20th October 2017

**CGP600 Advanced Games Programming  
AE1 – Group Project**

John McGrath (Q10931775) James Moran (Q12494305)

CGP600 Advanced Games Programming  
AE1 – Group Project

Contents

Contents

[Individual Report (James Moran) 4](#_Toc496277303)

[Task Breakdown 4](#_Toc496277304)

[Game Object 4](#_Toc496277305)

[Hover-Tank 4](#_Toc496277306)

[Player Controlled Tank 4](#_Toc496277307)

[AI Controlled Tank 4](#_Toc496277308)

[Static Obstacles 4](#_Toc496277309)

[Moveable Obstacles 4](#_Toc496277310)

[Collectibles 4](#_Toc496277311)

[Heads-Up-Display (HUD) 5](#_Toc496277312)

[Lighting 5](#_Toc496277313)

[Collision Management System 5](#_Toc496277314)

[Critical Reflection and Discussion of Group Work 5](#_Toc496277315)

[Individual Report (John McGrath) 6](#_Toc496277316)

[Project Inception 6](#_Toc496277317)

[Team Coordination 6](#_Toc496277318)

[Version Control 6](#_Toc496277319)

[Game Definition 6](#_Toc496277320)

[Planning 6](#_Toc496277321)

[Work Breakdown Structure (WBS) 6](#_Toc496277322)

[Software Development Methodology (SDM) 6](#_Toc496277323)

[Analysis 6](#_Toc496277324)

[Task Allocation 6](#_Toc496277325)

[Design 6](#_Toc496277326)

[Framework 7](#_Toc496277327)

[Input Manager 7](#_Toc496277328)

[Asset Manager 7](#_Toc496277329)

[Scene Manager 7](#_Toc496277330)

[Menus 7](#_Toc496277331)

[Error Handling 7](#_Toc496277332)

[Reflection 8](#_Toc496277333)

[Conclusion 8](#_Toc496277334)

[Appendix A: Supporting Material 9](#_Toc496277335)

[Project Inception (WBS 1.1) 9](#_Toc496277336)

[Version control 9](#_Toc496277337)

[Coordination and Team Info 9](#_Toc496277338)

[Concept Sketches (John McGrath) 10](#_Toc496277339)

[Concept Map (John McGrath) 11](#_Toc496277340)

[Planning (WBS 1.2) 13](#_Toc496277341)

[Work Breakdown Structure (John McGrath) 13](#_Toc496277342)

[Analysis (WBS 1.3) 15](#_Toc496277343)

[Design (WBS 1.4) 15](#_Toc496277344)

[Testing (WBS 1.6) 15](#_Toc496277345)

[Appendix B: Game Definition Document 16](#_Toc496277346)

[Intro 16](#_Toc496277347)

[Criteria 16](#_Toc496277348)

[Game Definition 16](#_Toc496277349)

[Environment 17](#_Toc496277350)

[Obstacles 17](#_Toc496277351)

[Collectables 17](#_Toc496277352)

[Vehicles 18](#_Toc496277353)

[Controls 18](#_Toc496277354)

[Viewport 18](#_Toc496277355)

[User Interface 18](#_Toc496277356)

[Appendix C: Design by James Moran 19](#_Toc496277357)

[Class Diagrams 19](#_Toc496277358)

[Figure 1: High Level Class Diagram 1.0.5 (James Moran) 19](#_Toc496277359)

[Class Method Pseudocode 19](#_Toc496277360)

[GameObject (James Moran) 19](#_Toc496277361)

[MoveableObject (James Moran) 21](#_Toc496277362)

[EntityControlledObject (James Moran) 22](#_Toc496277363)

[PlayerEntity (James Moran) 22](#_Toc496277364)

[AIEntity (James Moran) 23](#_Toc496277365)

[PlayerHUD (James Moran) 24](#_Toc496277366)

[CollisionManager (James Moran) 24](#_Toc496277367)

[AILogicSystem (James Moran) 24](#_Toc496277368)

[Use-Case Diagrams (James Moran) 25](#_Toc496277369)

[Figure 2 (James Moran): GameObject Use-Case diagram. 25](#_Toc496277370)

[Figure 3 (James Moran): Hover-Tank Use-Case diagram. 25](#_Toc496277371)

[Figure 4 (James Moran): Player Controlled Tank Use-Case diagram. 27](#_Toc496277372)

[Figure 5 (James Moran): AI Controlled Tank Use-Case diagram. 27](#_Toc496277373)

[Figure 6 (James Moran): Static Obstacles Use-Case diagram. 28](#_Toc496277374)

[Figure 7 (James Moran): Moveable Obstacles Use-Case diagram. 28](#_Toc496277375)

[Figure 8 (James Moran): Collectables Use-Case diagram. 29](#_Toc496277376)

[Figure 9 (James Moran): Heads-Up-Display (HUD) Use-Case diagram. 29](#_Toc496277377)

[Figure 10 (James Moran): Lighting Use-Case diagram. 30](#_Toc496277378)

[Figure 11 (James Moran): Collision Management System Use-Case diagram. 30](#_Toc496277379)

[Appendix D: Design by John McGrath 31](#_Toc496277380)

[Framework 32](#_Toc496277381)

[Pseudocode 32](#_Toc496277382)

[Input Manager 33](#_Toc496277383)

[Asset Manager 34](#_Toc496277384)

[Scene Manager 35](#_Toc496277385)

[Menus 36](#_Toc496277386)

[Appendix E: References 37](#_Toc496277387)

# Individual Report (James Moran)

## Task Breakdown

### Game Object

This class is for all objects within a level (such as any hover-tanks, static obstacles etc.). Encapsulating similar elements for other classes to build on.

This class is to be used as a base (abstract) class, for all entities within a scene. This class will hence, have basic properties that can be applied to all sub-class instances (such as their current location, rotation and scale), as well as manipulation of these properties (e.g. by moving, rotating or transforming the game object). The scene class (as detailed by John), will have a collection of all game objects within it, so that one can find any particular game object, no matter what sub-class it belongs to. (See Figure 2 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### Hover-Tank

The Hover-tank class inherits from the GameObject class and is in of itself, the super class for all controllable entities within the game (whether they are controlled by the Player via external input, or by an AI system, with internal input from the game).

