RMIT UNIVERSITY

COLLEGE OF VE, FUTURE TECHNOLOGIES

Course Code and Name: C2511C: Introduction to Programming
Semester 2, 2022 (2210)
PROGRAMMING PROJECT

Due Date: End of Week 15

Assessment weighting: This assessment is out of 100 and will contribute towards 30% of your overall course mark. *(This is not a hurdle assessment)*

Submission:

- The due date for this assessment is end of week 15
- Only **one** submission will be made per team
- Submissions will be made via Canvas

Assessment policy:

- Assessments submitted between 0-24 hours after the due date: 20% penalty
- Assessments submitted between 24-48 hours after the due date: 50% penalty
- Assessments submitted more than 48 hours after the due date will not be accepted

For more details on assessment policy check **here**.

What to Submit:

Your team should make one single submission in a .zip file named **cosc2511_sxxxxxxx.zip** (where **sxxxxxxx** is the project submitter's student number)

Your zip file should include the following:

- A document showing all algorithms and pseudocode used to solve the programming problems (gameplay, item interaction, npc encounters, etc)
- Documentation of your game including a description of the story, the game map, objective and instructions on how to run the game and play
- All required program files submitted as .java source code files

Extensions:

- Extensions will only be granted under exceptional circumstances and are intended to offer support and flexibility where unforeseen events have occurred preventing students from submitting projects on time
- If an extension is required, project teams must apply via email to their lab teacher prior to the due date with an explanation of the unforeseen circumstances experienced
- If an extension is granted by your teacher, it will be for a maximum of 7 calendar days
- If further extension is required all project team members must individually apply for RMIT Special Consideration here: https://www.rmit.edu.au/students/student-essentials/assessment- and-exams/assessment/special-consideration

Teaching Team:

Radhu Punchanathan: radhu.punchanathan@rmit.edu.au

Eddie Vanda: eddie.vanda@rmit.edu.au

Trevor Stone: trevor.stone@rmit.edu.au

Trevor Reynolds: trevor.reynolds@rmit.edu.au

Purpose:
The purpose of this project is to give you an opportunity to exercise your algorithmic thinking and Java programming skills to solve programming problems with tools practiced throughout the semester in Introduction to Programming COSC2511.

Team Composition: You will form teams of 1 – 3 students in Canvas and notify your instructor of your team composition via your section's Canvas discussion board by the end of week 13. Any students not forming a team by this date will be deemed to be submitting this project as an individual assessment task.

What to Create (*Core Requirements*):

The topic of this project is that your submission should be a console-based text adventure game written in Java and should show an understanding of topics covered in Introduction to Programming COSC2511.

Full creative control is with your project team, and the story line is completely up to you! You might be exploring an alien planet, attacking a castle, or trying to make your way out of a spooky forest. Let your imagination run wild!

Your game environment should be based on a **5 x 5 grid layout** (see visual example on page 3) and should have a minimum of **9 locations** that a player can visit throughout the game.

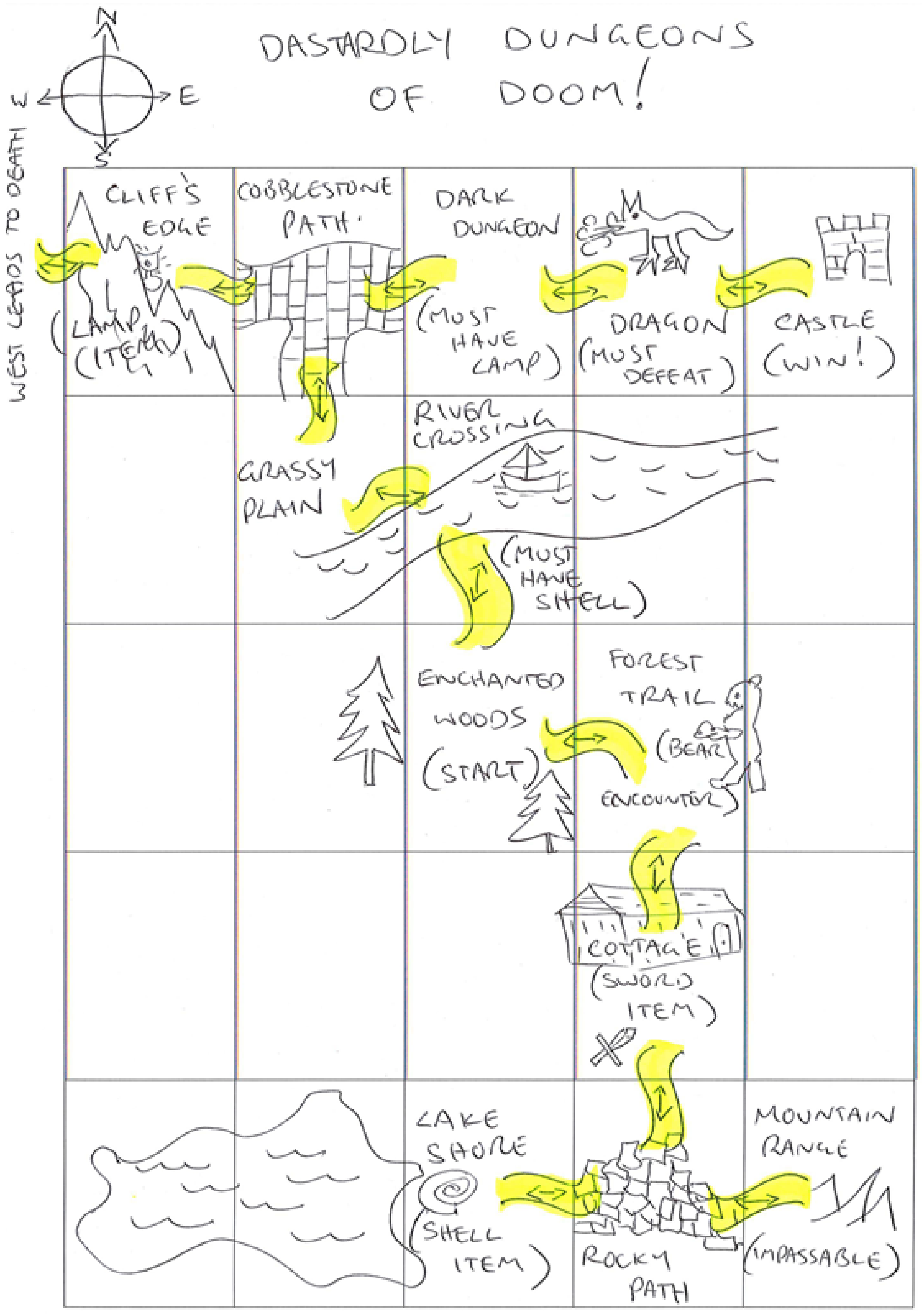
Your game **must have a clear objective** (to win the game), and potentially may have an alternate ending (optional) such as the player dying. Your game **must include an inventory** system allowing a player to **pick up, carry, list and interact with** a minimum of **5** items, and **must include** at least **one** interaction with a non-player character (**npc**).

