GENBLOCKS - INDOOR TERRAIN GENERATION

Final Year Project

Q5061094 – Charlie Maxwell

Contents

Indoor Terrain Generation by Charlie Maxwell Q5061094

Introduction

GenBlocks is a hand made plug-in for the Unity Engine (ver. 2019.1.14 and higher) designed specifically for use in the creation of indoor environments, these environments can then be exported for use as and where the user sees fit. GenBlocks is designed to be as user friendly as possible, aiming to bring the flexibility of a custom terrain tool to users who may have little to no experience within the Unity engine itself.

Setup

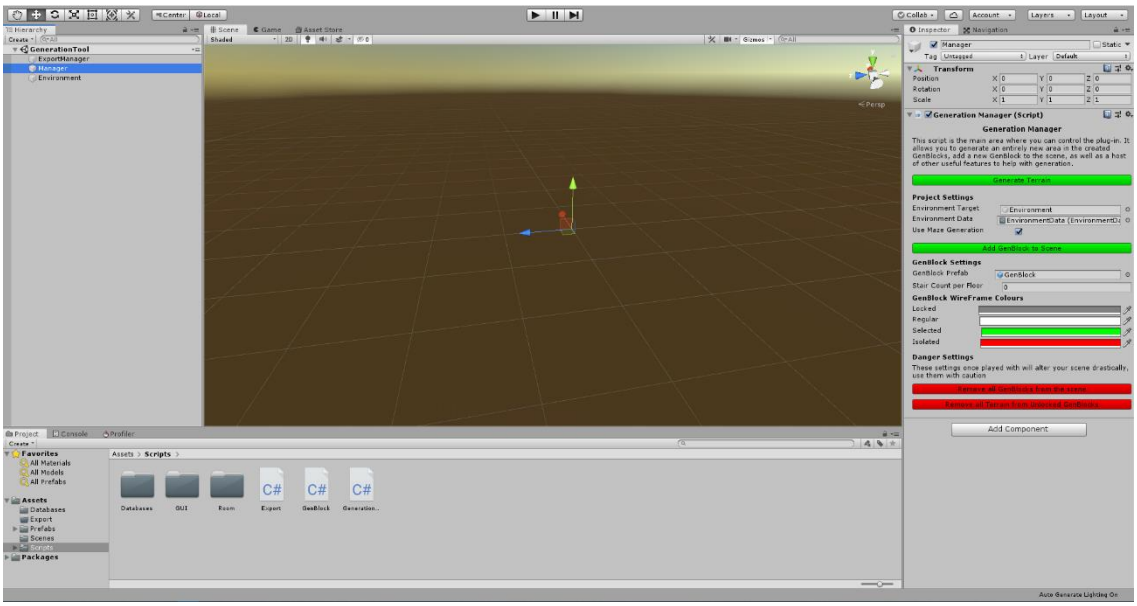
GenBlocks is a random generation too built specifically for indoor environments within the Unity games creation engine (versions 2019.1.14 and higher). Using the following steps, the solution can be setup ready for use.

1. Ensure that you have installed a compatible version of Unity.
2. Open the project folder through the Unity program.
3. Open the Generation scene.

Once the Generation scene is opened in editor, navigating to the Generation Manager reveals the controls to get started within the plug in.

Use

Once the Generation scene has been opened the user can begin the process of laying out their scene. The eponymous GenBlocks are the base of the generation method and are vital to the way that the plug-in functions. By adding these blocks in to the environment and positioning them in connected ways across the snapping grid, a 3D space can be mapped out in which the generation will take place.



Once GenBlocks are in place, there are various setting options that can be used to customise the generation technique to the user's content. Once the environment has be set up to the user's specification, the generation process will begin. This will use the settings that have been provided to

either generate an open space inside the defined area, or use the same area in a variation of a depth first Recursive Backtracking algorithm to provide a more maze like layout; with both options pulling from a user defined set of room prefabs that make use of the Unity engines recently integrated nested prefab system. This allows for a great degree of control in both the overall layout of the generated terrain as well as its room by room content.

Scope

While the project was in its infancy it was imperative to start and move forward with clearly defined milestones that would stretch throughout the development period to allow for a timely and precise delivery of certain large scale features, meaning that the project could progress at a steady pace and be delivered in a presentable manner. The project started with a base deliverable in mind, then the following milestones were set up and adjusted as the development time went on in a "stretch goal" mentality, meaning that features were arranged in order of feasibility to be included in the time frame that was provided for the project production and then tapered out or reduced in scale if they seemed unreachable.

