected to write between 500 and 1000 words excluding the implementation (C++ code), following the specific structure outlined above and in the submission requirement**.**

Learning Outcomes Assessed:

- Use the basic programming constructs of C/C++ to manipulate various datatypes, such as arrays, strings, and pointers.
- Manage memory appropriately, making use of memory allocation/deallocation procedures.
- Apply the Object-Oriented Programming approach to software problems in C++.
- Identify, isolate, and fix common errors in CC++ programs
- Write small-scale C/C++ programs using skills developed during the course.
- Develop and use test plans.
- Implement basic source control management e.g. git.
- Explain the ethical issues around open-source software, as well as trustworthy and secure software development

Submission Requirements:

The work you submit should comprise two parts: a Microsoft Word document or pdf file and a zip file with the C++ program file(s). The report should contain the design, and the results of running your program with an explanation of how it has been tested and all the C++ source code copied and pasted in the appendix.

How to avoid academic misconduct

You should follow academic conventions and regulations when completing your assessed work. If there is evidence that you have done any of the following, whether intentionally or not, you risk getting a zero mark:

Plagiarism & poor scholarship

- stealing ideas or work from another person (experts or students)
- using quotations from sources without paraphrasing and using citations

Collusion

- working together with someone else on an individual assessment, e.g., your work is corrected, rephrased or added to by another (both parties would be guilty)

Buying or commissioning work

- submitting work as your own that someone else produced (whether you paid for it or not)

Cheating

- copying the work of another student
- using resources or aids that are not permitted for the assessment

Fabrication

- submitting work, e.g., laboratory work, which is partly or completely made up. This includes claiming that work was done by yourself alone when it was actually done by a group

Personation

- claiming to be another student and taking an assessment instead of them (both parties guilty)

Specific formatting instructions:

You must type your assessment in Arial font 11, with single spacing.

You must submit the assessment electronically via the VLE module page. Please ensure you submit it via Turnitin.

Assessments submitted after the submission deadline may incur penalties or may not be accepted.

Addition submission information – check you have done the following:	
Formatting	Consistent font, spacing, page numbers, formatting and subheadings
Citations	Correct format and location throughout the report
Spell check	Spell check the report
Proof-reading	Proof-reading completed
Grammar	*Grammarly* has been used to check the report

How will this assessment be marked?

The assessment will be marked using the following areas and weightings:

- Design of the program. The design of your program using Class diagram and relationship between them (25%)
    - Description of the classes and their relationship (15%)
    - Design of the solution (10%)
- Description of how you tested the program and explanation of the results. (15%)
    - Clear description of problems encountered
    - Description of how you have solved the problems or were unable to solve them
- Implementation (60%)
    - Correct implementation of hierarchy of entity classes with appropriated constructors and overriding of methods in particular move and display. [20%]
    - Implementation of the option that displays the room [10%]
    - Implementation of the option to Reset the game [10%]
    - Implementation of the option that displays the entity by coordinates [5%]
    - All the entities move as expected [5%]
    - Programming style [5%]
    - Testing and Results [5%]. All the options of the menu should be tested.

You will receive a % mark in each of these categories. The overall mark will be a percentage (0-100%).

How will you get feedback?

Your tutor will mark the assessment and provide you with a written feedback sheet. You can use this feedback to guide your further learning on the module.