Fundamentals of Programming

Coursework specification

# Introduction

The coursework assignment for this module requires you to work with a software project provided by the module staff and to complete a series of tasks outlined in this document. The tasks will ask you to complete several pieces of functionality to complete a basic 2D game. You will also write and submit a short report on your submission, describing how your code works for each of the tasks.

The software provided to you is partially complete – the code to draw graphics to the screen and to handle input from a user are already working. You will complete methods in a specific class to build key parts of the game engine – the code will drive the rules of the game, deciding what is drawn and what happens when a user interacts with the system.

The software is fully documented and the methods that you need to complete are provided with both a task description in this document and documentation within the code describing what they should do. You must use what you have learned during the module to write the code to complete these methods. As you complete each task you should find the program grows more functional; each task when completed should produce some visible feedback (output to the screen) for you to test and assess.

# The software

The software is comprised of nine source files with nine classes. You should review each class, read the documentation for each class and understand how the software works before you start working. The modifications that you will make to the software are mostly limited to the GameEngine class, which you will see has a number of empty or partially complete methods for you to work on.

# Task specifications

To complete this coursework you will work through the tasks specified below. **It is strongly recommended that you complete the tasks in the order suggested**, as some build on one another and later tasks may not make sense if the preceding tasks are not completed. You may like to return to a task after you have completed it to add more features, or to increase the complexity of your solution/algorithm; more complete and challenging solutions will be awarded more marks.

**It is strongly recommended that you Export your project after you complete every task so that you have a working copy ready to submit at every stage.** You should submit the last working version of your solution – **your code must compile and run when it is tested** for marking. Submitted solutions with compiler errors will have a heavy penalty to the marks awarded. Details on how to export and submit your solutions are provided below.

Unless otherwise specified, all methods described below are in the GameEngine class, and the documentation in the code for each method describes the expected behaviour.

## Task 1 – Level Generation

You must complete the code for the generateLevel() method. The method must return an array of TileType values (TileType is an enumeration in the GameEngine class). Your code should return the array which is used to draw tiles to the screen by the GameGUI class. For example, a 2D array of only TileType.GRASS values will draw an empty level of grass tiles. You should code the method to produce an interesting and fun level for the player to move around. It is suggested that you design an algorithm and then implement the algorithm in code. The task requires levels be generated dynamically and you should avoid hard coding a completely fixed level layout. The level should contain exactly one tile with the type TileType.CAR.

## Task 2 – Player Spawning

You must complete the code for the spawnPlayer() method. The method must return a Human object that represents the player object in the game. You must set the maxHealth for the player, and the X and Y positions corresponding to the single Car tile in the level.

## Task 3 – Player Movement

The software already detects when the user presses a key on the keyboard if the GUI for the game is in focus. Currently the InputHandler class handles the key presses and calls the doTurn() method of the GameEngine class – any key press results in a single turn of the game taking place. If the key pressed was one of the arrow keys on the keyboard, it also calls a method in the GameEngine class to move the player. Your task is to complete the code for these four methods so that the player moves when an arrow key is pressed.

It is suggested that you code one of the methods, test it and then apply similar code and logic for the remaining methods. Design an algorithm first that specifies what should happen for a single direction, checking that the tile next to the player is suitable for them to move into (i.e. the player should not be able to move into a wall or off the edge of the level). If the target tile is suitable for the player to enter, the player object should have its X or Y attribute changed accordingly. The GUI is updated automatically as the input handler calls the already complete doTurn() method that redraws the level on the screen.

## Task 4 – Spawn seekers

You must complete the code for the spawnSeekers() method. This method should create and return an array of Seeker objects, by instantiating the Seeker class a number of times, setting the position of the Seeker obejcts, and adding them to an array of appropriate size. It is strongly recommended that you complete and use the getSpawns() method and use the spawns ArrayList to find places to add Seeker objects into the level, though it is possible to create and place Seekers without this..

## Task 5 – Seeker movement

You must complete the code for the moveSeekers() method. This method should process all Seeker objects in the seekers array and call the moveSeeker(Seeker a) method for each of them. The moveSeeker(Seeker a) method must also be completed, which should update the X and Y values for a specific Seeker object. The Seeker objects should move towards the player by checking their X and Y positions against the players, and updating one or both to decrease the distance between them and the player. Seeker objects should not be able to move off the map edge, or into a tile containing another Seeker. If they try to move into the tile containing the player, they should not move but should damage the player by reducing its health.

## Task 6 – New levels

You must complete code for the spawnFuel() method. The method should create and return a Fuel object with suitable X and Y positions in the level (i.e. not spawned in tiles with monsters or in the same tile as the player). It is suggested that the spawns ArrayList is used, which needs the getSpawns() method to work properly, although it is possible to create and place fuel without the ArrayList.

You must update your movement code to check if the player enters a tile containing the Fuel object. If so, the Fuel object should be deleted (it should be set to null) and the fuelCollected variable should be set to true. Once this variable is set to true, when the player returns to the Car tile the newLevel() method is called.

You must complete the newLevel() method to generate a new level, and reset appropriate variables as required. The cleared variable can be used to track completed levels and to increase difficulty by increasing monster amounts and/or speed.

## Task 7 – Chasers

You must add Chaser objects to the game. Chasers are similar to Seekers, but should move every two turns instead of every five. Chasers should be added to a level when a player enters a Nest tile (which look like Dirt tiles). When they do so, a new Chaser object should be added to the next available position in the chasers array, and its position should be set next to the Nest tile that the player is in.

## Task 8 – Additional Features

You should pick at least one of the sub tasks in this section and complete it. You may wish to complete more than one, and to add your own additional features, both of which can allow you to gain more marks for your submission. Some of these tasks require you to modify the other classes in the software – do so carefully and make sure you have exported a backup; a fully working version of your submission before you change anything in case you break existing functionality.

### 8.1 More items

Add items to the game. Use the Fuel object as a template and make the items have interesting properties or actions associated with them.

### 8.2 Unpredictable movement

Make seekers and chasers move at different times instead of all moving in a single turn.

(random generate number to be divided by turnNumber and be called by seekers)

### 8.3 Buildings

Use walls to draw (possibly complex) building shapes on the map and add items, fuel and/or traps to rooms.

### 8.4 Something else…

Design and implement your own feature(s) for the game, adding code and/or classes where needed and documenting them appropriately.