StormVile Game Documentation

PRGAP - Product Development Document

This documentation covers each phase from the requirements of the product through to the design and testing of the system.

Stormvile - 814853

Version	Date	Comments
1.0	*04/02/2019*	*Requirements were defined for the project.*

Contents

Introduction

Features

- Model Importer
- Random Targets (Health can be different)
- FPS
- Laser Bullets
- Flashlight
- Text

Gameplay

Fly around in tight corridors
Shoot random targets or turrets
Go as quickly as possible
Do not waste bullets

1. Purpose

a. This document details the design of an interactive 3-Dimensional product for submission along with the implemented product at the end of the DIRXENG course.

2. Scope

a. The development will be of a product that allows a user to interact and do the following:
   i. Go through 5 pre-made levels
   ii. Shoot at targets
   iii. Timed on how quickly they go
   iv. Counts the number of shots fired
   v. The object is to clear the trench and not waste shots

3. Definitions, acronyms and abbreviations

a. *O-O – Object Oriented.*
b. *DirectX – A Game SDK used to handle the input and output aspects of game play.*
c. *Procedural - Generation of a level/assets using random elements*
d. *Assets - Game objects/models e.g. Tree/book*
e. *Level - Premade/user made layout of objects place in a position to create a world*

System Requirements Specification

General Description

This Section describes the general factors that make the product and what requirements they have. IT does not state specific requirements, it just makes those requirements easier to understand

- ***Product perspective***
    a. *The product will be developed to run on Windows computers capable of effectively running DirectX 11.0 or greater.*

        #### i. Hardware

        1. PC Keyboard and Mouse. No controllers, joysticks
        2. Medium End Gaming PC
        3. 1920 *1080 monitor
        4. Static Window resolution

ii. Block Diagram

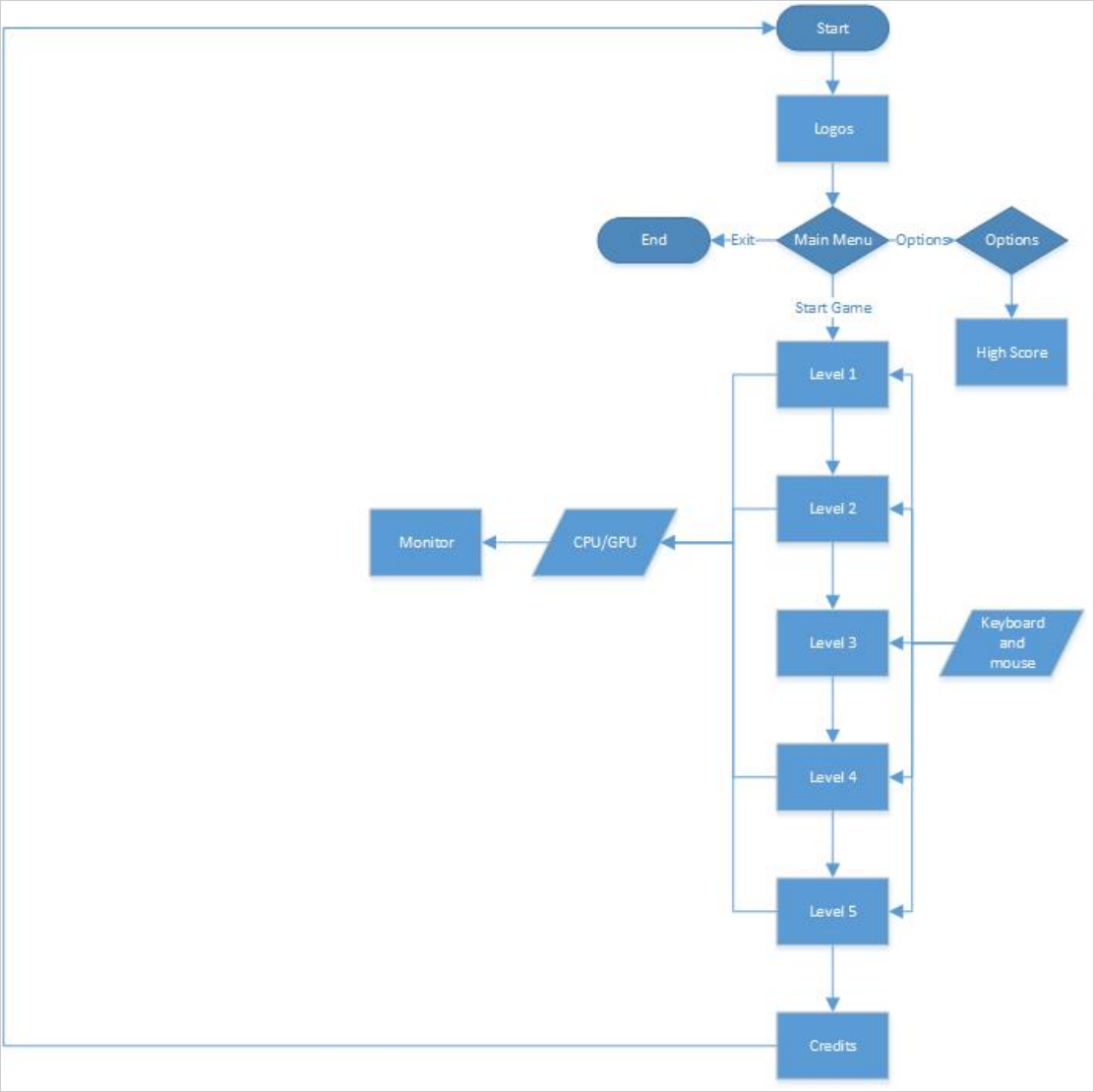


Figure 1 - Block Diagram

- ## *Product Functions*

a. The game will allows the user to take control of a star-ship going through tight and narrow corridors. The game will allow the user to move this ship in a 360 degrees with a 3RD person camera locked to the ship. The game will also allow the player to change this camera to be first person. There is possibilities for the game to have UI elements like crosshairs and additional information. When the user interacts with the ship they can accelerate it in the direction it is facing. The ship will be speed clamped to slow down. When the user hits a wall they take damage represented by the ships colour.

- ## *User Characteristics*

a. *The user will be academics and students within the University of Portsmouth that are familiar with the use of computers and 3D graphical applications.*

- ## *General Constraints*

    a. *The development of the product will be designed and implemented for the lab environments within the University of Portsmouth.*

    *b. The game requires a 1920 * 1080 resolution so the gameplay experience is more uniform.*

- ## *Assumptions and dependencies*

    a. The developers assume people playing this game have a “gaming” PC capable of handling basic gameplay, the game has multiple objects with differing textures and basic lighting, this can cause lag on older PCs

    b. I assume the user has Windows 10.

