Advanced Game Programming (CGP600) Assessment 1

[Cite your source here.]

Individual Report and Game Design Document

Swinging game

Francis Jones and Sergej Frank (Q14281708)

2018

Contents

[Introduction 3](#_Toc528838120)

[Preperation and Planing 4](#_Toc528838121)

[Research 5](#_Toc528838122)

[Syncing 5](#_Toc528838123)

[Pseudocode 5](#_Toc528838124)

[Shaders 5](#_Toc528838125)

[Timer 6](#_Toc528838126)

[Sound in DirectX 6](#_Toc528838127)

[Code Architecture 6](#_Toc528838128)

[Advanced Functionality 7](#_Toc528838129)

[Sound in DirectX 7](#_Toc528838130)

[Syncing and Swinging 8](#_Toc528838131)

[Reflection of work 8](#_Toc528838132)

[Individual Tasks 9](#_Toc528838133)

[Francis Jone’s Tasks 10](#_Toc528838134)

[Group Tasks 10](#_Toc528838135)

[Problems occurred 11](#_Toc528838136)

[Game Content 13](#_Toc528838137)

[Environment 13](#_Toc528838138)

[Levels 13](#_Toc528838139)

[Entities 14](#_Toc528838140)

[AI enemies 14](#_Toc528838141)

[Collectables 14](#_Toc528838142)

[Obstacles 15](#_Toc528838143)

[Mechanics 15](#_Toc528838144)

[Simple FPS Movement 15](#_Toc528838145)

[Jumping 15](#_Toc528838146)

[Air Acceleration (Syncing) 16](#_Toc528838147)

[Hooking/Swinging 17](#_Toc528838148)

[Cutting 18](#_Toc528838149)

[Shooting 19](#_Toc528838150)

[Death Timer 19](#_Toc528838151)

[Design Patterns 20](#_Toc528838152)

[Architecture of Code 20](#_Toc528838153)

[Functionality 21](#_Toc528838154)

[Phong Shading 21](#_Toc528838155)

[Timer 22](#_Toc528838156)

[DirectSound 22](#_Toc528838157)

[Lighting 23](#_Toc528838158)

[Tessellation 23](#_Toc528838159)

[Testing 24](#_Toc528838160)

[Appendix 27](#_Toc528838161)

[Appendix A 27](#_Toc528838162)

[A.1 OneDrive link 27](#_Toc528838163)

[A.2 Github repository 27](#_Toc528838164)

[A.3 Phong Shading 27](#_Toc528838165)

[A.4 Class diagrams 27](#_Toc528838166)

[A.5 Flow diagrams 27](#_Toc528838167)

[A.6 Harvard referencing tool 27](#_Toc528838168)

[A.7 Sound 27](#_Toc528838169)

[A.8 LongJump 27](#_Toc528838170)

[A.9 Kreedz Community 27](#_Toc528838171)

[A.10 Github AirSyncing 27](#_Toc528838172)

[A.11 Swinging Algorithm 27](#_Toc528838173)

[A.12 CS 1.6 reference 27](#_Toc528838174)

[A.13 Audio tool 27](#_Toc528838175)

[A.14 Game reference 27](#_Toc528838176)

[A.15 Time 27](#_Toc528838177)

[A.16 Patterns 27](#_Toc528838178)

[A.17 Maths 27](#_Toc528838179)

[Appendix B 28](#_Toc528838180)

[B.1 Class Diagrams 28](#_Toc528838181)

[B.2 Flow Diagrams 29](#_Toc528838182)

[B.3 Work Breakdown Structure 33](#_Toc528838183)

[B.4 Gantt-Chart 33](#_Toc528838184)

[B.5 User Stories 34](#_Toc528838185)

[B.6 Pseudocode 35](#_Toc528838186)

[Main 35](#_Toc528838187)

[GameObject 35](#_Toc528838188)

[Collectable 36](#_Toc528838189)

[Obstacle 36](#_Toc528838190)

[Character 37](#_Toc528838191)

[Enemy 37](#_Toc528838192)

[Player 38](#_Toc528838193)

[Input 38](#_Toc528838194)

[SoundEngine 39](#_Toc528838195)

[GameData (Singleton) 39](#_Toc528838196)

[Transform 40](#_Toc528838197)

[Command 40](#_Toc528838198)

[MoveCommand 41](#_Toc528838199)

[JumpCommand 42](#_Toc528838200)

[CrouchCommand 42](#_Toc528838201)

[AccSyncCommand 42](#_Toc528838202)

[ShootCommand 42](#_Toc528838203)

[Timer 43](#_Toc528838204)

(Sergej Frank)

# Introduction

This document describes the process of research and software design for the game ‘**Solent Royal**’ as well as an individual report from **Sergej Frank** within the scope of AE1(assessment1) for the course ‘**Advanced Games Programming**’ as part of the group work by **Francis Jones** and **Sergej Frank**. It covers all the necessary key points to properly implement all the functionality and graphics to create a former version of the game.

