# IN628 2018 Assignment 1: 2D Roguelike

Due Date: Tuesday, 23th October, 5.00 pm

Value: 40% of course mark

Group Size: This assignment is to be done in pairs.

For this assignment you will use Visual C++ with Visual Studio 2017 to build a 2D Roguelike game (a dungeon crawler with procedurally generated dungeons).

**Functional Requirements. The application must:**

|  |  |
| --- | --- |
|  | **System:** |
| **1** | Be written in Visual C++ using the .NET Graphics class. |
| **2** | Open without modification in Visual Studio 2017 |
| **3** | Display well at 1920x1080. |
|  | **Game world:** |
| **1** | Implement **procedurally generated** dungeons containing multiple rooms, connected by corridors and containing a randomly located stair (or equivalent portal) to the next dungeon. |
| **2** | Represent and display dungeons as tile-maps. The tile map must scroll, and be centred around the main player character. A "deadzone" of 1/2 the dimension of the viewable area is permitted at each edge of the world. |
| **3** | Implement “fog”. That is, the dungeon must be progressively revealed as the player character moves through the world. |
|  | **Entities:** |
| **1** | Contain one or more player characters under user (keyboard) control. |
| **2** | Contain at least three types of animated enemies placed randomly in the dungeon. Particular enemies may be confined to particular experience levels (or similar game play rule). Enemies must have distinct visual representations and behavior statistics. |
| **3** | Contain at least one type of item that directly impacts the game score (i.e. gold, treasure, etc.) |
| **4** | Contain at least two types of item which affect the player’s condition (e.g. increase or decrease health, increase or decrease attack strength, etc.) upon contact. |
| **5** | Demonstrate correct sprite to terrain collision detection. Players and enemies may not walk through walls. |
| **6** | Demonstrate correct sprite to sprite collisions. Collision between player and item affects the player’s condition and/or score. Collision between player and enemy initiates battle. |
| **7** | Implement a battle system. Turn-based, to-the-death is acceptable. |
| **8** | At least one enemy must exhibit complex programmed behaviour (i.e. AI). |
| **9** | Contain at least one game element whose behaviour involves trigonometric computation, as discussed in class (i.e. trajectory, rotation, and/or orientation). This computation may be part of the AI requirement, #8 above. |
|  | **Game Play:** |
| **1** | Allow control of the player character with the keyboard. |
| **2** | Have a clearly defined and displayed scoring system. |
| **3** | Have a clearly defined and displayed loss condition. |
| **4** | Provide appropriate user feedback. |
| **5** | Be visually attractive, with a coherent graphical theme and style. Please include correct citations for all externally-sourced graphic elements. All media must be royalty free (or legally purchased) for educational use. |
| **6** | Provide an interesting game play experience. |
|  | **Extra Credit:** |
|  | Up to 5% extra credit (each; for a possible total of 10% EC) will be given for the implementation of the following optional features:   1. Using a Finite State Machine to control the behavior of one or more entities. 2. Have ranged weapons or other projectiles. Correct collision detection is required for full-credit. That is, spells, lasers, flaming sheep, whatever, should not pass through walls. |

**Marking Schedule:**

|  |  |  |
| --- | --- | --- |
| **Component** | **Full Credit Criteria** | **Value** |
| **Code commenting** | * Both in-line and block commenting complete * Comments explain logic, do not simply translate the code. | 10% |
| **Planning document** | * Answers are clear and complete. * Planning was done prior to implementation. * Changes to planning, with justification, noted in detail. | 10% |
| **OO Architecture** | * Correct classes included * Inheritance and polymorphism are correctly employed * Roles assigned correctly to classes | 20% |
| **Code elegance** | * Appropriate modularity * Sensible algorithmic approaches * Correct flow of control * No integer literals in code * Sensible variable naming | 25% |
| **Functionality & Robustness** | * All functionality as described above * No bugs. Does not crash. | 25% |
| **Player Experience** | * UI is clear * Visual appearance is coherent * Score and player state feedback is clear. | 10% |

**Group Contribution:** All git commit messages must identify which member (or members) of the pair participated in the associated work session. Proportional contribution will be determined by inspection of the commit logs. If the commit logs show evidence of significantly uneven contribution proportion, the lecturer may choose to adjust the mark of the lesser contributor downward by an amount derived from the individual contributions.

**Submission:**

Visual Studio Project files, media files, and completed Planning Document (see below) must be submitted via Gitlab as described in class. Each pair will need to create a repo for development that both devs can access. Be sure to also set me as a collaborator. Additional paper planning documents, if any, can be uploaded as scans or photographs.

A minimum of two commits per week is required. Insufficient commit frequency is grounds for rejection of the submission and award of zero marks for the assignment.

# 2D Roguelike Planning Document

Fill in the following questions in detail before you begin to code your game. Please use a digital copy of the document, not a hard copy. For each question, justify your answer. If during implementation you make any changes to your originally articulated plan, amend the document, specifying the changes, and explaining your rationale.

Submit the completed document with your source code via Gitlab. To receive full credit, the completed document **must** be uploaded to the repo before any code files.

|  |
| --- |
| 1. Are your player, items and enemies the same class, different classes in the same family, or completely different classes? |
| 1. What logic will you put into your Form class? What logic will you put into your Game Manager class? |
| 1. What class(es) do you need to implement the dungeon? Briefly explain the job of each class, list the data members it must hold, and the methods it must expose. How do the Dungeon and the TileMap communicate? |
| 1. What data structure(s) do you need to hold collections of enemies and items? |
| 1. Does the dungeon need pointers to its sprites? Why or why not? |
| 1. Does the sprite class need a pointer to its dungeon? Why or why not? |
| 1. What enum types (if any) do you need? |
| 1. Does the player sprite need access to the collection(s) of enemy sprites? |
| 1. What class is responsible for creating the collections of enemies and items? |
| 1. If you are using an FSM, what class calls the FSM methods of the sprites? |
| 1. At each game cycle, you need to perform collision detection between the player character and each enemy and item in the dungeon. What class or classes hold a method to compare the areas of two entities to check for collision? What is the function header of this method? What other classes are involved in the collision detection logic? |
| 1. Describe the AI you are going to include.    1. Describe the behaviour    2. Describe the implementation logic |
| 1. Describe the trigonometry you are going to include (if not already contained in the AI from item 12). |
| 1. Describe, in detail, the logic of your battle algorithm and computations. |
| 1. Sketch the screen layout -- with -- controls that you will use to provide feedback during battle. |