Specific Requirements

This section provides comprehensive detail on all requirements. It should include all of the detail, which the designer will need to create the design

Textures Needed:

Text
Modern Brick
Ancient Marble
Futuristic Carbon Wall
LED Wall
1 Ship texture

Models Needed

Ship
Gun
Bullet
Turret
Wall

- ## *Hardware Requirements*

    a. Computer:
    b. i5
    c. Hard Drive 4GB space
    d. GTX 970
    e. 8GB Ram
    f. Monitor:
    *g. Any 1920*1080 monitor*

- ## *Software Requirements*

    a. DX11 Installed on PC

b. *Windows 10*

- ## *Interface Requirements*

a. U1 - Mouse (100 Mandatory)

i. Left Click: Fire Weapon
ii. Moue: Aims Gun Gimbals

b. U2 - Keyboard (100 Mandatory)

i. W: Rotate Down
ii. S: Rotate Up
iii. A: Strafe Left
iv. D: Strafe Right
v. E: Rotate Ship Right
vi. Q: Rotate Ship Left
vii. lShift: Thruster Increase
viii. Control: Thruster Decrease
ix. R: Centre Guns
x. X: Breaks
xi. Space: Fire Weapons
xii. Left: Rotate Camera Left Around Player
xiii. Right: Rotate Camera Right Around Player
xiv. Up: Rotate Camera Up Around Player
xv. Down: Rotate Camera Down around player

- ## *Functional Requirements*

a. F1 - Shooting (100 Mandatory)

i. The program will shoot bullets from a gun placed on the player ship

b. F2 - Spawning Targets (100 mandatory)

i. The program will spawn targets on walls or floors at random intervals and amounts
ii. Game logic should keep track of number of targets

c. F3 - Spawn Turrets (100 Mandatory)

i. The program will spawn turrets on walls or floors at random intervals and amounts
ii. Game logic should keep track of number of turrets

d. F4 - Create Corridor (100 Mandatory)

i. The program can create corridors at a set length (max 50)
ii. The floors will be separate so you can handle textures and lighting better

e. F5 - Create Room (100 Mandatory)

i. The program can create 4 rooms
   1. 2 x 2
   2. 3 x 3
   3. 4 x 4
   4. 5 x 5

f. F6 - UI (100 Mandatory)

i. The program displays stats like health and ammo used and time

g. F7 - Mesh Importer (100 Mandatory)

i. The program imports meshes from OBJ files

h. F8 - Texture Handler (100 Mandatory)

i. The program imports textures from DDS

- ***Performance Requirements***

  a. **P1 - The system will perform on average no less than 40 Frames per second**

  b. **P2 - The system will have no sudden frame drops**

  c. **P3 - The system will support the input of any keyboard and mouse**

  d. **P4 - The system will initialise and run from hard-drive storage in under 10 seconds**

  e. **P5 - The system will be able to run from removable storage formats given that the right DirectX version is installed.**

  f. **P6 - The system will load levels in under 5 seconds.**

  g. **P7 - The system will not close unless instructed to**

- ***Reliability requirements***

  a. **R1 - Mean time between failures will be 24 hours of game-play.**

  b. **R2 - Failures can be caused by unknown player inputs/devices**

  c. **R3 - The user will not be able to change controls these are hard coded in**

- ***Design Constraints***

  a. *The target hardware system will be a I7*
  b. *The memory available will be 4GB of main memory.*
  c. *The size of the final product is restricted to the size of the CD-ROM format (under 700Mb).*
  d. *The graphics hardware required will be an NVidia GeForce 4.*
  e. *The computer system will have the DirectX 11 runtime installed.*
  f. *The user will not be able to use a controller or joystick*
  g. *Textures and models are basic to save performance.*

Software Design Description

Design Overview

This section details the high-level look at the program and is used as a road-map to further define the component detail in the following sections.

- ***Software Structure Chart***
  - (In Project File Go to: Stormvile\Stormvile\Design)
  - Appendix 1: Software Structure Chart
- ***Screen Design***
  - **Main Menu**