The first section covers the individual report about research, content and reflection, and the second one that is the software design itself as a group work.

(Sergej Frank)

Preperation and Planing

We started by discussing what functionality the game we want to create needs and how it should look like.

From that point on we created the user stories (Appendix B.5) divided in four main components describing the players experience in a tree list where every individual latter point depends on the one before. The user stories have been reworked twice to ensure a precise description of the game.

Analysing the user stories, it was easy to create Work breakdown structure (Appendix B.3). Apart from relating tasks in the sections **Research**, **Design** and **Development** we added **Initialization**, **Planning** and **Submission** to the Work breakdown structure which meant to be part of the organisation.

Frank then created the Gantt-Chart (Appendix B.4) with **Microsoft Project.** With the Gantt-Chart created we had a good basis work on**.**

As part of the organisation we created a shared folder in our **OneDrive** (Appendix A.1) every team member could access and work on. Furthermore, we saved our data at home on our hard drives, USB-sticks and created a **Github** repository (Appendix A.2).

After that we created a list of subtasks that we saved on OneDrive to see which group member did what and what is already done (green text).

The next weeks we worked individually on our tasks and communicated over Facebook if problems occurred. As several tasks were meant to work in group, we tried to work them out in class.

(Sergej Frank)

Research

(Sergej Frank)

Syncing

To find out how the air movement in Counter Strike 1.6 works I had to look up the physics and equations behind the half-life game engine. The outcome of this research can be reviewed at the section Game Content->Air Acceleration (Syncing).

(Sergej Frank)

Pseudocode

Since I never wrote pseudocode before, I had to research the best practice for it. I had to think about how abstractly I write it. It had to be precise enough to describe the game yet simple enough to understand the logic of it on the fast glance. In the end I covered the most important parts regarding the game flow (Appendix B.2) leaving out code for initialization of the window and DirectX.

(Sergej Frank)

## Shaders

We decided to use lighting, vertex and pixel shaders. How to use vertex and pixel shaders we covered in class, but I had to research more about pixel shaders to apply global illumination. I found a good article about **Phong Shading** which I fully read to understand the algorithm behind it (Appendix A.3).

(Sergej Frank)

## Timer

A not so big challenge was finding a way to implement a timer because I have already worked on a project where I needed it. For that I used the code and algorithm of a blog post from mathematician **Gilles Bellot** (Appendix A.15).

(Sergej Frank)

## Sound in DirectX

Since I have not found only one relevant source for DirectX11 sound, I tried to understand everything I could get out of it.

(Sergej Frank)

Code Architecture

From classes I had last year at the b.i.b international college I used the common practice to write class diagrams. I used an online tool for that (Appendix A.4).

As the project is more than a main function doing something profane, I had to think of relationships and between classes and best use of object-oriented programming.

The most important class I had to think of was the **Timer** class (Appendix B.1). To let the game run on a constant speed while balancing hardware disadvantages due to different frames given per second. Every variable and method I want to use in the timer class are the same as in **Gilles Bellot’s** blog about timing in games.

When I read the book **Game Programming Patterns** (Appendix A.16) I adapted the State, Command and Singleton patterns.

My experience using the **Singleton** pattern as a container of game data (Appendix B.1) worked well in the past, so I decided to use it again.

For the input I had to think of a dynamic solution because I know how painful it is to repeat code for the input for different uses. This is where the **State** in combination with the **Command** pattern stands out (Appendix B.2). Commands can be bound dynamically and give the code an abstract layer in addition. For our project I wanted the programmer to be able to use the input for any kind of character in the game for the sake of testing. That is why I made the signature of the method expecting a **Character** class (Appendix 2.1) object to pass in.

The **GameObject** class was the best fit for the game’s requirements, since it describes all important properties of an object in a 3D world. Additionally, the classes that inherit from **GameObject** show that this class is reusable in sake of object-oriented programming (Appendix B.1).