The module-based approach to the development lends itself well to the sprint-based management system that was chosen at the beginning of the project as well as proving to be useful for seeing tangible improvement on iterations of the plug in as development time charts onwards.

Constrains

During the design and development process an eye was kept out for any constraints that would prove menacing for the project. Initially it was understood that the Unity engine itself would be a limiting factor for what was proposed when compared to programming a solution from scratch, but realistically creating a plug in through Unity was a much more feasible task and meant that throughout the project the Unity development pages could be used to stem any issues that arose through development. Throughout the design phase it was decided that the generation should be completed in real time and all entirely within the scene view of Unity with no need to enter a game view, this way the GenBlock framework could be added into an existing scene with as little interference to runtime scripts as possible, the environment generated and then the GenBlock framework removed with the generated environment being left behind for the users own machinations. This proved to cause issues during the initial development process, with the use of singletons for providing Instances of certain scripts needing to be scrapped in the pursuit of a more reliable result; eventually leading to all singletons being removed from the script designs, the script pipeline being slimmed down in favour of larger scripts, and the GenBlocks all keeping a reference to the main port of call script, the Generation Manager.

A further constraint that needed to be considered and decided upon within the project was the generation technique for the random room placement. As usability and user customisation was decided upon as the main focus of the plug-in, the generation needed to be robust as well as user definable. While there are many well documented techniques for generating maze like structures, the majority of implementations that are available for study that could be found at the beginning of the project were suited towards creating a maze in a pre-defined grid size along a 2D space rather than in 3D, which is what GenBlocks is designed around. This meant that along with the initial research, some form of adaptation was needed to existing formula. Taking this in to consideration

when planning the timescale of the project did prove difficult, but as various methods were tested and then discarded, a Recursive Backtracking style approach was taken, adapted to use the special data that had been set up within the gen blocks to create a linked list style tree to search through in the 3D space.

Initial Design and Subsequent Changes

During the initial research and design phase of the project research was done on solutions for Indoor Terrain Generation that were already widely available, specifically those geared towards use in the Unity engine, to see where they could be learned from when it came time to start production on the project, as well as to make sure that the solution incorporated enough unique properties to allow it to stand individually away from those like it and be a worthwhile addition to the scene. While there was a splattering of interesting and well-made plug-ins, not many of them had a great deal of visual feedback in their usage and as something that was a large focus of the initial design it was easy to see that production on the plug-in could be continued in earnest.

Although the initial design was decided upon early, there were aspects of the extra milestone goals that did have to be trimmed away from the sprint scheduling and added to a list of improvements that will be made in the future due to time allotted for specific tasks being over ran. Specifically finding freely available assets to use for the rooms and corridors themselves proved to be more difficult that originally anticipated, leading to the decision being made to simply make the environmental assets in house. Although this did take a chunk of development time away, it did lead to a more concise deliverable package in the long terms as all assets in the project were uniquely created for GenBlocks.

A further feature that ended up being cut and moved to a future implementation was the “Clutter” feature that had been part of the initial design to set the plug-in even further apart from others that are already available. Cutting these features left more time to achieve a more realistic and feature complete base product that also serves as a basis for future improvement, focusing on a random generation aspect allowing for the piecing together of environmental assets as well as allotting time to write the custom inspectors for the plug-in that are paramount to a user friendly experience.

Comparisons to Existing Approaches

In the Unity plug-in space there are a wide assortment of solutions available for users that are looking to create realistic terrains both for indoor and outdoor use cases, and as procedural terrain generation has been something that has been included in games for decades it is also something that has multiple sources of reference both in games and written format for those that want to create their own terrain generation. For this project’s generation method however, a unique mix of existing techniques as well as a bespoke room based linked list set up was created to allow for the user driven experience that was initially designed for was well as reliable and solid base for the generation aspect of the plug-in.