Any instance of this class can ‘float’ around the level, causing interactable obstacles to move or disperse, whilst static obstacles, will stop the tank and block movement into the area of such an object. (See Figure 3 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### Player Controlled Tank

The Player controlled tank is controlled by the Player, via external input (from a keyboard and mouse, or other control method such as a gamepad). This has the same functionality as the Hover-Tank base class, with the addition of being able to collect Energy Capsules (Collectibles), within the level area. If the Player collects these before time runs out, they will have completed the primary (and sole) objective for the level. The Player will have to attempt to avoid any static obstacles that get in their way, as well as any AI Controlled Tanks, that will actively try to harass the Player (causing them to lose time, that could be spent picking up the Energy Capsules in the level). (See Figure 4 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### AI Controlled Tank

The AI Controlled Tank is controlled by an AI System, granting input to this type of tank, internally. This sub-class also has the same functionality as the Hover-Tank base class, but as per the AI System this sub-class has, instances of this class will attempt to collide with the Player’s Hover-tank, interrupting them and wasting their time. The AI System of this class will have instances that simply move to the Player in a straight line, attempting to collide with them so as to waste the Player’s time. (See Figure 5 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### Static Obstacles

These are obstacles in the level, which are not movable by both the Player and any AI Controlled Tanks, blocking movement into their bounds, for both types of tanks. Both types of tanks will also have no interaction with these obstacles, other than colliding with them, to have their movement blocked by them. (See Figure 6 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### Moveable Obstacles

These are also obstacles in the level, which are moveable by both types of tanks (receiving translation to their position, or by getting scattered). These types of obstacles will also hinder movement to an extent (not by much if they are scatter-able, otherwise, they will impede movement appropriately), depending on the Hover-tank’s velocity. (See Figure 7 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### Collectibles

These are not obstacles (as they will not impede movement) and only interact with the Player Controlled Tank. These are known as Energy Capsules, with the primary objective for the Player, being that of collecting them. AI Controlled Tanks will have no interaction with them. (See Figure 8 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### Heads-Up-Display (HUD)

This will not be a 3D-HUD (based within objects in the game world), but a 2D-HUD superimposed upon the Player’s viewport. This will show the Player how much time they have left, as well as how many Energy Capsules they have collected and how many Energy Capsules remain in the level, to be collected. This will be reset for every level that the Player is loaded into, as per the number of Energy Capsules in that particular level. (See Figure 9 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### Lighting

Lighting in the game will either exist on a per-object basis, for each GameObject. Or as a global ‘sky-light’ granting lighting to all GameObjects in a scene. (See Figure 10 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

### Collision Management System

This is for handling collisions for each GameObject in the scene, that is to collide with or receive collision from other appropriate GameObjects (Any Hover-tanks, Static Obstacles and Moveable Obstacles), will be handled by Axis-Aligned Bounding Boxes (AABB). (See Figure 11 of Appendix C1: Design by James Moran, for a Use-Case diagram of this class)

Please see Figure 1 of Appendix C1: Design by James Moran, for a complete class diagram, for all of the user stories discussed here.

## Critical Reflection and Discussion of Group Work

(Section not attempted)

# Individual Report (John McGrath)

This report documents the design process for a 3D game, as specified by the Advanced Games Programming AE1 assessment brief. As this was a group project, I teamed up with class mate James Moran. Supporting material can be found in the appendices.

## Project Inception

After analysis of the assessment brief, it was felt that a simple open map driving game would easily fulfil all minimum requirements. In the interest of keeping vehicle models simple, the concept of ‘hover-tanks’ was suggested, as hover-tank models don’t need any moving parts to float over the map.

### Team Coordination

The Trello application had been introduced to us in a previous unit and had proven a useful tool for organising a project, so we opted to use it for this assessment (Trello, 2017). The board was divided into ‘TODO’, ‘In progress’, and ‘Done’ lists, with tasks moving from list to list as appropriate. Additional lists held miscellaneous information, such as contact details (appendix A).

### Version Control

GitHub version control was considered for centralising our documents, but it was felt would be simpler and more convenient to use Google Docs for non-code documents (Google).

### Game Definition

The concept was refined further in a ‘Game Definition’ document (appendix B), which included a table of assessment requirements and justifications. This ensured that all minimum requirements would be met and highlighted additional features that might earn extra credit.

Some concept sketches were drawn to aid visualisation of the game (appendix A).

A concept map covering the basic assessment requirements was created, and then modified according to the game definition.

A high-level class diagram was also drawn up.

These sketches, maps, and diagrams helped to further refine the game definition.

## Planning

The Engineering Software Systems (ESS) unit provided us with some direction to help plan and push the project onwards.

### Work Breakdown Structure (WBS)

The project was broken down into six phases of development: Inception, Planning, Analysis, Design, Implementation, and testing. These were recorded on a WBS dictionary table (appendix A), as demonstrated by M. Piscopo (2017). Each phase was populated with tasks as suggested by the ESS unit. The table was updated throughout the planning and analysis phases.

### Software Development Methodology (SDM)

A brief discussion about choices of SDM led us to agree that an iterative approach (Agile) would suit development of our project better than the waterfall model, as the game could be built in layers, starting with a basic framework and adding to it for each iteration.

## Analysis

With a reasonably solid game definition in place, we set about attempting to break the project into smaller tasks. This was accomplished mostly through analysis of the concept map and game definition, and task were added to the design phase section of the WBS.

### Task Allocation

After a brief discussion, it was clear that James had a preference for the game mechanics, while I had already worked on the core framework on previous projects. Allocation of task was a simple division between game mechanics and everything else. On the WBS this divided into an approximately equal number of tasks each, and the division kept most connected systems together.

## Design

With tasks assigned, we each set about the design process. My assigned tasks began with a class relationship diagram, showing how the framework and core components fitted together. These were designed with reusability in mind, intended to be useful as a generic base for applications to build on. Refer to appendix D for supporting material for these design tasks.

### Framework

The Windows Application and D3D Application classes encapsulate Windows and Direct3D specific functionality respectively, thereby decoupling game code from the complexity of the underlying code, and separating Windows code from DirectX.  
This allows the Windows Application class to be reused, as it is designed to be used as a base class for wrapper classes using different APIs, such as DirectX, Vulkan, SDL, or just as it is, plain Win32.

### Input Manager

The Input Manager class records user input sent from the Windows message handler via the ProcessMessage() method.  
Only keys already present in the map container need be recorded, other key presses will be ignored. Keys can be added to the map by requesting the key status via the IsKeyDown() method.