Players must be able to navigate the game using a simple control system based on the four main points of a compass *(north(**n**), south(**s**), east(**e**) & west(**w**))*, and each time a location and any available items or interactions is visited a description of the location must be printed to the screen. **Once an item has been picked up, or a foe vanquished**, subsequent visits to that location must omit the item or npc from the description (or in the case of a slain enemy, change its state to dead in the description).

Your goal is to display an understanding of the following programming tools used throughout your game:

- Exemplary coding etiquette showing good indenting, consistent block bracing style, data type and class naming conventions and appropriate use of comments
- At least one text-based menu
- User (player) input
- Appropriate selection of and use of variables
- Selection statements including switch, if, else if and nested if
- Iteration with the appropriate type of loops
- Use of random numbers
- Arrays
- Functions
- Classes and methods

Game Layout Example



Marking Guide	**Not Quite There**	**Starting to Improve**	**Getting Better**	**Nice Work!**	**Above and Beyond!**
	Objective not achieved ***(0 marks)***	Objective partially achieved ***(3 marks)***	Objective mostly met ***(6 marks)***	Objective fully met ***(9 marks)***	Exceeded expectations ***(10 marks)***
Coding Style	Little to no commenting and/or poorly implemented indenting and block bracing	Limited use of commenting with inconsistent block bracing and/or indenting style and class and data type naming conventions	Good use of commenting with appropriate information and consistent block bracing and indenting style in addition to sound class and data type naming conventions	Thorough use of commenting techniques with consistent block bracing and indenting style in addition to good class and data type naming conventions	Thorough and detailed use of advanced commenting techniques with consistent block bracing and indenting style in addition to industry standard class and data type naming conventions
Algorithmic Thinking	Programming problems not solved, and core requirements not met	Partial thought given to problem solving with little algorithmic thinking or pseudocode provided to support the program. Core requirements partially met	Good problem solving and algorithmic thinking displayed and pseudocode documentation provided to support the program. Most core requirements met	Thorough problem solving with additional algorithmic thinking and pseudocode documentation provided to support the program. All core requirements met	Thorough and efficient problem solving with additional algorithmic thinking and pseudocode documentation provided to support the program. All core requirements met with some exceeded
User Input	User not asked for input	Limited user input accepted with no data validation	Appropriate user input accepted with some data validation techniques used	A range of user inputs are accepted with consistent data validation	Complex user input mechanisms implemented with advanced data validation techniques utilised
Data Types	Inappropriate data types used throughout program	Some data types used in the program but not consistently the best data types	Acceptable data types used throughout the program	A range of completely appropriate data types used showing an understanding of all data types covered	A large range of data types used including advanced techniques such as resizable implementations of arrays etc.
Selection	No selection tools used	Limited use of selection statements but not all types used	All selection types used in a limited way. Could be improved with better type selection or nesting etc	All selection types used in an efficient manner	Extensive usage of a full range of selection statements including nesting and implementation of user menus
Iteration	Loops not used in the program	A single loop type used throughout the program in a limited way	A range of loops used but not always the most appropriate type	Appropriate types of loops used showing a good understanding of each type of loop method	Advanced use of iteration using a range of techniques to control program flow and duration and to initialise data types
Arrays	Arrays not implemented in the program	One array implemented but not the most appropriate type or not accessed correctly	Arrays used mostly appropriate with fixed initialisation parameters	Good use of multiple arrays showing dynamic initialisation techniques	Extensive use of multiple arrays showing advanced initialisation and access methodologies
Object Oriented Programming Principles	No object-oriented principles used in the program	Objects instantiated in the game do not have an ***'is a'*** relationship and/or are not accessed	Use of objects in a limited way with basic methods available	Good use of objects displaying an ***'is a'*** relationship to game elements and appropriate get and set methods for object attributes	Extensive use of classes and object instantiation showing an understanding of advanced OOP principles
Game Objective	No clear objective defined to win the game	Game objective present but not clear or achievable	Defined game objective present but either too challenging or too easy to achieve *(no puzzle solving required for example)*	Clearly defined game objective that is challenging but achievable	Clearly defined game objective that is challenging but achievable with more than one alternate ending provided
Creativity	No storyline implemented in the game scenario	Limited and shallow story used in the game	Interesting but very short story with limited documentation	Engaging and entertaining story with good documentation provided	Deep, complex and engaging story implemented showing exceptional creative process and excellent documentation