Jack Porter, Lewis Rosier, Ed Smith

Q12313882, Q11585706, Q12934917

Abstract

Software design and implementation of a 3D game using a variety of good software practices and technical components

AE1 - GROUP PROJECT

CGP600 Advanced Games Programming AE1

Contents

[Analysis and Design 2](#_Toc528873822)

[Functionality and core requirements 2](#_Toc528873823)

[Flow/Class Diagrams 2](#_Toc528873824)

[Development Techniques 3](#_Toc528873825)

[Object Oriented Design 3](#_Toc528873826)

[Task Breakdown and Rational 4](#_Toc528873827)

[User Stories 4](#_Toc528873828)

[Work Breakdown Structure 4](#_Toc528873829)

[Critical Paths, Tasks, Timescale, Dependencies Grid Tasks and Times relating to WBS 4](#_Toc528873830)

[Testing Plans 4](#_Toc528873831)

[Critical Reflection and Discussion of Group Work 5](#_Toc528873832)

[Evidence of Equal Distribution of Work 5](#_Toc528873833)

[Reflection of the Design Process 5](#_Toc528873834)

[Identification and Resolution of Problems 5](#_Toc528873835)

[Software Backup Methodology 6](#_Toc528873836)

[Appendix 6](#_Toc528873837)

[1 – Design 6](#_Toc528873838)

[1.1 – User stories 6](#_Toc528873839)

[1.2 – WBS - Lewis 8](#_Toc528873840)

[1.3 – Gantt chart - Jack 8](#_Toc528873841)

[1.4 - Equation research - Lewis 8](#_Toc528873842)

[1.5 – Game flow diagram – Lewis 9](#_Toc528873843)

[1.6 – Class diagrams - Ed 12](#_Toc528873844)

[1.7 – Pseudocode - Ed 17](#_Toc528873845)

[1.8 – Test plan – Jack 19](#_Toc528873846)

[References 23](#_Toc528873847)

# Analysis and Design

## Functionality and core requirements

The game that I have designed with my group that I will be making is a 3D platformer inspired by games like Super Mario 64 (Nintendo Co., 1996), and A Hat In Time (Gears for Breakfast, 2017). 

Super Mario 64 (Venture Beat, *Super Mario 64,* 2016. )

The player will be tasked with navigating the world picking up 8 map pieces to locate the treasure chest which must be collected to complete the level. In the world are enemies that will patrol between points and if the player gets too close to them, they will chase the player down. The player can kill the enemies by jumping on top of them and will take damage if they collide with the enemy in any other way. If the player takes damage they will be able to collect coins to heal themselves.

## Flow/Class Diagrams

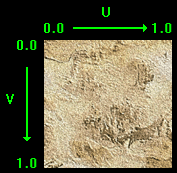
Ed took the lead role on the class diagrams (Appendix 1.6) and pseudocode (Appendix 1.7), however we all had input into their design and we made sure that we were in constant review of the design so that we could make changes that suited our needs.

Lewis worked on the game flow diagrams (Appendix 1.5) in which he mapped out some basic logic for the AI and also a flow map for the player with all the input options available to them.

## Development Techniques

For collision in this project I plan on using capsule collision (Thunderfist, 2012) on the player and enemies, sphere collision (cursorweb ed., 2018) on the pick-ups and box collision for the environment. Capsule collision is a good way to do fast collision detection for a humanoid shape with sphere collision being the simplest form of collision, making it perfect for collectables, finally box collision, or AABB collision allows us to have basic geometry for a level that can be traversed.

Textures are a way to enhance the look of a 3D model, these textures could be a diffuse map (dArroway Textures, n.d.) to give detail to the models surface by applying an image to it, a normal map (Opengl-tutorial, n.d.) to give more realistic lighting to a model or even give it a more detailed surface, and make a flat surface look 3D, or even a displacement map which actually allows you to alter the geometry of the model to give more detail.



(rastertek, *UV example on a texture, n.d.*)

This is an example of a texture, it has a U and a V coordinate which goes from 0.0 to 1.0.s

Shaders (Satran, M., 2018) are used to turn the original code and turn them into a 2D image to be displayed on the screen, this is called rasterizing (Techopedia a, n.d.), DirectX uses HLSL or High Level Shader Language to program shaders.

## Object Oriented Design

I will be using Classes (Moore, K. et al., Classes (OOP), n.d.), abstraction (Techopedia b, n.d.), inheritance (Adobe, 2012) and virtual functions (Mikeblome, n.d.) in this project.

Classes are the bases for objects in the program that can store data, functions and references to other objects.

Entity will be an abstract class, this is because Entity will not be an instantiated object its self, however it will be used for Player Pickup and AI to inherit from, this is because they will all share the same basic functions and features such as position and movement, but will also all have their own functions and features that the others will not have need for, such as the player needing input or the AI needing pathfinding and states.

Virtual functions are functions declared in a class that is later filled out by an inherited class, this allows you to have many classes use the same function call from an abstracted class, but have them all do different things, for example, having a base collider class and having sphere, capsule and AABB collision all use the same “CheckCollision()” function.

Lewis also designed a small maths library with functions (Appendix 1.4), however we later found out that DirectX has its own library built in, and so we will be using that instead.

# Task Breakdown and Rational

## User Stories

We looked at the core requirements for this development product and at our basic game design that we had created and set about writing the user stories for this project. We thought about both the stories from the player’s perspective, as well as from the client’s perspective, trying to cover every one of the core requirements in our user stories and every aspect of the gameplay.

The user stories can be found in (Appendix 1.1).

## Work Breakdown Structure

Lewis designed the WBS for this project (Appendix 1.2), using the user stories that we created as a group, he broke them down into tasks representing their smallest form that encompasses a module, after this, we all went over it as a group to make sure that we were happy with those tasks, we didn’t feel like any changes had to be made to these as Lewis had done a good job.

## Critical Paths, Tasks, Timescale, Dependencies Grid Tasks and Times relating to WBS

Using the WBS that was created, I then took all of the tasks and formatted them in to a Gantt chart (Appendix 1.3). I attempted to change the daily working hours and add an hourly work field to the chart, however they caused more problems than they solved and so I opted to just use days as a measure for working time. Since we are all students and have many other units as well, I assumed that we would have less time in a day to work on this project than if we were working solely on it, as such the day is assumed to have around 4 hours of working time instead of the default 8 hours.

For the time allocated to each task I estimated how long I would take to complete a task, I also included some extra time to allow for some testing of the feature that will be implemented. Finally I took the time that I was left with and doubled the value, this allows for some wiggle room if a task is particularly tough to figure out.

After I had finished I had the rest of my team look and go over it to make sure that they were happy with the critical path shown by the Gantt chart and also with the time allocated to each task, thankfully they were pleased with my estimations and were happy with the final result.