### Asset Manager

The Asset Manager class handles the loading and unloading of assets. Each type of asset (images, fonts, sounds, videos, etc) has its own class, which are in turn derived from an abstract base Asset class. These assets classes can only be requested via a templatised method in the Asset Manager class. Successfully created assets are stored by the manager in a map container, where they can be retrieved or destroyed.

Asset data may be loaded and unloaded at any time, allowing memory to be freed without destroying the asset.

### Scene Manager

The application will utilise a modified state pattern (Gamma et al, 1995) to manage ‘scenes’, where each scene represents a separate area of the application (intro screen, menu, a game level, etc).

The SceneManager class contains a collection of objects that extend the abstract Scene base class, which are updated each frame using the SceneManager Update() method.

The Update() method returns true if there is at least one Scene present, otherwise it returns false, giving a convenient way for the main loop to detect when to quit the application.

All Scenes are updated, from oldest to newest, allowing for paused Scenes to continue animating, or to display a spoiler.

The AddScene() method adds a new Scene as the current scene, pausing the previous Scene (if present).

The ChangeScene() method adds a new Scene as the current scene and removes the previous Scene (if present).

Any time a Scene is added, the SceneManager will call its OnStart() method.  
Any time a Scene is removed, the SceneManager will call its OnExit() method, and Resume() the previous State (if present).

### Menus

The Menu class is generic, allowing it to be reused for many purposes. In this case we have three menus: The front-end menu, an options menu, and an in-game menu.

Once instantiated, a Menu must be initialised with its location and size (Rectangle), a background texture, button size, a sprite sheet with three button images for on/off/hover states, and text to display on the buttons.

### Error Handling

This application will throw exceptions (std::runtime\_error) as they provide good feedback for debugging, allow for graceful exits, and they make the code relatively tidy. Consequently, most functions and methods have a void return type.

## Reflection

Most of the design work I have done for this project has been to reuse code and techniques used in previous assessments and personal projects, so new design work has been minimal.

I found many of the software engineering diagrams to be confusing and unhelpful, particularly when using existing concepts instead of designing from scratch.

The concept map proved to be useful: its visual representation and expansive nature make the project details more obvious, allowing design gaps to be filled. A good concept map quite easily translates into a class relationship diagram, and from there into more detailed individual class diagrams.

Time management was once again an issue, with a few holes in the design and an incomplete report resulting from leaving things till the last minute.

The Trello application has obvious benefits and useful features that largely went unused or were abandoned when time became short.

### Conclusion

Better time management will improve results. Given a preference for iterative development methodologies, I will attempt to employ Agile techniques to future projects. Making full use of tools such as Trello and Microsoft Project may help with this.

# Appendix A: Supporting Material

This appendix consists of notes, images, diagrams, and other artefacts produced during for the CGP600 AE1 assignment. All work listed here is a joint effort, except where noted.

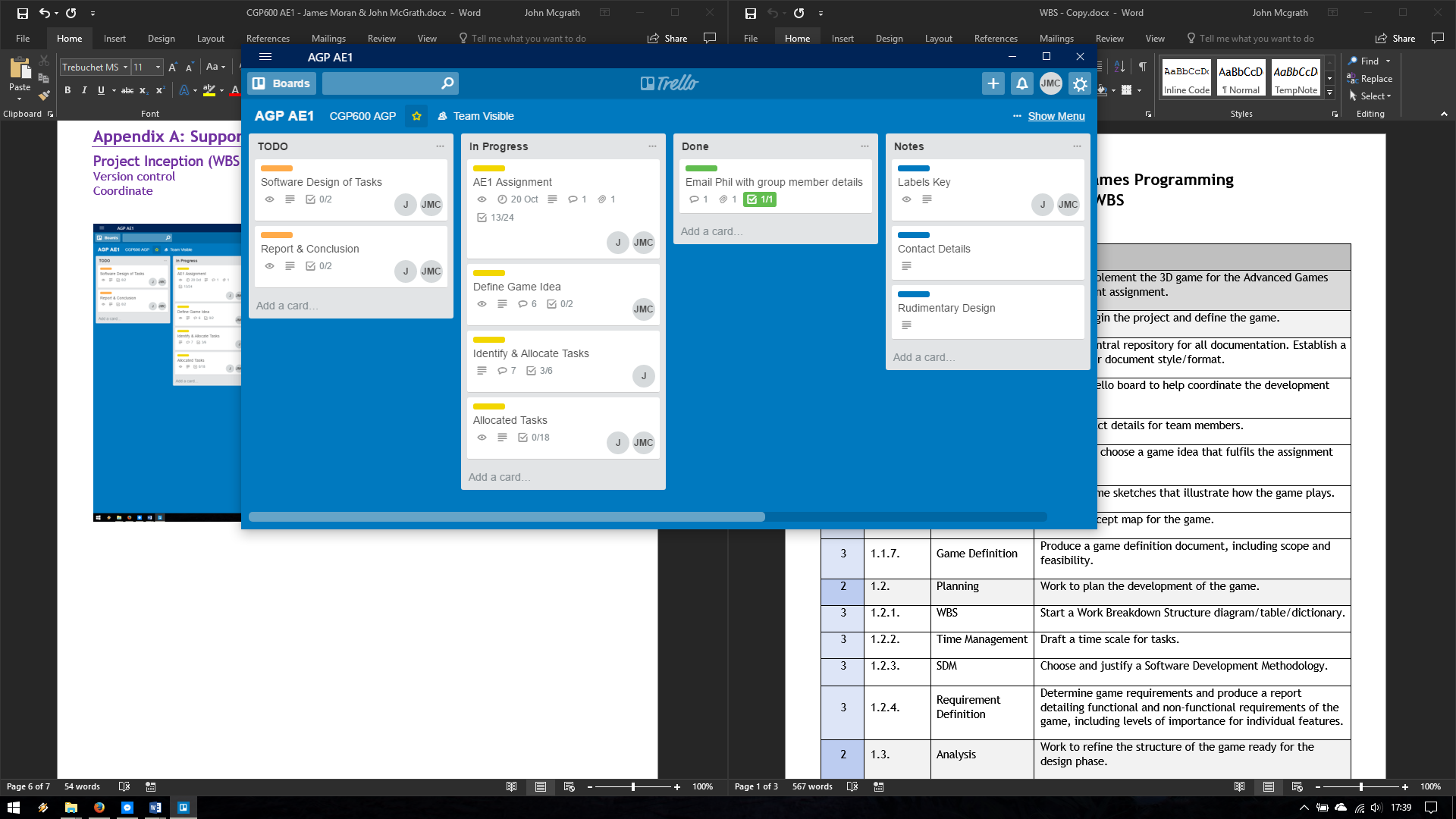
## Project Inception (WBS 1.1)

### Version control

To keep our work together, we opted to use Google Docs as this works well for sharing and working on live documents. For the second assignment we would use GitHub version control to manage code.

