**{ PPCG }**

|  |
| --- |
| Project 2 |

**Overview**

Project 2 focuses on the basic implementation of a simple procedural level. You may choose ONE of the following THREE options.

**OPTION 1** – **BULLET HELL game level**

**Overview**

The main focus of Option 1 is the genre of bullet-hell style games, which provides plenty of scope for procedural content generation.

There are two main approaches to bullet hell games:

* each bullet is a separately controlled object
* bullets are generated by a particle system

One of the main reasons for using particle systems is performance. You can see some discussions about this at:

* <https://www.reddit.com/r/Unity3D/comments/2x96r7/optimizing_danmaku_style_games>
* <https://www.reddit.com/r/Unity3D/comments/29qaes/sprites_or_particles_for_a_bullet_hell_shooter/>

Here are some posts about Bullet Hell game patterns in general:

* <https://answers.unity.com/questions/199736/2d-bullet-hell-game-bullet-patterns.html> (but *don’t* use the Instantiate method in the sample code!)
* <https://www.gamedev.net/forums/topic/605497-theory-behind-common-bullet-hell-patterns/>

Here are some videos about creating common Bullet Hell patterns:

* <https://www.youtube.com/watch?v=UbBxGxGzZJ0>
* <https://www.youtube.com/watch?v=6JclriikDt0>

The basic ideas are fairly straightforward, mostly involving firing bullets outwards from a moving and rotating enemy.

Figure 1.1 below shows a very simple collection of patterns using particle effects. This Unity project is available on OLIVE for you to look at. Figure 1.2 shows a simple pattern made from combining a number of scripts, each of which specifies a simple bullet movement. The combination of these simple movements can give rise to complex patterns such as spirals and other swarm-style motion.



**Figure 1 Particle systems**

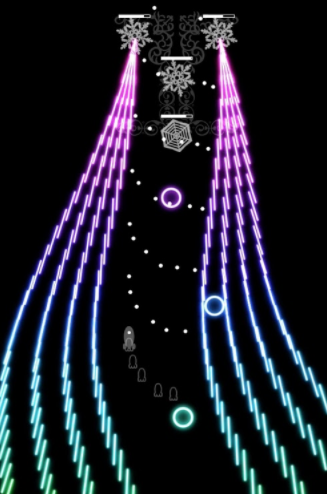
**Separate objects**

**Requirements**

IMPORTANT! You will score higher marks if you implement your bullet hell game using only separately controlled game objects.

The BASIC requirements, for a passing grade, are:

1. You must create a bullet hell level that lasts for at least *THIRTY* seconds (assume that the player will survive for this long)
2. Your level must include at least *THREE* major pattern changes. For this project, a major pattern change may be considered a change in bullet movement that is not due to a change in any parameter(s) of an existing pattern. Each of the patterns in Figure 1.3 below would be considered a major pattern.



**Figure 1.3: Bullet Hell patterns**

1. Your level must be *survivable*. This means that, after practice, an “average” player must be able to survive until the end of your level.
2. Your level must show *intelligent design*, in terms of the patterns used, pattern changes, and movement of enemies. You must discuss the design of your level in your Project Report (see below).
3. The level must be created entirely by code. You may create particle-system based or other prefabs, but there should be no gameplay-related objects in the Scene View before the game starts.

ADVANCED requirements, to score a higher grade, are:

1. Your level must include a player object. This player object must be able to move to avoid bullets, react when hit by a bullet, and fire at the enemies.
2. Your level must include at least one pattern that uses game objects, not particles. This means that each bullet is controlled by one or more scripts. In addition, your pattern generator must use *object pooling*. See this link for a good tutorial about object pooling: <http://catlikecoding.com/unity/tutorials/object-pools>
3. At least ONE pattern should include bullets that move in a complex path. For example, bullets could: move towards the player’s current position; change direction in mid-path in a non-random way; form complex geometric patterns; etc. See: <http://catlikecoding.com/unity/tutorials/basics/building-a-graph> for ideas.
4. Include at least THREE satellite enemies that move across the screen in a more complex path while firing bullets; each path must be different.
5. Your level must NOT be the same every time the game is played, i.e. the patterns or patterns combinations must change every time the game is played.

