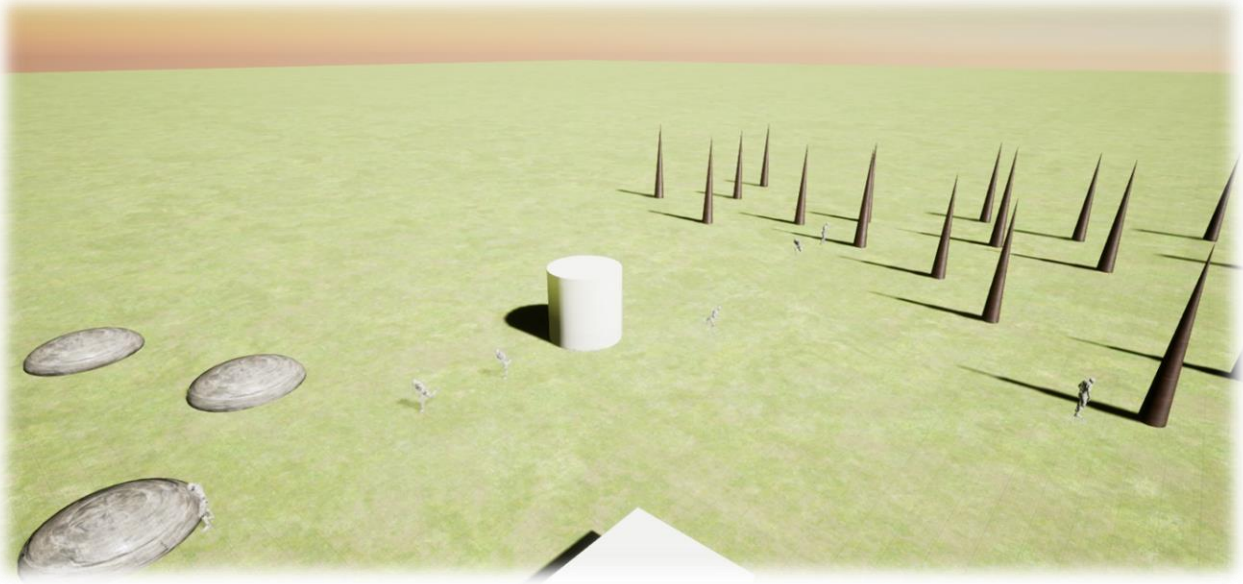


Gameplay Mechanics Development

Coursework Report

Gavin George

1501029

Document Version History

Current
Version 1.0.0 –
April 29th, 2019

V 0.1.5 –
April 10th, 2019
V 0.1.0 –
March 31st, 2019
V 0.0.9 –
March 24th, 2019
V 0.0.8 –
March 20th, 2019
V 0.0.7 –
March 11th, 2019
V 0.0.6 –
February 27th, 2019
V 0.0.5 –
February 25th, 2019
V 0.0.4 –
February 23rd, 2019
V 0.0.3 –
February 19th, 2019
V 0.0.2 –
February 12th, 2019
V 0.0.1 –
January 30th, 2019

Table of Contents

Preface

This document is a comprehensive report intended to accompany the submitted coursework project, “CMP302 Gameplay Mechanics Development” by the author, Gavin George. Included within this report is a detailed specification of each facet of the chosen mechanic. A summary description of the mechanic is as follows: the mechanic features RTS style camera controls, selectable units, base-building and resource management. These features are typical of a real time strategy game, which was the intended objective; to implement RTS base building mechanics. The system allows the user to direct units to construct buildings and harvest resources, navigating and viewing the environment using a scrollable/panning/rotatable spring arm camera. The aim of this report is to deliver an in-depth explanation of the requirements and specifications of the system, to delineate the technical aspects and techniques used to achieve the implementation and finally to explain the development process of the project, providing UML diagrams as a visual aid and an online video demo to present the system.

System Requirements Specification

1.1 Introduction

1.1.1 Objective

The overall objective of this project was to provide a solution to a concept that the author synthesized in accordance with the coursework brief [6.1]. The concept of real time strategy base building was popularised through games such as Warcraft [6.3, 1] & Age of Empires [6.3, 2]. The RTS genre has bloomed over the last ten years with the same mechanics at its core and this project exists to fulfil and demonstrate said mechanics.

1.1.2 Intended Audience

Users of the application are expected to use this report as a guide to understand the applications functionality. Assessors will use the document to critically evaluate the project as part of the coursework submission. Furthermore, the project is intended as an academic resource for any who wish to use it for educational purposes.

1.1.3 Project Scope

The scope of this project limits it to an application with a single scene to act as a demo environment, with the intention of keeping the project compact and minimising un-necessary features. The scene will contain all the necessary elements to thoroughly present and demonstrate the full range of features available in the system. This approach is not that of a regular game project but more that of a specific system prototype that can be used for demonstration purposes.

The essential elements of the project are the five core features of the system, forming the base building mechanic. Stretch goals for this projected included: building and unit stats, simple AI other than pathfinding and a small variety of different building types, resource types and unit types. Graphics was the stretch goal with the least priority.