Room prefabs that are pulled from the Room database have a Room script attached to them, in this script there is an enum flag that sets where in the room there is an entry point, these flags are also assigned at generation to each GenBlock, with the block checking its surroundings using a raycast and updating its spatial data to reflect where it is attached to other blocks. These neighbours are added in to a dictionary that is indexed using the special enum to allow for direction in 3D space relative to neighbouring blocks to be accessed at run time for the generation algorithm.

```
public void UpdateSpacialProperties()
{
    // Reset spacial data
    mSpacialData.mProperties = 0;
    mSpacialData.mNeighbours = new Dictionary<GenBlockSpacialProperties, GenBlock>();

    // Cast out to find neighbours on the correct collision layer.
    int layerMask = 1 << 9;
    RaycastHit hit;
    if (Physics.Raycast(transform.position, Vector3.left, out hit, mGenerationManager.mGridSize / 2, layerMask))
    {
        mSpacialData.mProperties |= GenBlockSpacialProperties.LEFT;
        mSpacialData.mNeighbours.Add(GenBlockSpacialProperties.LEFT, hit.collider.gameObject.GetComponent<GenBlock>());
    }
    if (Physics.Raycast(transform.position, Vector3.right, out hit, mGenerationManager.mGridSize / 2, layerMask))
    {
        mSpacialData.mProperties |= GenBlockSpacialProperties.RIGHT;
        mSpacialData.mNeighbours.Add(GenBlockSpacialProperties.RIGHT, hit.collider.gameObject.GetComponent<GenBlock>());
    }
    if (Physics.Raycast(transform.position, Vector3.forward, out hit, mGenerationManager.mGridSize / 2, layerMask))
    {
        mSpacialData.mProperties |= GenBlockSpacialProperties.FORWARD;
        mSpacialData.mNeighbours.Add(GenBlockSpacialProperties.FORWARD, hit.collider.gameObject.GetComponent<GenBlock>());
    }
    if (Physics.Raycast(transform.position, Vector3.back, out hit, mGenerationManager.mGridSize / 2, layerMask))
    {
        mSpacialData.mProperties |= GenBlockSpacialProperties.BACK;
        mSpacialData.mNeighbours.Add(GenBlockSpacialProperties.BACK, hit.collider.gameObject.GetComponent<GenBlock>());
    }
    if (Physics.Raycast(transform.position, Vector3.up, out hit, mGenerationManager.mGridSize / 2, layerMask))
    {
        mSpacialData.mProperties |= GenBlockSpacialProperties.UP;
        mSpacialData.mNeighbours.Add(GenBlockSpacialProperties.UP, hit.collider.gameObject.GetComponent<GenBlock>());
    }
    if (Physics.Raycast(transform.position, Vector3.down, out hit, mGenerationManager.mGridSize / 2, layerMask))
    {
        mSpacialData.mProperties |= GenBlockSpacialProperties.DOWN;
        mSpacialData.mNeighbours.Add(GenBlockSpacialProperties.DOWN, hit.collider.gameObject.GetComponent<GenBlock>());
    }

    // Set isolated if we are
    mSpacialData.mIsolated = mSpacialData.mProperties == 0 ? true : false;
}
```

From here the generation method decided upon by the user is then executed. If the user has opted for a maze-like terrain, the program will conduct its specialised tree search to plot an accurate course through the room nodes. Starting from a random block on the lowest level of the terrain, a random path is followed through the neighbour nodes that have already been generated, adding them to a stack to be processed in sequence until a dead end is found, from here we trace back through the stack until we find a node that still has viable paths to take. This breadth first tree search generates winding paths through the terrain that don’t loop back on each other.

```
// Generate our terrain using a backwards recursive search to define the paths.
private void GenerateMaze(GenBlock startBlock)
{
    // Begin Recursive Backtracking
    Stack<GenBlock> blockStack = new Stack<GenBlock>();
    blockStack.Push(startBlock);
    while (blockStack.Count > 0)
    {
        // Pop our current block and process its spacial data
        GenBlock block = blockStack.Pop();
        if (!block.mSpacialData.mDeadEnd)
        {
            // Now that the block is popped, we have visited here
            block.mSpacialData.mVisited = true;

            // Check for available directions
            var candidatesDirections = GetViableDirections(block);
            if (candidatesDirections.Count > 0)
            {
                // Get a random Direction
                var direction = candidatesDirections[Random.Range(0, candidatesDirections.Count - 1)];

                // Set our used path in this direction
                block.mSpacialData.mUsedPaths |= direction;

                // Set our new rooms used path in the opposite direction
                block.mSpacialData.mNeighbours[direction].mSpacialData.mUsedPaths |= GetOppositeDirection(direction);

                // Add our current block and then our next block to the stack
                blockStack.Push(block);
                blockStack.Push(block.mSpacialData.mNeighbours[direction]);
            }
            else
            {
                // We have nowhere to go, so we are a dead end
                block.mSpacialData.mDeadEnd = true;
            }
        }

        // We're a dead end, so generate our terrain
        if (block.mSpacialData.mDeadEnd)
        {
            GenerateBlockTerrain(block);
        }
    }
}
```

Although there are more intricate ways of generating paths through 3D space than this, this method was chosen to be adapted so that there would be a solid and reliable backbone to the generation that could then be expanded upon.

