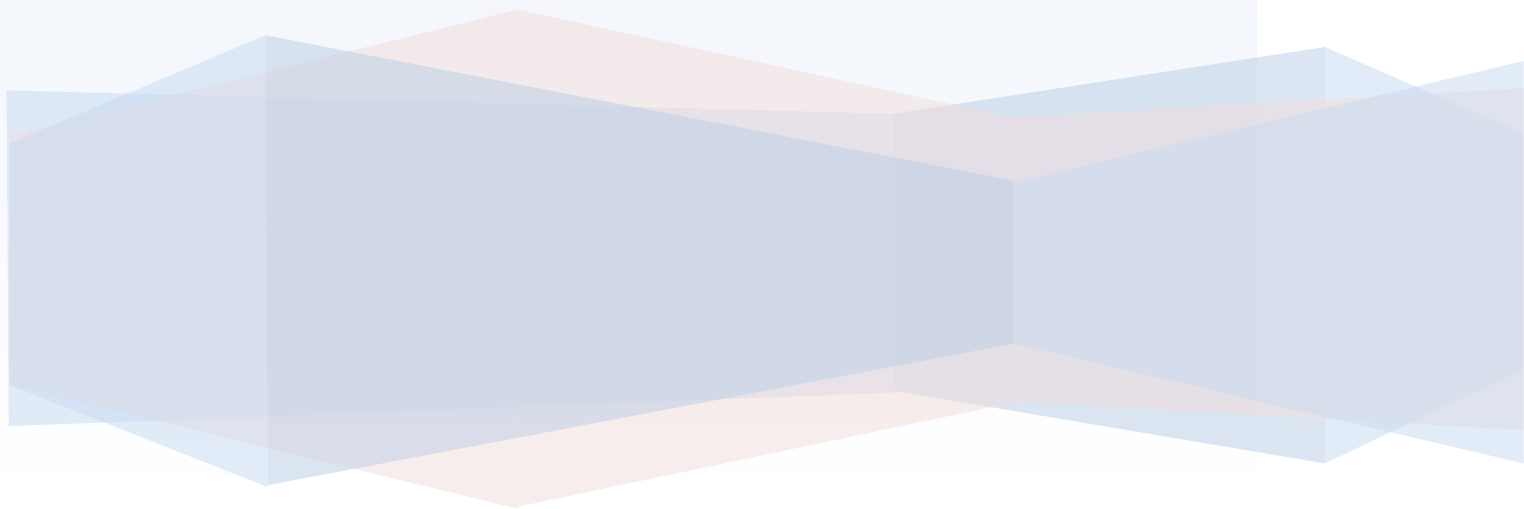


COS30031 Games Programming

Learning Summary Report

Peter Argent (7649991)



Introduction

This report summarises what I learnt in COS30031 Games Programming. It includes a self-assessment against the criteria described in the unit outline, a justification of the pieces included, details of the coverage of the unit learning outcomes, and a reflection on my learning.

All Spikes can be found at <https://github.com/stormcroe/GamesProgramming2016/>

Self-Assessment Details

The following checklists provide an overview of my self-assessment for this unit.

	Adequate (Pass)	Good (Credit)	Outstanding (Distinction)	Exemplary (High Distinction)
Self-Assessment (please tick)		✓		

Self-assessment Statement

	Included? (tick)
Learning Summary Report	✓
Lab Test 1 and 2 + feedback + responses	✓
Complete "core" spike work	✓

Minimum Pass Checklist

	Included? (tick)
Additional non-core spike work (or equivalent)	✓
One or more <i>non-code</i> pieces	✓

Minimum Credit Checklist, in addition to Pass Checklist

	Included? (tick)
Code for non-trivial program(s) of own design	
Document for the program(s) included (structure chart etc)	

Minimum Distinction Checklist, in addition to Credit Checklist

	Included? (tick)
Research report, and associated pieces	

Minimum High Distinction Checklist, in addition to Distinction Checklist

Overview of Pieces Included

This section outlines the pieces that I have included in my portfolio...

The Pieces Contained in this portfolio can be accessed at:

<https://github.com/stormcroe/GamesProgramming2016>

The Pieces are:

Spike 1: Simple Game Loop. (With Extension): This is a simple Object Orientated based Game loop.

Spike 2: Intro to Visual Studio: This is a short introduction to using Visual Studio. It shows the basics of the IDE

Spike 3: Debugger Use: This was to show off how we can use Visual Studio's inbuilt tools to debug and test our programs when something goes wrong during runtime.

Spike 4: Non-Blocking Game Loops: This was to show off how we can create non-blocking game loops. I used threads which was a credit extension

Spike 5: Game State Management: Shows off the game state pattern.

Spike 6: Basic Data Structures: The report shows my knowledge of Data structures in C++. The code implements one of these as an inventory in Zorcish

Spike 7: Graphs: This is spike based about the abstraction of graphs as a data structure. They were used to create the world for Zorcish.

Spike 8: Command pattern: This spike shows the command pattern in action, Implemented to allow the player to move and look at things in the Zorcish world

Spike 9: Composite Pattern: This pattern is used in conjunction with the command pattern to change values of in game objects or move where those objects are, implemented as PickUp, PutDown, and sharpen commands

Spike 10: Component Pattern: This is where we use a component to set up the item for the sharpen command

Spike 11: Spike 12: These two spikes show my understanding of Messaging systems.

Spike 13: Unit Tests: Another way of testing things in our programs, this way is more for testing logic.

Spike 14: Unreal Engine Familiarisation: This is showing the basics of the unreal blueprint system. Is also extended to show the differences between Unreal and Unity game engines.

Spike 15: Simple Scene: This spike looks at a very simple scene in unreal engine 4

Spike 16: Create a blueprint in c++: This shows how I exposed some C++ code to the blueprint system in unreal

Spike 17: Input Handling: This shows my knowledge of how to access various input functionality in Unreal

Spike 18: Sounds: This shows my knowledge of how to use sounds in Unreal

Spike 19: Collision: This is about using the blueprint system to change actors upon collision, it also shows some of the blueprint systems control structures

Test 1: This shows my knowledge on Data Structures and Game Loops

Test 2: This one shows my knowledge on client structures and My knowledge on the difference between engines, frameworks and libraries.

Coverage of the Intended Learning Outcomes

This section outlines how the pieces I have included demonstrate the depth of my understanding in relation to each of the unit's intended learning outcomes.

ILO 1: Design

Discuss game engine components including architectures of components, selection of components for a particular game specification, the role and purpose of specific game engine components, and the relationship of components with underlying technologies.

The Following Spikes show my proficiency with the Design Learning outcome:

Spike 1, Spikes 4 – 12, Spike 16.

These spikes all show how I can use and create code to specific designs.

1 and 5 are about OOP, 1 and 4 are about game loops, blocking and non-blocking respectively.

5 – 12 are about following design briefs in parts, in addition to knowing what patterns work for which parts of a game.

Also showing good to deep knowledge of this ILO are the two tests.

ILO 2: Implementation

Create games that utilise and demonstrate game engine component functionality, including the implementation of components that encapsulate specific low-level APIs.

The Following Spikes show my proficiency with the Implementation Learning outcome:

Spike 1, Spikes 4 – 12, Spikes 15 - 19.