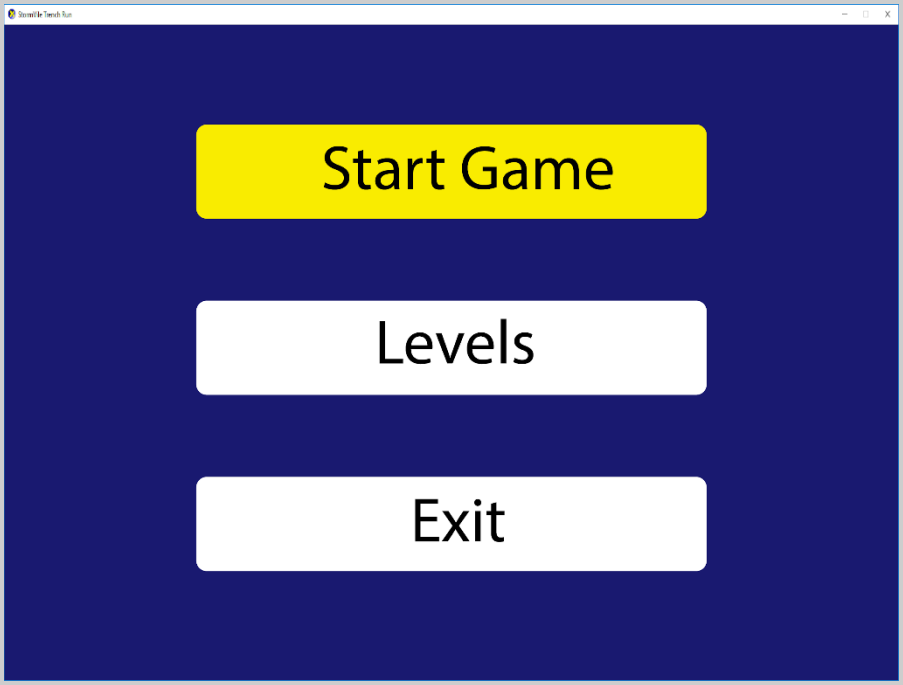


Figure 2 Main Menu Screen design

Game

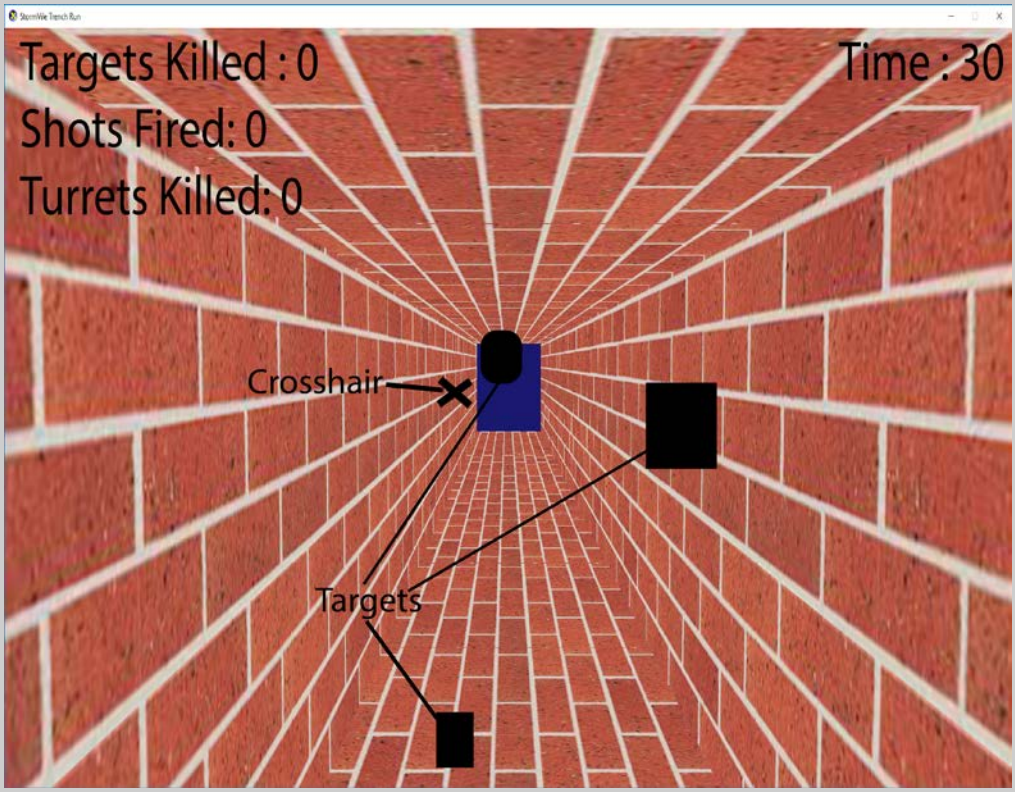


Figure 3 Game Screen Design

Level Selector

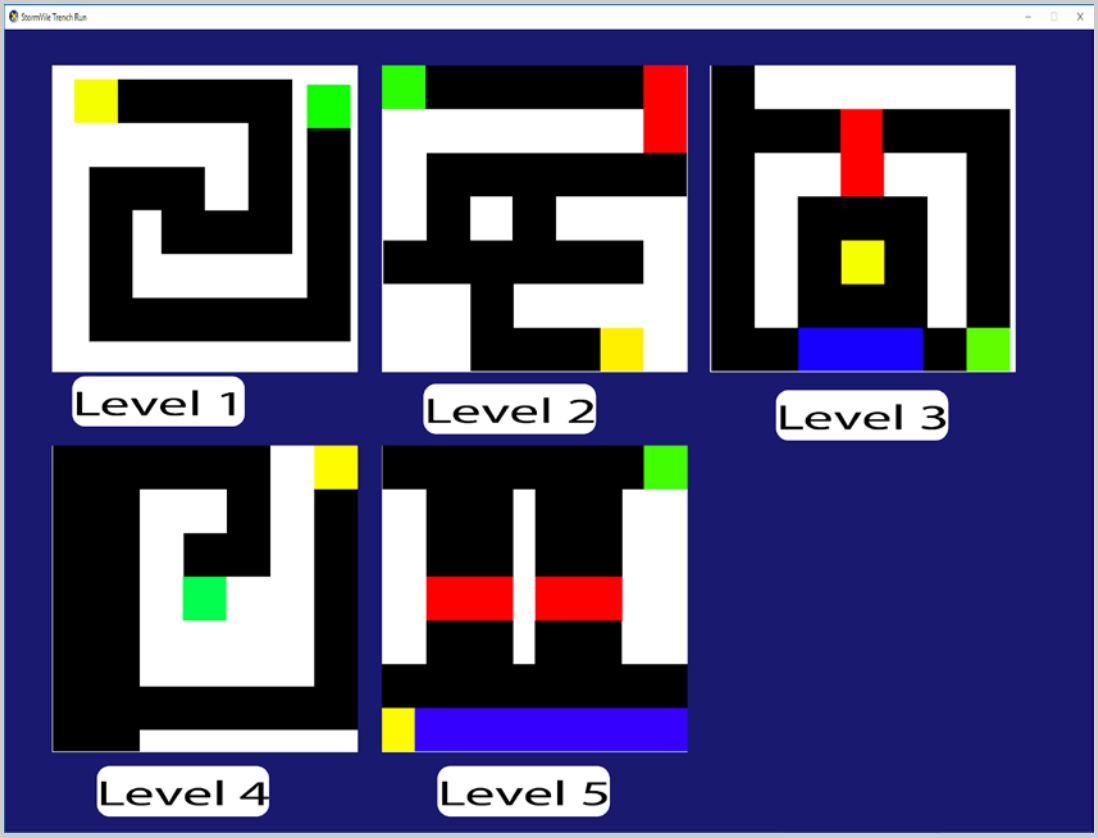


Figure 4 Level Selector Screen Design

○ *Level Design*

Level 1 Level 2 Level 3

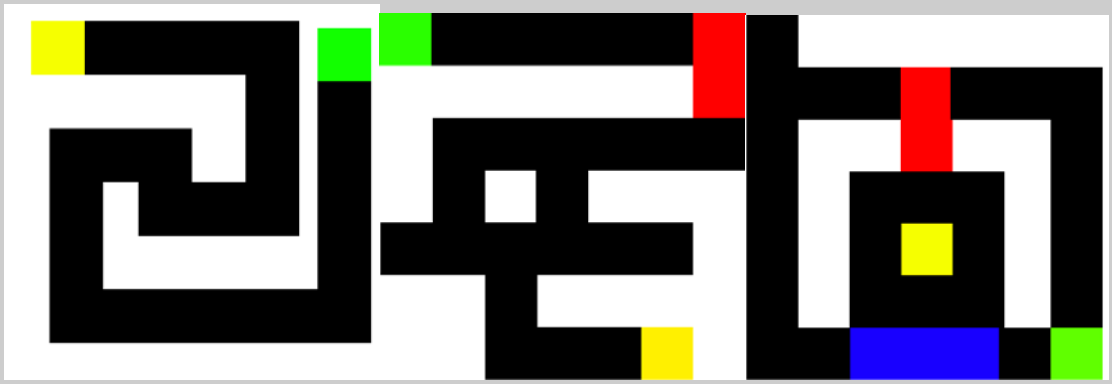


Figure 5 Level 1 Design Figure 6 Level 2 Design Figure 7 Level 3 Design

Level 4 Level 5

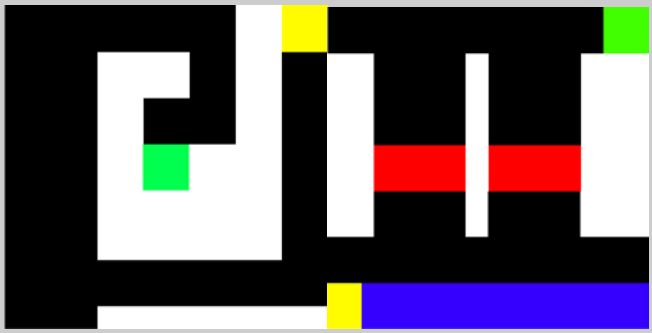


Figure 8 Level 4 Design Figure 9 Level 5 Design

Level 1 is a simple snake like level, black represents the corridors positions, Yellow is where the player starts and green is the finish. The level designs are not what the final level will look like but more of an idea of how to layout the level for the player and a general structure needed.
Level 2 brings in the concept of multiple pathways and shorter corridors, it also adds a new colour red, red is a vertical shaft up or down for the player to fly in. The player must again reach the green cube to go to the next level. Turrets and targets can spawn on either end of the connecting corridors and this can add challenge to the player

Level 3 starts in a room, the room size will be determined in the programming phase, to get out of the room the player must fly up/down the shaft to enter a corridor, this corridor will go around in a circle, with a dead end, and the dead end is guaranteed to have a turret. The blue colour is used to represent that it is on a different level to the black room the Green cube is on a corner

Level 4 goes back to a more basic concept but with a wider corridor/ room this will give the players more space to shoot and fly around in and the level is more build for fast speeds. The Corridors go tight quickly giving the player a chance to crash and die
Level 5 the most complex level to create in game this level will require a lot of manual setups with the prefabs. The player starts in the bottom left, the blue corridor is separate to the black corridor with an entrance at the end. The player then has a choice of two wide corridors, both of these corridors have a shaft going up to the next level the player can then make their way to the green end.