1.1.4 Online Demo

An online demonstration of this product was uploaded to YouTube, as per the coursework brief for this submission. Available at the link below:

https://youtu.be/L5wbltwT3qQ

1.2 Overall Description

1.2.1 Product Functions

The product must allow the user to perform a number of functions, these functions are listed thusly:

- The product must let the user control the camera dynamically
- The product must let the user select and interact with the units and buildings in the scene
- The product must let the user assign tasks to individual AI workers units
- The product must perform three worker tasks, move, construct & collect
- The product must perform with a high level of robustness, reliability and efficiency

1.2.2 Product Perspective

The project is a new, self-contained product representing a gameplay component of an RTS style game. The system demonstrates the building and resource management mechanics of this genre of game.

1.2.3 User Guide

For information regarding the setup and gameplay control of the product consult the User Guide [See Appendix A] which is included within this documentation.

1.2.4 Operating Environment

The system shall operate through the Unreal Engine 4 editor on a Windows machine. The minimum hardware specifications for this are as follows, sourced from the Epic Wiki [6.3, 3].

- Desktop PC or Mac
- Windows 7 64-bit or Mac OS X 10.9.2 or later
- Quad-core Intel or AMD processor, 2.5 GHz or faster
- NVIDIA GeForce 470 GTX or AMD Radeon 6870 HD series card or higher
- 8 GB RAM

1.2.5 Assumptions & Dependencies

The system depends on the correct setup of the Unreal Project used by the assessor. The makeup of the demo level and project settings could affect the project if incorrect. The contents of the System Requirements Specification were produced upon the assumption that the project had been setup as instructed in the User Guide [See Appendix A].

1.2.6 Design & Implementation Constraints

Constraints regarding the design and implementation of the product were stipulated in the coursework brief [6.1].

1.3 External Interface Requirements

1.3.1 User Interface

- Project files interfaced via the Unreal Engine
- Custom HUD class acts as interface between player and game via UI elements
- The demo environment via the level editor & inspector
- Editable class attributes via hooks in the BP editor
- System interaction via gameplay using input peripherals

1.3.2 Hardware Interface

- The application shall be developed for the Windows PC platform
- Input requires keyboard and mouse

1.3.3 Software Interface

- The application was built in Unreal Engine 4 v_4.22
- Blueprints were created in the Unreal Engine Blueprint Editor
- C++ Classes were implemented using the UE4 C++ class constructer
- IDE used for programming was Microsoft Visual Studio 2017

1.4 System Features

1.4.1 Demo Environment

1.4.1.1 Description & Priority

The scene that contains the landscape, nav-mesh and custom classes that comprise the demo environment.

<Priority Low>

1.4.1.2 Stimulus/Response Sequences

The demo environment level is imported into the Unreal Engine editor and is loaded with the correct project settings.

1.4.1.3 Functional Requirements

(Req.1) Level Design:

R 1.1 Terrain

The scene shall consist of a landscape terrain.

**UE4 generated landscape to be used with nav-mesh*

R 1.2 Object Layout

The terrain shall not have obstacles and will be a flat plane.

**The focus is not on movement mechanics, so no complex terrain is required*

R 1.3 Resources

Resource patches should be placed around the map.

**Resources placed for core function demonstration purposes*

(Req.2) User Interface:

R 2.1 Build Menu

The system shall include a menu system for selecting a building to construct, which is visible only when a worker is selected.

**Provides visual indication of selection and provides interaction opportunity for core function*

R 2.2 Train Menu

The system shall include a menu system for training new workers, which is visible only when a unit building is selected.

**Provides visual indication of selection and provides interaction opportunity for core function*

R 2.3 Resource Bar

Resource shall be visible on a bar at the top of the screen.

**Provides visual indication of collection of resources to demonstrate resource management core function*

R 2.4 Selection Window

The system may display the currently selected units/buildings.

**Visual indication to increase polish*

1.4.2 Camera Movement

1.4.2.1 Description & Priority

Birds-eye view camera with full dynamic movement mechanic typical of a real time strategy game.

<Priority Medium>

1.4.2.2 Stimulus/Response Sequences

Inputs from the mouse and the position of the mouse cursor on the screen determine the pan, pitch and yaw of the camera.

1.4.2.3 Functional Requirements

(Req.3) WASD and Edge Scroll:

R 3.1 Moving Type One

The camera shall be transformed left, right, up and down using the keyboard or by moving the cursor to the relevant screen edge.

**In-depth camera controls are a core function*

(Req.4) Pan and Rotate:

R 4.1 Moving Type Two

The camera shall be tilted up and down and rotated using the mouse.

**In-depth camera controls are a core function*

(Req.5) Zoom:

R 5.1 Moving Type Three

The camera shall be zoomed in and out using the mouse wheel.

**In-depth camera controls are a core function*

(Req.6) Editable Settings:

R 6.1 Blueprint Edit Hook

The camera settings should be alterable via a blueprint.

**This is useful for artists or designers to balance the game*

1.4.3 Mouse Selection

1.4.3.1 Description & Priority

Most of the core functions of the mechanic rely on being able to click and select actors to interact with the game. The system feature includes the ability to drag a selection box and select multiple units or buildings.

<Priority Very High>

1.4.3.2 Stimulus/Response Sequences

Initiates when the user holds down left click and completes selection process when the user releases the left mouse button.

1.4.3.3 Functional Requirements

(Req.7) Select:

R 7.1 Click or Drag Select

The system shall set an actor or actors as currently selected when the actors are in the zone of selection upon execution.