Spikes 15 – 19 show how Unreal's Blueprint system can be used to implement actions within a game.

Including: Playing sound, collision testing, printing to a console for logs etc.

1, 4 and 5 – 12 show how different methods are required to be combined to create a full game. It also shows how those methods can differ.

ILO 3: Performance

Explain and illustrate the role of data structures and patterns in game programming, and rationalise the selection of these for the development of a specified game scenario.

In terms of Performance, these spikes show an understanding of this concept.

Spike 6: In this I talk about how different data structures have an impact on the performance of the program, it is talked about in terms of complexity $O(1)$ vs $O(n)$ for example.

Spike 19: This spike uses Unreal's collision engine to try and quickly calculate performance, the credit task if completed would have had a difference in performance than the unreal collision engine.

ILO 4: Maintenance

Explain and illustrate the role of data structures and patterns in game programming, and rationalise the selection of these for the development of a specified game scenario.

All Spikes have some relation to the maintenance ILO due to how the spikes are structured to be built upon one another. But the spike with the most relation to this ILO are Spikes 1, 3, 13, 14 and 7.

3 and 13 are about testing and debugging, ensuring that the program works as intended.

14 is about a familiarity with the Unreal Engine and why engines are used.

1 and 7 are about modularity and expandability, 1 through OOP and 7 through the Graph structure.

Reflection

The most important things I learnt:

Was about the applications of software design patterns to video game design and programming.

The things that helped me most were:

My classmate's ability to help explain some concepts, especially the state pattern.

I found the following topics particularly challenging:

The State Pattern was the hardest of all concepts to grasp, but once one of my friends explained it to me and walked through his code, it made more sense.

I found the following topics particularly interesting:

The Patterns were interesting to work with, but the amount of already working code to change to implement each successive pattern became annoying over time. Especially when the code to be changed messes up with certain patterns.

I feel I learnt these topics, concepts, and/or tools really well:

Command, Composite and State Patterns: Once Given a modicum of guidance from friends and the internet I was able to help explain these concepts to others in my tutorial to help them as others helped me.

Game Loops: A very simple concept that I feel I learnt well enough to be able to use later.

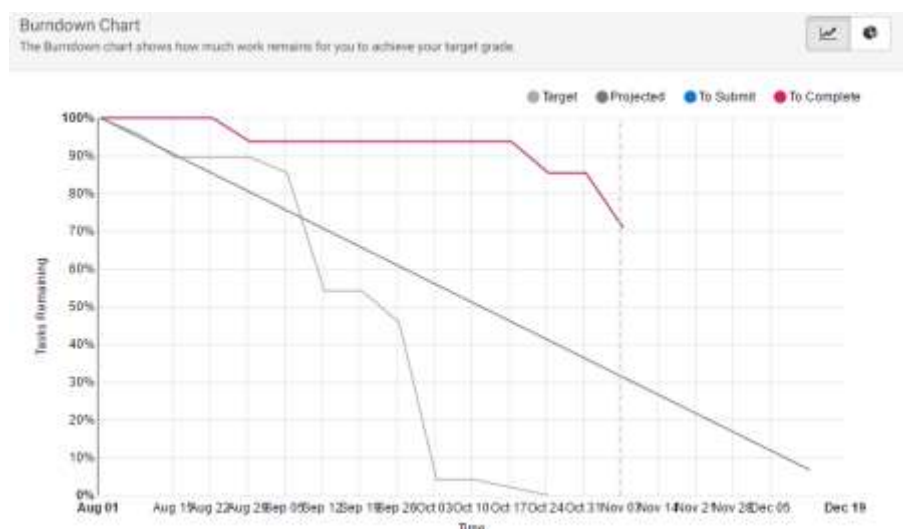
Visual Studio 2013. This is my current IDE of choice for console applications and windows form applications. I like to use this when creating applications in other units that don't require specific IDEs for complicated programming. In addition I learnt some debugging and comfort tools for the 2015 edition.

I still need to work on the following areas:

Unreal Engine 4: I find UE4 very unwieldy to use, which is due to the low res on my laptop and the inability to unclutter the screen when working with it. I am still much more used to Unity.

My progress in this unit was ...:

Poor if you look at the doubtfire graph, but that is due to the fact that it doesn't show when you submitted things vs when you get them marked off, and because I never had anything marked off by my own tutor in the first few weeks, I decided to do all the spike reports at the end and the actual work throughout the semester. Thus I left the reports too late to do to get all marked off, but I did complete the work I needed for my grade.



Overall progress was good but it cannot be shown through doubtfire due to incompetence of the tool and the lack of feedback from my tutor during the semester.

This unit will help me in the future:

The Unit will help in the future as it showed me some interesting patterns to use for when I create my own games. It also helped cement some knowledge of Visual Studio, and it gave me an insight into how Unreal Engine works as an IDE.

If I did this unit again I would do the following things differently:

I would force my tutors to look at my work earlier as well as doing the spike reports more regularly instead of doing them all at the end.

Conclusion

In summary, I believe that I have clearly demonstrate that my portfolio is sufficient to be awarded a low credit grade.

I completed a fair number of credit tasks, and completed the pass tasks to a satisfactory level.

Spike: Spike_1

Title: Simple_Game_Loop

Author: Peter_Argent, 7649991

Goals / deliverables:

In this Spike, We created an object-orientated base version of source code for GridWorld. This includes: GameController.cpp, Grid.cpp, GridSquare.cpp, Player.cpp and Source.cpp

Also Included are some screenshots of my initial pen and paper planning, and the IDE.

The source Code can be found here:

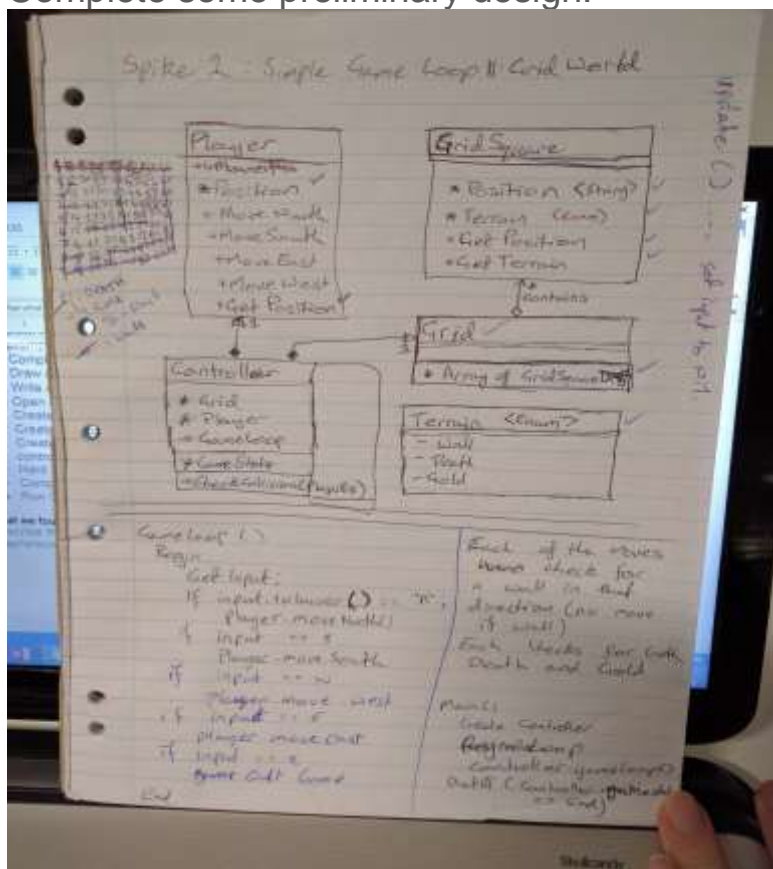
<https://github.com/stormcroce/GamesProgSpike1.git>

Technologies, Tools, and Resources used:

- Visual Studio 2013
- CPlusPlus.com for the c++ STL syntax and reference.