Detailed functionality description

This section Details the functionality of classes, their purpose, pseudo code of what they do and any dependencies of sub components

- ***D1 - Texture Importer***
  - **Description:**
    - The Texture importer, imports textures from DDS files, the purpose of this is to have external textures such as bricks. DDS files can be read from file and applied in main
  - **Sub-Components:**
    - DX11
  - **Dependencies:**
    - DX11
    - DDS
  - **Used by:**
    - Main/Transformation
    - Objects and Meshes
- ***D2 - Model Importer***
  - **Description:**
    - The model importer reads obj from file and applies their model data to the mesh structure used by DX11, this can include UVS to apply textures. This is optional
      - Read file from directory
      - Vertex Buffer = Obj Model Vertices by reading the file line by line and getting the vertex point
      - UV = Obj Model UV
      - Store In memory
  - **Sub-Components:**
    - DX11
  - **Dependencies:**
    - DX11

- **Used by:**
  - Transformation
  - Main

D3 - Transformation

- **Description:**
  - Main Transformation component which is used by all objects to define where they are in the world, how big they are, their rotation and their colour
    - Update Transformation:
    - Position = new Position (XMFLOAT3 (0, 0, 0));
    - Scale= new Scale (XMFLOAT3 (0, 0, 0));
    - Rotation= new Rotation (XMFLOAT3 (0, 0, 0));
    - Red = 50
    - Blue = 10
    - Green = 1

  - **Sub-Components:**
    - DX11
    - Mathf
    - Vector Maths
    - Collision Detection

  - **Dependencies:**
    - DX11

  - **Used by:**
    - All Game Objects
    - Collision Detection

D4 - Collision Detection

- **Description:**
  - AABB Based Collision detection using basic collision checks to increase performance all objects have a AABB box that automatically updates every frame
  - Intersect tests are done in main but the functionality is in collision detection
    - Update
    - AABB MIN = Transformation - Scale
    - AABB Max = Transformation + scale

    - Intersects (AABB BOX1, AABB BOX2)
    - Check if Min is more than max is so swap them

- If (BOX1 is in BOX2)
- Return true
- Else
- Return false

Sub-Components:

- Transformation

Dependencies:

- Transformation

Used by:

- All Game Objects

D5 - Targets

Description

- Class that handles and spawns targets in for the player to shoot, these targets are static but still need to check for collision detection
  - Update
  - Get Bullet list from player's gun
  - Check bullet I box against target box
  - If collision is true
  - Set game logic score to +1
  - Set Shots fired to +1
  - Destroy target

Sub-Components:

- Transformation
- Collision Detection

Dependencies:

- Collision Detection
- Transformation

Used by:

- Level Creation

- ## *D6 - Shooting system*
  - ### Description
    - Handles the shooting of bullets from guns, handles any physics involved and the timeout destructor
      - Fire
      - Get Gun Position
      - Get Gun Rotation
      - Shoot Bullet Direction from gun rotation
      - Apply Speed
      - Update for 10 seconds
      - Destroy
      - Update Bullet direction + speed
  - ### Sub-Components:
    - Transformation
  - ### Dependencies:
    - Transformation
  - ### Used by:
    - Gun

D7 - Turret

Description

- The turret module handles and creates the turret, it gets the player position and see if they are in distance to start shooting, and it then shoots at the current player position at a set speed.
  - Update
  - Get players location
  - If Distance is less than 50
  - Shoot at player
  - Get players bounding box
  - Check Turrets Bullet list against player's collision
  - If collision is true reduce health

Sub-Components:

- Transformation
- Shooting system

- **Dependencies:**
  - Shooting system
  - Transformation

- **Used by:**
  - Level Creation

- ***D8 - Player Input***
  - **Description:**
    - Get keyboard and mouse input from windows and complete actions based on player input
      - Switch (Keyboard)
      - Case A
      - Strafe Left
      - Case D
      - Strafe Right
      - Case W
      - Pitch Down
      - Case S
      - Pitch Up
      - Space
      - Shoot
  - **Sub-Components:**
    - Windows SDK
  - **Dependencies:**
    - Windows SDK
  - **Used by:**
    - Player Input

- ***D9 - Ship Flight***
  - **Description:**
    - Part of the player functionality accurately flies and moves the ship depending on player input
      - If Acceleration = true
      - Speed = 5;
      - Direction += speed

- **Sub-Components:**
  - Transformation
  - Player Input
- **Dependencies:**
  - Transformation
  - Player Input
- **Used by:**
  - Player

D10 - Corridors and Rooms

- **Description**
  - Creates Corridors And rooms
    - Create a floor with a scale of 5
    - Creates walls with the same size and offset
    - Get Corridor Size
    - Create floors for how long the corridor is
    - Put the corridors in the direction intended
    - Make sure corridor collision detection is updated
    - Per floor randomise if it gets a turret / target
    - Choose a turret / target
    - Randomise wall/floor/roof
- **Sub-Components:**
  - Transformation
  - Turrets
  - Targets
- **Dependencies:**
  - Transformation
  - Turrets
  - Targets
- **Used by:**
  - Player

○ *D11 - Text*

■ Description:

- Converts String to Displayable text
  - Get Number
  - Get Text to display before number
  - Combine the two
  - Convert it to wchar
  - Display wchar using sprite batch

■ Sub-Components:

- DXUT
  - Sprite Batch

■ Dependencies:

- DXUT
  - Sprite Batch

■ Used by:

- Game logic

Requirements Mapping

This section details a trace-ability matrix to map the requirements to the design functionality.
Requirement

Requirement	Design	Mandatory	Importance (0-100)
F1	*D6*	*Y*	*100*
F2	D7	Y	100
F3	D5	Y	100
F4	D10	Y	100
F5	D10	Y	100
F6	D11	Y	100
F7	D2	N	70
F8	D1	Y	100

Full function Plan

This sections details all the functions needed per class, data types will be determined when the function is created. The plan does not include data types of structures as these can be changed function by function, it does not include optional

Maths

- The maths class is a more generalised lengthy class of functions to handle vector maths related to the game mechanics and objects. This is from simple operations like addition and subtraction to more complicated functions like vector lerp and forward direction function used for ship movement

Text

- DXUT function that handles the text display in game

Collision Detection

Update

- Get position
- Get Scale
- Min = Position - Scale
- Max = Position + Scale

Intersects

- Get Box1
- Get Box2
- Intersect test for two box
- If intersecting return true

Transformation

Draw

- Get Position, Rotation, Scale, Colour Red, Green blue
- Transform Rotate Then Scale
- Apply Colours

Menu

Create Button

- Location
- Colour
- Text
- Button Action

Update

- If mouse is on button highlight = true
- If highlight is on button highlight = true
- Highlight = keyboard input W up D down

Boundaries

Update

- Check grid.
- Check if player is in grid
- If player is in grid check collision detection

Spawn Target

- If number is in the percentage chance of a target being spawned target = true
- Randomise wall
- Offset according to wall
- Store In memory

Spawn Turret

- If number is in the percentage chance of a turret being spawned turret = true
- Randomise wall
- Offset according to wall
- Store In memory