**Necessary for access to unit and building menu functions and control of workers*

R 7.2 Hover Over Actor

The system may change the cursor when hovering over actors of different types.

**Provides a more polished visual indication of potential selection*

1.4.4 Worker Units

1.4.3.1 Description & Priority

Custom actors with the ability to follow multiple commands and queue multiple tasks; movement, construction & collection.

<Priority High>

1.4.3.2 Stimulus/Response Sequences

On each tick a worker seeks to complete the tasks it has been assigned in order and can only receive new tasks when it is selected by the player.

1.4.3.3 Functional Requirements

(Req.8) Move to Position:

R 8.1 Move to Location

Units shall move to the position of the mouse when a move action is given.

**Units are guided by the player's mouse cursor as is the norm in RTS games*

R 8.2 Selected Check

The system shall allow workers to move only when they are selected.

**To prevent incorrect allocation of tasks*

R 8.3 Formation Move

The system shall space the units apart when multiple units are selected.

**To reduce traffic jams when multiple workers move to a position*

(Req.9) Build Structure:

R 9.1 Move to Location

A selected worker shall move to construct when a building is placed with left click

**An instant reaction to the player's command is intended*

R 9.2 Continue Construction

Workers not building should be sent to construct an unfinished building by right clicking on it.

**To allow an interrupted construction to be completed*

(Req.10) Harvest Resource:

R 10.1 Move to Location and Harvest

Selected units shall move to harvest from the right clicked resource.

**Workers follow the players commands*

R 10.2 Deposit Resources

Units shall bring the resource back to the nearest depot when they have reached the max carry.

**A limit on how much resource can be carried is important for balance*

R 10.3 Repeat Task

Once the unit deposits resources it shall automatically repeat the task.

**Automation is a key feature of this system in an RTS game*

R 10.4 Display Carried Resource

The system may display a different mesh being carried by the unit for each different resource type.

**Visual indication to increase polish*

1.4.5 Resources

1.4.4.1 Description & Priority

A core function of the product is the resources which contribute to the resource management aspect of the mechanic. The product includes multiple types of resource with different specifications.

<Priority High>

1.4.4.2 Stimulus/Response Sequences

The resource deposits in the demo level can be interacted with by a single selected worker or a group of selected workers.

1.4.4.3 Functional Requirements

(Req.11) Resource Depletion:

R 11.1 Decrease Resource

As the resource is harvested it shall decrease in resource remaining.

**Resource management involves the spending of limited resources*

R 11.2 Destroy Resource

If the resource remaining becomes zero, the resource shall be deleted.

**Once the resource is deleted, it allows workers to move to another resource of the same type*

1.4.6 Construction

1.4.5.1 Description & Priority

The construction system includes the different types of building that are available to construct in the scene and forms the basis of the project game mechanic.

<Priority Very High>

1.4.5.2 Stimulus/Response Sequences

Interacting with the menu buttons that are displayed when a worker is selected allows the user to place buildings to be built.

1.4.5.3 Functional Requirements

(Req.12) Placing a Building:

R 12.1 Choose Building to Place

Selecting a building from the menu shall spawn a unfinished building that will follow the mouse cursor.

**The player is to choose where the building is placed*

R 12.2 Place Building

When placing a building left click shall finalize its position in the world

**A left click would indicate the player has chosen a build position*

R 12.3 Collision Check

Building should not be able to be placed on top of other buildings, units or resources.

**Clipping actors together can cause issues with collisions and worker AI*

(Req.13) Build Cost:

R 13.1 Time to Build

Buildings shall take 10 secs to build.

**Build times are used to balance the mechanic and adds realism*

R 13.2 Cost to Build

Buildings should cost resources to place

** The cost factors into the resource management balance - costs ranging from tens to low hundreds*

(Req.14) Multiple Builders:

R 14.1 Faster Construction

Assigning multiple builders to a construction shall increase the speed at which it is built by the number of workers.

**Increases the complexity of the building mechanic*

1.5 Other Non-functional Requirements

(Req.15) Performance:

R 15.1 Frame Rate

The product should be able to function between 30-60 FPS with multiple workers running multiple tasks.

**Stable performance contributes to a good user experience*

(Req.16) Software Quality Attributes:

R 16.1 Implementation

The software implementation should be robust and reliable, and not prone to errors/crashes.

**Crashes are game breaking and deprecate the user experience*

R 16.2 Gameplay

The gameplay should be intuitive and the AI responsive.

**This allows for a balance of ability between users who have and have not experienced the RTS genre*

R 16.3 Code Base

The code base should be readable and therefore maintainable.

**A good code base assists with further development and maintenance*

UML Diagrams

2.1 Use-Case Diagram

The Use-Case Diagram [See Appendix B, Figure 1] for the system was used to visualise the system from the user's point of view.

2.2 Class Diagram

The Class Diagram [See Appendix B, Figure 2] was created to model the static structure of the system and its classes, attributes operations and relationships.

Method

3.1 Selection Control

3.1.1 Summary of Techniques

- Screen space to world space conversion
- Array data structures to hold references to found units

3.1.2 Detailed Description

The primary method of control for the player was through mouse selection input. As specified in the system features functional requirements it was necessary to be able to drag a selection box over units or buildings. This was achieved by using UE4 input management to bind the mouse controls to functions that fire when the left mouse button is pressed and released. On pressed, the system finds the screen position of the mouse cursor and draws a rectangle between the start position and the current position each frame. On release of the button, the system uses the Unreal function "GetActorsInSelectionRectangle" to convert to world space and find highlighted actors. Found actors are stored in an array that is accessible through a Getter.

