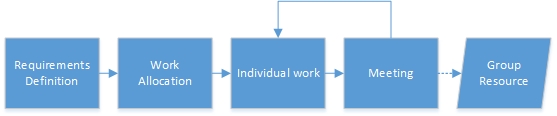
**CGP600 AE1 – Group Project**

Konstantin Schaper

**Figure 1:** Course of Group Work

Computer Games (Software Development), Southampton Solent University



**Abstract**

This document goes into detail about the design process of the video game ‘3DTanks’ (Working title) and is meant to be a guideline for developers to implement the game concept below.

**CR Categories:** System architectures, Documentation

Keywords: game engine, component based, documentation, software design

1. Game Concept

The game is a very basic ‘Third Person Arena Shooter’. The player controls a small tank which can fire its canon by pressing a spacebar and move around using the ‘WASD’ keys. The camera is fixed behind the player and rotates/moves with him. The angle of view on the character is static, both in horizontal and vertical direction. There might be several levels in the future but the demo level consists of basic shapes and blocks so the players can move around and hide from enemy attacks. Pickups spawn in defined intervals giving players either health back when collected or enhancing their weapon for some time. There is a game timer ticking down. When it reaches zero, the player with the most take-downs wins.

1. Group Work

The group consisted of three moderately practiced programmers of which one already had a background in DirectX development. Initially, requirements had to be defined. Resultantly, the group brainstormed what the game should look like and how the single features could be defined in a short sentence. Those requirements were then divided into Must-have, Should-be and Nice-to-have (See appendix A). All dependencies between requirements had been noted afterwards.

When the requirement definition had been complete, each group member was assigned a number of them to research upon. The allocated tasks were grouped by category so everyone could work independently. From this point onwards the team met regularly to share and discuss the results of their research. Each meeting produced a protocol and a set of team resources everybody could use for their final design document. A sketch of the described sequence is displayed in Figure 1 above.

A link to the GitHub repository can be found in the references.

1. Software Design

This section and its subsections showcase the various parts the architecture of the game engine is built of and how they connect with each other.

* 1. Component Subsystem

This system is the core of the software architecture, designed to encapsulate and decentralize functionalities which are distinct for each single object or certain groups of objects. The scene graph consists of a hierarchy of modular game objects, each with a range of components attached to them. Each component has a specific purpose such as rendering or collider definition. Therefore, new gameplay features are an ease to implement and test individually which has an appreciable impact on project management. Each game object has at least a transform component that defines its position within the world space. Because all game logic is implemented as a script attached to a game object as well, swapping out the gameplay to create a completely different game is definitely possible. This would be intensely time consuming when hardcoded.

The described concept has been a proven part of established game engines like Unity3D and, to some extent, Unreal Engine 4 for several years now.

* 1. Components

The following subsections give a brief overview of the various components that can be attached to game objects.

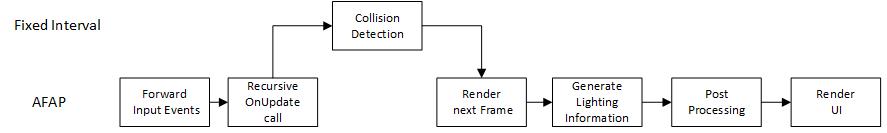
* + 1. *Transform*

Every game object has at least a transform component to represent its position within the game’s coordinate system. Every transform component definition contains position, scale and rotation. The transform component class also implements basic translation, rotation and scaling operations.

* + 1. *Renderer*

An object without a render component will simply not be visible in the scene as it is not being processed by the graphics manager. Each specific renderer holds the information needed to display the object it is attached to in the game world.

The game currently only uses mesh renderers which are linked to both a mesh resource and a material resource.

* + 1. *Tank Behavior*

**Figure 2:** Game Loop

This script is shared between both the player and all computer controlled tanks. It manages functionality such as movement, shooting, ammo and respawning.

* + 1. *Player Behavior*

This script builds up on the tank behavior script and reads user input to move the character and camera around, fire the gun or switch weapons.

* + 1. *AI Behavior*

The AI is based up on the tank behavior script and utilizes a decision tree (See appendix B) to choose the correct action to take.

* + 1. *Audio*

An audio component stores information about a specific sound or music track as well as its base volume. It can also play, pause and rewind the track.

* + 1. *Camera*

Each camera declares all necessary information to generate a valid view matrix and can be set as the active scene’s main camera. A camera can be transformed through the transform component, which every game object has by default.