## Testing Plans

I took on the lead role for designing and building the testing plan for this project (Appendix 1.8), I have based the design around 2 types of black box testing, two types of white box testing and a section for logging ad-hoc testing

The black box testing that will be used is Systems testing, which makes sure that the program is meeting the functional requirements of the design and is the first level of testing of the product as a whole. We are also doing acceptance testing, which is used to test the product against the client’s requirements to make sure that they meet the design specified at the start of the project.

The white box testing that will be used is unit testing, which allows a developer to test specific functions or areas of code against a confirmed outcome to make sure that the code is performing as it should be. We are also going to include bottom up integration testing, which requires us to build and test each system from the lowest level upwards to make sure that each system will be working completely alone and with others before moving onto the next one.

I learned about these forms of testing in a lecture for Mobile Applications (Thomas, N., 2018)

# Critical Reflection and Discussion of Group Work

## Evidence of Equal Distribution of Work

For this assignment, I was tasked with creating the testing plan and the Gantt chart, This is because I was the only one of our team who was familiar with the software required to make a Gantt chart, and I was also confident in creating a thorough test plan. Ed was tasked with creating the Class Diagrams and Pseudocode, Lewis was tasked with creating the WBS and a maths library.

Whilst we all focused on specific parts of this project, we all made sure to constantly check each other’s work to ensure that we were happy with how the progress was going and to suggest any changes that could be added to the design.

The design of the game and the user stories were things that we all worked on at the same time in the early stages of planning as they were the basis for the rest of the design.

## Reflection of the Design Process

On reflection of this design process I am mostly happy with how it has gone. I would personally make some small changes to the class diagrams in that I would add an abstract collision class that would be inherited from to create a sphere, capsule and AABB collider classes.

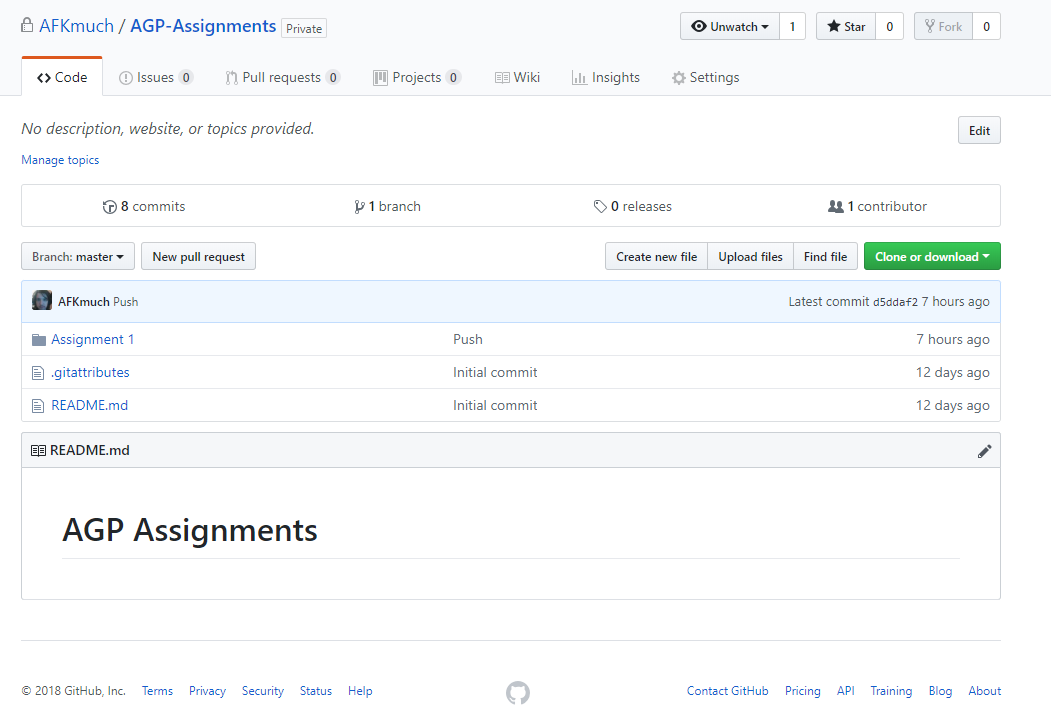
## Identification and Resolution of Problems

For this assessment we had one major problem, the IT staff at our university had failed to install the required software for the course until 5 weeks into term, this made it hard to design a game for DirectX when we couldn’t put into practice our knowledge from the lectures about the subject. However it did allow for us to spend more time on the less technical aspects of the design document and so we were able to balance it out.

It was also difficult to design a project that we would be working on solo as a group, as we had to find a balance between techniques that some of us had already worked on before and other hadn’t and techniques that we were interested in learning through this project, for me those were A\* pathfinding for AI and binary space partitions, both of which Ed has already developed, where I have not.

## Software Backup Methodology

For this project I will be using Github (GitHub, Inc, 2018) as a method of source control, this will allow me to work on my assignment from home and at university and store my work in the cloud in case of hardware failure, it also will allow me to revert any changes that would otherwise take too long to re-write.



This is an image of the GitHub repository that I have created for this unit, I have used it to keep a copy of this written report and will use it to keep my programming assignment as well.