### Coordination and Team Info

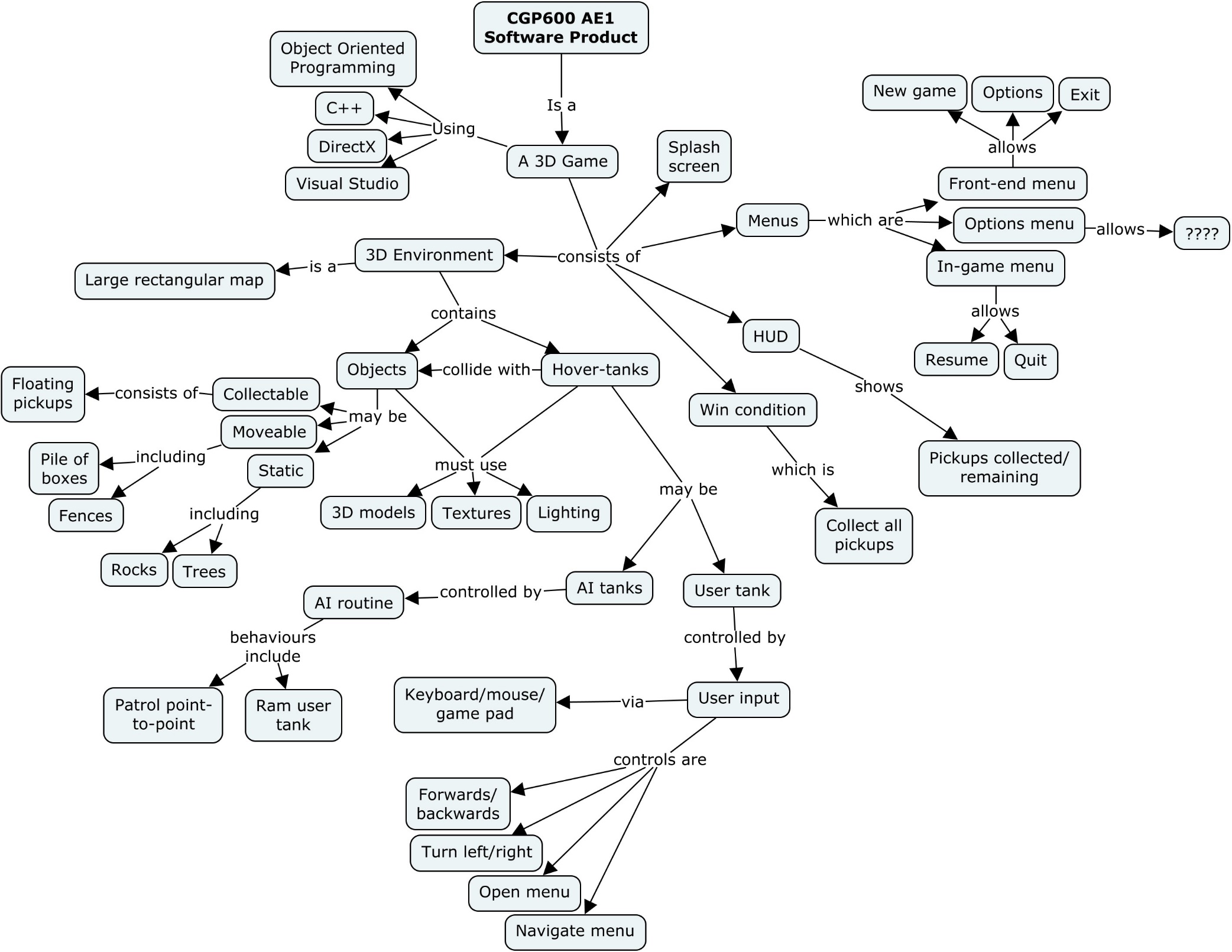
To coordinate our team, we opted use a Trello board due to its useful features. We kept contact details for team members here.



### Concept Sketches (John McGrath)

|  |
| --- |
| C:\Users\drayc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\concept02.png |
| C:\Users\drayc\AppData\Local\Microsoft\Windows\INetCache\Content.Word\concept01.png |

