IN628 Programming 4
Semester 2, 2020

Assessment 01: Roguelike Assessment Rubric

	10-9	8-7	6-5	4-0
Planning Document	Planning document submitted before the due date. The provided set of questions are thoroughly answered in detail. System design thoroughly planned and no changes to the submitted planning document.	Planning document submitted before the due date. The provided set of questions are mostly answered in detail. System design mostly planned and a few changes to the submitted planning document.	Planning document submitted after the due date. The provided set of questions are answered in some detail. System design planned and several changes to the submitted planning document.	Planning document submitted after the due date or not submitted. The provided set of questions are answered, though in minimal or no detail. System design poorly planned or not planned.
Code Commenting	All header comments thoroughly explain the input, output, effect and computational logic of each class and method. All inline comments thoroughly explain the logic of construct of each computational statement.	Most header comments explain the input, output, effect and computational logic of each class and method. Most inline comments explain the logic of construct of each computational statement.	Some header comments explain the input, output, effect and computational logic of each class and method. Some inline comments explain the logic of construct of each computational statement.	Minimal or no header comments explain the input, output, effect and computational logic of each class and method. Minimal or no inline comments explain the logic of construct of each computational statement.

Code Elegance	All class files contain no integer literals except for 0, 1 and 2. Application demonstrates thorough elegance on all of the following: • Correct use of intermediate variables, e.g., no method calls as arguments • Idiomatic use of control flow and data structures • Sufficient modularity, e.g., classes, methods have a single purpose • Efficient algorithmic approach	Most class files contain no integer literals except for 0, 1 and 2. Application demonstrates clear elegance on most of the following: • Correct use of intermediate variables, e.g., no method calls as arguments • Idiomatic use of control flow and data structures • Sufficient modularity, e.g., classes, methods have a single purpose • Efficient algorithmic approach	Some class files contain no integer literals except for 0, 1 and 2. Application demonstrates elegance on some of the following: • Correct use of intermediate variables, e.g., no method calls as arguments • Idiomatic use of control flow and data structures • Sufficient modularity, e.g., classes, methods have a single purpose • Efficient algorithmic approach	Class files contain frequent integer literals. Application does not demonstrate elegance on any of the following: • Correct use of intermediate variables, e.g., no method calls as arguments • Idiomatic use of control flow and data structures • Sufficient modularity, e.g., classes, methods have a single purpose • Efficient algorithmic approach
OO Architecture	All classes adhere to a general OO architecture, e.g., classes, methods, concise naming and methods assigned to the correct classes. Inheritance fully and carefully implemented in classes, e.g., player sprite inherits from sprite. Finite State Machine (FSM) implemented fully and stores three states and actions.	Most classes adhere to a general OO architecture, e.g., classes, methods, concise naming and methods assigned to the correct classes. Inheritance mostly implemented in classes, e.g., most classes are deriving from base classes. Finite State Machine (FSM) mostly implemented and stores two states and actions.	Some classes adhere to a general OO architecture, e.g., classes, methods, concise naming and methods assigned to the correct classes. Some inheritance implemented in classes, e.g., some classes are deriving from base classes, though some are incorrectly implemented in the wrong classes. Some Finite State Machine (FSM) implemented and stores one state and action.	Classes adhere to minimal or no general architecture, e.g., classes, methods, concise naming and methods assigned to the correct classes. Minimal inheritance implemented or not attempted. Minimal Finite State Machine (FSM) implemented or not attempted.