In addition, you may score up to 5 bonus marks if you implement a transition effect before each pattern change. You can see examples at:

<https://www.youtube.com/watch?v=7VK5GkQz4_c>

IMPORTANT! Simply adding any of the advanced requirements does not *guarantee* a higher grade. The *quality* of your work will be assessed before more marks are awarded.

For both basic and advanced requirements, your code must be suitably commented.

The marking scheme is given at the end of this document. If you have any questions about the marking criteria, please ask your tutor to explain.

**BULLET HELL - PROJECT REPORT**

You must submit a report with the following sections:

1. A screenshot of each pattern that you use.
2. For each pattern, a brief explanation of why you believe the pattern to be “fun” or “challenging” to play. This is obviously subjective, so treat this as a competition between developers who must “pitch” their patterns to a lead game designer for inclusion in the game. (*50 to 100 words for each pattern*)
3. A discussion of how you change patterns as the game progresses, i.e. a discussion of your “intelligent design.” (*150-250 words*)
4. A reflection on your game development process and thoughts while implementing this project. (*150-250 words*)
5. A brief review of an existing bullet hell style game. (*150-250 words*)

Your report MUST include a cover page with your NAME, MATRIC #, and CLASS.

**BULLET HELL – VIDEO PRESENTATION**

You must also submit a video presentation, as follows:

1. Minimum length of 5 minutes
2. Commentary (voice narration)
3. Overview of gameplay
4. Overview of at least ONE section of code used for procedural content generation – you must explain how the code works

*Option 2 is on the next page*

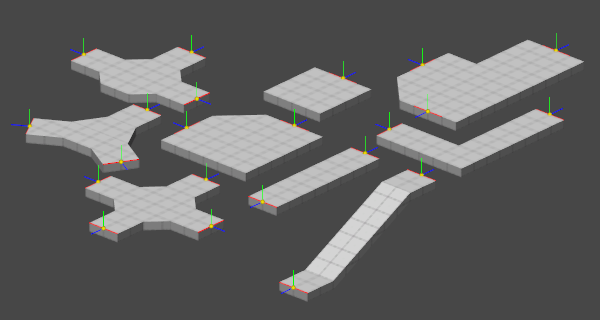
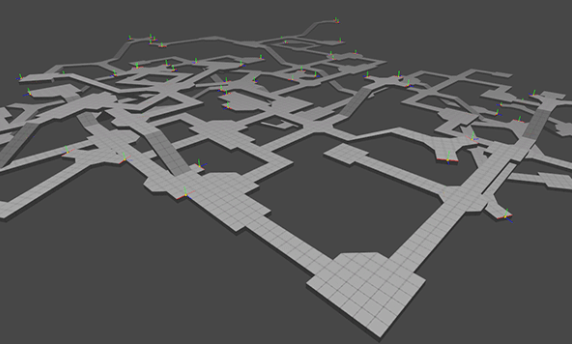
**OPTION 2** – **MODULAR level generation**

**Overview**

Option 2 is based on the tutorial at:

<https://gamedevelopment.tutsplus.com/tutorials/bake-your-own-3d-dungeons-with-procedural-recipes--gamedev-14360>

This covers how to create a procedurally generated level from a collection of modular meshes, as show in Figure 2.1 below.

**Figure 2.1**

The tutorial also covers the procedural placement of items in a room.