Bullet

Update

- Get velocity
- Timeout Destructor
- Transformation + Velocity

- Check Bullet Collision
- Set Transformation

Target

- Update

If Bullet is in Grid 10
If Target Is in Grid 10
Perform Collision Detection Test
If Colliding Destroy Target
Add one to target score

Player

- Update

Forward Direction = Rotation
Right Direction = Cross Product of Forward Direction
If Q == true
Rotation.y -= 0.05;
If e == true
Rotation.y +=0.05
IF w == true
Rotation.x -= 0.05
If s == true
Rotation.x +=0.05
If A== True
Velocity.X -= Right Direction;
If D == true
Velocity.X +=Right Direction
If Space == true
Player Gun.Fire
If (Shift == true)
Velocity += Forward Direction
If (LControl == true)
Velocity -=Forward Direction

Check against Boundaries in Grid
If Colliding Player dead == true
If Colliding in Finish Grid stop time
Next level

Rooms

Setup

- Get Room Size
- If 2 BY 2
- Create 4 Floors
- Have Walls Open
- Close Outside Walls
- Create Percentage Chance Of Turrets and targets
- Setup Collision

Update

- Check Room Grid Against Players and Bullets
- If Collided Kill Player

Gun

- Setup
  - Set Gun Position
  - Set Bullet Pool Size
  - Set Gun Rotation
- Update
  - Update Bullet Array
  - Update Position and Rotation
- Fire
  - If bullet fired = false then
  - Get Gun Position and Rotation
  - Velocity = 10
  - Bullet fired = true

Corridors

- Setup
  - Get Corridor Length
  - Get Corridor Direction
  - If Corridor Direction = left
  - Create Floor Prefabs going to left direction
  - Length = floors used
  - Max is 50
  - Have Walls Open
  - Close Outside Walls
  - Create Percentage Chance Of Turrets and targets

        - Setup Collision
    - Update
        - Check Collision Against Player

Main (Engine)

- Main/Render
    - This functions handles most of the DX11 functionality, in broad terms it reads a texture from a file and stores it, the device context is set and updated in the render code where the draw functions are called.
    - DX11 handles most of the graphics API however I still need to set the device context, P3D devices and the context buffer to get the drawing working. I also need to make sure each object has a position and colour in world space otherwise the vertex buffer will not draw.

Project Plan

Project organisation

Life cycle model

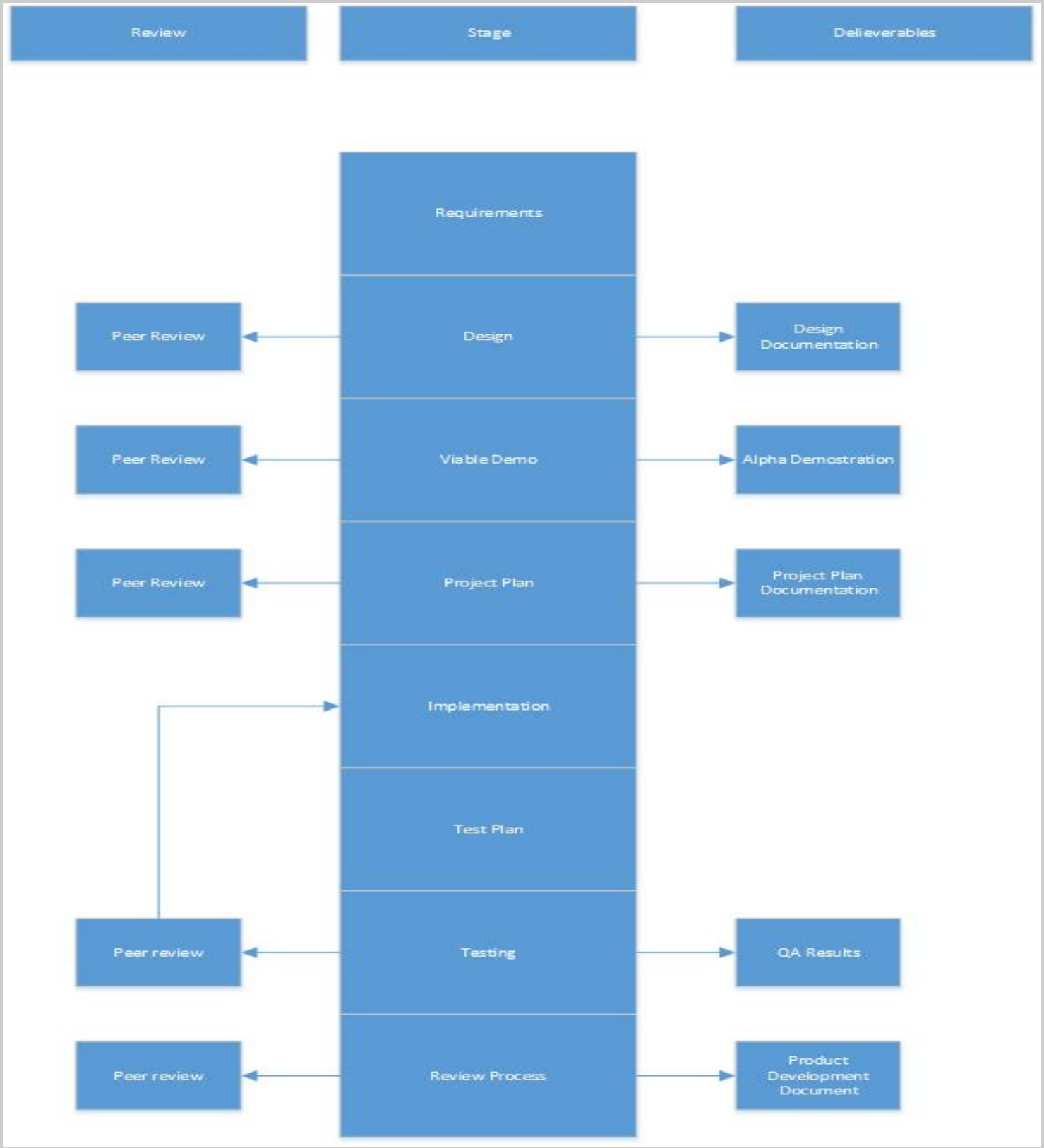


Figure 10 - - Project Life cycle model.

Project organisational structure

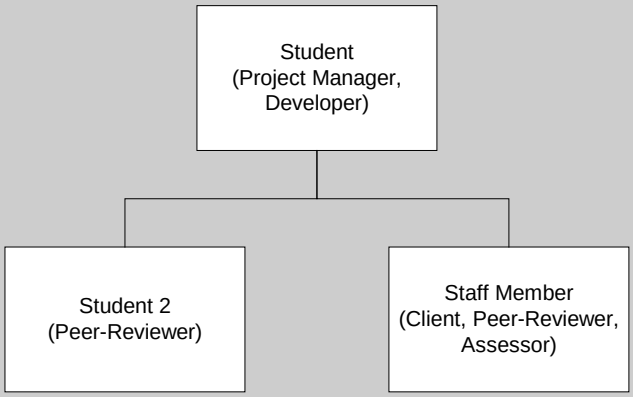


Figure 4.1.2.1 – Organisational structure chart.

Managerial process

Management objectives & priorities

The progress of the project will be discussed informally in the workshop session for the course each week.