Tasks undertaken:

- Complete some preliminary design.



- Draw a simple UML Class diagram (Shown above)
- Write a basic pseudo-code game loop. (Shown above)

- Open a new console project in Visual Studio -> Make it an Empty project.
- Create each class using the IDE from the UML diagram
- Create all properties and methods in the UML Diagram, making sure they are in the right classes.
- Create Input(), Update() and Render() methods for the game controller classes' GameLoop().
- Hard Code the map in the Grid Class.
- Ensure that
- Build the GridWorld code and you now have a game.

What we found out:

Describe the outcomes, and how they relate to the spike topic + graphs/screenshots/outputs as needed

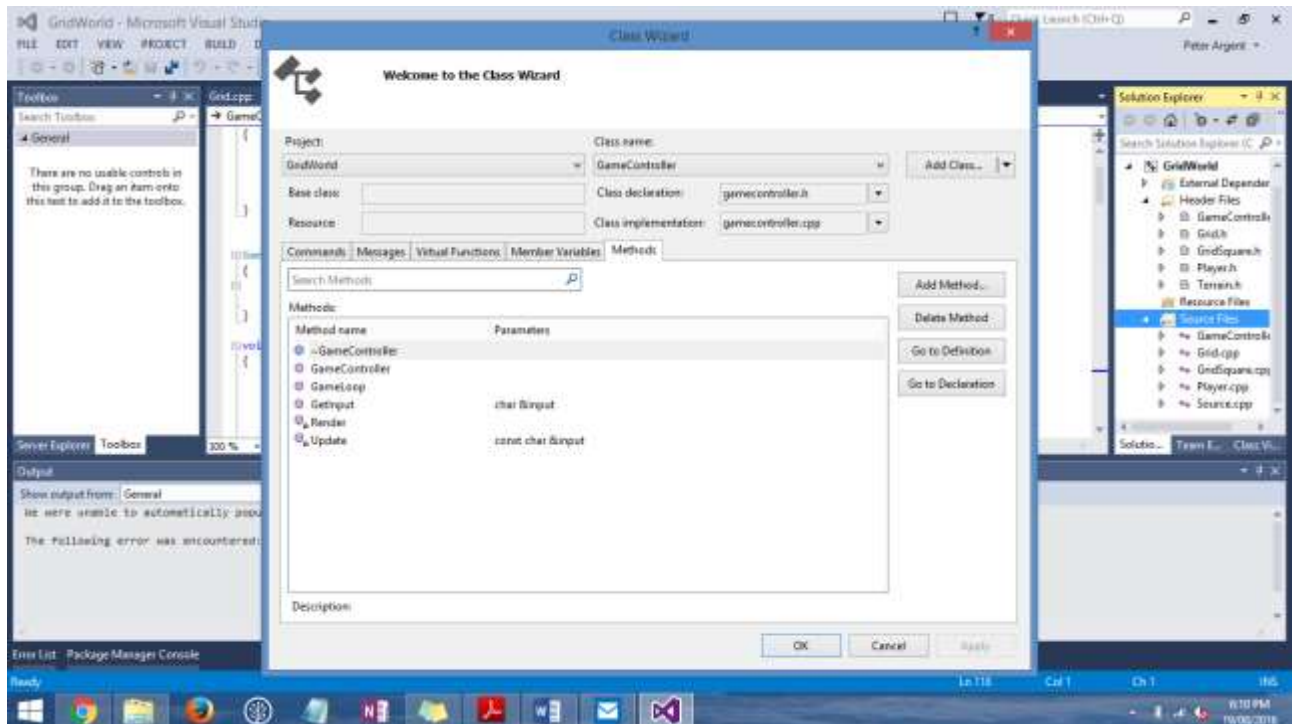
From this spike, we learn how to create a new c++ project in Visual Studio and learn the basic structure for a game loop. We also produced a paper design to show how we understand the game loop and the class structure for a simple game.

Specific things we found out were:

- The Input, Update, and Render functions separate each section of the game loop for ease of readability and modifiability.
- We learn how to use a single Dimensional Array to simulate a 2D grid.

We hardcoded East to be +9 and West to be -9. There are better ways to simulate the 2D grid, but for the scope of this project this was the easiest way.

- We learned how to use the Visual Studio's Class Wizard (ctrl-shift-x)



- We used GetX() and SetX() to allow other classes to access one classes' private fields (Where X is the field name)
- We used an enum (terrain.h) to create a definition for the GridSquare class to easily define what type of terrain it is.

Spike: Spike_2

Title: Visual Studio

Author: Peter Argent, 7649991

Goals / deliverables:

The Goal of this spike is to familiarise the user with Visual Studio

Deliverables Include:

- Short report entitled "IDE Report"

Technologies, Tools, and Resources used:

- Visual Studio 2013
- Microsoft Word

Tasks undertaken:

- Create a new project in Visual Studio and screenshotting the process for the short report
- Create a breakpoint in a Visual Studio project and detail how to do so and how to use it
- Note down in the short report where variables can be monitored during this time.

What we found out:

How to use Visual Studio's basic IDE functionality.

Spike: Spike_3

Title: Debugger Use

Author: Peter Argent, 7649991

Goals / deliverables:

- There is a fixed copy of Spike3.cpp at <https://github.com/stormcroe/GamesProgramming2016/tree/master/Spike3>

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Visual Studio 2013
- Cplusplus.com is handy for syntax, and general help.

Tasks undertaken:

- Compile the spike3.cpp that we were given in Visual Studio.
- When the program brakes, choose to create a breakpoint in the code where the program stopped working.
- Check that all variables have been initialized using the “Autos” and “Locals” Tabs in the bottom left of the IDE
- If you cannot find bad variables try using the “Call Stack” window to step back from the breakpoint and find if the variable you were dealing with had bad values from elsewhere.
- To find the memory leak, make sure that each resource has had delete called on it, check the main() function especially, and then each class separately.

What we found out:

Describe the outcomes, and how they relate to the spike topic + graphs/screenshots/outputs as needed

- Logic Errors can only be found by noticing where a program does not do what you expect it to and setting breakpoints there so you can step back and forth in code to find the error.
- Syntactical errors are often picked up in Visual Studio using the inbuilt ‘Intellisense’ that underlines wrong or missing syntax, if it would not compile, the intellisense picks it up. If an error would break a program in runtime, the intellisense will not pick it up
- By using a breakpoint and the Locals, Autos and Call Stack tabs, we can step through code and read what variables are doing at certain points in the program. This allows us to see whether a variable is acting in the correct way.