If the user decides on an open plan generation, rather than charting a path through the available neighbours, a terrain will instead be generated where every possible path is opened up; leading to a shell-like construction.

During the next section of the generation phase, as each GenBlock becomes a dead end in the tree search, the block loops through the available terrain pieces that match its used paths and tries to find rooms that either perfectly match the spatial data or are a rotated variation of the GenBlocks spatial data. These rooms are then picked from randomly and instantiated within the scene.

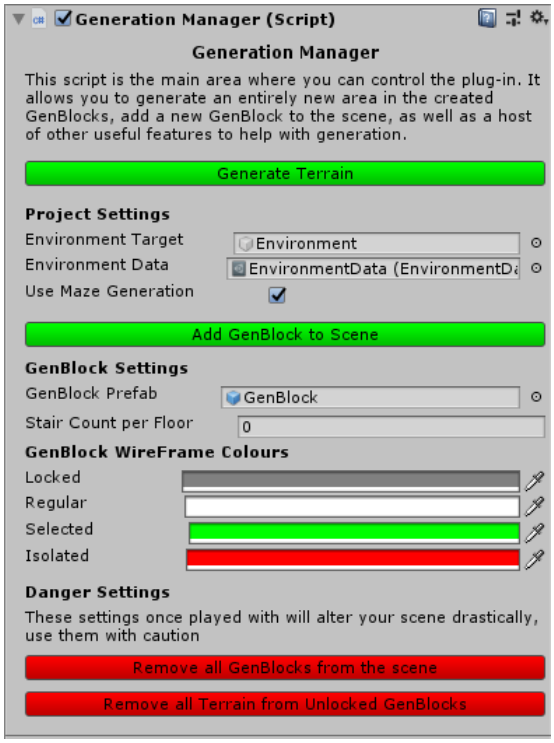
Although this approach works well for the current projects needs, there are multiple improvements to the solution that could be made in the future that were picked up on during development that would lead to an even deeper amount of user customisation in the generation technique, such as blending other methods of path generation to provide options for looping and more varied terrain.

Ethics

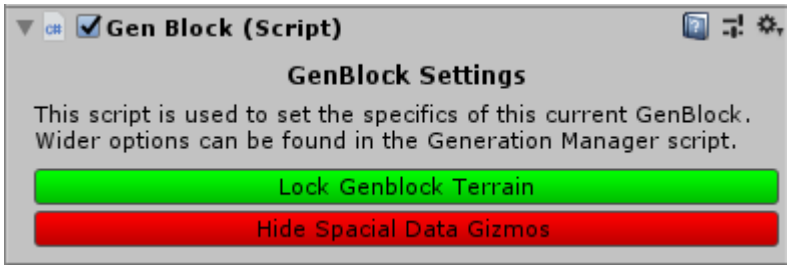
As was discussed in the initial project propos, there wasn't much need for consideration of any large scale ethical issues when designing the solution and writing the subsequent report as it had been planned from the beginning that development would be kept entirely internal, with no real world users to be exposed to the project during the development time. During the implementation process as features were being added and tweaked it was a consideration that user feedback sessions could be ran to guage the usefulness of certain features or how the custom inspector was being layed out for each individual aspect of the project but this was dismissed relatively early on in the process as running user tests would have eaten in to the development time that was allotted for the project.

In terms of industry related concerns of ethics, the projects use case of the Unity engine falls well within the Academic and Personal licenses extended to those users so that was not a concern and as environment assets did end up being created internally there was no accrediting needed to any assets that may have been found elsewhere.