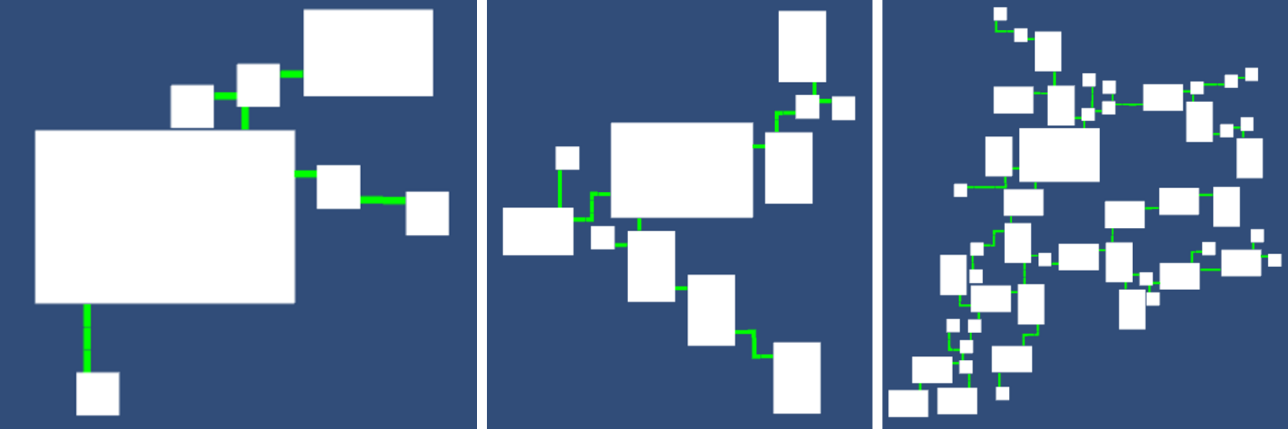
The source code Unity project can be downloaded from:

<https://github.com/tutsplus/3d-dungeons-with-procedural-recipes>

The technique covered here is fairly straightforward, involving the connecting of compatible modules based on their tags.

The tutorial is easy to follow. The only problem you might have is figuring out where the meshes are. The developer discusses this in the comments section. Basically, they exist as Mesh components of game objects in the Scene.

You will use this as the basis for your project, but in 2D, as shown in Figure 2.2.



**Figure 2.2**

This shows 5, 10 and 20 iterations of the algorithm.

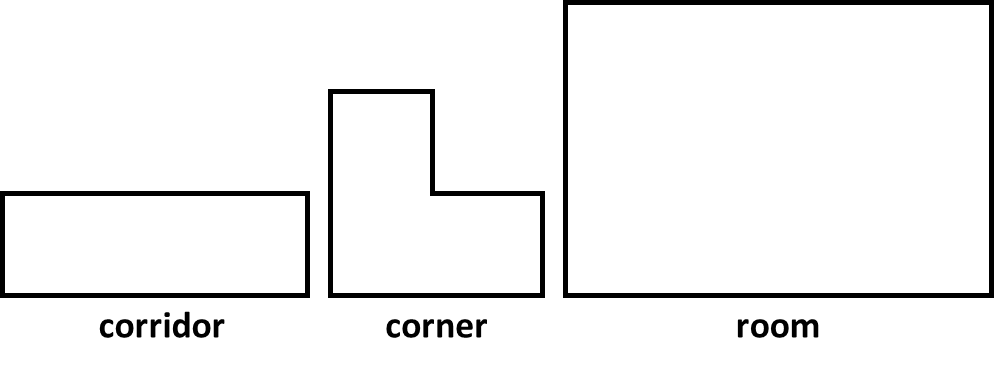
**Requirements**

Note that this option requires a good understanding of algorithm design, data structures, and general programming. It has been scoped to be achievable for this project in the time given.

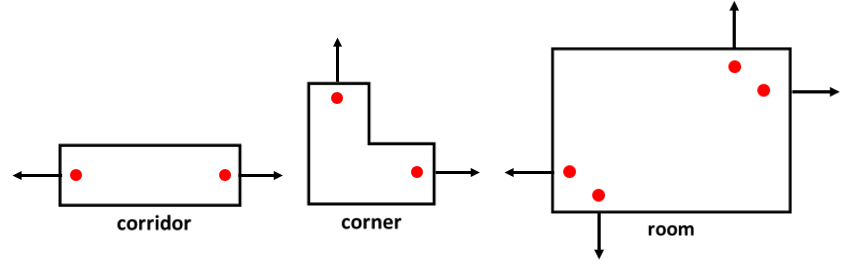
If you understand the tutorial, the basic requirements are straightforward. The advanced requirements will need careful thought.