Functionality & Robustness	Application opens in Visual Studio 2017 without errors and does not need to be modified to be run. Application demonstrates thorough functionality & robustness on all the following: • Displayed at the correct screen size of 1920x1080 • Dungeon represented as a tile map • Edge of the world has a dead zone which is ½ the dimension of the viewable area • Dungeon procedurally generated at each new level, e.g., multiple non-overlapping rooms, walls, corridors and portal tiles correctly placed • Fog of war reveals the dungeon as the player character progressively navigates through the dungeon • One or more player characters are controlled by user keyword • Two distinct animated enemies • Careful sprite and terrain collision detection, e.g., sprite to enemy, sprite to wall collision detection	Application does open in Visual Studio 2019, though needs to be modified to be run. Application demonstrates most functionality & robustness on all the following: • Displayed at the correct screen size of 1920x1080 • Dungeon represented as a tile map • Edge of the world has a dead zone which is ½ the dimension of the viewable area • Dungeon procedurally generated at each new level, e.g., multiple non-overlapping rooms, walls, corridors and portal tiles correctly placed • Fog of war reveals the dungeon as the player character progressively navigates through the dungeon • One or more player characters are controlled by user keyword • Two distinct animated enemies • Careful sprite and terrain collision detection, e.g.,	Application needs to be modified to be open and run in Visual Studio 2019. Application demonstrates some functionality & robustness on all the following: • Displayed at the correct screen size of 1920x1080 • Dungeon represented as a tile map • Edge of the world has a dead zone which is ½ the dimension of the viewable area • Dungeon procedurally generated at each new level, e.g., multiple non-overlapping rooms, walls, corridors and portal tiles correctly placed • Fog of war reveals the dungeon as the player character progressively navigates through the dungeon • One or more player characters are controlled by user keyword • Two distinct animated enemies • Careful sprite and terrain collision detection, e.g.,	Application cannot be opened in Visual Studio 2019or application is empty. Application does not demonstrate functionality & robustness on any of the following: • Displayed at the correct screen size of 1920x1080 • Dungeon represented as a tile map • Edge of the world has a dead zone which is ½ the dimension of the viewable area • Dungeon procedurally generated at each new level, e.g., multiple non-overlapping rooms, walls, corridors and portal tiles correctly placed • Fog of war reveals the dungeon as the player character progressively navigates through the dungeon • One or more player characters are controlled by user keyword • Two distinct animated enemies • Careful sprite and terrain collision detection, e.g., sprite to enemy, sprite to

	• Careful collision detection that affects the score and condition, e.g., sprite to coin, sprite to health potion • Working battle system, e.g., turn-based or/and to-the-death • Immediate gameplay feedback including battle system feedback, score, win and loss • One enemy that exhibits artificial intelligence behaviour. This may be implemented using trigonometry	sprite to enemy, sprite to wall collision detection • Careful collision detection that affects the score and condition, e.g., sprite to coin, sprite to health potion • Working battle system, e.g., turn-based or/and to-the-death • Immediate gameplay feedback including battle system feedback, score, win and loss • One enemy that exhibits artificial intelligence behaviour. This may be implemented using trigonometry	sprite to enemy, sprite to wall collision detection • Careful collision detection that affects the score and condition, e.g., sprite to coin, sprite to health potion • Working battle system, e.g., turn-based or/and to-the-death • Immediate gameplay feedback including battle system feedback, score, win and loss • One enemy that exhibits artificial intelligence behaviour. This may be implemented using trigonometry	wall collision detection • Careful collision detection that affects the score and condition, e.g., sprite to coin, sprite to health potion • Working battle system, e.g., turn-based or/and to-the-death • Immediate gameplay feedback including battle system feedback, score, win and loss • One enemy that exhibits artificial intelligence behaviour. This may be implemented using trigonometry
Player Experience	Highly attractive, with a coherent graphical theme and style Application is highly appealing and has an engaging game play experience	Mostly attractive, with a coherent graphical theme and style Application is mostly appealing and has an engaging game play experience	Somewhat attractive, with a graphical theme or style Application is somewhat appealing and has an engaging game play experience	Minimal attempt or no coherent graphical and style Application is not appealing or engaging, e.g., no game play

Marking Cover Sheet

OTAGO POLYTECHNIC
Te Kura Matatini ki Otago

Assessment 01: Roguelike

IN628 Programming 4
Level 6, Credits 15
Bachelor of Information Technology

Name: _____ Date: _____

Learner ID: _____

Assessor's Name: _____

Assessor's Signature: _____

Criteria	Out Of	Weighting	Final Result
Planning Document	**10**	**10**	
Code Commenting	**10**	**10**	
Code Elegance	**10**	**25**	
OO Architecture	**10**	**20**	
Functionality & Robustness	**10**	**25**	
Player Experience	**10**	**10**	
		Final Result	**/100**
This assessment is worth 45% of the final mark for the Programming 4 course.			