* + 1. *Collider*

Colliders hold information for the collision manager to calculate possible collisions from. Each collider should register with the collision manager to reduce the number of elements to iterate over. For this game, only box colliders will be implemented.

* + 1. *Game Logic Script*

This script does all the game logic.

* + 1. *HUD Script*

This script builds the heads-up display at the start of the game and continuously updates it.

* + 1. *Light*

There are going to be four different types of lights: ambient-, directional-, point- and spot lights. Every one of them registers with the lighting manager and stores information such as color and intensity.*3.2.11* *Pickup Script*

The pickup script enables certain game objects to be collected via stepping on them. There are two kinds of collectibles that need to be implemented: weapon boosts (adds a second weapon to the user’s repertoire) and health boosts (regenerate a set amount of health).

* + 1. *Projectile Script*

The projectile script moves the missile which is fired by the tank. It also contains functionality for when it hits another tank.

* 1. Managers

Each functionality that should be centralized (e.g. file loading, window or input management) is wrapped in a singleton manager class. This way it can be accessed from everywhere within the application.

* + 1. *Application Manager*

This manager serves as entry point to the application and performs basic setup such as window creation and message loop management. Besides that, all responsibility is forwarded to other managers. There might be different implementations of this manager, dependent on the target platform.

* + 1. *Game Manager*

The game manager is the application’s game core. It initializes the DirectX device and other managers, executes the main game loop (See Figure 2) and delegates tasks to other parts of the game engine.

* + 1. *Graphics Manager*

All functionality and data related to the graphics pipeline is maintained by the graphics manager. It holds the DirectX device context and is called by the game manager to render the next frame.

* + 1. *Input Manager*

The input manager holds the state of all kinds of input. In the current design, this will only be keyboard and mouse. Input events are generated by the application manager, which reads them from the Windows API, and forwarded to the input manager.

* + 1. *File Manager*

The file manager is responsible for reading and writing any kinds of data from and to the file system. This will mainly be resources such as textures, meshes and shaders.

* + 1. *Collision Manager*

Just as the graphics manager is responsible for the rendering step during the main game loop, the collision manager manages the collision between collider components.

* + 1. *Lighting Manager*

As the game engine uses deferred shading to simulate illumination, the lighting manager generates lighting information from the various light sources within the scene and writes it to a texture for the graphics manager to post process.

* + 1. *Preference Manager*

The preference manager loads and stores the player’s settings and utilizes the file manager to persist them to the hard drive.

* + 1. *Level Manager*

The level manager is responsible for saving and loading scene graphs through the file manager. This manager also holds the active scene.

* 1. Resources

All resources are data classes which are, with the exception of materials, loaded through the file manager.

* + 1. *Texture*

Texture resources simply enclose two dimensional color information.

* + 1. *Material*

A material is a combination of a texture and an appropriate shader.

* + 1. *Mesh*

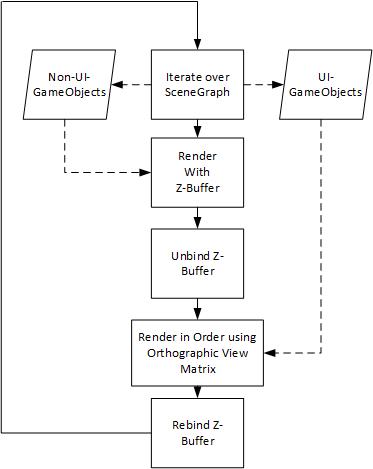
A mesh resource contains all vertices, which store position, normal and texture coordinates, and indices to describe the shape of an object or character.

* 1. Shaders

The vertex shader will calculate world view transformations and depth (z-buffer) but not lighting, as this information is created by the lighting manager.

* 1. User Interface

Due to the use of a custom component subsystem, there is no need for a third party library. The canvas component will simply be part of the scene graph and each child game object is going to be rendered in 2d screen coordinates rather than world coordinates, utilizing an orthographic projection view matrix.



**Figure 3:** Simplified UI Render Process

The Figure above shows the basic concept of integrating the ui rendering process into the 3d graphics pipeline. All 22 components are being rendered using the painter’s algorithm, because allowing manual ordering, besides using the order of the scene graph, would lead to an overall worse performance. Lighting is applied before the user interface is being rendered.