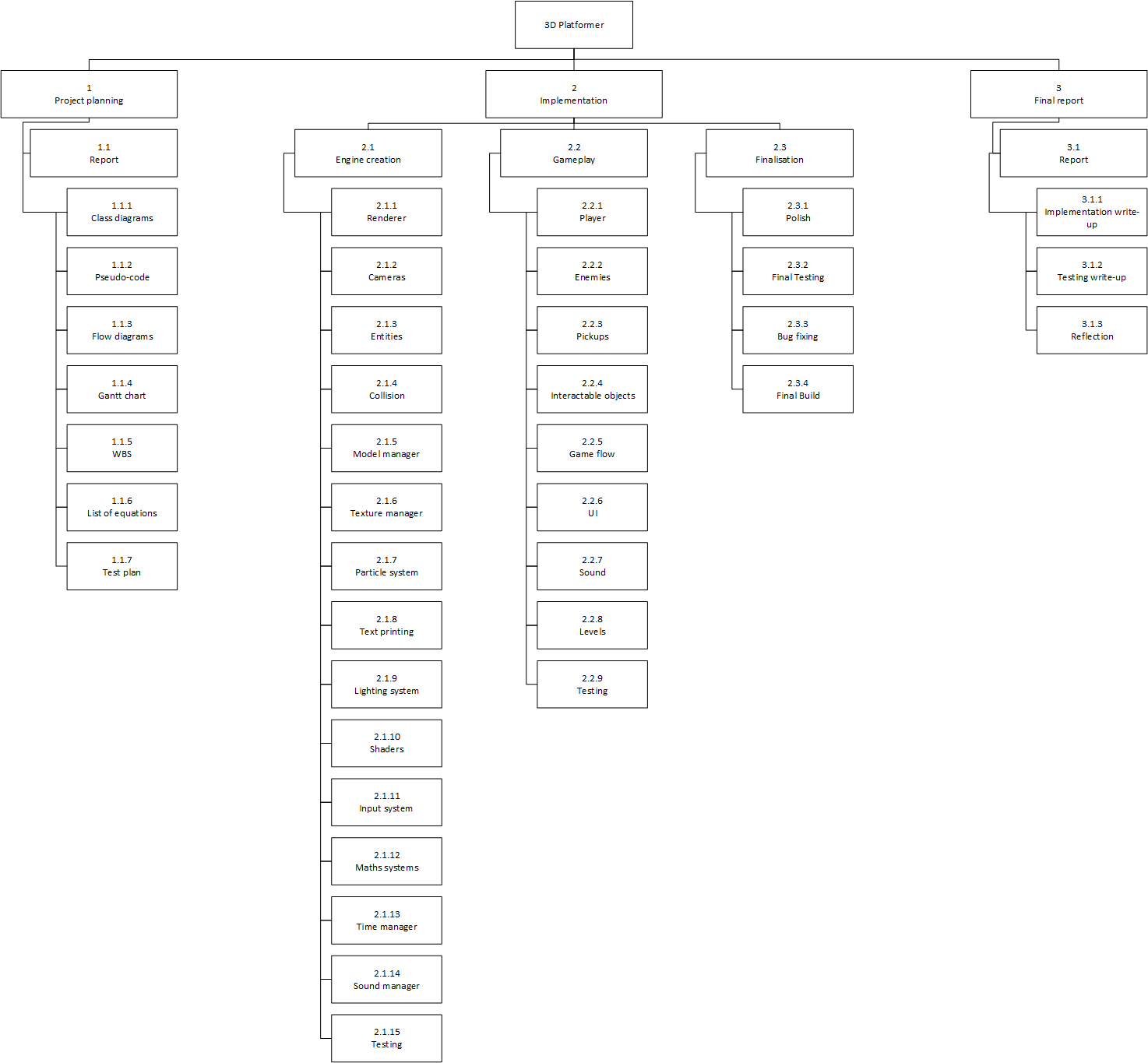
# Appendix

## 1 – Design

### – User stories

* As a Player I want to move around so that I can navigate the level
* As a Player I want to jump so that I can kill enemies and activate buttons
* As a Player I want to collect coins so that I can increase my score
* As a Player I want to collect map pieces so that the chest will spawn
* As a Player I want to avoid enemies so that they will not kill me on contact
* As a Player I want to see enemies navigate the world so that the world feels more alive
* As a Player I want to navigate around obstacles so that I can solve problems and have fun
* As a Player I want to collide with the world and entities in the game so that the world is shown as a set of obstacles to navigate
* As a Player I want to stay over 0 health so that my character doesn't die
* As a Player I want to have the game be rendered at a high frame rate so that I can see the game world update smoothly
* As a Player I want to move the camera so that I can look around the game world
* As a Player I want to have a UI so that I can see information on the game
* As a Client I want enemies to move between nodes so that it shows patrolling to make enemy movement more interesting to player
* As a Client I want to have enemies make use of a finite state machine so that it breaks up behaviour into states making code cleaner and easier to debug
* As a Client I want the game be 3D and written with Object Oriented C++ using Visual Studio and DirectX so that the game fulfils the requirements
* As a Client I want objects in the environment have textures and lighting so that the game looks presentable while showing off technical features
* As a Developer I want to have a model manager to reduce load times and memory usage
* As a Developer I want to have a texture manager to reduce load times and memory usage
* As a Developer I want to have a particle system so that effects can be created easily
* As a Developer I want to have a lighting system so that the world is believably lit
* As a Developer I want to have text displayed on the screen so that UI elements can be created
* As a Developer I want to use shaders so that objects can be drawn as intended
* As a Developer I want to have a sound system so that the game world can feel more alive with audio
* As a Developer I want to have a time system so that I can use the time between frames to optimise my game

### – WBS - Lewis



### 1.3 – Gantt chart - Jack

Microsoft Project file: <https://drive.google.com/open?id=1x7BhkVccdtUKBzcSTPrR0G2GCVuVuXeg>  
PDF: <https://drive.google.com/open?id=1tJbs2dFzddk7WC3t2yawReVkdfp7dlJa>

### 1.4 - Equation research - Lewis

#### Vector maths

**Vector3.Angle**

**Vector3.ClampMagnitude**  
ClampedVector =

**Vector3.CrossProduct**  
CrossProduct =

**Vector3.Distance**  
Distance =

**Vector3.DotProduct**  
DotProduct =

**Vector3.Lerp**

**Vector3.Magnitude**  
Magnitude =

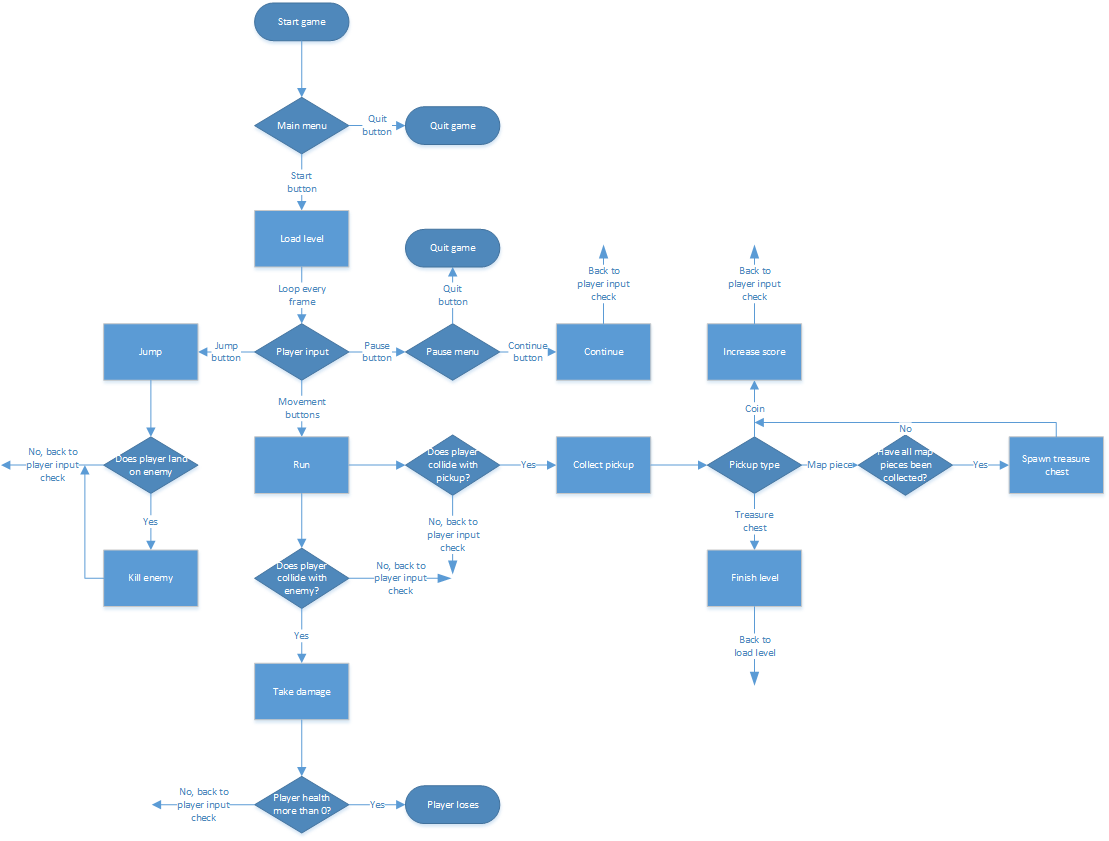
**Vector3.Normalise**  
NormalisedVector =