### Concept Map (John McGrath)



## Planning (WBS 1.2)

WBS, SDM, Time, Requirements

### Work Breakdown Structure (John McGrath)

|  |  |  |  |
| --- | --- | --- | --- |
| **Level** | **WBS Code** | **WBS Label** | **Definition** |
| 1 | 1. | CGP600 3D Game | Work to Implement the 3D game for the Advanced Games Development assignment. |
| 2 | 1.1 | Inception | Work to begin the project and define the game. |
| 3 | 1.1.1. | Version Control | Set up a central repository for all documentation. Establish a standard for document style/format. |
| 3 | 1.1.2. | Coordinate | Set up a Trello board to help coordinate the development team. |
| 3 | 1.1.3. | Team Info | Store contact details for team members. |
| 3 | 1.1.4 | Brainstorm | Discuss and choose a game idea that fulfils the assignment criteria. |
| 3 | 1.1.5. | Concept Sketches | Produce some sketches that illustrate how the game plays. |
| 3 | 1.1.6. | Concept Map | Make a concept map for the game. |
| 3 | 1.1.7. | Game Definition | Produce a game definition document, including scope and feasibility. |
| 2 | 1.2. | Planning | Work to plan the development of the game. |
| 3 | 1.2.1. | WBS | Start a Work Breakdown Structure diagram/table/dictionary. |
| 3 | 1.2.2. | Time Management | Draft a time scale for tasks. |
| 3 | 1.2.3. | SDM | Choose and justify a Software Development Methodology. |
| 3 | 1.2.4. | Requirement Definition | Determine game requirements and produce a report detailing functional and non-functional requirements of the game, including levels of importance for individual features. |
| 2 | 1.3. | Analysis | Work to refine the structure of the game ready for the design phase. |
| 3 | 1.3.1. | Domain model | Produce a domain model diagram, showing hypothetical classes in the game and how they interact. |
| 3 | 1.3.2. | Use-cases | Produce use-case diagrams and corresponding use-case tables detailing how the user interacts with the game. |
| 3 | 1.3.3. | Robustness | Produce a robustness diagram that encapsulates interaction between the user and the game, as detailed by use-cases. |
| 3 | 1.3.4. | Sequence | Produce a sequence diagram that shows step-by-step game execution. |
| 3 | 1.3.5. | Classes | Produce a class diagram that details classes and their relationships, as identified from domain, use-case, and robustness diagrams. |
| 3 | 1.3.6. | Decomposition | Produce a decomposition diagram that details game processes. |
| 2 | 1.4. | Design | Work to determine what will go into the code prior to the implementation phase. |
| 3 | 1.4.1. | Framework | Windows, DirectX, and game start up and shutdown, and main loop. |
| 3 | 1.4.2. | Input Manager | Central input management. Start with basic Windows input events, explore DirectInput options. |
| 3 | 1.4.3. | Resource Manager | Central resource management (memory & assets). |
| 3 | 1.4.4. | Scene Manager | Central FSM for transition between game scenes. |
| 3 | 1.4.5. | Splash Scene | Front-end splash scene. |
| 3 | 1.4.6. | Front-end Scene | Front-end main menu and sub menu scene. |
| 3 | 1.4.7. | Game Scene | Core game logic. |
| 3 | 1.4.8. | Menus | Reusable menu system, for all front-end and in-game menus. |
| 3 | 1.4.9. | Game Objects | Abstract base class for all objects. |
| 3 | 1.4.10. | Hover-tank | Extends game object class for the Hover-tank vehicle. |
| 3 | 1.4.11. | Player Tank Control | Extends the Hover-tank class with a user control method. |
| 3 | 1.4.12. | AI Tank Control | Extends the Hover-tank class with an AI control method. |
| 3 | 1.4.13. | Static Obstacles | Solid game objects within the environment that do **not** move when Hover-tanks collide with them. |
| 3 | 1.4.14. | Movable Obstacles | Solid game objects within the environment that **do** move when Hover-tanks will collide with then. |
| 3 | 1.4.15. | Collectables | Non-solid game objects within the environment that de-spawn only when user Hover-tanks collide with then. |
| 3 | 1.4.16. | HUD | Screen overlay that shows collected/remaining collectables in the environment. |
| 2 | 1.5 | Implement | Work to write the actual code. |
| 2 | 1.6 | Test | Work to test the game. |

## Analysis (WBS 1.3)

Domain, use-case, robustness, sequence, decomposition

## Design (WBS 1.4)

Framework, Input, resource, scene, splash, front-end, game, menus, game objects, Hover-tank, human control, AI control, static obstacles, movable obstacles, collectables, HUD.

## Testing (WBS 1.6)

Test plans

# Appendix B: Game Definition Document

## Intro

For the Advanced Games Programming AE1 assignment, James Moran and myself are to produce the software design for a simple 3D game. We have decided to design an open world driving game that contains elements that will fulfil all the basic assignment criteria, and allows for additional enhancements for extra credit. This document serves to define the game concept in detail, prior to the main design process.

## Criteria

As per the assessment brief:

* A real-time, interactive, 3D game (C++, OOP, DirectX 11, Visual Studio
* Large world environment that the user can move around
* Static obstacle (model with textures)
* Movable obstacle (model with textures)
* Collectables (model with textures)
* Collision with obstacles and collectables
* Lighting
* Non-player AI entities (model with textures) that move around the environment
* Collision between entities and obstacles
* Player/entity interaction

*Additional grades are achieved through enhancements to the basic game requirements (e.g., innovation and robustness).*

*Examples of such enhancements include:*

* *extending the game to use advanced features, such as, procedural content, physics, or AI;*
* *using advanced DirectX/Windows/Shader techniques;*
* *using more sophisticated techniques for core game features, such as, managing the scene and collision detection;*
* *optimising game performance;*
* *good object-orientated design, with the game and its constituent objects extensible and reusable.*

*These are just a small set of possibilities, look at what other games do and use your imagination to come up with others. Some of these enhancements will require additional research of 3D, object-oriented and game programming techniques not explicitly covered in the unit.*

## Game Definition

The user controls a hover-tank vehicle in a large open environment, with a chase-cam view. The objective of the game is to collect a predefined number of pick-up items that are scattered around the map. Several AI controlled hover-tanks patrol areas of the map that will attempt to ram the user vehicle when it comes with their visual range. Static objects are placed across the map that vehicles will collide with, potentially preventing access to certain areas.

|  |  |
| --- | --- |
| Requirement | Hover-tank Game |
| 3D game | Chase-cam view of a vehicle in a 3D environment |
| Large environment | Free roam over an open map |
| User can move around the environment | User flies a hover-tank around the map |
| Obstacle (static) | Map features; trees, rocks, buildings, rivers – impassable by hover-tanks. |
| Obstacle (movable) | Automated objects; drawbridge Smaller map features; fences, boxes – destroyed or scattered when rammed by hover-tanks |
| Collectable | Energy capsules; de-spawn when user hover-tank collides, find them all to complete the level |
| Lighting | Environment lighting, model based lights |
| AI entities | Enemy hover-tanks patrol, and chase/ram user when tank passes within range |
| Entity collision | Enemy has same collision rules as user tank, except it will not pick up collectables |
| User/AI interaction | Player proximity triggers enemies |

### Environment

A large map that defines the limits of vehicle movement and object placement, including player spawn point.

Note: Can start with a flat map, add variable height at a later stage.

Ideas: add map-to-map transitions.

### Obstacles

Obstacles are reusable objects that can be repeated any number of times around a level.

Static: immovable obstacles such as trees, large rocks, buildings, and rivers block vehicle movement.  
Movable: Smaller features, such as fences, and boxes, which do not block vehicle movement, but are scattered or destroyed on contact.

### Collectables

Static objects that de-spawn on contact with only the user vehicle. Collected versus uncollected objects are noted on the user HUD. Collection of all objects triggers the game level win condition.