(Sergej Frank)

# Advanced Functionality

Basics of the vertex and pixel shaders were covered in class, so I had to find some advanced usages of them to get the game the little extra.

(Sergej Frank)

Sound in DirectX

Sound engineering is a whole field of study, so I used only the simplest basic ways – with the potential to expand it – to include it in the project. That is why I neither wrote all the basic initialization methods in my class diagrams nor in the pseudocode. A variety of sound effects would be great thing to have in the project, but I would put that in the polishing/stretch goals section to ensure the more important functionality is completed before. If there is some time left near the end of the project, I would use it to implement the sound.

(Sergej Frank)

## Syncing and Swinging

Both syncing and swinging had some not so easy algorithms behind them to understand. As soon as I understood how I could implement that, it showed me that I need at least a global gravity force in the physics pulling the entities down.

(Sergej Frank)

# Reflection of work

All in all, the whole project worked fine. The learning experience can be described as a slow starting curve that rose exponentially. Without any doubt it was a lot of work to do concerning research and thinking about solutions for the game’s properties. What helped accomplishing that goal was the task breakdown. I even did a further breakdown for myself for the assigned tasks in my head. Maybe it was not the best idea to not write that down. If I would have written that down, things could have gone faster. At least I have done that afterwards.

(Sergej Frank)

## Individual Tasks

* Explanation of mechanics, graphics, design patterns and logic
  + Workflow on this was constant
  + This task made the most fun
* Relevant equations with titles and images for mathematics
  + This task has driven me to get even more interested in maths
  + Understanding the logic behind all equations raised a good feeling of success
* Class diagrams
  + First, I thought that this would be one of the easiest tasks, but the diagrams had so much depth that I wanted to add something every time I looked at them. That is why they took me the longest time to finish.
* Flow charts
  + As the class diagrams were finished, this task was a piece of cake
  + I only took the classes that have the most influence in the game process to be described
* Pseudocode
  + A little research and a lot of words
  + I do not like pseudocode, because the time it took me to write it I could have also written code directly. This doubles the work in my opinion. But I understand that it is good for the sake of documentation.
* Pixel Shader
  + As the pixel shader is a good tool to apply local object illumination, I decided to use the **Phong Shading** for some extra graphic style.
  + It was quite a lot of fun understanding the equation behind it. What helped me to understand it was that I parallelly read the book Foundations of Game Engine Development that our lecturer **Philip Alassad** told us about.
* Timer
  + As stated before, the timer class (Appendix B.1) was something I had already done and used in one of my other projects. Therefore, it was nothing too hard to include in the design.
* Sound
  + I had a hard time understanding this field, because it is so big
  + What really helped me – ironically – was, to listen to music while reading and learning about DirectX sound
* Discussion of object-oriented design
  + After I read Nystrom’s Game Programming Patterns, it was easy to include some of his useful patterns from his book (Appendix A.16).
  + Still I had to think of a reusable solution that makes the code as abstract as possible

(Francis Jones)

## Francis Jone’s Tasks

* Gantt-Chart
* Lighting
* Tessellation shader
* Overworked Work Breakdown Structure
* Grid Tasks relating the Work Breakdown Structure
* Testing (Ad-Hoc, Black box, White box)

(Francis Jones and Sergej Frank)

## Group Tasks

* Referencing
* User Stories
* Work Breakdown Structure

(Sergej Frank)

## Problems occurred

* A lack of communication between group members
  + Even though we used Facebook as a medium to communicate, we did not explicitly point out what we have done so far and what and how we put things together. At least we did that in the last week before submission, so everybody would surely know that all parts of the project are finished till submission day.
* Illness of a group member
  + Because of the illness of a group member we had a delayed workflow and task completion. Fortunately, the submission date was postponed either, so we had a new opportunity to finish the work with less stress.
* Software issues in the University
  + Although someone could argue that the software that was not available for us to use could have been ignored in order to complete this assignment, I think it was a big throwback for the workflow, because it demotivated the students and we lost valuable time learning about foundations of DirectX 11 that could have helped us to understand the basic conditions of a DirectX application.
  + Enough students complained about that, so the IT department put more focus on solving the problem.

Advanced Game Programming (CGP600) Assessment 1

AAssessment 1

[Cite your source here.]

Individual Report and Game Design Document

Swinging game

Francis Jones and Sergej Frank (Q14281708)

2018

(Sergej Frank)

# Game Content

(Sergej Frank)

## Environment

Mainly the look of the game should convey a feeling of a brightness and happiness paired with an extreme contrast of roughness. This combination is identified by a cartoonish look with pirates as main characters for the rough part and bright shiny colors for the happy part.