Recommendations [Optional – **remove** heading/section if not used!]:

I recommend having multiple programmers looking at the same code base when debugging, as having multiple viewpoints and people to explain code to helps immensely during testing/debugging.

Spike: Spike_4

Title: Non-blocking Game loops

Author: Peter Argent, 7649991

Goals / deliverables:

- This Spike was centred on using threads to make a non-blocking game loop.
- The source code for the Spike can be found at https://github.com/stormcroe/GamesProgramming2016/tree/master/Spike_04
-

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Visual Studio 2013
- <http://www.cplusplus.com/reference/thread/thread/?kw=thread>
- Cplusplus.com as a reference site for more C++ syntax
- A copy of spike one's source code to work off.

Tasks undertaken:

- Open a Copy of spike 1's GridWorld in Visual Studio 2013
- Open the gameController.cpp file
- Move the "input" variable to be a field and have it initialize to NULL
- Change the parameters on GetInput() to GetInput(char* input, bool *gameover)

The input parameter is a pointer to the _input field, this allows us to pass the input information between threads, the gameover parameter allows us to check for the game finishing so that the thread's internal loop ends.

- Add an If statement to each threads' while loop so that the code within activates only on: (input == NULL) for the input loop, and (_input != NULL) for the game loop.
- Write the code that would create and join the input thread to the main thread.

```
thread GetUserInputThread(GetInput, &_amp;_input, &_amp;gameover);
```

```
//... The game loop code is here.
```

```
GetUserInputThread.join();
```

- Note: Make sure the _Input is reset to NULL after the game loop, otherwise the input thread won't trigger again.

What we found out:

Describe the outcomes, and how they relate to the spike topic + graphs/screenshots/outputs as needed

- One thing that we found out during this spike is that passing by reference in C++ cannot be done if used in a method being passed to a new thread. However using pointers works perfectly. [Search cplusplus.com for how to use pointers]
- We need to ensure that the game loop code did not run if there was no input and the input not run while there was an input otherwise the console would quickly fill and move at a pace that a human could not read. Using Non-blocking game loops is not a reasonable solution for console games running on a basic game loop; only if there is a time sensitive flag should they be used eg. A physics engine or day-night cycle

Open issues/risks [Optional – **remove** heading/section if not used!]:

- Risk/Issue 1: Using pass by reference in the GetInput() method breaks down when attempting to send data back across threads.

Recommendations:

I recommend that people have a requisite understanding of how pointers are used in C++ before attempting this spike, without understanding how they are used, someone cannot get the code to work properly and send the input data across threads to the game loop.

Spike: Spike_05

Title: Game_State_Management

Author: Peter Argent, 7649991

Goals / deliverables:

The goal of this spike is to implement the game states of Zorcish stage 1 design using Object orientated principles.

Deliverables Include:

- Paper Design in Appendix
- Code at: <https://github.com/stormcroce/GamesProgramming2016>

Technologies, Tools, and Resources used:

- Visual Studio 2013
- CppReference.com
- Classmates
- MS One Note for Android for the Design
- MS Word

Tasks undertaken:

- Plan the Zorcish Project on Paper using the Design given and consulting others.
- Create a new Project for Zorcish
- Create a stateManager Class
- Create a State super class
- Create a Class for each State required from the Zorcish stage 1 design
- Create a main function that uses the state manager to run the program
- Test each state

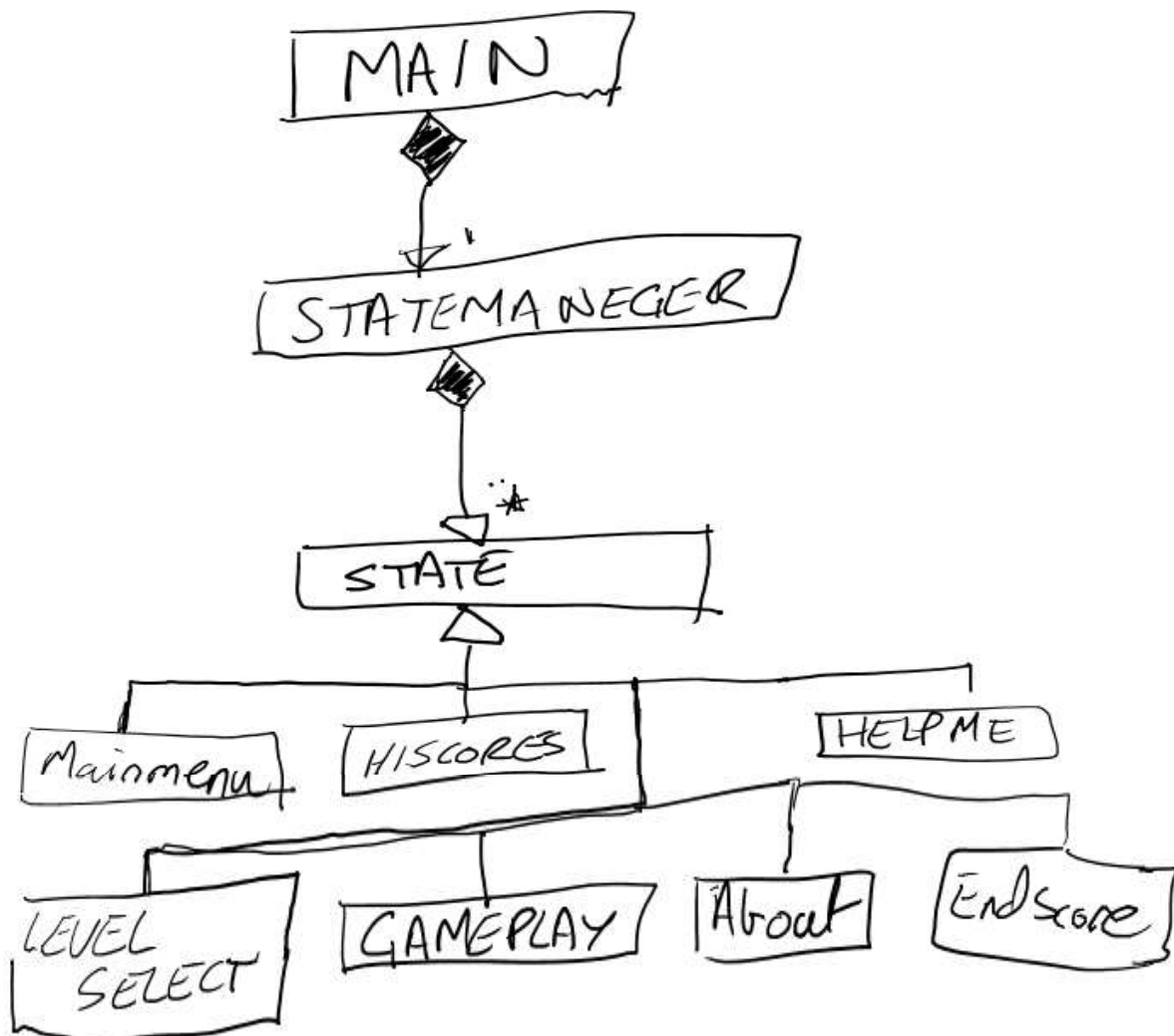
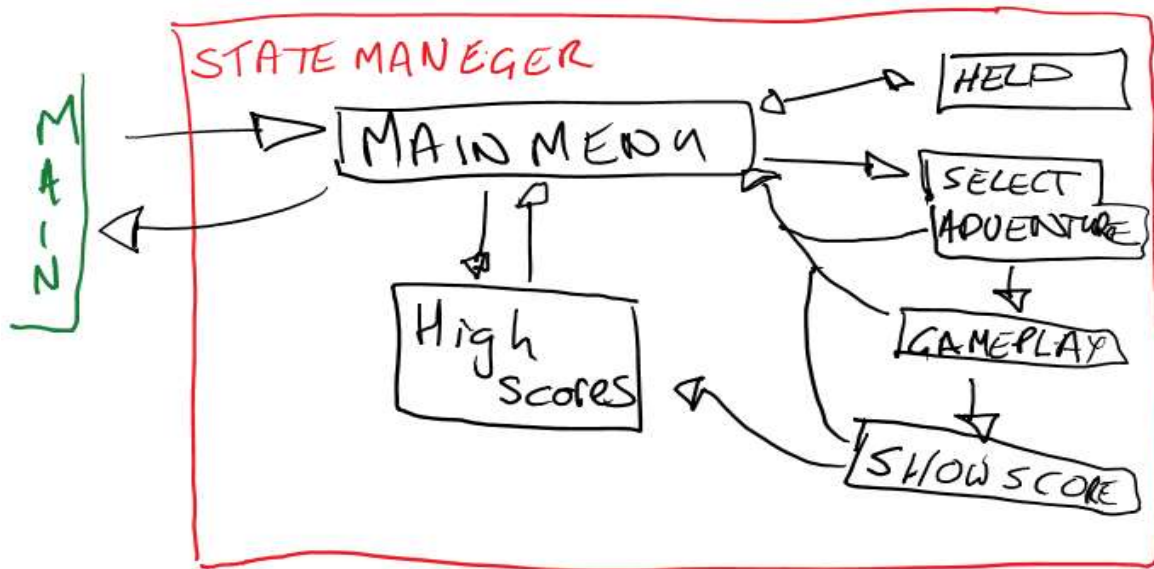
What we found out:

- Learnt how to implement gameStates and the State Pattern in C++

Open issues/risks [Optional – **remove** heading/section if not used!]:

- **Issues in this current iteration of the code:**
- The state manager does not have a robust system for checking if the code entered is a correct input
- There is no way to exit the application using the console itself, Must use the kill switch

Appendix:



Spike: Spike_6

Title: Data Structures

Author: Peter Argent, 7649991

Goals / deliverables:

This Spike is about exploring options for Inventory systems and implementing one into Zorkish

Deliverables Included:

- Code at <https://github.com/stormcrore/GamesProgramming2016>
- Report on different data structures

Technologies, Tools, and Resources used:

- Visual Studio 2013
- CppReference.com
- Classmates
- MS Word
- <http://en.cppreference.com/w/cpp/container>
-

Tasks undertaken:

- Research Data types
- Choose a data type to implement for Zorkish
- Create an Identifiable Item class
- Create a Game Object class, this is an Identifiable Object
- Create a class for Player, Inventory and Item, These are GameObjects
- Because Player causes a circular reference when implementing an Inventory we create an abstract class iCanHazInventory to inherit from, this includes the inventory class to inherit. Surprisingly this is similar to a component/composite pattern

What we found out:

We found pros and cons for four Data types: Maps, Vectors (Dynamic Arrays), Static Arrays and Lists.

We figured out how to implement a user defined class containing a Vector in Zorkish

Spike: Spike_7

Title: Graphs

Author: Peter Argent, 7649991

Goals / deliverables:

The Goal of this Spike is to use graphs to represent Locations and their connections

Deliverables included are:

- The Design for the Graph, as well as how the Text File that loads the graph in looks. Included in Appendix
- Code for Spike 7 included at <https://github.com/stormcrore/GamesProgramming2016>

Technologies, Tools, and Resources used:

- Visual Studio 2013
- CppReference.com
- Classmates
- MS One Note for Android for the Design
- MS Word

Tasks undertaken:

- Create classes for Location, World and Path
- Create a text file for the adventure
- Create a method "setUpGame()" in the Gameplay state that loads the data for the adventure
-

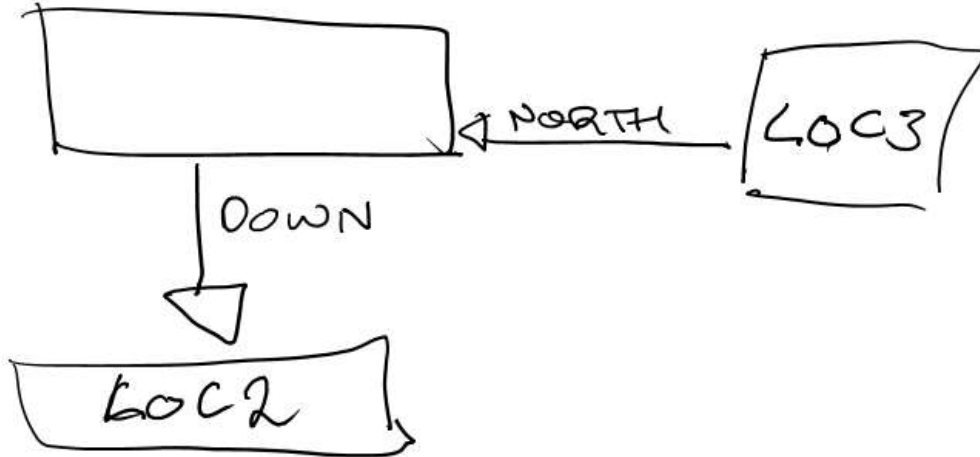
What we found out:

- The ability to load data from a file
- How to represent Locations and paths as an abstract Graph.

Recommendations:

Do this spike at the same time as spike 8, due to overlapping content. Spike 8 is for implementing movement and look commands, and thus can be used to test the Paths and locations of this spike.

Appendix



→ PATH : Directional



AdventureMap.txt

LOCATION

ID

NAME

DESCRIPTION

PATH

FROM

TO

<LOCATION
ID>

DIRECTION

DESCRIPTION

PLAYER

LOCATION

NAME

DESCRIPTION

ITEM

LOCATION

ID

NAME

DESC

ISINVENTORY

ISPICKUPABLE

Spike: Spike_No

Title: Spike_Title

Author: Peter Argent, 7649991

Goals / deliverables:

This Spike asked us to implement our game commands using the command pattern

Deliverables Include:

- Code at: <https://github.com/stormcroe/GamesProgramming2016>
-

Technologies, Tools, and Resources used:

- Visual Studio 2013
- CppReference.com
- Classmates
- MS Word

Tasks undertaken:

- Create a Command superclass
- Create a command processer
- Create a moveCommand
- Create a lookCommand
- Add the command processer to Gameplay
- Add the move and look commands to the processer

What we found out:

- This spike teaches the command pattern.
- How to create an expandable command processer and command classes

Open issues/risks:

- Issues In this Spike Include:
- Not adding identifiers

Recommendations:

Do this with spike 7, due to that being where we learn how to define the world and locations as a graph.