The aim is to create a product that meets the requirements and design and is delivered on time to a high-level of quality.

Peer reviews will be performed within the workshop environment to help review the progress of the project and reassess the projects risks

Peer reviews will be informal discussions and feedback on the current version of the product.

Assumptions, dependencies & constraints

The project will be completed by the due date of the final submission, which is the date of April of the 5th.

The hours allocated to implement the project is 8 hours per week totalling 176 hours for the project.

The Project will be constrained in development time by other projects including Unity and PS4 Projects.

Risk management

The results of the latest risk management performed is detailed in the following table:

Event or Risk	Probability	Severity	Preventative steps	Contingency
Corruption of work	*Low*	*High*	*None*	1. *Internet backups* 2. *Zip backups*
Death	Medium	End of project	Good exercise and food	Posthumous

Loss of work	Low	High	Backup Work Source Control	Revert to latest source
Fire	Medium	Low	Fire-extinguishers	Move to library
Flooding	Low	Medium	None	Move to library
Internet Down	Medium	High	Internet Maintained	Memory Stick Backups
GitHub Shutdown	Low	Medium	None	Move to new source control
Building Closed	Medium	High	Extend Open hours	Work at home
Computer Failure	High	High	Computer maintained	Use latest backup Work at new PC
Virus	Medium	High	Anti-virus software	Revert to source
Microsoft shutdown windows	Low	End Of Project	None	Rebase project on new operating system
Personnel Injury	Low	Medium	Be careful	Work at home

Monitoring & controlling mechanisms

This is an individual project with the use of peer review techniques with fellow students as well as staff during workshop times to review project progress according to the timeline detailed in the document.

Staffing plan

The project is an Individual approach with peer reviews performed by one other student and one other staff member.

Quality Planning

This section determines procedures and details for how quality will be maintained on the product. Detail source code commenting standards, file naming scheme, and methods used for version control of both source-code and art assets.

Source Control

Source control will be performed on GitHub, using the desktop application git kraken. To ensure the code is backed up and to revert any major changes that brake the game. The source will be updated after an implementation of a function of feature of the game. The version control has multiple branches depending on what is being developed.

File Name Scheme

Files will be named in plain English with no syntaxes or code so anyone can read the file name. Folders will be named for what they contain E.g. Design contains the images and files that were created for this documentation.

Code Readability

To ensure code readability I have used my own naming convention in which I stray away from using PP1 instead having Player.Position which gives whoever is reading the code a higher chance of understanding it. There are also comments on the blocks of code which gives users even more indication of what a section is doing.

Work packages, schedule & budget

This section defines the schedule of work. In a team environment you would also include a work break down structure to allocate work across a team.

Schedule

Difficulty:
Low: 1 Hour implementation
Medium: 3 Hour implementation and test
High: 1 day implementation and tests
Risk –
Low: No risk to project
Medium: Can cause computability issues or minor errors
High: Can break the project leading to crashes and major errors

Function	Difficulty	Risk	Time
F1	Medium	Low	1 Day
F2	Low	Low	1 hour
F3	Low	Low	1 hour
F4	Low	Low	2 hours
F5	High	Medium	4 hours
F6	Low	High	3 days
F7	High	High	4 days
F8	High	High	2 days

Budget

Function	Cost
F1	£100
F2	£10
F3	£10
F4	£20
F5	£50
F6	£00
F7	£400
F8	£200

Project training plan

No training will be required for the users of this product.

Test Plan and Results

Test Items

This section details the test cases for the product along with the test result. This document will be revised in version for each time the set of test cases is carried out.

Requirement	Test	Test Case	Status
F1	*T1*	*User Can shoot*	*Passed 2/04/19*
F2	*T2*	*Level Spawns in Targets*	*Passed 2/04/19*
F4	*T3*	*Level Spawns in Corridors*	*Passed 2/04/19*
F5	*T4*	*UI updates and creates*	*Passed 2/04/19*
F6	*T5*	*Meshes Are Imported correctly*	*Passed 2/04/19*
F7	*T6*	*Textures Are imported correctly*	*Passed 2/04/19*
F8	*T7*	*Import a second model*	*Failed 2/04/19*

Table 5.1.1 – Trace-ability matrix from section 3.3 with test cases and test result information.

Test Report Results

Date: 2/4/19

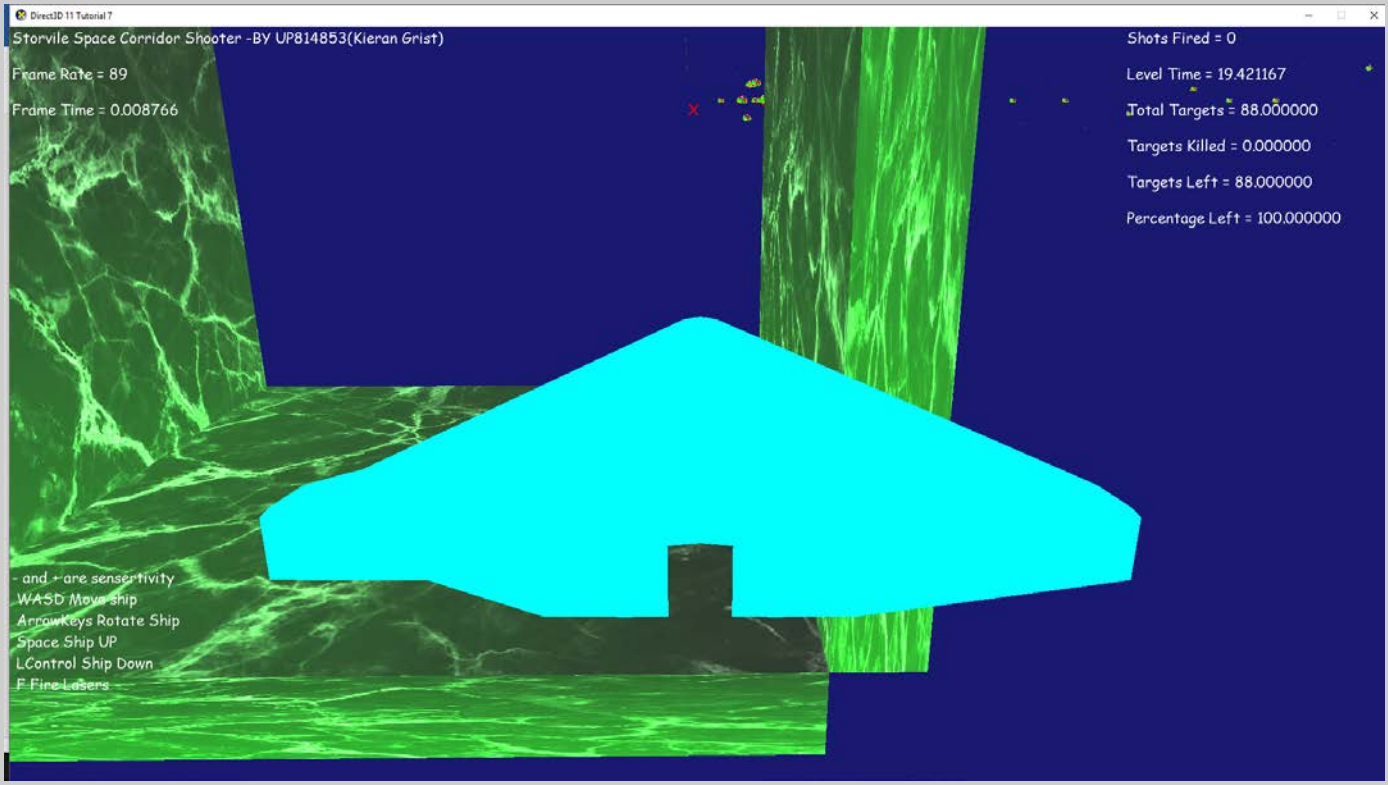
Tested: T1:T6

Issues Encountered: With this test I went through all the functions to ensure they were working. When I tested this I found little to not issues. The only issue I found was when the player pointed up the camera would gimble lock causing you to look at the ships backside and not be able to see where you are going.