The BASIC requirements, for a passing grade, are:

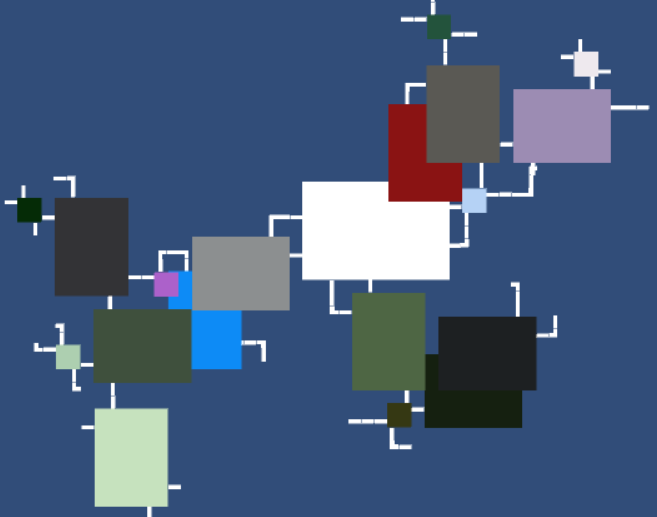
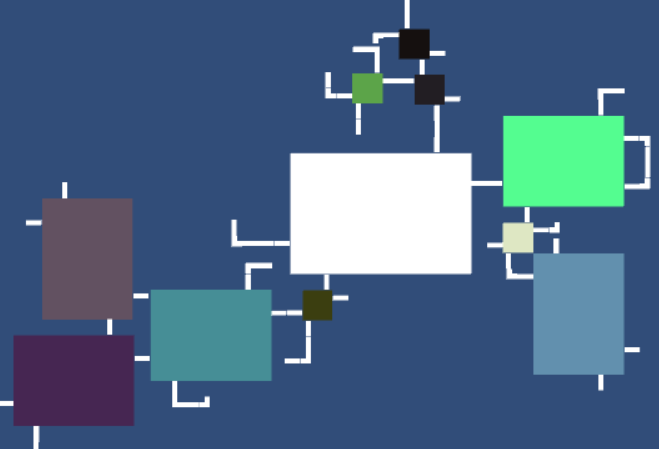
1. You must create the following sprites:



The links, or exits, for each sprite module are shown below:

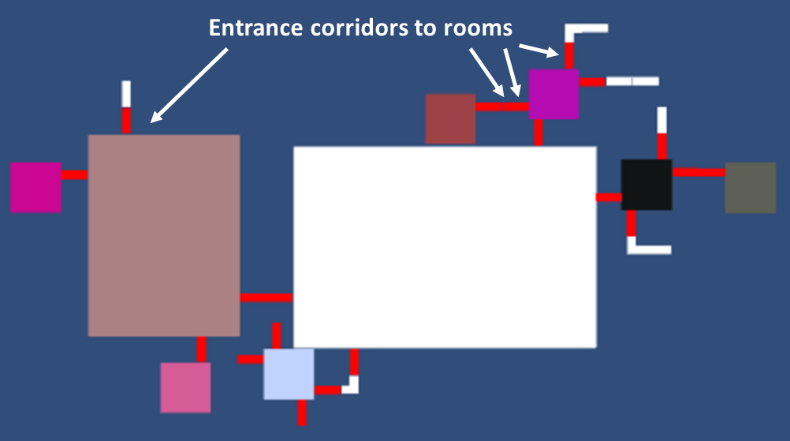


1. These sprites must be used to generate a map of connected rooms (see Figure 2.2). You must use the tutorial as the basis for your map generation. As in the tutorial, you must be able to change the number of iterations used to generate the map.
2. As shown in Figure 2.3, if the colours are randomised you can see a lot of overlap between sprites. You must implement a way to eliminate this overlap, as shown in Figure 2.4.

**Figure 2.3 Figure 2.4**

1. You must highlight the entrance corridor sprites to every room, as shown in Figure 2.5. This is not a necessary part of map generation, but it does show that you understand how the algorithm works. (Also, in a game you may want to identify certain regions for the design of different gameplay situations, such as more powerful guard enemies outside each entrance.)