### Vehicles

The hover-tank is a triangular shaped vehicle that floats just above the environment floor. It can turn, and thrust either forward or backward.

Both user and AI use the same hover-tank model, although the user tank has a unique texture to distinguish it.

Collisions may push tanks sideways, supporting the hovering appearance.

Ideas: add tank abilities as upgrades (jump/strafe/weapons/defenses/phase-shift/HUD), add additional vehicle models. Improve AI path-finding and ramming routine. Implement health/damage system.

### Controls

Use keyboard arrows and/or WASD to control the user tank.

Ideas: add analog gamepad support.

### Viewport

The camera is above, behind, and attached to the user tank, always turning to face a point above and ahead the line of the velocity of the tank.

Ideas: allow camera rotation, elevation, and zoom controls.

### User Interface

Show number of collectables remaining.

Ideas: HUD map showing collectables and enemies. Display menu, FPS, health/damage/shield, ammo...

# Appendix C: Design by James Moran

## Class Diagrams

This contains all the classes that have been identified in the Task Breakdown section:

### Figure 1: High Level Class Diagram 1.0.5 (James Moran)

## Class Method Pseudocode

This appendix is for the methods/functions of each class noted above (Wikipedia, 2017):

### GameObject (James Moran)

**Method** GameObject() **is** Class Constructor

**Input:** *None.*

**Output:** *None.*

*Initialise a particular instance of this class*

**End Method**

**function** GetLocation() **is** a ‘getter’ function

**Input:** *None.*

**Output:** *Vector3 Location: This GameObject’s current location.*

*Return this GameObject’s Location member property*

**Return** Location

**End function**

**function** GetRotation() **is** a ‘getter’ function

**Input:** *None.*

**Output:** *Vector3 Rotation: This GameObject’s current rotation.*

*Return this GameObject’s Rotation member property*

**Return** Rotation

**End function**

**function** GetScale() **is** a ‘getter’ function

**Input:** *None.*

**Output:** *Vector3 Scale: This GameObject’s current scale.*

*Return this GameObject’s Scale member property*

**Return** Scale

**End function**

**Method** MoveObject() **is**

**Input:** *None.*

**Output:** *None*.

*Handle translation of this object*

**End Method**

**Method** RotateObject() **is**

**Input:** *None.*

**Output:** *None*.

*Handle rotation of this object*

**End Method**

**Method** ScaleObject() **is**

**Input:** *None.*

**Output:** *None*.

*Handle scaling of this object*

**End Method**

### MoveableObject (James Moran)

**Method** MoveableObject() **is** Class Constructor

**Input:** *None.*

**Output:** *None.*

*Initialise a particular instance of this class*

**End Method**

**Method** OnCollisionBegin() **is** a class event handler

**Input:** *None.*

**Output:** *None*.

*Handle the event of this object beginning collision with another object*

**End Method**

**Method** OnCollisionEnd() **is** a class event handler

**Input:** *None.*

**Output:** *None*.

*Handle the event of this object Ending collision with another object*

**End Method**

### EntityControlledObject (James Moran)

**Method** EntityControlledObject() **is** Class Constructor

**Input:** *None.*

**Output:** *None.*

I*nitialise a particular instance of this class*

**End Method**

**function** GetAIControlled **is** a ‘getter’ function

**Input:** *None.*

**Output:** *bool AIControlled: This flag indicates whether this object is controlled by the Player, or by an AI system.*

*Return this object’s AIControlled member property*

**Return** AIControlled

**End function**

### PlayerEntity (James Moran)

**Method** PlayerEntity() **is** Class Constructor

**Input:** *None.*

**Output:** *None.*

I*nitialise a particular instance of this class*

**End Method**

**Function** GetCollectiblesPickedUp() **is** a ‘getter’ function

**Input:** *None.*

**Output:** *int CollectiblesPickedUp: The number of collectibles that the Player has picked-up on this current level.*

**Return:** CollectiblesPickedUp

**End function**

**Method** InitiateControlSystem() **is**

**Input:** *None.*

**Output:** *None.*

*Initialise the control system of the input Method, given the control system the Player is using*

**End Method**

### AIEntity (James Moran)

**Method** AIEntity() **is** Class Constructor

**Input:** *None.*

**Output:** *None.*

I*nitialise a particular instance of this class*

**End method**

**Method** InitiateAILogicSystem **is**

**Input:** *None.*

**Output:** *None.*

*Initialise the logic system (FSM, Behaviour Tree etc.) that this AIEntity utilises*

**End method**

### PlayerHUD (James Moran)

**Method** PlayerHUD() **is** Class Constructor

**Input:** *None.*

**Output:** *None.*

I*nitialise a particular instance of this class*

**End method**

### CollisionManager (James Moran)

**Method** CollisionManager() **is** Class Constructor

**Input:** *None.*

**Output:** *None.*

I*nitialise a particular instance of this class*

**End method**

### AILogicSystem (James Moran)

**Method** AILogicSystem() **is** Class Constructor

**Input:** *None.*

**Output:** *None.*

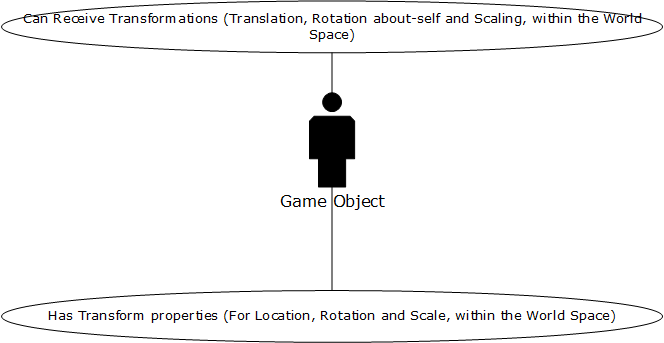
I*nitialise a particular instance of this class*

**End method**

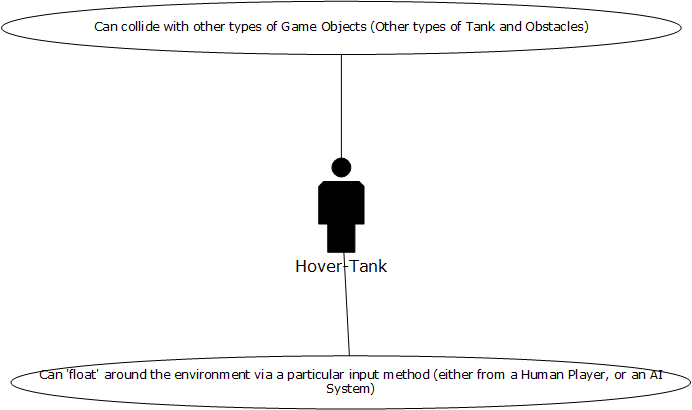
## Use-Case Diagrams (James Moran)