3.2 Worker Control

3.2.1 Summary of Techniques

- Array structure of tasks to be completed
- Finite state machine responsible for the completion of tasks
- Tasks dynamically added to the stack by player
- Inventory struct for carried resource
- Finding the nearest resource building to act as depot
- Finding the nearest resource of the type that was last gathered
- Custom UI elements

3.2.2 Detailed Description

The worker class was derived from the ACharacter parent class. The class uses a capsule component for collision, a skeletal mesh with animation and a select icon. The main system of the worker is the finite state machine that begins, runs and removes tasks from the "taskStack" array data structure based on the status of the current task. Tasks are added to this array in the "RunTask" and "AddTask" functions. The RunTask function differs from the AddTask as it clears the taskStack before adding another task to begin the new task immediately.

The system features an inventory struct to allow resources to be stored in a specific area of the class along with the settings for constraints. Access to the data is through a getter and a setter for the struct.

The worker class has functions to locate the nearest resource building and nearest resource and store the location and an object reference respectively, which can be accessed via Getters. This was implemented by running a search through the scene to find objects of the desired type and then looping through each to find the closest option.

When the workers booleon variable “isSelected” is set to true the select icon is set to visible to provide a visual indication of the selection. When a worker is selected the HUD is instructed to display the unit menu which is where the buildings can be selected and placed. Custom UI element structs are stored in two arrays, one for units and one for buildings.

3.3 Tasks

3.3.1 Summary of Techniques

- Virtual abstract base class task with subclasses for different tasks
- Subclass functions override parent
- Each subclass has a unique function

3.3.2 Detailed Description

There are three types of task which derive from the parent task class. The three tasks receive the necessary data to perform the task when constructed. The three types are moveTask, constructTask, collectTask and each has unique polymorphic functions which direct the worker.

The move task has functionality to allow the worker to interact with the nav-mesh in the level using The UAIBlueprintHelperLibrary function “SimpleMoveToLocation”. When the system detects that the worker is close to the move point it sets the task status to complete.

The construct task combines movement functionality with functionality to increase the percentage of construction when the worker is nearby the building placement location. If the building has been constructed more or equal to the threshold variable fully the task is set to complete.

The collection task combines aspects of the movement & construction tasks. The main system feature is the automatic repetition of the task, where the worker is able to transition between collecting and depositing resources. The collect cycle works in a similar way to construction, but instead of constructing the building the worker removes resource from the target and adds resource to its inventory. When depositing, the worker moves to the nearest depot and clears its inventory variables values. The cycling of the task takes place if the task queue has only the individual collect task, otherwise it sets the task status to complete.

3.4 Buildings

3.4.1 Summary of Techniques

- Virtual base class with subclasses for different buildings
- Subclass functions override parent
- Each subclass has specific characteristics
- Custom UI elements

3.4.2 Detailed Description

Buildings of type Unit and type Resource are subclasses of the Construction. This polymorphic method allows new building types to be easily added with the simple creation of a new subclass. Only the unit building has a custom UI system and function. The resource building acts as a depot location for worker resources. The unit building has functionality to spawn new worker actors into the level, nearby the building. When the train unit button is clicked a new train unit order is added to a queue structure. After a few seconds counted by a timer, a new worker will spawn. This allows several units to be queued for production. Furthermore, if multiple unit buildings are selected and the training button is pressed, each building will have a training order added to the queue. This is the type of macro system that is preferred in RTS games.

3.5 Resources

3.5.1 Summary of Techniques

- Resource manager attached to game mode to monitor resources
- Virtual base class with subclasses for different resources
- When collect task is performed resources are subtracted from the class
- Each subclass has specific characteristics

3.5.2 Detailed Description

A separate class was created to act as a resource manager. The manager is instanced in the Custom Game Mode class to allow it to be accessed from any UClass via the “GetWorld” reference. The class is where the values for the player’s current resources are stored and read for the HUD.

Each specific resource derives from the same base class to allow for addition of new resource types as subclasses of parent resource. Each resource keeps track of the yield it has remaining and when the yield reaches zero the resource sets itself to depleted and destroys the object. This sequence of events is what prompts the worker who depleted the resource to change to another resource of the same type and prevents it from being harvested from again.

Development

4.1 Development Process

The development process that was utilized for this project was based on the Agile development process. This was chosen for its flexibility and iterative nature, which would likely prove useful as the project had a moderate to high probability of evolving during its course and the planning, design and documentation would need to adapt with it.

The initial design of the mechanic was developed in UML which acted as guidelines for the main systems of the product. This approach combined with the Agile process was beneficial to the development of the process as it provided a reference to work off of during construction of the entire system, while accounting for potential changes in the project. While the initial layout of the system was not left unchanged by the end of development, the material and development process were useful in preparing for potential project risks.

4.2 Concept Design

The concept that was chosen was the first of a variety of concepts that were synthesised and checked with the project client/assessor. This mechanic was chosen as it had been deemed the most suitable for the brief and offered the right amount substance for assessment and the right amount of technical challenge. The concept was the preferred choice as it was a good balance between a well-known mechanic and a mechanic that would demonstrate systems that were less commonly submitted.

