**Description**

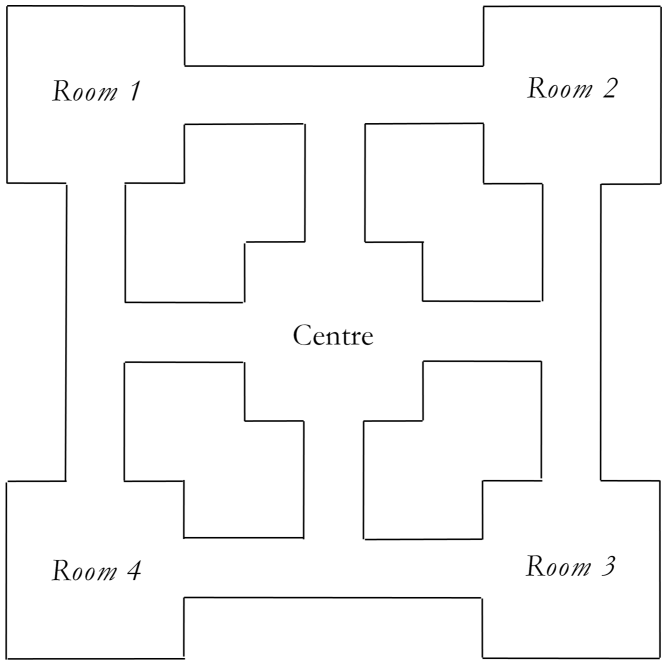
You will produce a 3D maze game and a game engine developed from the provided Entity-Component-System (ECS) framework. You will also critique and evaluate your design in terms of performance against an Inheritance design.

The object of the game is to collect 3 keys that will allow you to switch on an exit portal that will allow you to escape an enemy drone robot. The game will be rendered in 3D.

**Use the Rubric below to see a breakdown of the marks.**

Production of the Game and Engine Features - 55%

* Extend the provided Entity-Component-System (ECS) framework from lab 2 to include all the necessary Entities, Components, Systems and Utilities that you require
* Produce a maze. The design of the interior of the maze is up to you, however there will be a centre section and four other rooms around the maze.  An example can be found in Fig 1.  The height of all walls is 1m high (the player can see over all of the walls).  There are no ceilings.



**Fig 1 – The floor plan of the maze**

* There is a large skybox around the maze that will be visible from inside of the maze. The centre of the sky box will remain at the position of the camera to give the illusion that the sky box is very large, but it will rotate appropriately.
* Room 1 contains a key, and an object rolling around the floor. If the player touches the object, they will lose a life.
* Room 2 contains a key, and an object that bounces on the ground going from one side of the room to the other and then back again. If the player touches the object, they will lose a life.
* Room 3 contains a key.
* Room 4 contains the exit portal. The portal will be visible always.  The portal will initially be switched off but visible. The portal can only be switched on and used once all 3 keys have been collected and is operated by simply walking into it.  You will change the appearance of the portal when it is switched on.
* The collection of keys will be done by simply walking into them. A visual list will be presented to the player to indicate which keys have been collected.
* There will be a simple AI drone that will try to find the player. The drone will therefore move around the maze.  If the drone touches the player, then the player will lose a life and the positions of the player and drone will reset appropriately.
* The game starts with the player in Room 4 and the drone in the centre of the maze.
* The game ends if the player loses 3 lives or the player exits in the portal.
* Elements of the Head-Up Display (HUD) and Menu system will be developed, e.g. hearts for lives, game start screen, game over screens, option menus, etc.
* The walls, floors and skybox will be textured.
* All assets and settings will be loaded from files, e.g. geometry, textures, map, config etc.
* Provide sufficient lighting to enhance the virtual environment.
* The keyboard and mouse can control the movement of the player throughout the maze:
  + ‘UP’ arrow key or ‘W’ key – moves the player forward
  + ‘DOWN’ arrow key or ‘S’ key – moves the player backward
  + ‘LEFT’ mouse movement or ‘A’ key – rotates the user anti-clockwise
  + ‘RIGHT’ mouse movement or ‘D’ key – rotates the user clockwise
* Appropriate sound effects will be present in the scene. All sound files will be loaded from a file and played at runtime. The minimum sound effects are:
  + A sound will continuously be played from the portal’s 3D position. This sound will change when the portal becomes enabled – e.g. change the sound when it is switched on.
  + A sound will be played from the Drone’s 3D position – but only when the drone sees the player otherwise the drone will be quiet.
  + A sound will be played when each key is collected.
* For debugging/demo purposes, the Drone’s movement will be enabled/disabled using the ‘1’ key on the keyboard. This will pause the Drone but allow the user to move around and play the game.
* There will be simple collision detection between the player and the walls of the maze. For debugging/demo purposes, this will be enabled/disabled by pressing the ‘2’ key on the keyboard.