Spike: Spike_9

Title: Composite Pattern

Author: Peter Argent, 7649991

Goals / deliverables:

The Goal of this spike was to recognise the component pattern and implement it into the Zorkish code

Deliverables include:

- Code at <https://github.com/stormcrore/GamesProgramming2016>

Technologies, Tools, and Resources used:

- Visual Studio 2013
- CppReference.com
- Classmates
- MS Word

Tasks undertaken:

- Add a new command, Sharpen
- Add to game object the ability to set the description
- Allow Sharpen to set the description of the item to sharpen to the items description + "It is sharper than it was."
- Add pickUp and putDown commands.
- These will move items from the player's inventory to their current location and vice-versa.
-

What we found out:

- Learnt how to use commands to change properties for ingame objects for example location or description.
- Learnt that the world class is a composition of locations which is a composition of paths and items
- Learnt that what we had done with the Inventory class inheriting from a gameObject and containing gameObjects (Items).

Data Type Containers Report:

Maps:

An associative container of Key-Value Pairs

Pros:

- Each piece of data has a Indexical Key
- The Value of the map can be any Base or User-Defined Data type'
- Very Fast Search and Sort algorithms, due to low complexity
- Each Key is Unique
- Dynamic Number of entries

Cons:

- Each Key is Unique

Vectors (Dynamic Arrays):

Pros:

- Easy to add and remove items from end, $O(1)$ complexity
- Easy to implement in C++ due to automatic storage methods.
- Fast access time, $O(1)$ complexity due to Random access.
- Dynamic number of entries

Cons:

- Slow to remove items not at end, Defined by constant $O(n)$, where n is number of items in from the end.
- Can overflow the index, able to access other parts of memory using it
- Each value is of one data type (though this can be removed as a con if you have that data type as a class)

Static Arrays:

Pros:

- Easy to implement

Cons:

- Can overflow the index, able to access other parts of memory using it
- Hardcoded number of items in it
- Each value is of one data type (though this can be removed as a con if you have that data type as a class)
- As complex as the number of items in it.

Lists:

Pros:

- Iterator does not expire when members of the list are moved around between lists, or additional members added to a list or removed from a list, only when the member itself is deleted.

Cons:

- Only has a reference to the item before and after in the list, Fast access not supported
- $O(n)$ is the complexity of insertion and removal of objects in the list
- Hard to implement on larger scales

User-Defined Class

Pros:

- Can contain different types of data
- Can container other containers
- Implements methods that act on the data stored within

Cons:

- Harder to code without knowing Object Orientated Programming
- Much more complex than any other type of data structure

Decision on what to use for Player Inventory in Zorkish:

I used a Class based Container with a Vector of Items contained within. The reason I used this method is that I knew how to create Vectors in c++ already as well having need of special methods acting upon the data contained in the Vector.

I should use a map, however I am less confident in using a map, and this project will not strain the computers resources, thus I can avoid it at this time.

Spike: Spike_10

Title: Component Pattern

Author: Peter Argent, 7649991

Goals / deliverables:

The goal of this spike is to show an understanding of how a component can modify a class made by the programmer.

Delivered is:

- Code at: <https://github.com/stormcroce/GamesProgramming2016>

Technologies, Tools, and Resources used:

- Visual Studio 2013
- CppReference.com
- Classmates

Tasks undertaken:

- Created a SharpenComponent
- In this we had a Sharpen() method and an isSharp Boolean
- We added this to the Item class as well as its own sharpen method
- This Sharpen method will call the one on the sharpenComponent if it exists, otherwise it will output something else.
- Then we changed the SharpenCommand's execute method to call upon the items sharpen method when called.

What we found out:

We found out how the component pattern relates to the command and composite patterns as well how developers need to keep in mind how they are structuring their games during design, because it is hard to accommodate changes to class structures once the program has been developed for so far.

Recommendations:

My recommendation is for developers to have clear Ideas about what components are going to be necessary before implementing the main game systems. The experience has been for so much to be changed whenever we needed to implement a different way of completing a command; currently there is some vestigial code in the program that unfortunately has been superseded by the command pattern implementation and then again by the component and composite pattern implementation.

Spike: Spike_11

Title: Simple Messaging System

Author: Peter Argent, 7649991

Goals / deliverables:

The Goal of this spike was to create a simple messaging system.

Besides this report, what else was created?

- Code at <https://github.com/stormcroce/GamesProgramming2016>

Technologies, Tools, and Resources used:

- Visual Studio 2013
- Classmates

Tasks undertaken:

- Create A class for your Message object
- Anything that can accept a message object must have a PerformAction(message * messageArgument) method
- This method checks the messages action and if it can do something with it, it will
- EG Player can pick up things, Thus the players PerformAction does something if the messages action == "PICKUP"

What we found out:

We found another way for objects in a game to call actions on each other. The way messaging works is such that the object calling for the action only needs to know that there could be the other object and it won't crash if the other object has no way of completing the action asked of it.

Note:

This spike was completed in conjunction with spike 12: Create a messaging board system. Thus the way the messages are implemented, it is not direct from one object to another.

Spike: Spike_12

Title: Announcements and Blackboards

Author: Peter Argent, 7649991

Goals / deliverables:

This spike was to create a messaging board system that allowed for a more versatile messaging system.

For example: UML diagram, code, reports

- Code at <https://github.com/stormcroe/GamesProgramming2016/Spike11>

Technologies, Tools, and Resources used:

- Visual Studio 2013
- Classmates

Tasks undertaken:

- Create a blackboard Class
- This Class allows for Adding of Messages, Checking of Messages and Removal of Messages.
- Each Class that subscribes to a blackboard should be able to call an UpdateBoard() method that looks at each message on the blackboard and performAction(message on the board) for each message addressed to it or to all objects subscribed.

What we found out:

We learnt how to use blackboards to send multiple messages at once. It allows for the game to send and receive messages that can be accessed by multiple objects at once, while this functionality is not fully implemented in this build, it is easily expandable.

Note:

This was completed at the same time as Spike 11, thus is within the spike 11 folder on github

Spike: Spike_13

Title: Unit Testing

Author: Peter Argent, 7649991

Goals / deliverables:

This Spike shows a way of testing logic in code in such a way that you don't have to step through Breakpoints, It is often used within defensive programming

- Code at https://github.com/stormcroe/GamesProgramming2016/Spike_10
- Xamirin C# Code to show unit testing in action due to bad linker errors in the C++ code. View at <https://github.com/stormcroe/GamesProgramming2016/Spike13>

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Visual Studio 2013
- [https://msdn.microsoft.com/en-us/library/hh419385\(v=vs.120\).aspx](https://msdn.microsoft.com/en-us/library/hh419385(v=vs.120).aspx) for a short guide on how to create unit tests.
- Xamirin Studio for C# unit test examples.

Tasks undertaken:

- Create a new Test Project in the solution that you wish to test (We are using Spike10/Zorcish to test)
- Include files necessary to test
- Create a test method for each method you wish to test
- Use Assert:: functions to check the logic of your code.

What we found out:

We found out that Linking Test files to our actual code can be awkward and can mess up, the steps above should work but unfortunately we did not have time to go through linker errors to find out what file we keep forgetting to include.