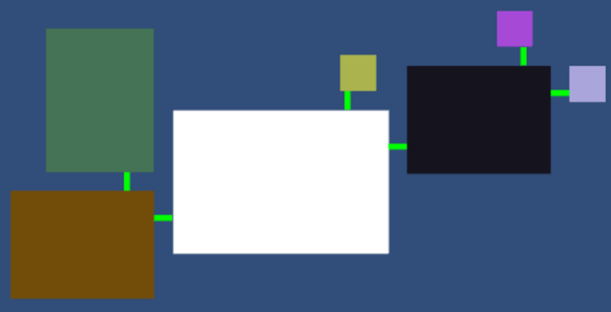


**Figure 2.5**

1. Your map must have at least THREE different sizes of rooms, with different X and Y scales. Figure 2.5 has small, medium, and large rooms.

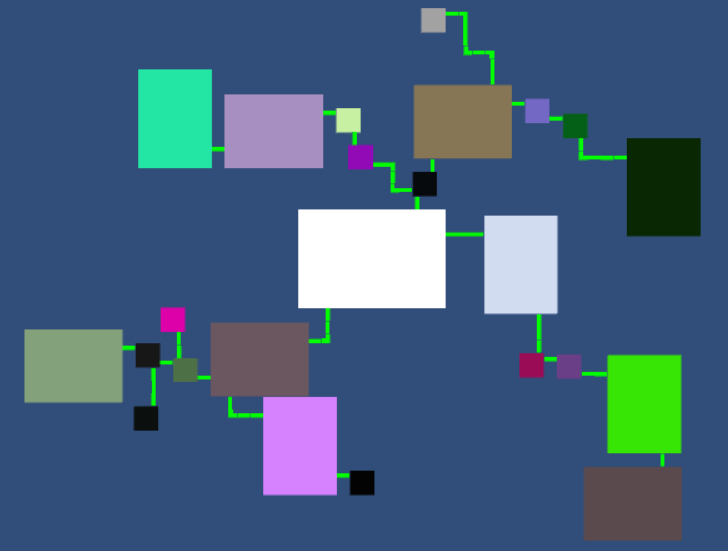
ADVANCED requirements, to score a higher grade, are:

1. You must make the room sizes procedural, i.e. each room may have a different size. You must specify minimum and maximum room dimensions.
2. Note that the map has many dead ends that do not lead to a room. You must eliminate these, as shown in Figure 2.6.



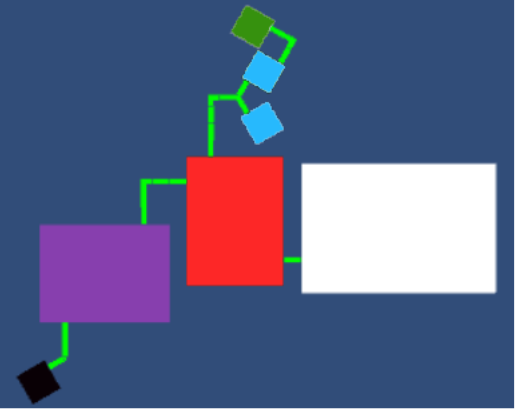
**Figure 2.6**

The pruning of dead ends must work for all map sizes, as shown in Figure 2.7. There must be a path to every room, i.e. no isolated rooms.



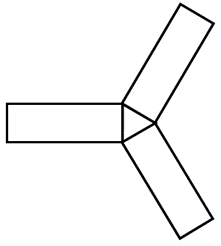
**Figure 2.7**

1. You must add an additional sprite module that acts as a three-way junction, as shown by the white arrow in Figure 2.8. Dead-end pruning must work with this.

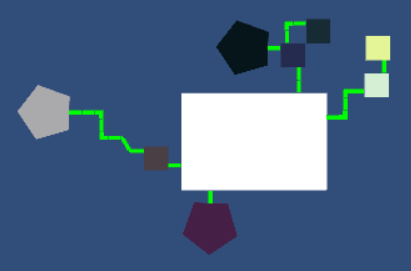


**Figure 2.8**

The junction sprite should look something like:



1. You must add at least ONE different shaped room, as shown by the white arrows in Figure 2.9. You can make the room any shape you like. Your room must have an appropriate number and posititioning of exits. In the example below, the pentagonal rooms would have five exits, one for each side. Again, dead-end pruning must work with this.