4.3 Prototyping

The first step of construction was blueprint prototyping. The intention was to prototype the entire mechanic while also developing an understanding of the UE4 blueprint editor. For this, a tutorial [6.2.2] was employed, which provided the basics on using this part of the editor. However, after completing a blueprint prototype of the camera control system [See Appendix C] it was apparent that this process would be vastly time consuming. The size of the event graph for simple camera movements, when compared to the small input function of the C++ GG_RTS_Camera pawn, was evidence of this. The blueprint prototype wasn't a necessity of the product and the workflow was too slow to be sustainable, so the production was stopped in favour of C++ development.

4.2 Documentation

The documentation had undergone many alterations throughout the course of the project and several different versions were made. These alterations were mostly focused on the functional requirements sections of the system requirements specification, as a number of the sections, for example, the product perspective and objective were constant throughout. This was mainly due to iterations of the Agile development process through which further levels of detail were extracted from the requirements of the mechanic.

Initially, the functional requirements gave a top-down description of a system feature, in one all-encompassing requirement. This was later broken down into sub levels of requirements, each with their own unique identifier, requirement and rationale which appeared as much more detailed. Furthermore, each requirement was altered to conform to the iEEE formal language specifications 'Shall', 'Should' and 'May' as stated in the recommended practice for SRS documents section Std-830-1998 [6.4, 5].

Conclusions

5.1 Shortcomings

The project set out to achieve a number of goals and features, however, not all of them were completed by the end of the project. These shortcomings were due to a number of reasons ranging from revised requirements that rendered the feature obsolete, to cancellation due to reduced priority and in some cases certain features had just not been implemented due to lack of time. These shortcomings were limited to the gameplay mechanic and none of them resulted in a failure to achieve the outcomes of the project brief.

One of such shortcomings was the number of resource types implemented. The concept brief that was proposed specified three separate types of resource. However, as the project progressed it was decided that two individual resource types would be suitable to demonstrate the system.

A further shortcoming was the lack of a “defensive” building which suggested some kind of system where buildings/units could take damage. The defensive building was scrapped due to time constraints as it would involve not only the construction of the building, which would be simple to implement, but also the implementation of its function of defence. Typically, in an RTS game this would be some kind of enemy type. A result of this was the exclusion of unit and building stats. This also took less priority as the game mechanic concept was base building and resource management.

5.2 Known Issues / Areas for Improvement

The system has a known issue that causes a fatal error only some of the time, which was unable to be rectified in time. As stated, the issue only arises occasionally, crashing the game, and when it does it has little indication to the cause, making it difficult to debug. A less than ideal issue, it can be worked around for demonstration purposes by restarting the application.

The main area for improvement is the overall level of polish that the mechanic displays. There is a degree of polish but there is further room for improvement within the art and UI design. Another area of improvement lied on the technical side of the project and this was mainly because this was my first experience using Unreal Engine and it was as much of a learning experience as it could have been. By the end of the project I had learned enough about the Engine that my overall technical design at the start of the project might have been different, had I possessed the level of knowledge that I do now. However, I believe that the technical design I implemented was suitable enough, upon consideration.

5.3 Possible Solutions

With regards to the above issue, the suspected source is and unhandled exception where a null pointer is being called.

As for the improvements, the art could be overhauled to replace the current assets which are all just different placeholder meshes from the starter content to tell each object apart. The UI would benefit from further development as it hasn't a large amount of detail or features. Features such as; more windows to display increased information, for example, a current selection pane with unit or building details for unit stats and building construction times, etc.

5.4 Extending the Mechanic

A possible extension to the mechanic would likely involve expanding the depth of the construction systems. An ideal expansion would be increasing the number of building types that are available to construct, which would open the mechanic to further functions such as the construction of modular, node based walls, with the function of defence. Another extension could be development of the resource system to account for more types of resource and interactions between workers and resource patches, such as limits to the number of active workers, which would contribute to the balancing factor.

5.5 Personal Outcome

During the course of the project I have developed my knowledge of Unreal Engine and upon reflection I have identified elements of the project that went right for me, parts that went wrong and the parts that proved the most challenging. The biggest challenge I faced in this project was ultimately learning the different facets and nuances of Unreal Engine for the first time ever. We were given a certain degree of direction on how to use this software in the labs, but a lot of the learning was our own responsibility. This ties into the main part of the project that I felt I could have done better. I found that because I spent the majority of my time early on learning the engine I started the construction of the mechanic in C++ later than was advisable. This caused the development to reach a crunch in the run up to submission.

One of the factors I found went right was the overall development of the documentation, specifically the SRS and some other areas of the report. I made sure to treat the report as a 'living' document and did my best to keep the contents updated as the project evolved. At the beginning of the project I focused on the technical design so that the construction of the development process would run smoother, using the documentation and UML as reference. Furthermore, I feel that the progression from concept to completion was a success and the final product featured unique functionality.

References

6.1 Brief

Bett, M. (2019). [online] Blackboard.abertay.ac.uk. Available at: https://blackboard.abertay.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_8571_1&content_id=_524873_1&mode=reset [Accessed 30 Jan. 2019].