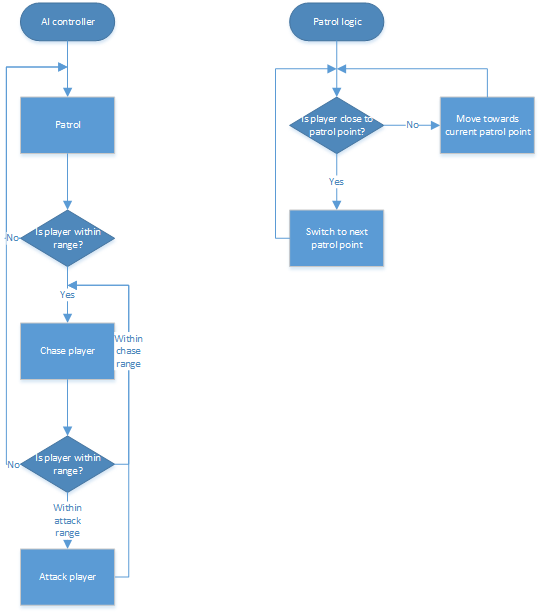
#### World/Local

**World to local**  
Local = position – target’s position

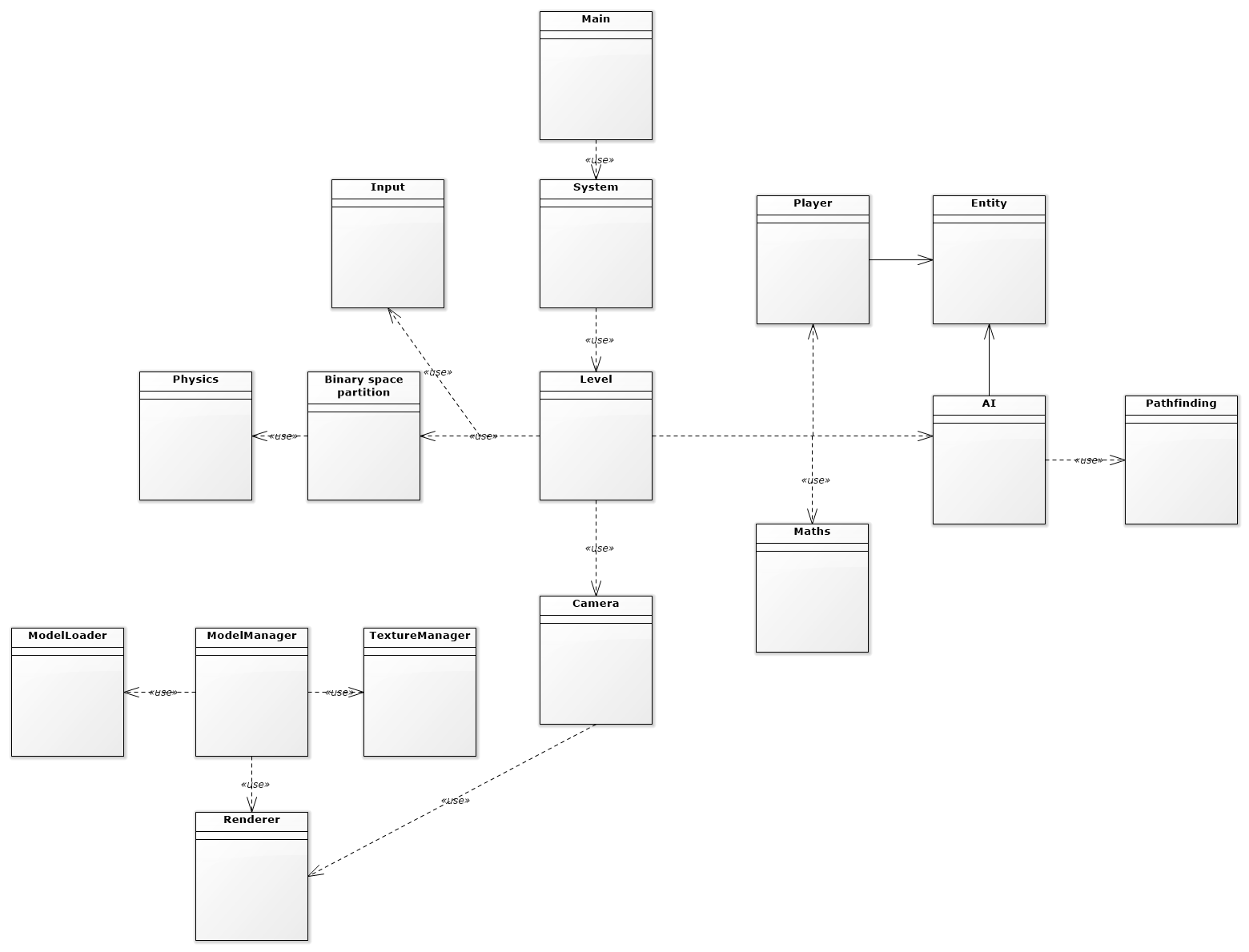
**Local to world**  
World = position + target’s position