**Figure 2.9**

1. You must come up with ONE additional requirement to extend the map. You should discuss this with your tutor before implementation.

In addition, you may score up to 5 bonus marks if you can figure out how to add a *four*-way junction, and make this work with dead-end pruning.

IMPORTANT! Simply adding any of the advanced requirements does not *guarantee* a higher grade. The *quality* of your work will be assessed before more marks are awarded.

For both basic and advanced requirements, your code must be suitably commented.

The marking scheme is given at the end of this document. If you have any questions about the marking criteria, please ask your tutor to explain.

**MODULAR level generation - PROJECT REPORT**

You must submit a report with the following sections:

1. At least 5 screenshots of your map, with an increasing number of iterations.
2. A brief explanation of how the map generation algorithm works, with diagrams if necessary. (*150-250 words*)
3. A brief explanation of your solution to the overlap problem. (*150-250 words*)
4. A reflection on your development process and thoughts while implementing this project. (*150-250 words*)
5. A brief discussion of an alternative 2D map generation algorithm, with a comparison to the approach taken in this project. (*150-250 words*)

Your report MUST include a cover page with your NAME, MATRIC #, and CLASS.

**MODULAR level generation – VIDEO PRESENTATION**

You must also submit a video presentation, as follows:

1. Minimum length of 5 minutes
2. Commentary (voice narration)
3. Overview of the approach you took to constructing the 2D map, with reference to appropriate and relevant sections of your code.

*Option 3 is on the next page*

**OPTION 3** – **SELF-DIRECTED**

**Overview**

You may suggest your own project topic. If accepted, you must devise your own set of basic and advanced requirements for the implementation and project report assessment components, which must be agreed upon by your tutor.

The same requirements as for Option 1 and Option 2 must be submitted.

The same marking scheme (below) will be used for your project, if accepted.

Your tutor must be confident that your suggested project is an original piece of work that cannot be copied from the Internet or any other source.

Your suggested project may *originate* from a tutorial or some other source, but you must scope it so that there is room for original extension and enhancement.

This may be in the form of a technical demonstration of a procedural content generation technique, or the modification of an existing game to use procedural content generation.

Note that it will not be easy to have your suggested project approved!

*The marking scheme is on the next page*

**Name:**

**Class:**

**TOTAL MARK:**

**PPCG Project #2 - Marking Scheme**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **F: <5**  **Poor** | **D: >= 5 < 6**  **Adequate** | **C: >= 6 < 7**  **Good** | **B: >= 7 < 8**  **Very Good** | **A: >= 8**  **Excellent** | **Raw Mark (/10)** | **Final Mark (weight)** |
| **Implementation Requirements and code commenting** \*  **(40%)** | One or more basic requirements NOT completed at all. | One or more basic requirements NOT completed to a high standard. | ALL basic requirements completed to a high standard. | ALL basic requirements, and at least TWO advanced requirements, completed to a high standard. | ALL basic and advanced requirements completed to a high standard. |  | **x4** |
| **Video Presentation**  **(10%)** | Less than 5 minutes in length. | No commentary, inadequate overview of gameplay/ approach, inadequate code overview. | Has commentary, adequate overview of gameplay/approach, adequate code overview. | No commentary, good overview of gameplay/ approach, good code overview. | No commentary, excellent overview of gameplay/ approach, exellent code overview. |  | **x1** |
| **Reflection Report (15%)** | Poor attempt with some parts missing or incomplete. | Generally adequate attempt, but with at least one individual section of poor quality. | Adequate attempt, but with overall poor analysis and reflection. | Good attempt, with good analysis and reflection. | Excellent attempt, with excellent analysis and reflection. |  | **x1.5** |
| **Bonus Marks** |  | | | | | **/5** |  |
| **TOTAL** |  | | | | |  |  |

**Comments**

\* Note that although you may have implemented all requirements, your score will be based on the assessed standard of your work. The marking scheme indicates the minimum requirements needed for a particular grade.