6.2 Techniques

6.2.1 C++

Wiki.unrealengine.com. (2015). HUD, Canvas, Code Sample of 800+ Lines, Create Buttons & Draw Materials - Epic Wiki. [online] Available at: https://wiki.unrealengine.com/HUD,_Canvas,_Code_Sample_of_800%2B_Lines,_Create_Buttons_%26_Draw_Materials [Accessed 20 Mar. 2019].

YouTube. (2017). [UE4 C++] RTS Select and Move Units. [online] Available at: https://www.youtube.com/watch?v=TLp4XyX6y4o [Accessed 20 Mar. 2019].

Docs.unrealengine.com. (2004). Programming Guide. [online] Available at: https://docs.unrealengine.com/en-us/Programming [Accessed 11 Mar. 2019].

Api.unrealengine.com. (2004). Unreal Engine Programming and Scripting Reference. [online] Available at: http://api.unrealengine.com/INT/index.html [Accessed 11 Mar. 2019].

6.2.2 Blueprints

YouTube. (2015). Unreal Engine 4: RTS Tutorial Series. [online] Available at: https://www.youtube.com/watch?v=FZK5T-vAVFA&list=PLDnygpcOYwFW2XtNyiandrLDG__OAZs7Q [Accessed 23 Feb. 2019].

6.3 Research

[1] *En.wikipedia.org. (2019). Warcraft: Orcs & Humans (1994). [online] Available at: https://en.wikipedia.org/wiki/Warcraft:_Orcs_%26_Humans [Accessed 19 Feb. 2019].*

[2] *En.wikipedia.org. (2019). Age of Empires (video game) (1997). [online] Available at: https://en.wikipedia.org/wiki/Age_of_Empires_(video_game) [Accessed 19 Feb. 2019].*

[3] *Wiki.unrealengine.com. (2004). Recommended Hardware - Epic Wiki. [online] Available at: https://wiki.unrealengine.com/Recommended_Hardware [Accessed 10 Apr. 2019].*

6.4 Resources

[1] *Docs.unrealengine.com. (2004). Unreal Engine 4 Documentation. [online] Available at: https://docs.unrealengine.com/en-us [Accessed 24 Feb. 2019]. -> Unreal Engine 4 -> Starter Content*

[2] *Visual Studio. (2017). Visual Studio IDE, Code Editor, Azure DevOps, & App Centre - Visual Studio. [online] Available at: https://visualstudio.microsoft.com/ [Accessed 24 Feb. 2019].*

[3] *Cite This For Me. (2019). Save Time and Improve your Marks with CiteThisForMe, The No. 1 Citation Tool. [online] Available at: http://www.citethisforme.com/ [Accessed 12 Feb. 2019].*

[4] *Wiegers, K. (1999). [online] Web.cs.dal.ca. Available at: https://web.cs.dal.ca/~hawkey/3130/srs_template-ieee.doc [Accessed 30 Jan. 2019].*

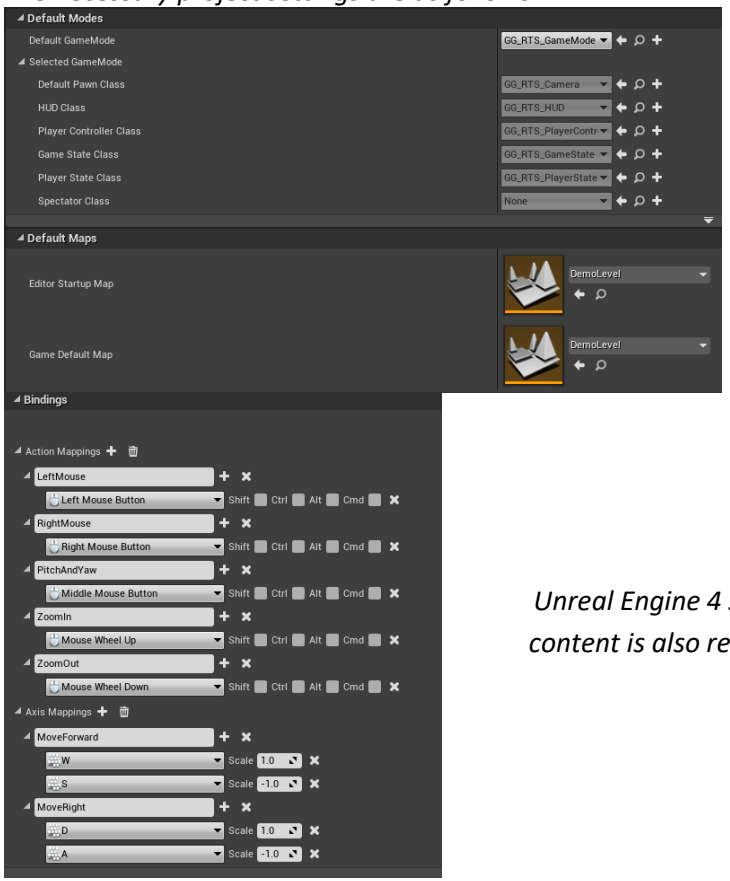
[5] *Cse.msu.edu. (2009). Pg 23, 5.3.2. [online] Available at: http://www.cse.msu.edu/~cse870/IEEEXplore-SRS-template.pdf [Accessed 10 Apr. 2019].*

Appendices

Appendix A: User Guide

1 Setup

The necessary project settings are as follows:



Unreal Engine 4 starter content is also required.