Date: 2/4/19

Tested: T7

Issues Encountered:



Features not to be tested

Identify functions that are not to be tested and specify reasons.

Function	Reason
User Input	No reason to test if it works as the input is hard coded
End Screen	The end screen is dependent on collision detection, we would just end up testing collision detection again
Room	Failure to implement feature
Turret Creation	Failure to implement feature

Post Project Review

The post-project review is an information gathering and evaluation exercise to determine what worked well and what did not during the personal project. From this recommendations are drawn about how projects can be run more effectively in the future.

Effective Project Procedures

I have encapsulated the code so that someone can spawn in a boundary with a structure and all they have to do is call update and the update function handles the rest. This was done with all classes and uses basic polymorphism techniques to insure inheritance is done correctly.

The Boundaries class went well when creating it and using it, I came up with a method to generate a closed block around a point, each side of the block. My original project plan would be ineffective and unfit for purpose, I had planned to check if the player was in the grid which would cause framerate issues and possibly lag the game till it is unplayable. I kept the idea and implementation of the Spawn target feature.

The Collision Detection class is very effective at its job, being attached to the game object class, any object can collide with another object and use the collision detection functionality. Comparing my implementation to the project plan I have kept to the plan and not differed from it. For this functionality I kept it simple and low on the processor with it only being called when collision detection is needed within the game. I do this in the collision detection loop inside level update, the game checks if bullet 1 is alive, it then checks if the target is alive, if both are true it checks a collision between the both.

This saves power with bullet 50 being dead it will not check against any objects. I have also implemented this with the walls with the collision detection loop going through the walls and saying, if left wall is true check for Collison against player. If this was false it would allow the player to fly through the invisible wall.

Ineffective Project Procedures

For the Room class a lot of techniques and methods went wrong at the same time, the room class was created to create and update a 10 by 10 room made out of the boundaries. However as soon as there was any collision detection the program took a heavy performance dive. My implementation of this was very much like the project plan indicating that the method I chose to implement the rooms was incorrect and not fit for purpose. The next procedure would be to try and import a mesh of a room as this will be less intensive then 600 game objects.

For Bullets I used a pooling technique to go through 50 bullets which are already in memory, however this is ineffective and there are better techniques to create and update the bullets including vectors and dynamic arrays. Comparing to the project documentation I have kept to the plan with the classes' functionality, the only feature that needs to be changed is the pooling technique.

For Game object I should have used more advanced polymorphism techniques so that I can pass any object that has a game object and process it all under-one function. This function can call update versions on any game object and it will run the code. Instead I have a basic game object class which has transformation and collision detection, and can contain a texture for it to be drawn with. Specific techniques

Estimation Accuracy

Function	Estimated Time	Actual Time
F1	1 Day	3 Days
F2	1 hour	4 Hours
F3	1 hour	- Not Implemented
F4	2 hours	10 Hours
F5	4 hours	8 hours
F6	3 days	4 Days
F7	4 days	1 day
F8	2 days	1 day

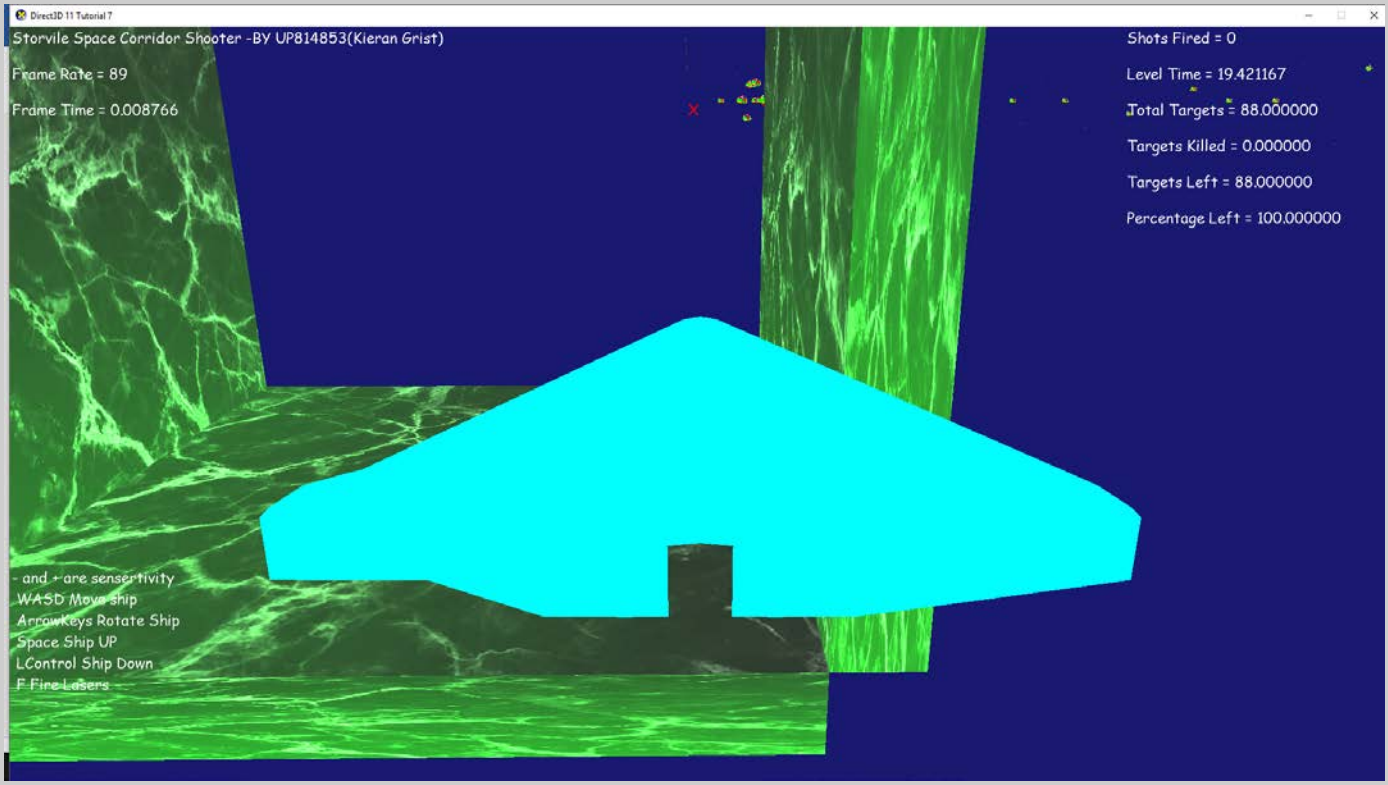
My estimations were incorrect for the project, the implementation of functionality in most cased took longer than expected, in one case with the room the class was not implemented at all due to a lack of time and experience. Having estimated times in my project did not help as they were layaways inaccurate as I would often run into bugs and graphical issues with the project when I added in a new class.