Project outcome



Though it has been paired down slightly from the vision in the initial design, the plug-in, design, and implementation process went on in a steady and controlled manner to produce a final product that delivers a user friendly and highly customisable indoor generation toolkit. On the side of the artefact itself there is much to discuss. The GenBlock project is set up to allow users to plan out the area of their generation using the blocks themselves in a mock-up of the plug-in being added into a Unity scene. The main control point for users is the Generation Manager script within a scene object; from here the general settings for the plug-in can be calibrated to the users liking, as well as providing the buttons for adding in a new GenBlock to the scene and generating the environment itself.



GenBlocks also have their own individual options such as being locked into the environment, making them exempt from the next set of random generation, this can be useful for if the user has a particular group of blocks that they like the layout of but don't want to keep the rest of the terrain. The "Lock" functionality also makes the GenBlock exempt from the Terrain Clear options that are also present in the Generation Manager script. Adding to the user customisation of the scene, the GenBlock wireframe will change to different colours to represent that specific blocks current status of which there are four: Regular, Isolated, Locked and Selected; all of which the user can define colours for in the plug-in. (Isolated blocks are ones that aren't attached to any other GenBlocks and as such are also skipped over in the generation process.)

Once GenBlocks have been added to the scene and rearranged to the users liking, the generation can be commenced from the Generation Manager script; looping through the GenBlocks, organising them into floors, and finally generating terrain using rooms that have been decided to be suitable by each individual block. From here the user can customise the generation as they please, locking down blocks and regenerating until they find a layout that they are happy with, then can start the export process. The Export script allows users to take their generated terrain and save it to a specified folder with a user defined name as a Unity prefab if they don't want to use the terrain within the current scene, allowing for even further tweaking and customisation of the environment generated.

Though the generator works as intended and is easy to navigate and use, there are limitations within the current set up that need to be addressed. First of all due to the environmental assets having to be created rather than sourced as originally intended and that adding to the development time, there is currently only one variation of each type of room meaning that although the current generation is accurate it doesn't produce a large amount of variance. This however can be assuaged from a user stand points. Within the plug-in there is a folder of environmental pieces each of which has a Room script attached to it containing the rooms "Spatial Information"; users can add environment pieces into this folder, add a calibrated room script to the prefab, and then add them to the Environment database. Once the piece is included in the database it is simply added to the pool of available rooms and included within the generation. Although this problem is technically a limitation of the artefact, it is not an issue with the logic of the plug-in itself.

A further limitation of the plug-in however is that although the original design called for the plug in to be as user friendly as possible, there is still some base familiarity with the Unity engine that is presumed. Ideally this would have been circumnavigate in a more intuitive to navigate control panel, perhaps being implemented with Unity's custom window scripts that were recently added to the engine itself.

Codebase

The main body of the project can be found in two sections, code devoted to the custom UI elements and that which handles the plug-ins functionality itself. Knowing that one of the core targets to hit

with the project was to gain a better understanding of the Unity editor and deliver a positive user experience much of the development time was spent refining the layout and functionality of the editor scripts to make sure that they had a positive User Experience design.

The rest of the code base however is split between the core systems of GenBlocks project. The Generation Manager script houses most of the functionality of the plug-in, this script in particular used most of the conventional methods of programming that were already understood before the project began, with exception of being created to execute in editor. Executing in editor means that the Update() function fires not as fast as possible as is the case in regular game view within the Unity engine but instead will execute whenever there is a change to the scene view such as a GenBlock being moved or a change being made in an objects inspector – like when the Generate button within this script is clicked by the user in the inspector. This didn't prove too much of an issue once the Singleton instances that were originally in use within my scripts had been removed, but it did mean that the original planning of having the Grid and the Generation settings being attached to two different objects had to be scrapped and both of those scripts folded together to produce far more reliable results.

The editor execution did prove to be an issue when it came to programming the GenBlocks themselves. The GenBlocks are the place where the user interacts with the plug-in the most so it was paramount to the experience to make sure that they worked quickly and reliably as to make sure that the user experience wasn't slow or frustrating, as having a plug-in that works but is cumbersome can sometimes be worse than one that just doesn't work. The main issue that revealed itself in their implementation was making sure that the code that governed them snapping to the user defined grid updated correctly and provided the visceral "pop" feedback you get when watching something snap in to place; after some tinkering and research I'm really happy with how it's turned out. The GenBlocks are also in charge of checking for their own room matches within the room database, which they do with a flag comparison between the two, then finding the amount of 90 degree rotations between the matching pair.