This appendix shows the Use-Case Diagrams for all the classes identified in the descriptions for each feature (User-Story):

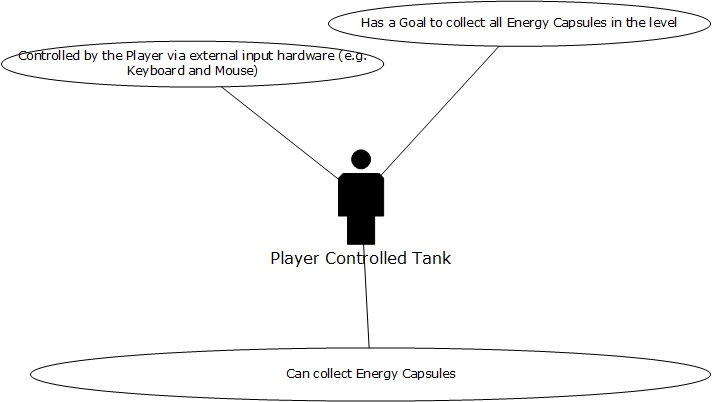
### Figure 2 (James Moran): GameObject Use-Case diagram.



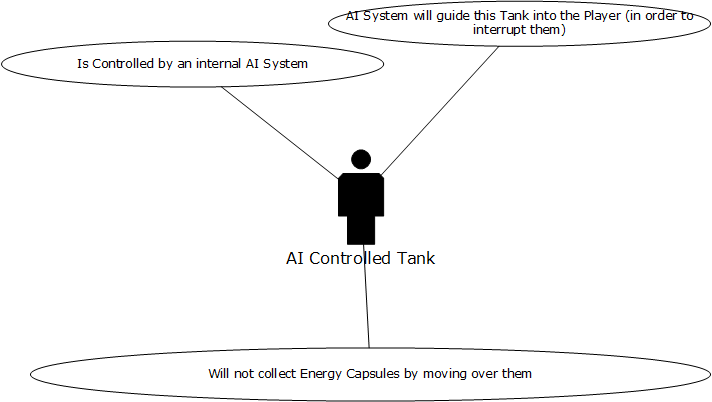
### Figure 3 (James Moran): Hover-Tank Use-Case diagram.



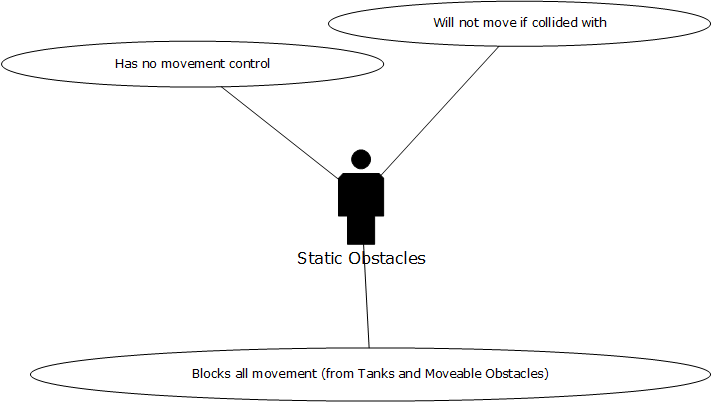
### Figure 4 (James Moran): Player Controlled Tank Use-Case diagram.



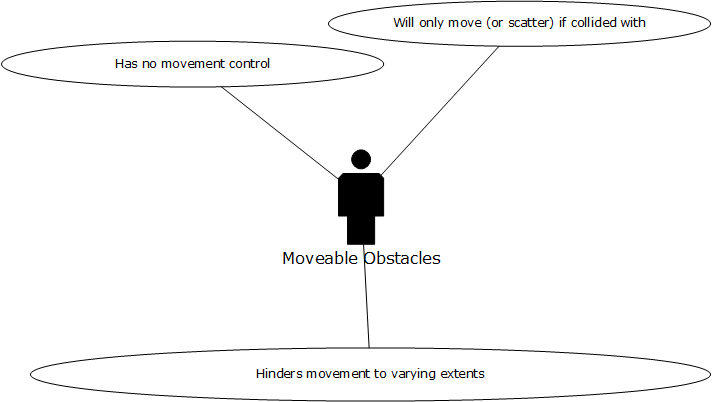
### Figure 5 (James Moran): AI Controlled Tank Use-Case diagram.



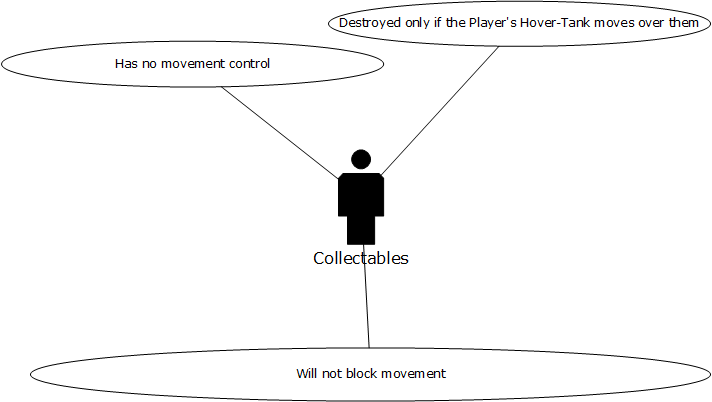
### Figure 6 (James Moran): Static Obstacles Use-Case diagram.