1. **Project Management**

The core of the project management is a single Microsoft Project file (See appendix C), which contains all necessary information about tasks, dependencies, times and deadlines. Because of the given time restrictions, no extra buffer would fit into the project schedule and some features had to be cut (e.g. the particle system).

1. Testing

As the expected programmer’s ability to write code that can be easily unit-tested is not guaranteed and the time frame for development is quite small, the requirements are going to be solely black box-tested using a testing sheet (See appendix D for a blueprint). Each requirement has a test case which tells the tester what to do and which result to expect. Whenever the developer feels that a certain use case has been implemented, he or she can prove correct completion by the executing the corresponding test. If the test succeeds, it can be marked as passed for further reference. Otherwise, the implementation has to be revisited to pass the test.

1. Debugging

To enhance the development process, the third-party logging library Boost.Log (See <http://www.boost.org/>) is going to be used in conjunction with some self-made macros to increase execution speed in production mode. Said framework provides built-in support for advantageous functionalities such as log files, log levels and log filtering as well as additional metrics like thread information and timestamps.

1. **Assets**

Due to the small scope of the project, all models will be either imported from third-party websites or represented by self-made dummy objects created with Autodesk 3ds Max. The same goes for textures, which could be downloaded from <http://www.textures.com/> or created with Adobe Photoshop, and sounds (various websites).

1. **Reflection**

Most of the design process went simply perfect. All members helped working out the requirements, did their individual work in the given timeframe and participated during the meetings to get each other the foundation they needed to create the requested document. Even during private time, my colleagues were open to questions and helped where they could.

There has only been one major problem in the start of the workflow, which was delayed by at least two weeks as a result of incomplete understanding of the assignment definition as well as the team, which consisted exclusively of German exchange students, accustoming themselves to the new surroundings and the general working habits at Solent University. Because of this, research was not as impeccable as it could have been and the number of academic sources was limited.

1. Contact Information

If you have questions or suggestions regarding this document, please contact me at [3SCHAK76@solent.ac.uk](mailto:3SCHAK76@solent.ac.uk).

References

ANON., 2016. *DirectX 10, DirectX 11, and DirectX 12 tutorials* [viewed 21 October 2016]. Available from: <http://rastertek.com>

DUNN, F. and I. PARBERRY, 2002. *3D math primer for graphics and game development*. Plano, TX: Wordware Publishing

POLICARPO, F., F. FONSECA and C. GAMES, n.d. *Deferred shading Tutorial*. [viewed 22 October 2016]. Available from: http://gamedevs.org/uploads/deferred-shading-tutorial.pdf

SCHAPER, K., 2016. *Pilopa/Advanced-Games-Programming\_Group* [viewed 22 October 2016]. Available from: <https://github.com/Pilopa/Advanced-Games-Programming_Group>