2 Gameplay

Pan the camera with WASD or by scrolling to the edge of the screen. Rotate the camera by holding down the middle mouse button and moving the mouse. Zoom the camera by rolling the mouse wheel.

Click or click and drag to select units or buildings. When selected right click to initiate a task. Hovering the mouse over an unfinished building and right clicking cause the worker to resume construction. Hovering over a resource patch and right clicking will cause the workers to begin harvesting that patch. When the mouse isn't over any objects, right clicking will move the units to the location of the cursor.

Assign multiple tasks by holding shift when giving a task. This works with movement, construction and resource collection.

When placing a building press Q to cancel the placement. You can only construct a building if you have enough resources.

Interact with the menu buttons by putting the cursor in the bounds and left clicking. Clicking the train worker button when the Unit Building is selected will add a new production order to the buildings queue. The new units will be produced after the set time.

Appendix B: UML Diagrams

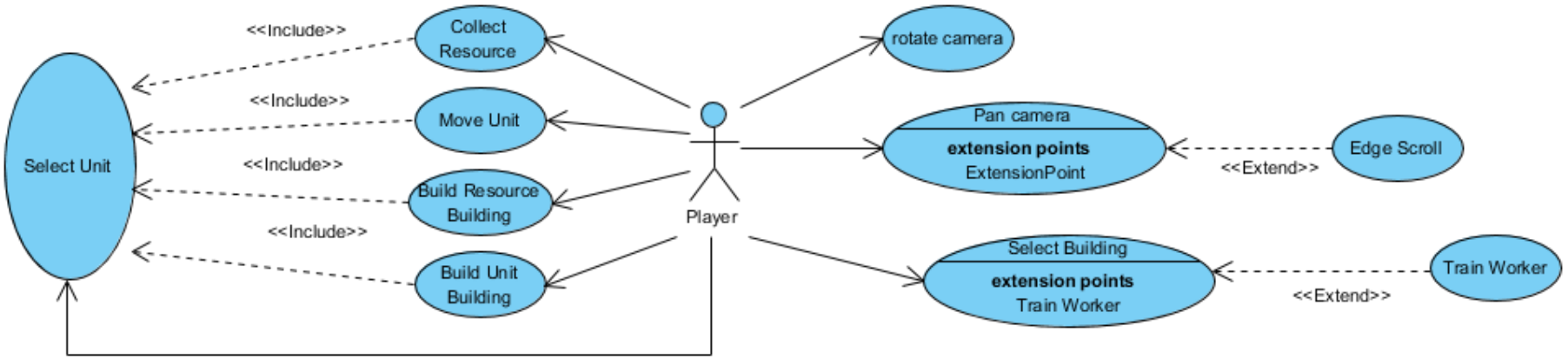


Figure B.1, Use-Case Diagram V2

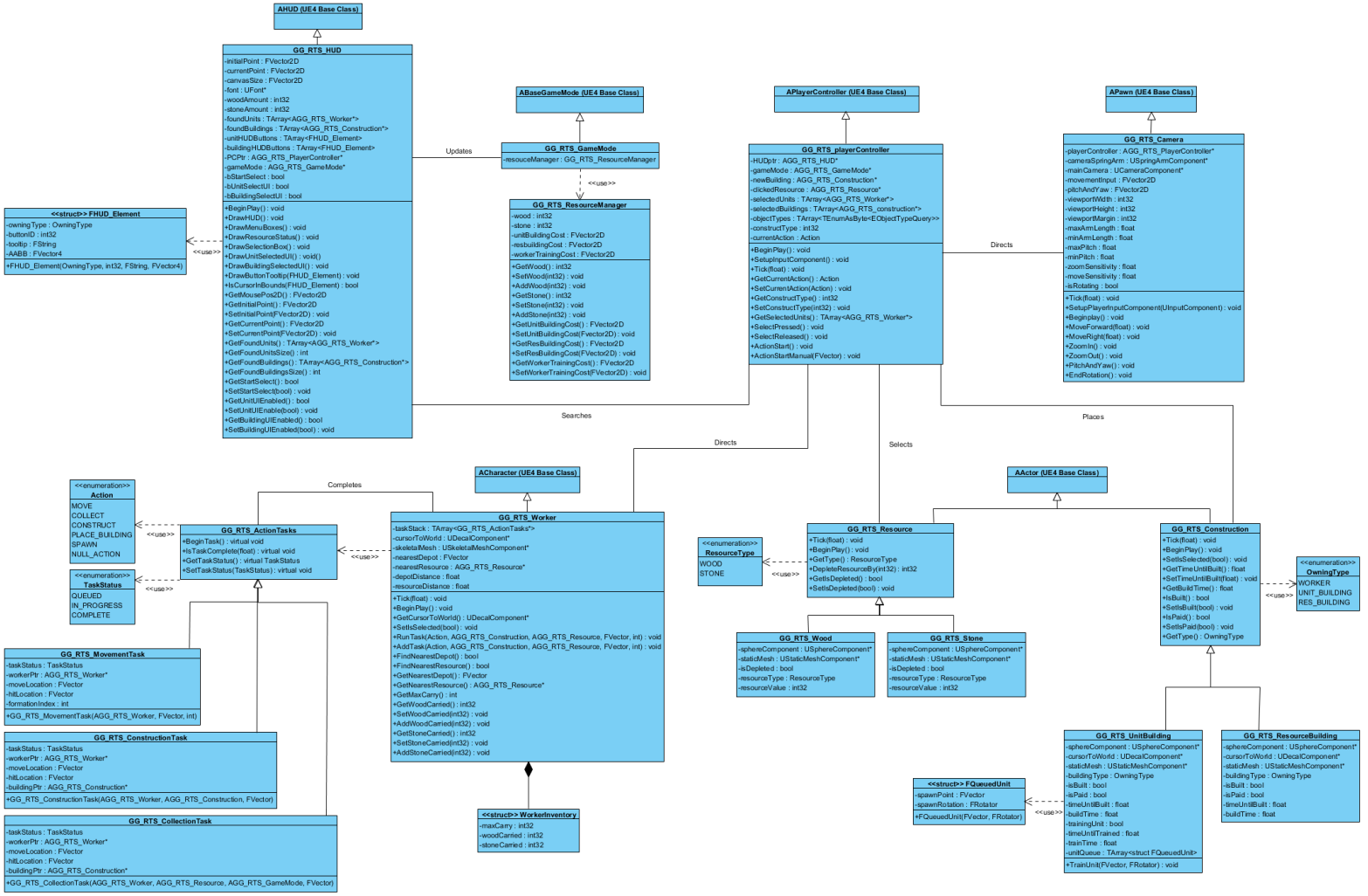


Figure B.2, Class Diagram V2

Appendix C: Blueprint Prototype

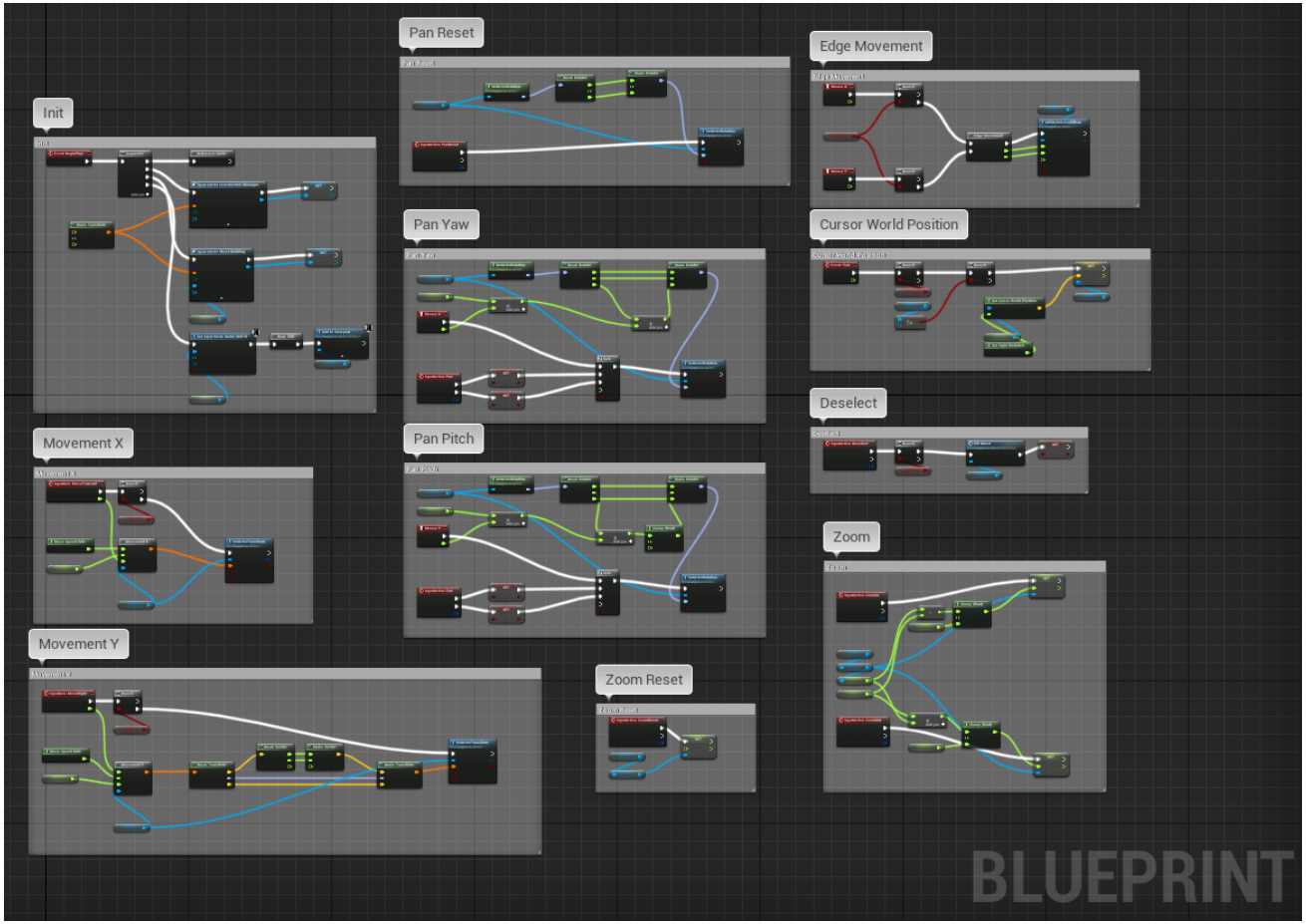


Figure C.1, Camera Blueprint Prototype Scale