The Tests are currently commented out. Included in the Portfolio is some C# unit testing to show how to properly unit test. These ones should work.

Open issues/risks:

- Linker Errors on trying to access the sharpenable component, unsure what was wrong, limited time to figure it out, might be something to do with includes.

Recommendations:

Take time to include all files into the test cpp. Linker errors are a pain to deal with.

Spike: Spike_15

Title: Unreal Engine Familiarisation

Author: Peter Argent, 7649991

Goals / deliverables:

- Report at <https://github.com/stormcroe/GamesProgramming2016/tree/master/spike15>
- The unreal engine project is at <https://github.com/stormcroe/GamesProgramming2016/tree/master/spike15>

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Unreal Engine 4.12.5
- The Inbuilt tutorials in Unreal studio.
- Wikipedia List of Unreal Engine games
- My own experience with Smite, Injustice and Bioshock
- <http://blog.digitaltutors.com/unreal-engine-4-vs-unity-game-engine-best/>
-

Tasks undertaken:

- Completed the tutorials associated in Unreal Engine 4 itself.
- - Welcome to Unreal Engine
- - Blueprints/Tips for editing blueprints
- - Blueprints/Working with components on actors
- Created a new blueprint named “ ”
- In the blueprint editor created a visual script to print the frames since start to the console log
- Created a Report to put each task required into it.
- Added a section to the report screenshots of the visual script and its comments to the report.
- Added a section to the report on three games and how and why they use Unreal Engine
- Added a section to the report detailing a comparison between Unity 5 and Unreal Engine 4

What we found out:

Describe the outcomes, and how they relate to the spike topic + graphs/screenshots/outputs as needed

- The outcomes for this spike was to find out about the blueprinting options in Unreal Engine, It gets us to find out about these options by pointing us towards the tutorials in Unreal Engine.
- The report submitted with this report also shows 3 games and why I believe that they made the right decision to be made with Unreal.
- In addition the report also shows a small blueprint made using the visual scripting that Unreal offers. The visual scripting is a very handy tool to have as it allows non-programmers a decent chance to be able to help with the design of a game or scene in Unreal.
- Lastly the spike mentions how not every Engine is right for every game. Included in the report is a comparison table between Unreal Engine 4 and Unity 5.

Unreal Engine Games:

Smite

Hi-Rez Studios; MOBA; UE3

Smite uses a fairly generic over-shoulder camera angle to display the player character and actions, it also has a fairly in-depth physics system. Thus using Unreal Engine 3 to start is a reasonable decision by the developers of the game. The game is multi-platform thus creating the game engine from scratch there would be more pain than it is worth, thus using a genericised game engine like Unreal helps massively for multi-platform builds.

Injustice: Gods among Us

Nether Realm Studios; Fighting; UE3

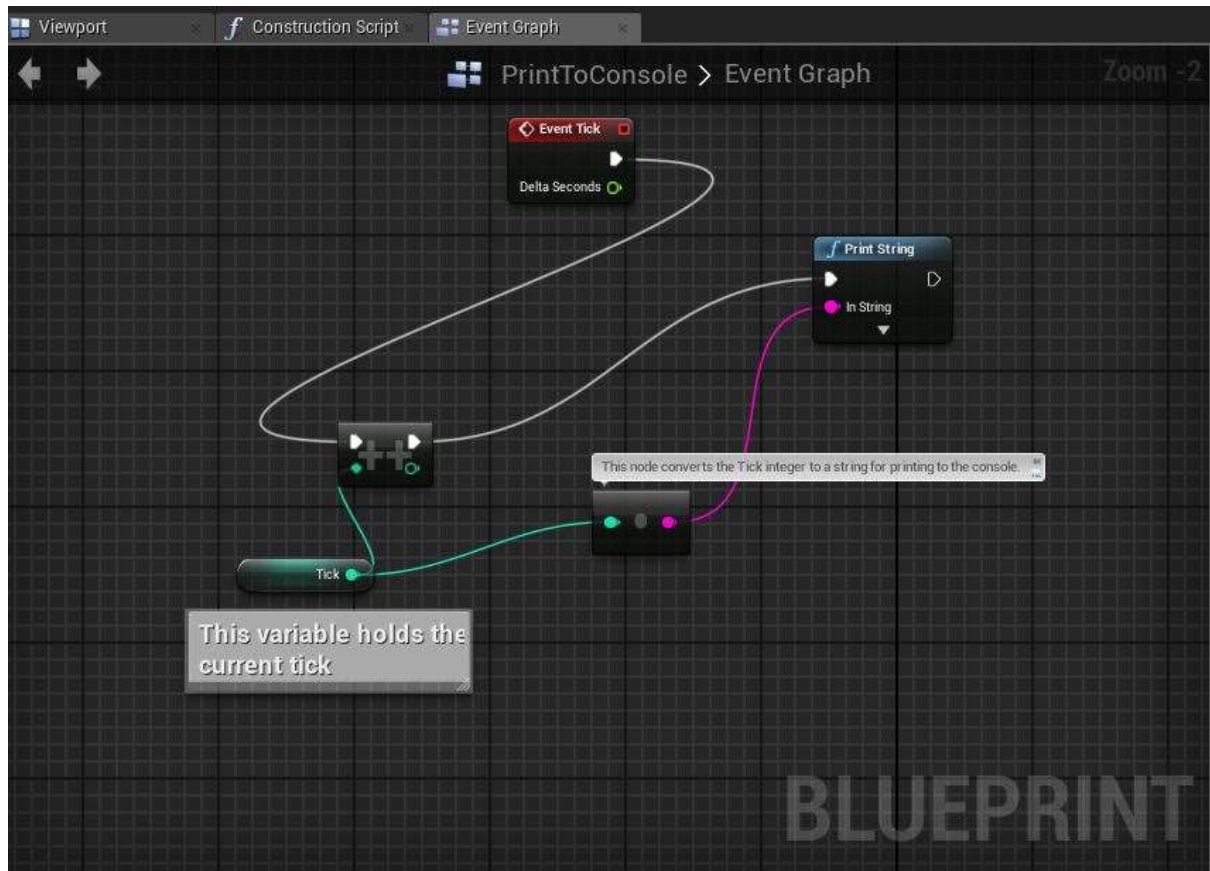
Nether Realm Studios uses the Unreal engine to quickly drop characters in and start animating them so that they can quickly iterate and improve Unreal Engine's systems to optimise of the fighting game genre. Injustice has a 60fps framerate that was easily accessed using the engines properties.

Bioshock

2K Studios; First Person Shooter; UE2


The Original unreal engine was made for the Unreal series of games. This series of games are first person shooters, thus 2K's decision to base their first person shooter on the Unreal Engine was a good one that enables them to quickly get to coding the unique parts of their game.

Unreal Blueprints:



This screenshot shows the blueprint system in Unreal Engine 4. It shows two types of comments used within the editor, as well as a simple tick counting method which outputs to the console.

Credit Task: Comparison between UE4 and Unity.