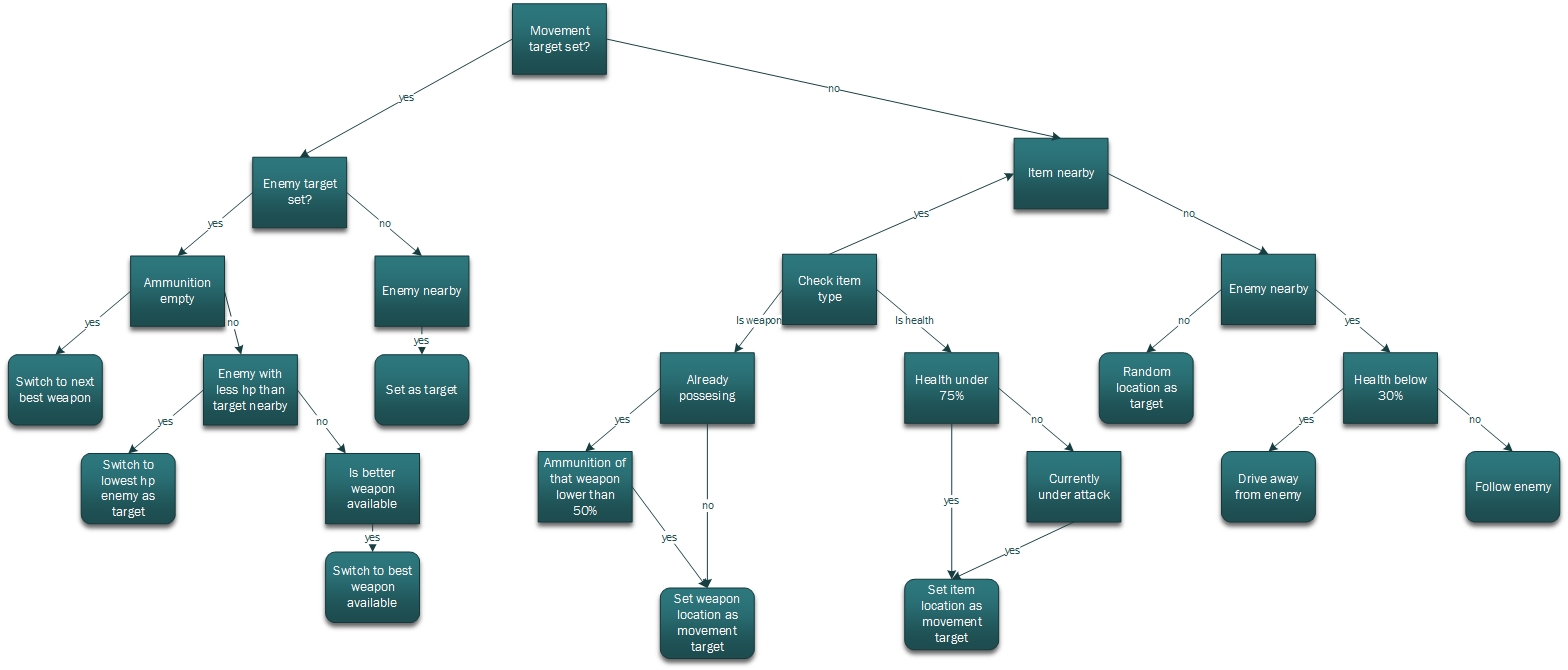
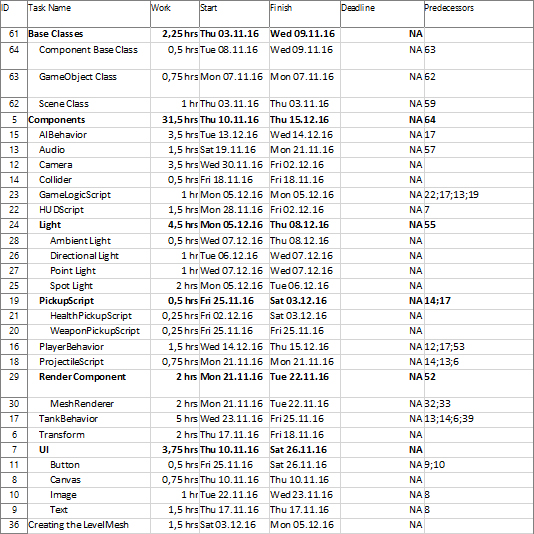
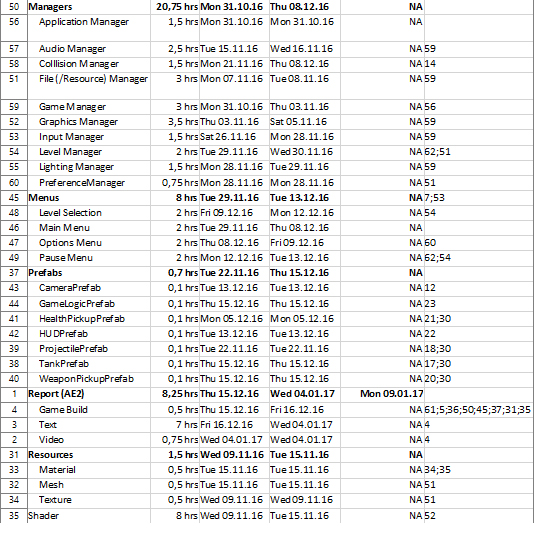
SHERROD, A. and W. JONES, 2011. *Beginning DirectX 11 game programming*. Boston, MA: Delmar Cengage Learning

SHERROD, A., 2007. *Ultimate 3d game engine design and architecture*. Boston, MA: Charles River Media