*NOTE: The look and representation of the graphics, including geometry, menus, screens, objects and textures, will be considering in the marking of your work.*

When you have completed the main functionality and features listed above, you can choose to implement one of the following.  Note, that you must specify which one of these you have chosen to implement in your report.  Failing to identify which one you have chosen to implement may result in no marks being awarded for your choice.

**CHOICE 1:** Add at least 4 different coloured spotlights located above the maze (in different locations from each other away from the centre) that face downwards onto the floor so that they illuminate the maze.  The direction of the spotlights should move around so that the player sees the lights move around the maze.

**CHOICE 2:** The OBJ loader loads all textures that are associated with an OBJ file.  These can include textures for bump mapping etc.  Use these extra textures to add visual effects to your objects by creating the appropriate shaders and components if required.  You can extend the OBJ loader if required.

**CHOICE 3:** Produce an 2D overlay map of the maze in one of corners of the screen.  This map will show the locations of all objects (including player and drone).

**CHOICE 4:** Compile your engine into a true library. The game will use this library rather than the source code of the engine.

Game Engine Design and Implementation - 30%

The code you produce will be as portable/modular as possible, so you will maximize the use of Abstraction in your design.

Your game engine code will be stored separately from your game code in Visual Studio.  Your game engine code is then used by your game.  You do not need to compile your game engine as a separate library, but you do need to keep the code separate from the game code in Visual Studio, so that a developer can see that the engine code is different to the game code.  This can be done simply by having the game engine code in a different folder than that of the game code.  A developer should be able to take only the engine code and start their own game without the existing code game.

You will load all settings from a file when the game starts. These settings will include information for Entities, general environment information\values, etc.

You will implement the general collision detection as part of the engine using Abstraction with an Engine Collision Manager and appropriate Systems.  Then you will implement a Game Specific Collision Manager that extends the Engine Collision Manager that can decide what to do in the game when there are collisions.

You will implement an Input Manager in your Engine.  You may also want to consider having a Game Specific Input Manager that extends the Engine Input Manager.

You will declare your data members in each class so that they are memory efficient.

Reports - 15%

***These two reports are worth a total of 15% of this assessment.  Therefore, you need to spend enough time to describe and justify your design explicitly and critique your design.  The reader must be able to understand your final design.***

**Game engine design and critique report**

This takes the form of a PDF document.  This document will include:

1. UML diagrams illustrating your final ECS design. Your design will show all classes that have been used in your framework.  The diagrams will only consider the class relationships (inheritance, associations, singletons, etc.) and should not include any member methods or member data ([NClass (Links to an external site.)](http://nclass.sourceforge.net/" \t "_blank) is good for creating UML diagrams). These will be used when marking your engine design and implementation. **If your design is complex, then you must create an overview of the whole design, and then include class diagrams of sub-sections**.
2. You will include descriptions and justifications of your design. You will briefly describe the role of each class and any relationships to other classes.  This could include a table of classes with a description of what they do.  These will be used when marking your engine design and implementation.
3. You will include a matrix/table illustrating each composition of each Entity like the example shown in lectures.
4. Download the **600098-Inheritance\_Basic\_Example.zip** file from lab 1 and explore the code and its structure.  Consider this basic Inheritance-based design and how you would improve it.  You do not need to write any code, but you will produce a better Inheritance-based design diagram.  Include your better Inheritance-based design diagram in the report.  Then compare and critique your improved Inheritance design with your final ECS design.  This will include reference to the four design principles of Abstractness, Modularity, Simplicity and Recyclability.  You may want to pose questions, e.g. “which methods or classes need to be changed by adding a new artefact in the game?”, “how much work is required to create each engine design compared to how much work is required to create the game based on these engines?” and/or “how would networking be implemented into each engine design?”.

**Evaluation report**

This takes the form of a PDF document.  This document will include:

1. A statement of whether you implement one of the choice implementations, and if so, which one you have implemented. You will include a description of how you implemented your choice. ***Please state even if you have not implemented one.***
2. A description of your AI algorithm(s) that you have implemented and how it works. You will provide appropriate diagram(s) as required.  You will also critique this algorithm in terms of its performance characteristics.
3. An evaluation of the game production problems in terms of programming that you encountered whilst producing the game, and how you overcame these problems.

Submission Requirements

You will submit a **single zip**file to Canvas.  The zip file will contain three folders, namely "**Code**", "**Reports**" and "**Run**".

All code and associated files (including Visual Studio files, resources etc.) will be placed in the ‘**Code**’ folder.

The reports will be placed in the ‘**Reports**’ folder.

A program executable (and associated resources and correct folders etc.) will be placed into a folder called ‘**Run**’.  From here, your software can be executed without having to compile your code. Therefore, test by copying this folder to a different location to your code to make sure that the user can simply run your software from the **Run** folder before submitting your work.