	Unreal Engine 4	Unity
Programing Language	C++	C# or Javascript
Visual Scripting	Blueprints	N/A
Basic 3D modelling	In engine modelling using basic shapes + Easy importing	Easy importing for 3D models, no basic shapes
Assets store	Well defined	Larger in scope, mostly free.
Profiler	All versions	Only UnityPro
Graphical Engine	Complex Particle Simulations and Dynamic Lighting	Basic lighting inbuilt, Dynamic lighting can be found on the asset store. Particle interactions are simple, but there are more complex interactions in the asset store for sale.
Pricing	Free but takes 5% of sales	Free version and Pro Version. No royalties; Pro version costs US\$75/month
Physics System	Detailed collision system	Detailed Collision system, inbuilt mathematics (Quaternions)
Mascot	None	Unity-Chan 

Reference: <http://blog.digitaltutors.com/unreal-engine-4-vs-unity-game-engine-best/>

Comparing when to use Unreal Engine 4 to Unity, your choice boils down to what you expect to be doing with the engine, and how experienced you are in OO programing. Both engines lend themselves to 3D games over 2D and to First and Third Person perspectives when creating games. Both Engines have detailed and significant collision and physic systems; However Unity supports quaternions simpler in the base engine. In terms of other features, Unity has less features built into the base engine but has a larger asset store with many assets and functionalities free for purchase from this store. Unity is in a simpler language for moderately experienced programmers to use; however most simple functionality that is hard to create using C++ is easily accessible using the blueprint visual scripting system. Which engine the programmer uses comes down entirely to preference, both are generic systems that can implement good functionality for any basic genre of game.

Spike: Spike_15

Title: Creating Simple Scene

Author: Peter Argent, 7649991

Goals / deliverables:

- The goal of this spike is to show how we place actors in the world and how to import 3d models.
- The unreal engine project is at <https://github.com/stormcrore/GamesProgramming2016>

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Unreal Engine 4.12.5
- The Inbuilt tutorials in Unreal studio.
- Online 3D Models, Searched for a Corpse Model in Google

Tasks undertaken:

- Download/Obtain some 3D mesh objects
- Import them in Unreal Engine
- Place them in the world
- From here we can use Blueprints to make the objects do stuff like move in a direction constantly, or even more complicated things.

What we found out:

- We found out how to import 3d models into an Unreal scene.
- In addition we used the blueprint system to have them move around and collide.

Spike: Spike_16

Title: Creating Blueprints from C++

Author: Peter Argent, 7649991

Goals / deliverables:

The Goal of this spike is to show how C++ can be integrated with the blueprint system of Unreal Engine 4

Deliverables:

- Code at <https://github.com/stormcrore/GamesProgramming2016>

Technologies, Tools, and Resources used:

- Visual Studio 2015
- Unreal Engine 4.12.5
- Unreal C++ Blueprint Tutorial/Guide at https://wiki.unrealengine.com/Blueprints,_Creating_C%2B%2B_Functions_as_new_Blueprint_Nodes

Tasks undertaken:

- In Unreal: Add New -> New C++ Class
- Parent Class is Actor
- In Visual Studio 2015:
 - Create the Private Data Variables
 - Create the Getters, These are preceded by "UFUNCTION(BlueprintCallable, BlueprintPure, Category = "classname")"
 - Create the Setters, These are preceded by "UFUNCTION(BlueprintCallable, Category = "classname")"
- Save and Build the C++ files in Visual Studio
- In Unreal:
 - Find Where you created the cpp class in the content browser
 - Add one to the scene
 - Compile on the Scene
 - Select The Actor that represents the class and click the Blueprint/AddScript button in the details pane
- In The Blueprint Editor we can now Use this to visually script the program we had in mind.
- Compile the blueprint and scene again. And then you can run the program.

What we found out:

We found out how to expose C++ files to the blueprint system to allow for visual scripting in addition to the complex and customised programming ability that C++ provides.

UFUNCTION () allows a method to be used within Blueprints, with BlueprintPure as an argument to this allows for functions to be separate from the exec chain and thus be called upon to provide accessors and calculations that do not require changing the game state.

UPROPERTY () allows a field to be accessed in a property like state in the Blueprint editor, with the argument it is allowed to be accessed in the actual editor as well.

Open issues/risks:

A Major issue I had with task is because I usually use Visual Studio 2013, I didn't have the right dependencies required to use it in addition with Unreal Engine 4.12.5. The Developer doing this spike must be aware of these concerns.

Spike: Spike_17

Title: Input Handling

Author: Peter Argent, 7649991

Goals / deliverables:

The Goal of this spike is to familiarise the user with Unreal's user input system.

Delivered is:

Code at: <https://github.com/stormcroce/GamesProgramming2016>

Technologies, Tools, and Resources used:

List of information needed by someone trying to reproduce this work

- Unreal Engine V 4.12.5
- Classmates
- Unreal Tutorials

Tasks undertaken:

- Navigate to the Project Settings (Edit > Project Settings)
- Go to Engine > Input
- In Bindings > Action Mappings
 - Set MoveForward > S to a scale of 1
 - Set MoveForward > W to a scale of -1
 - Set MoveRight > D to a scale of -1
 - Set MoveRight > A to a scale of 1
- Then in Bindings > Action Mappings we Created a new action mapping
- Any Key will 'Print'
- Going to The FirstPersonCharacter Blueprint
- We Right Click to Create an event for the Print Action Mapping
- Then we link that to a generic print screen, thus whenever the player makes an action "Hello World" is printed to the console
- We also created an output string to a pinch input that reads "You Pinched"

What we found out:

We learnt how to modify how unreal handles inputs using the Blueprints.

Spike: Spike_18

Title: Sound

Author: Peter Argent, 7649991

Goals / deliverables:

The Goal of this spike is to allow the user to play around with sounds in Unreal Engine.

Deliverables

- Code <https://github.com/stormcroce/GamesProgramming2016>

Technologies, Tools, and Resources used:

- Unreal Engine 4.12.5
- Classmates

Tasks undertaken:

- We create a new Unreal Project.
- On the Player's Pawn we attach an Audio Component, adding a sound cue to it
- On another actor in the world we attach a sound cue to an audio component, but it defaults to always playing.
- To have sound fall off over distance we need to in the Attenuation tab of the details:
 - Allow Spatialization
 - Override Attenuation
- Then we can change how far out the sound plays by adjusting the radius in this manu

What we found out:

How to play sounds attached to the player and to objects in the world

Spike: Spike_No

Title: Spike_Title

Author: Peter Argent, 7649991

The Goal of this spike is to allow the user to understand how to possibly use collisions in a game.

Deliverables

- Code <https://github.com/stormcrooe/GamesProgramming2016>

Technologies, Tools, and Resources used:

- Unreal Engine 4.12.5

Tasks undertaken:

- Create a new FPS blueprint project
- Open the First Person Projectile Blueprint
- Between the Event Hit and the Physics Impulse Branch, Add a Sequence, We will use this to split from the Event to implement additional functionality
- From here we create a branch that asks if the thing we hit was simulating physics
- If True, we set the material of the thing the projectile hit and change its material
- I have expanded this to also randomise the material the object changes into.

What we found out:

I found out how to use the blueprint systems logic using Branch and Sequence nodes.

I found out that there are logic operators in the Blueprint system

I found out how to use the blueprint system to call events, ie the Collision.

Appendix