Appendix

1. Requirements Breakdown
2. Decision Tree (Felix Boehner)
3. Project Schedule
4. Testing Sheet
5. Requirements Breakdown

|  |
| --- |
| Must-Have (A): |
|  |  |
|  | RA001 : There should be a main menu. |
|  | RA002 (Requires RA001) : The main menu should contain a "Start Button", which loades the main scene on click. |
|  | RA003 (Requires RA001) : The main menu should contain an "Exit Button", which terminates the game on click. |
|  | RA004 : There is an environment for the player to drive around. |
|  | RA005 : The Player character, represented by a tank, is being displayed on screen. |
|  | RA006 : The Player has to be able to move by using WASD. |
|  | RA007 : The Player is able to shoot by pressing Spacebar. |
|  | RA008 : The game should have some sort of physics system (eg. The bullet flies in a realistic way, the Player falls into holes). |
|  | RA009 : The game should have some sort of collision detection system (eg. The bullet collides with enemies and explodes) |
|  | RA010 : The player must be able to pickup weapons and items |
|  | RA011 (Requires RA010) : The player must be able to select a weapon to use |
|  | RA012 : The game must have enemies wich are represented as tanks |
|  | RA013 : There is an UI |
|  | RA014 (Requires RA013) : The UI shows the player's points |
|  | RA015 (Requires RA013) : The UI shows player health |
|  | RA016 (Requires RA013 & RA011) : The UI shows currently selected and available weapons |
|  | RA017 (Requires RA013 & RA011) : The UI shows ammunition for the currently selected weapon |
|  | RA018 : The tanks in the game can take damage |
|  | RA019 : The vertical orientation of the canon is controlled via the arrow keys |
|  | RA020 : Pressing ESCAPE allows the player to exit the game |
|  | RA021 (Requires RA018) : Tanks respawn on a random position (though the locations may be predefined) on death |
|  | RA022 : Each tank has a basic canon, which has infinite ammunition but does minor damage |
|  | RA023 : A rocket launcher pickup spawns in specific intervals on the map (randomly or predefined positions) with limited ammunition, |
|  | but high damage and area of impact |
|  | RA024 : A repair pickup spawns in specific intervals on the map (randomly or predefined positions) which restores some of the tanks hitpoints |
|  | RA025 : There is some basic form of lighting, coming from different sources within the world |
|  | RA026 : Each object in the game is textured and the texture is being displayed correctly |
|  | RA027 : The game lasts for a certain amount of time and the player with the most points wins at the end. You gain points by destroying |
|  | opposing tanks and lose points by dying. One cannot have less than 0 points. |
|  | RA028 : The camera follows the player and is third person. The camera is always facing in the direction of the player model. |
|  |  |
|  | Should-Be (B): |
|  |  |
|  | RB101 : Every implemented requirement must be blackbox-tested with a corresponding test-document(ation). |
|  | RB102 (Requires RA001) : The main menu should contain an "Options Button", which opens the options menu on click. |
|  | RB103 : There should be an options menu. |
|  | RB104 : The volume should be adjustable. |
|  | RB105 (Requires RA012) : The enemies should move around. |
|  | RB106 (Requires RA012) : The enemies should attack the player. |
|  | RB107 : There should be different tank models or textures |
|  | RB108 (Requires RA002) : The player can set the amount of enemies in the game, before the level loads |
|  | RB110 (Requires RA005) : The play model represents the orientation of the canon |
|  | RB111 (Requires RA020) : The player should be prompted before exit. |
|  | RB112 : There is a skybox/skysphere around the world displaying a simple texture |
|  | RB113 : The player can define the duration of the match before the level loads |
|  | RB114 (Requires RA021) : The respawn is delayed, the time left to respawn display on screen. |
|  |  |
|  | Nice-To-Have (C): |
|  |  |
|  | RC201 : The bullets have an explosion effect on impact. |
|  | RC202 : The tank canon should emit a particle effect each time it shoots. |
|  | RC203 : The tracks of the tank emit particle effects while driving. |
|  | RC204 : There are different maps to choose from |
|  | RC205 (Requires RA002 & RB107) : The player can choose their tank/color before the level loads |
|  | RC206 (Requires RA002 & RC204) : The player can choose a map to play on before the level loads |
|  | RC207 (Requires RB107) : Different tanks have slightly different stats |
|  | RC208 : There are different difficulties for the AI |
|  | RC209 (Requires RA019) : The UI shows the orientation of the canon |
|  | RC210 : Menu items are animated |
|  | RC211 : The tanks take damage depending on where they have been hit (i.e. Hitboxes) |
|  | RC212 : There is an option to let the game generate a semi-random map to play |
|  | RC213 (Requires RA021) : The screen is grayed out while respawning. |
|  | RC214 (Requires RC208) : The player can set a difficulty for the AI before the level loads |

1. Decision Tree (Felix Boehner)
2. Project Schedule
3. Testing Sheet