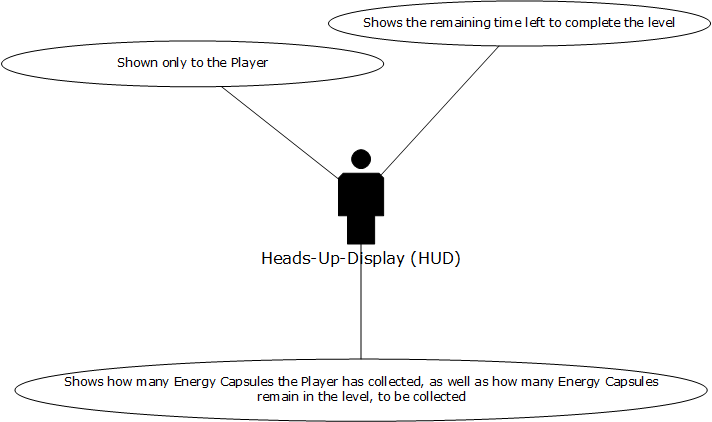
### Figure 7 (James Moran): Moveable Obstacles Use-Case diagram.



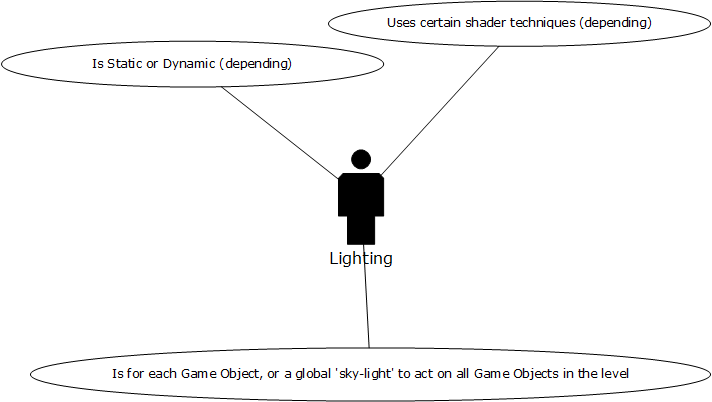
### Figure 8 (James Moran): Collectables Use-Case diagram.



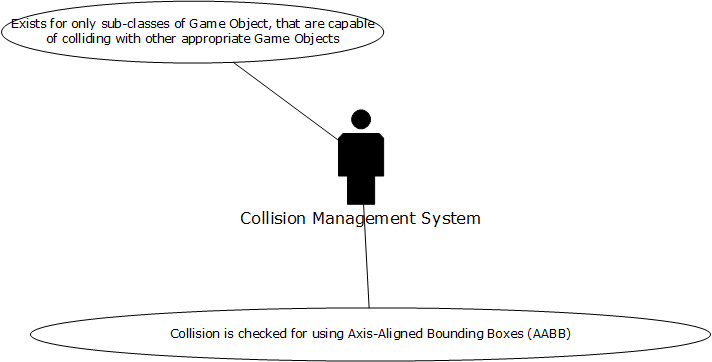
### Figure 9 (James Moran): Heads-Up-Display (HUD) Use-Case diagram.



### Figure 10 (James Moran): Lighting Use-Case diagram.



### Figure 11 (James Moran): Collision Management System Use-Case diagram.



# Appendix D: Design by John McGrath

The aim of this design is to make use of reusable and extensible code.



# Framework

## Pseudocode

1. Start application
2. Initialise Game:
   1. Initialise Direct3D:
      1. Initialise Windows:
         1. Register window class
         2. Create a window
      2. Enumerate adapters
      3. Create device & swap chain
3. Run application
4. Shut down application:
   1. Clean up Game objects
   2. Shut down Direct3D:
      1. Destroy D3D objects
      2. Shut down Windows:
         1. Close the window
         2. Unregister window class

# Input Manager



# Asset Manager

Assets such as images, sounds, music, video, etc, are represented here by Asset objects.



# Scene Manager

The application will utilise a modified state pattern (Go4) to manage ‘scenes’, where each scene represents a separate area of the application (intro screen, menu, a game level, etc).

The SceneManager class contains a collection of objects that extend the abstract Scene base class, which are updated each frame using the SceneManager Update() method.

The Update() method returns true if there is at least one Scene present, otherwise it returns false, giving a convenient way for the main loop to detect when to quit the application.

All Scenes are updated, from oldest to newest, allowing for paused Scenes to continue animating, or to display a spoiler.

The AddScene() method adds a new Scene as the current scene, pausing the previous Scene (if present).

The ChangeScene() method adds a new Scene as the current scene and removes the previous Scene (if present).

Any time a Scene is added, the SceneManager will call its OnStart() method.  
Any time a Scene is removed, the SceneManager will call its OnExit() method, and Resume() the previous State (if present).

Example:

1. The user starts the application, which instantiates a new SceneManager.
2. A new LevelOne object (derived from Scene) is instantiated and passed to the SceneManager::AddScene() method, which in turn calls LevelOne::OnStart() method, loading the relevant game data as detailed in the LevelOne class.
3. The application enters the main loop, calling the SceneManager::Update() method on each pass, which in turn calls LevelOne::Update() method.
4. During an update, the user presses the escape key to open the in-game menu, which is captured in LevelOne::Update() method. A new InGameMenu object (also derived from Scene) is instantiated and added via SceneManager::AddScene(), which causes InGameMenu to OnStart() and LevelOne to Pause().
5. The user closes the menu, which removes the InGameMenu object, leaving the SceneManager to resume the previous LevelOne Scene.  
   -or-  
   the user opts to quit the game, causing removal of both the InGameMenu and LevelOne Scenes.

## Menus



# Appendix E: References

Gamma, E., Helm, R., Johnson, R. & Vlissides, J., 1995. *Design Patterns: elements of reusable object-oriented software.* 30th ed. Indianapolis(Indiana): Pearson Education Corporate Sales Division.

GOOGLE INC., 2017, *About Google Docs* [viewed 18th October 2017]. Available from: <https://www.google.co.uk/docs/about/>

PISCOPO, M., 2017, *Work Breakdown Structure (WBS)* [viewed 12th October 2017]. Available from: <http://www.projectmanagementdocs.com/project-planning-templates/work-breakdown-structure-wbs.html#axzz4X432ZBbH>

TRELLO INC., 2017, *About Trello* [viewed 18th October 2017]. Available from: <https://trello.com/about>

Wikipedia, 2017, Wikipedia: WikiProject Computer science/Manual of Style [Viewed on the 14/10/2017]. Available from: [https://en.wikipedia.org/wiki/Wikipedia:WikiProject\_Computer\_science/Manual\_of\_style#General\_guidelines\_for\_writing\_pseudocode](https://en.wikipedia.org/wiki/Wikipedia:WikiProject_Computer_science/Manual_of_style" \l "General_guidelines_for_writing_pseudocode)