Recommendations

On reflection there is a number of ways, methods and techniques that I could of used to my advantage to make this a better game and a more viable product. The software as is, can spawn in corridors of a various length and update the contents inside and keeps all of this code away from the user so all they have to do is call setup and the function does all the heavy work for them. However there was techniques I could have used to make the product better.

The first method is to use programming techniques such as polymorphism in my code to make the code more usable and more efficient. Under this I would also use pointers and object types more writing a system that allows me to handle a class at once instead of one object. This system would work using the As Technique, this lets me say, If Object 1 is a Bullet Call Bullet Collision System, allowing me to dynamically check object types. I can also use this technique to create a bullet from a game object, Simply going Create Bullet = Game Object As Bullet, with a constructor function dealing with that behind the scenes and having a bullet fire from the game objects position.

The second method is a more robust graphics pipeline that allows me to load more than one object. When I was creating the mesh loader I had a massive issue for when I imported a teapot and the space ship model where the cubes went invisible with other graphical issues as well.



Using ENUMS and other methods to handle the keyboard inputs and also implement file readers so users can change and save their key bindings to how they want. This integration would also include joysticks and controllers which was one of my far of stretch goals to allow the user more fine input. The file readers would also allow the user to have high scores, however the random element of the game makes it harder to have a fair compotation.

Personal Reflection

For this project. I modified Tutorial 7 to make a 3D game and with a 3D camera system that follows the player and has a physic and collision system. However at the start I struggled with how the systems of DirectX worked my primary struggle was understanding how they draw their objects and how their co-ordinate system worked. After 6 weeks of creating prototypes I had a moderate understanding of direct x, this was until I done a small game jam held by our lecture. I found this game jam super useful as it put me under time pressure to create a game in direct x, within this week I created a game where you could place objects on a ship and got collision detection working.

After I had a “Game” Working I moved onto thinking about this project, the project started out as a first person shooter, where you would go into a kill-house and shoot targets and have a score at the end. This changed in the middle of me debugging the corridor generation I found it fun to fly through the corridor so I changed the game to be a space corridor shooter.

I decided to implement one class at a time, instead of all at once which could cause a cluster of issues and computability errors and make debugging the game harder to do. With this I focused on target generation and collision detection, as the corridor generation was working from an early stage creating a game from what I already had was a high priority.

I struggled to get the bullets to draw on screen, they were updating in the code but they were not showing on the screen, to fix this issue I simply made sure that the bullets were not destroying their force after it had been applied. While this does cause a constant force to be applied in one direction it fixed the bullets not drawing. After I had the bullets colliding and the player colliding it was then moving onto the level generation.

The level generation took me 3 hours to get fully set up, this was as the code to set up a corridor was tedious as I would often lose reference of where I was in the 3D world and the level generated with trial and error methods.

On reflection with this project the collision detection should have been done with some form of special partition system so bullets and the player would only check for collisions with objects that are around them, this would improve framerate and allow for the room to be implemented. Also instead of finding out why the bullets were not drawing I opted for an easy and lazy fix which did not fix the original code instead created a work around.

I could have made the setup more efficient by not including unneeded initialise values that were not being needed such as rotation and scale for most initialisers were being overridden, an example of this is the boundaries class which takes in a scale value but does nothing with it.

There is no simple way for me to create a corridor and then set it up latter or dynamically change it and move it around, I have to do it using the setup functionality otherwise it will not work. This is one disadvantage for abstracting my code and using structures as I should have had smaller functions that would allow me to more dynamically create a corridor.

At the start of the project I was head strong on creating a game and ignored the documentation. This was not the wisest approach to this project as it would leave me often guessing what I need to do and have no plan to follow. After I got ahead of the workshops I got hubristic leaving the documentation and working on game and this left me needing to do the documentation before I could carry on working on the game. When I finally did the game documentation I had more of a plan and idea of what I was doing and what was needed to be created, I quickly learnt the systems of DirectX and how to create a video game from this. I personally feel like I done very well with the game made and the time I created it in. Within 4 months I created a game that can load in one model and shoot lasers in a corridor.

However I struggled at certain points often hitting roadblocks which seemed impossible to overtake and errors in code that would leave me wanting to reset the whole project. The mental strength to carry on the project and the lessons I learnt have made me a better programmer towards industry learning how to handle the pressure of a big assignment and still complete the tasks that I set out.

I missed out some sections of the plan and did not have a schedule writing down ahead of time, this was a personal flaw as it left me needing to do a schedule after I created the code and I lost track of when I done items and how much time it took. I had a rough schedule in mind thinking how long a certain function would take but did not realise I needed the dates.

If I was to do this project again I would not do the first 3 sections instead I would get up to the testing plan and make sure before I even make a game I had a full plan of how I want to make it and what I want to do with it.

I learnt how to handle a project on my own from this, I also learnt that wanting to make something without a plan is not always the best action. I also learnt that I need to pay the attention to the plan more during the start of the project to get the best out of myself when creating the code.

Appendix

Appendix 1: Software Structure Chart
Either Go to:
https://drive.google.com/open?id=1YcxAYsja0VB1w9WtGG_gH2_Fa0XfdhdvZOjf-CZcIwc
Or Project File:
Stormvile\Stormvile\Design\Software Structure Chart

Appendix 2: Screenshots of project
Go To: Stormvile\Stormvile\Screenshots

References

australbricks. (2019, April 03). *Homestead*. Retrieved from australbricks.com.au/: https://australbricks.com.au/sa/product/homestead/

Bly7. (2018, July 17). OBJ-Loader. Retrieved from https://github.com/Bly7/OBJ-Loader

Christy, S. (2011, Novemeber 26). Stopwatch Class. Retrieved from https://gist.github.com/SamChristy/1395840

Fernando, R., & Kilgard, M. (2003). *Chapter 5. Lighting*. Retrieved from http://developer.download.nvidia.com: http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter05.html

Idea Catalyst. (2019, April 03). *20 Incredible Geometric Designs Which Will Amaze You!* Retrieved from www.pinterest.co.uk: https://www.pinterest.co.uk/pin/327707310363304027

Lucas, G. (Director). (1977). *Star Wars: Episode IV - A New Hope* [Motion Picture].

Luna, F. D. (2013). *Introduction to 3D Game Programming with Directx 11*. Dulles, Virginia, Boston, Massachusetts: Mercury Learning And Information.

Murals Wallpape. (2918, April 03). *Black Marble Wallpaper*. Retrieved from www.muralswallpaper.com/: https://www.muralswallpaper.com/shop-murals/black-marble-wallpaper/

Parallax. (1995, Feburay 13). Descent [Video Game]. *Descent [Video Game]*. Champaign, Illinoi , United States Of America: Interplay Inc.

Srimal, S. (2019, April 03). *Discover ideas about Textured Walls*. Retrieved from www.pinterest.co.uk: https://www.pinterest.co.uk/pin/30188259980408133/

Walbourn, C. (2019, March 28). Directx-sdk-Samples. Redmond, Washington, United States. Retrieved from https://github.com/walbourn/directx-sdk-samples