Finally, careful consideration had to be paid to the logic and upkeep of the codebase when it came to the generation techniques themselves. During the creation of the tree-search that is used to govern the maze generation in the terrain, several revisions to both the over all structure as well as the code readability had to be made in order to make the generation as fast and as reliable as possible.

Development Process

Throughout the development process there were a few routines that helped with the production of the project. Having a base design document as a leaping off point helped enormously with the planned continual design of the plug-in, as understanding within the engine of choice grew, it became clear that certain features would have to be tweaked or removed to ensure a useable and cohesive product was delivered by the time the hand in came around. This paired with vigilance in keeping tidy scripts and cohesive naming conventions lead to a repeated process of being able to do sections of work, make any updates needed to the design, update the dev blog that was being kept during the development window when applicable, then pick up where development had been left off in a timely and reasonable manner when coming back to do work.

The development process during the time allotted to for the creation of GenBlocks started off incredibly strong, within the first week of the proposal being submitted a GitHub repository had been set up to house the source control for the project, a Trello board had been set up to keep track of design and progress being made, as well as having set up a calendar style tracker for specific milestones that needed to hit be hit at specific points in time throughout the development cycle. Although it would have been additive to have squeezed in some of the extra features that had been planned out such as the addition of Editor Windows and a more advanced customisation of the terrain generation, the production schedule that was used resulted in a deliverable product that stuck to the general design of the initial proposal.

Improvements to the project

In terms of improvements to the project a few ideas were made evident during the incremental design process on what can be expanded or reworked to provide a smoother or more in depth experience, although they proved unattainable during the initial development of the plug-in the expansion ideas have been noted and could help make the plug-in much more feature rich if it were to be revisited.

First and foremost, the features that had to rescoped out of the design for the plug-in would be ones that should be added back into the project. The clutter feature that was outlined in the initial proposal would be something that would help make the plug-in stand out if it were to ever be released to consumers in the future and allow for a deeper customisation of the generation. Adding points within the room and corridor prefabs that are designated as “Clutter” points, with clutter amounts being designated in the Generation Manager as well as having each GenBlock be given the option to override the global clutter amount.

In terms of the generation technique itself it currently only generates maze paths in one way, this is something I would like to change and improve upon in the future. This would entail a rewrite of how generation is currently handled but ideally it would allow users to choose how open or closed off the design of the terrains is, with a slider in the inspector, rather than simply choosing if a maze generation technique should be used or not. Open design terrains would generate as they currently do when maze generation is deselected, where if there is a connection to an adjacent block, that connection is traversable by the player, whereas more closed off maze like terrains would take fewer of the connections available to them and generate a more twisting environment. A further boon to the generation would be allowing users to have GenBlocks of multiple sizes in the same scene, leading to more interesting room layouts where some rooms span multiple floors, but this again would require a re-write of the current generation method.

Expanding on the initial focus of wanting to learn more about the Unity engine as well as proving a user-friendly experience a further improvement that could be made would be the inclusion of custom Editor Windows. Although there are custom Inspectors for the majority of my scripts there are still areas where some experimenting with the Editor Window system would benefit the artefact greatly; one place in particular where this could prove beneficial is in the Export script. Instead of having the user define the prefab name and path by text it would be much more intuitive to instead have a popup Editor Window that allows the user to save the environment as a prefab or even as a Unity Package for greater control over their creation.

Overall Evaluation

Overall the final deliverable product, while missing some of the features originally detailed in the initial design document, delivers a user friendly, editor driven generation method that produces customisable environments that can be exported and used elsewhere to the user's specifications. During development and upon reflection there are areas of the system that can be tweaked and improved, but with what is present and current in this version of GenBlocks, the plug-in itself delivers a majority of what it strove to do from the get-go. Over the time that was given for the development of the project, using a continual design process and a sprint-based methodology, the system went from infancy to the form that it is being delivered in in a measurable way that displays progress at every step, as well as providing the sought after opportunity to learn more about the engine that it was developed in and the developmental process of creating a plug-in and a system based around random generation.

The computing project as a who proves to be an invaluable learning experience for those that partake in it, from those that have been in education for there whole lives up to and including those that have had work setting experience. Having an extended period of time to develop an idea from start to finish entirely under the steam of the creator, with careful guidance from the tutors assigned to them, helps them grow and extend their skills and provides a good insight into active development.