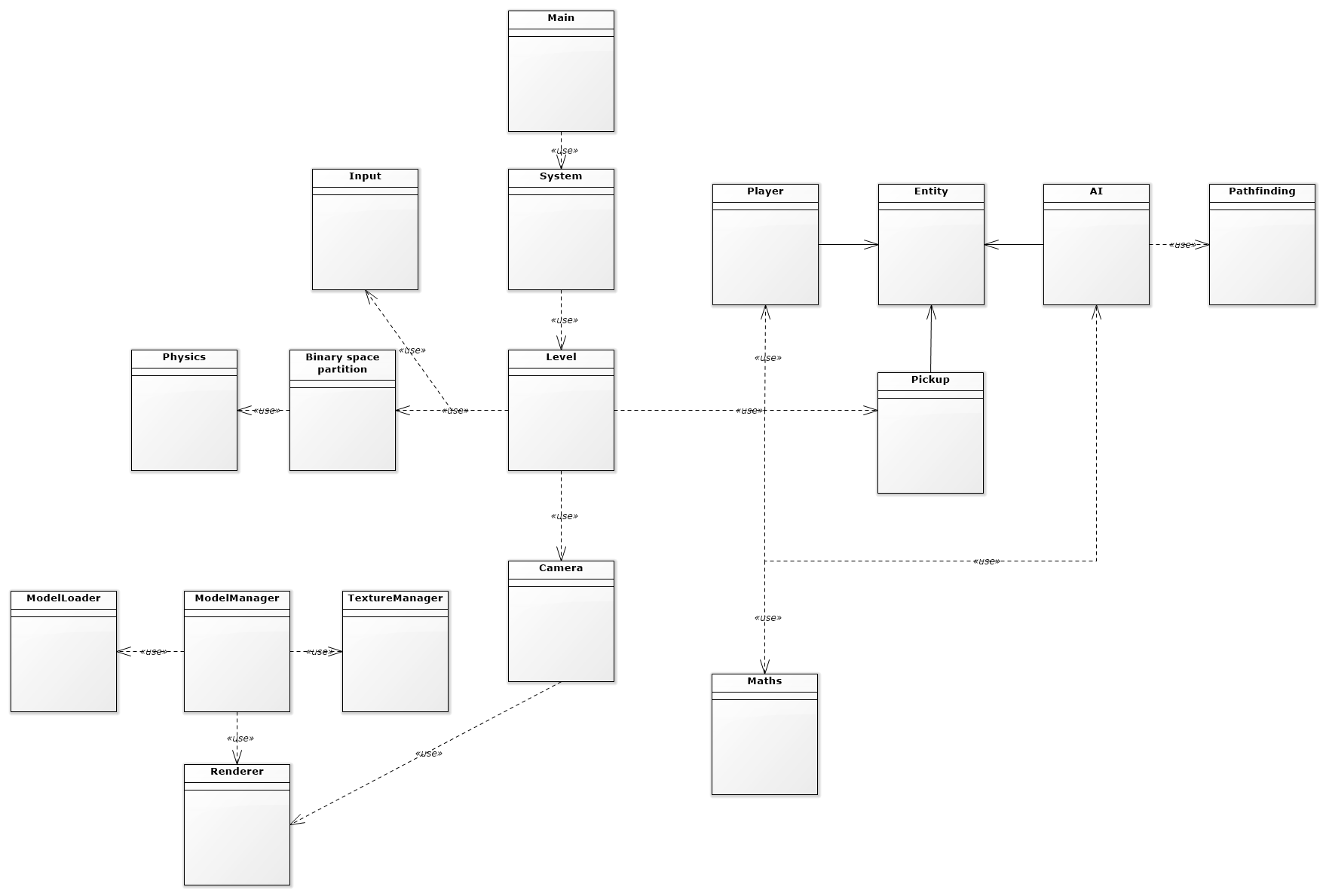
### 1.5 – Game flow diagram – Lewis

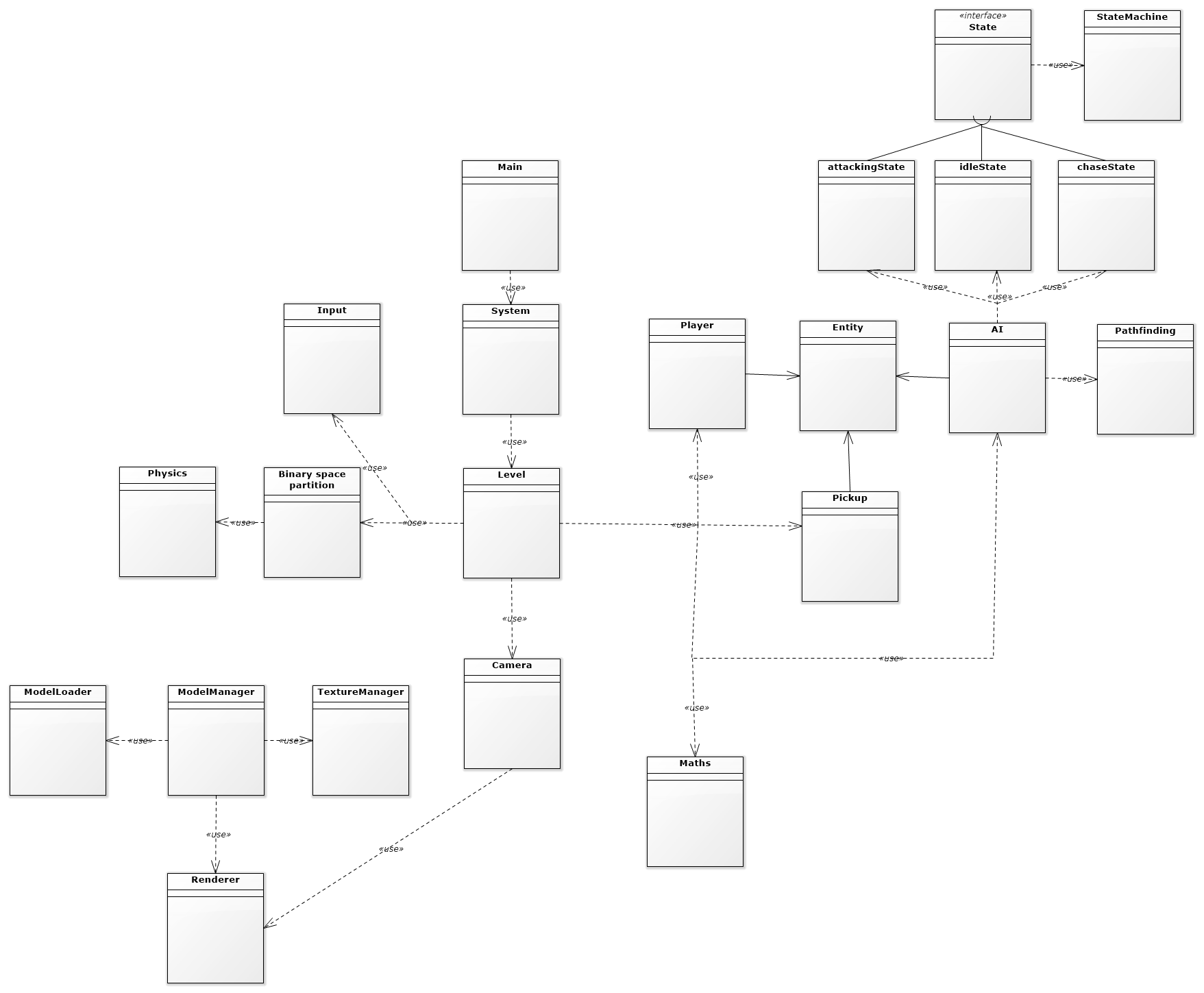


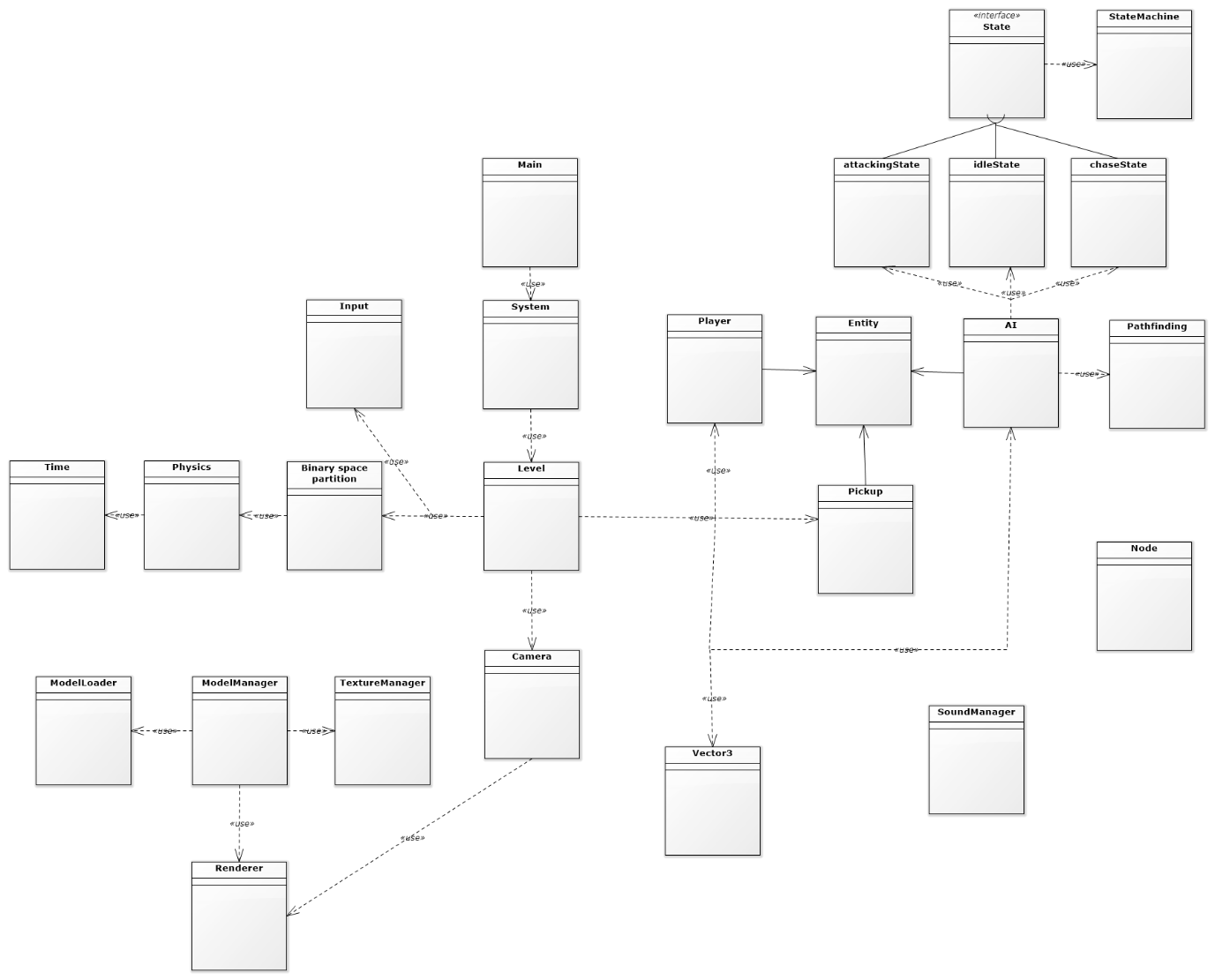


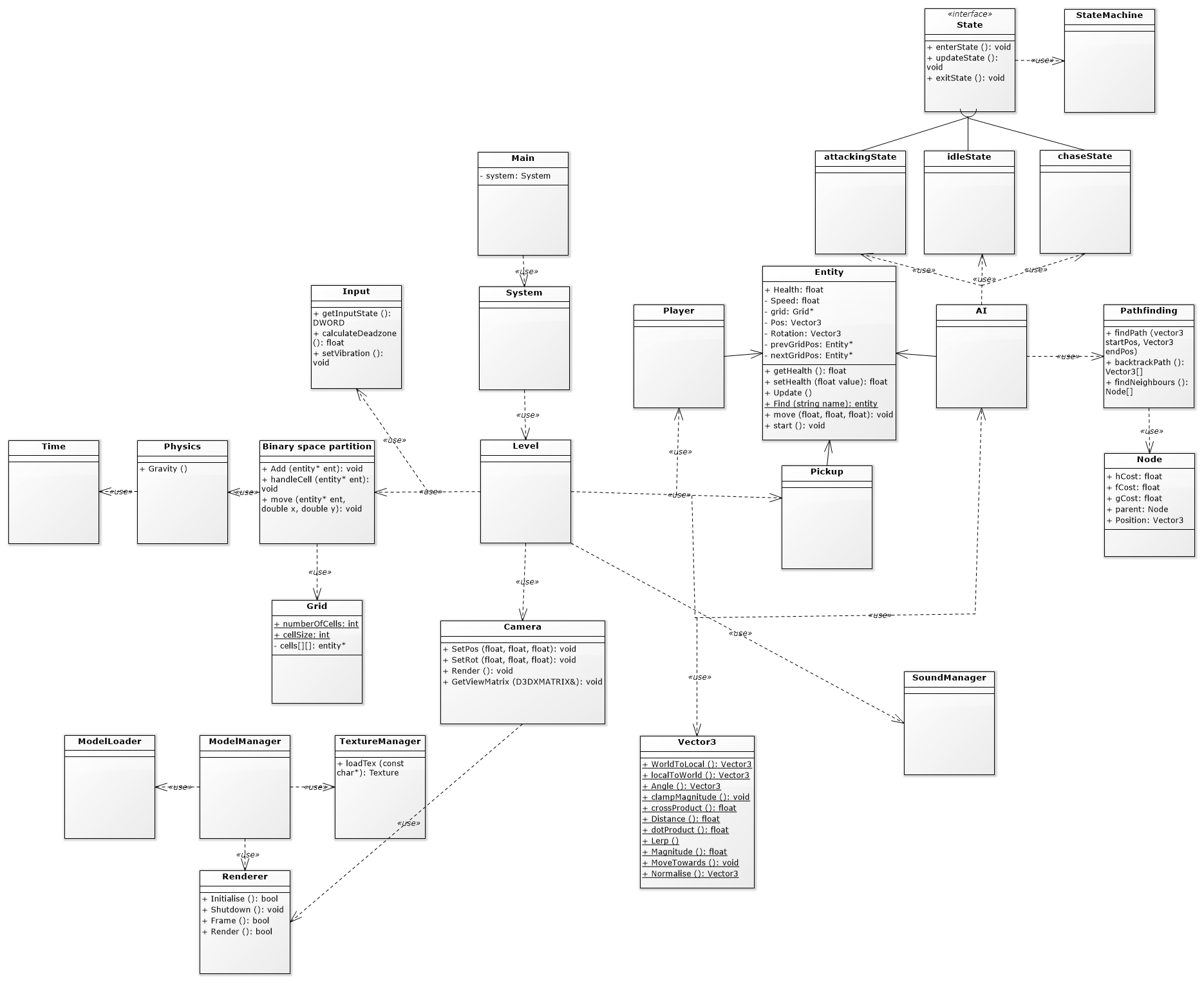
### 1.6 – Class diagrams - Ed

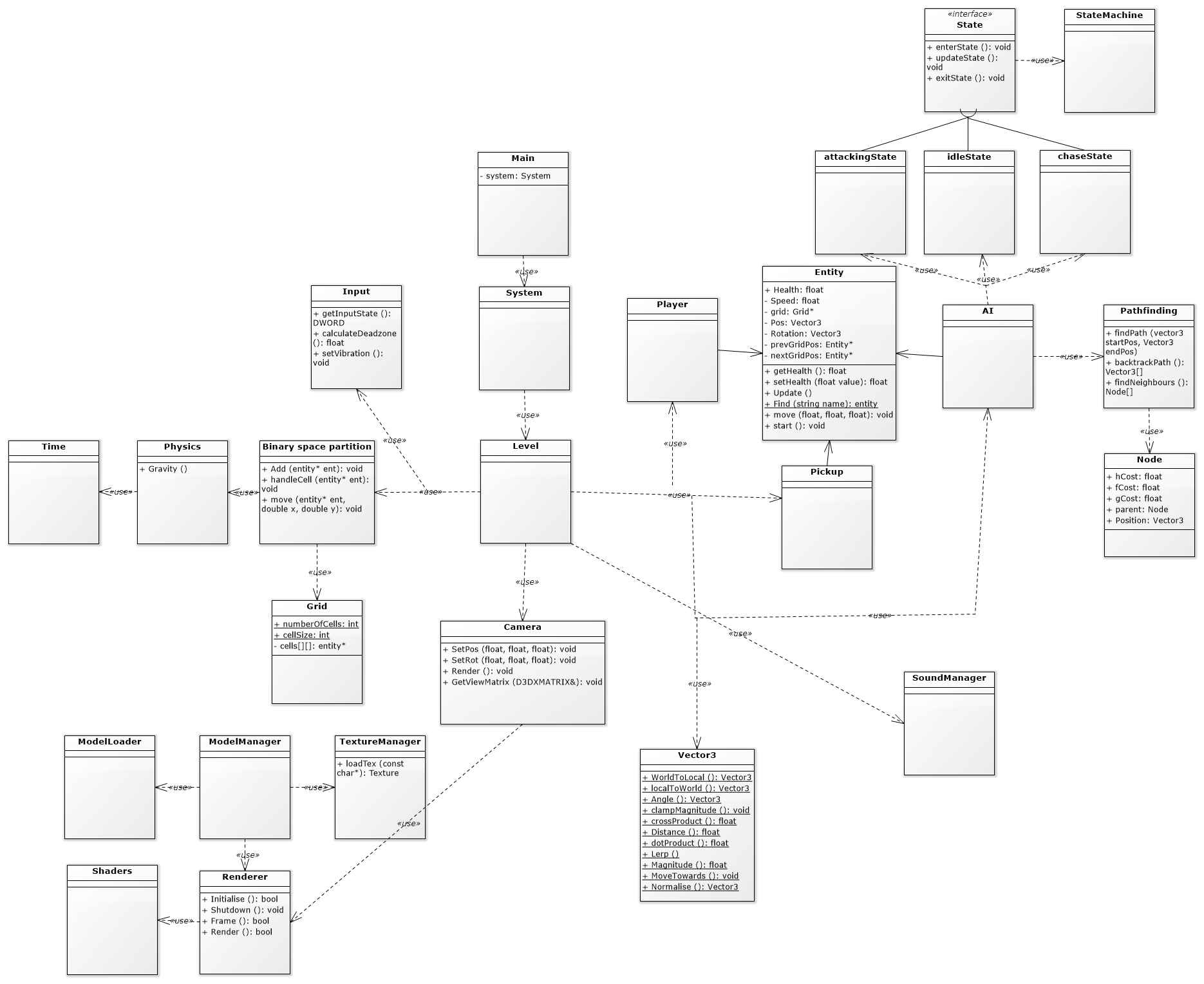
Revision 1:  


Revision 2:  


Revision 3:  


Revision 4:  


Revision 5:  


Revision 6:  


### 1.7 – Pseudocode - Ed

**Renderer**

**Initialise**

|  |
| --- |
| Initialise window  Initialise Shaders  Initialise viewport and perspective |

**Render**

|  |
| --- |
| Clear target view  Set render target  Set buffers  Set shaders  Draw |

**Binary space partition**

**Add**

|  |
| --- |
| Determine position in grid  Previous grid move is equal to null  Next move is equal to grid position  Grid position is equal to this entity |

**handleCell**

|  |
| --- |
| While this entity is not null      Other entity is equal to this entity next move      While other entity is not null          if(collision)              Handle collision      Other entity is equal to this entity next move  This entiy is equal to next move |

**Move**

|  |
| --- |
| Calculate old cell  Calculate next cell  Entity position is equal to desired move pos  If entity didn’t move,      Return  Unlink entity from list of old cells  If this entity is at the head of a list of cells      Remove entity from list  Add unit back to grid at new cell |

**textureManager**

**Load**

|  |
| --- |
| If texture is not already in map      Load texture      Add texture to map  Else      Return texture from map |

**Pathfinding**

**findPath**

|  |
| --- |
| Add startpos to open list  Current pos is equal to start pos  while  openlist is not empty  currentPos is equal to node with lowest fcost  Remove currentNode from open list  Add currentnode to closed list  If current pos is equal to target          Return path  findNeighbours()  For each neighbour      If neighbour not walkable or its in closed list          Continue      If neighbour is not in open list list          neighbour g cost is equal to current node g cost plus distance between          neighbour h cost is equal to distance between it and target node          Neighbour parent is equal to current node          If neighbour not in open list              Add to open list          Else              Update neighbour in open list |

**backtrackPath**

|  |
| --- |
| Initialise path list  current node is equal to end node  While current node is not equal to start      Add current node to path      current node is equal to current node parent  reverse path  Return path |

**findNeighbours**

|  |
| --- |
| Initialise neighbour list  For i equal to -1 and less than or equal to 1      For j equal to -1 and less than or equal to 1          If x is equal to 0 and y is equal to 0              Continue          If nodePosx plus i is greater than or equal to 0 and less than gridSizeX              If nodePosY plus j is greater than or equal to 0 and less than gridSizeY  Add node to neighbour list |

### 1.8 – Test plan – Jack

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Unit Testing** | | | | |
| **Function** | **Input** | | **Expected Output** | |
| Phyiscs.Gravity |  | |  | |
| Phyiscs.AddForce |  | |  | |
| Phyiscs.AddImpulse |  | |  | |
| Vector3.Normalise |  | |  | |
| Vector3.ClampMagnitude |  | |  | |
| Vector3.CrossProduct |  | |  | |
| Vector3.Distance |  | |  | |
| Vector3.DotProduct |  | |  | |
| Vector3.Magnitude |  | |  | |
| Vector3.Angle |  | |  | |
| **Integration Testing** | | | | |
| **Module** | **Pass/Fail** | | **Action Required** | |
|  |  | |  | |
|  |  | |  | |
| **System Testing** | | | | |
| **Input** | **Expected Outcome** | **Actual Outcome** | **Pass / Fail** | **Action Required** |
| Move Forward | Player moves forward |  |  |  |
| Move Right | Player moves right |  |  |  |
| Move Left | Player Moves Left |  |  |  |
| Move Backwards | Player Moves Backwards |  |  |  |
| Move Camera Right | Camera rotates around the player to the right |  |  |  |
| Move Camera Left | Camera rotates around the player to the left |  |  |  |
| Jump | The player Jumps into the air |  |  |  |
| Gravity (Passive) | The player will fall if there is nothing under them |  |  |  |
| Player lands on enemy | The enemy will be killed and the player will bounce off them |  |  |  |
| Enemy collides with player (player is not landing on top of enemy)` | The player will take one damage |  |  |  |
| Player collides with a coin | The coin is removed from the game and the player gains 1 coin to their inventory, if the player is missing any hp, they will regain one hp |  |  |  |
| Player collides with a map peice | The map peice is removed from the game and the player gains that map peice to their inventory |  |  |  |
| the player collects all the map peices on the level | the treasure chest (goal) will spawn on the map |  |  |  |
| The player collides with the treasure chest | The level will be completed |  |  |  |
| The player reaches zero hp | The player will be killed and the level will restart |  |  |  |
|  |  |  |  |  |
| **Acceptance Testing** | | | | |
| **Feature** | **Functions** | **Function Outcome** | **Functions Required** | **Pass/Fail** |
| The player should be able to move around the environment in real time |  |  |  |  |
| Envrionment has static objects |  |  |  |  |
| Envrionment has dynamic objects |  |  |  |  |
| Player can move dynaic objects |  |  |  |  |
| Player can collect collectables |  |  |  |  |
| Objects should have textures |  |  |  |  |
| Player should collider with objects |  |  |  |  |
| Game must contain non-player entities |  |  |  |  |
| The non-player entities should collider with the world |  |  |  |  |
| The non-player entities should collider with the player |  |  |  |  |
| If the player and non-player entities collide, there should be an interaction |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| **Adhoc Testing** | | | | |
| **Function** | **Expected Outcome** | **Actual Outcome** | **Action Required** | **Pass/Fail** |

URL to spreadsheet version: <https://docs.google.com/spreadsheets/d/1OyKwXpvqMb8bcSE1zqaU1gUdI9nCfm1Lk5DcQwv0Ixk/edit?usp=sharing>

# References

Adobe, 2012. Object-oriented programming concepts: Inheritance. *Object-oriented programming concepts: Inheritance | Adobe Developer Connection*. Available at: https://www.adobe.com/devnet/actionscript/learning/oop-concepts/inheritance.html [Accessed November 1, 2018].

Arroway Textures, Arroway Textures. *What's the purpose of all the different map types? | Arroway Textures*. Available at: https://www.arroway-textures.ch/en/faq/whats-purpose-all-different-map-types [Accessed November 1, 2018].

cursorweb ed., 2018. 3D collision detection. *MDN Web Docs*. Available at: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D\_collision\_detection [Accessed November 1, 2018].

Gears for Breakfast, 2017. A Hat In Time. *A Hat In Time*. Available at: http://hatintime.com/ [Accessed November 1, 2018].

GitHub, Inc, 2018. GitHub. *GitHub*. Available at: https://github.com/ [Accessed November 1, 2018].

Moore, K. et al., Classes (OOP). *Brilliant Math & Science Wiki*. Available at: https://brilliant.org/wiki/classes-oop/ [Accessed November 1, 2018].

Nintendo Co., 1996. Super Mario 64.

Opengl-tutorial, *Tutorial 13 : Normal Mapping*. Available at: http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-13-normal-mapping/ [Accessed November 1, 2018].

rastertek, *UV example on a texture*, Available at: http://www.rastertek.com/pic0012.gif.

Satran, M., 2018. Work with shaders and shader resources. *Microsoft Docs*. Available at: https://docs.microsoft.com/en-us/windows/desktop/direct3dgetstarted/work-with-shaders-and-shader-resources [Accessed November 1, 2018].

Technopedia a, What is Rasterization? - Definition from Techopedia. *Techopedia.com*. Available at: https://www.techopedia.com/definition/13169/rasterization [Accessed November 1, 2018].

Technopedia b, What is an Abstract Class? - Definition from Techopedia. *Techopedia.com*. Available at: https://www.techopedia.com/definition/17408/abstract-class [Accessed November 1, 2018].

Thomas, N., 2018. Testing. *Lecture delivered to Mobile Applications Level 6*.

Thunderfist, 2012. Capsule-Capsule Collision in Games. *Thunderfist*. Available at: http://thunderfist-podium.blogspot.com/2012/02/capsule-capsule-collision-in-games.html [Accessed November 1, 2018].

Venture Beat, 2016. *Super Mario 64*, Available at: https://venturebeat.com/wp-content/uploads/2016/06/supermario64.png [Accessed November 1, 2018].

Mikeblome, Virtual Functions. *Microsoft Docs*. Available at: https://docs.microsoft.com/en-gb/cpp/cpp/virtual-functions?view=vs-2017 [Accessed November 1, 2018].