(Sergej Frank)

### Levels

The game consists of 3 levels of height describing the map. Every level has its own purpose for the gameplay.

Players start at the mid-level and want to shoot their hook on the upper levels platforms to start swinging.

The **only** purpose of the upper level’s platforms is to let the player hook on it.

The ground level has the following properties:

* On the ground of the lower level which is mad out of platforms items can be found and used as projectiles to shoot other characters off the map
* Players who exceed the time limit of being allowed to stay at the lower level will instantly die
* Ascending blocks will connect this level to the mid-level to let the player hook back to the swinging area if he manages to get in range in time
* Under the platforms will be lava that heats up the platforms to justify a death counter

(Sergej Frank)

## Entities

Three different entities exist in the game:

* AI enemies
* Collectables
* Obstacles

(Sergej Frank)

### AI enemies

There are two kinds of enemies.

One kind of enemy – called **Grounder** – can move on the ground at the lower level of the map without dying with the goal of harassing other players to shoot them off the map.

The other one – called **Swinger** – would swing around trying to cut players down.

(Sergej Frank)

### Collectables

Collectables are mostly power-ups that give the collector a slight advantage in gameplay. These advantages would be speed-ups or invulnerability for a few seconds.

(Sergej Frank)

### Obstacles

Some of the entities are obstacles preventing players to follow straight paths to create a challenge in gameplay experience.

(Sergej Frank)

## Mechanics

* FPS Movement
* Jumping
* Air Acceleration
* Hooking/Swinging
* Cutting
* Shooting
* Death Timer

(Sergej Frank)

### Simple FPS Movement

For the movement on ground there will be a standard First-Person-Shooter view with W, A, S, D controls for directions and the mouse input to look around.

(Sergej Frank)

### Jumping

Every character can jump while grounded by pressing the space bar. Its velocity will be affected by a gravity force.

(Sergej Frank)

### Air Acceleration (Syncing)

In general, the movement in the air consists of accelerating and changing direction while airborne.

*Example*:

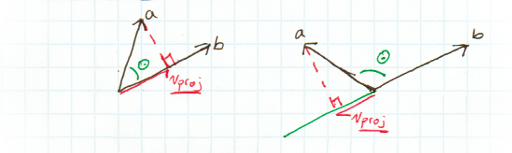
*Moving to left would require pressing and holding the button ‘a’ and moving the mouse synchronically to left to accelerate.*

The mechanic used is like a main mechanic in Counterstrike 1.6 that let the player community create a climbing mod. It works as shown in the Video (Appendix A.8) tough it only applies to syncing and not the rest of the movement in this game.

Additionally, there is a whole community supporting this game mod of Counter Strike 1.6. Explanation videos and paragraphs of how the air-movement mechanic works in Counter Strike 1.6 can be found on the community website (Appendix A.9).

As a condition to implement this mechanic players it is important to understand it.

Research about how that mechanic works in Counter Strike 1.6 showed that the projection of a vector is key to it. Limiting the current velocity of a player in mid-air by using a projection of a vector of an alternative direction. This means that the projection of a vector 90 degrees to the velocity vector has no limit, since the projection is 0 and the player can accelerate endlessly while airborne. An equation for the projection would be:



*Source: Appendix A.10*

(Sergej Frank)

### Hooking/Swinging

What it should look like?



*Source: Appendix A.14*

Hooking/Swinging will be implemented by pressing the hooking key while the player looks at the direction of the desired platform he wants to hook on. Basically, the whole algorithm behind this would add a force each frame depending on the angle of the worlds down vector and the direction vector pointing from the hook the to player. The equation to implement and translate in code later would be:

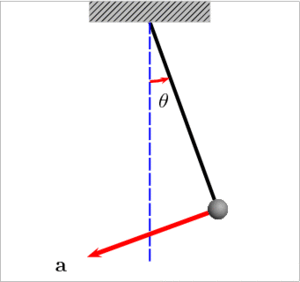


Figure 1.3

While this equation is pure mathematically, it could be easier to implement it differently using only the current angle and the horizontal direction you get if you look the origin of the hook. A possible solution would look like this:

(Sergej Frank)

### Cutting

Every player has a cutting tool in his inventory that can be used to cut enemies off the hook. Only when the player gets near enough the rope of an enemy he can cut him. After a player got cut, he can not use his hook for a specific time.

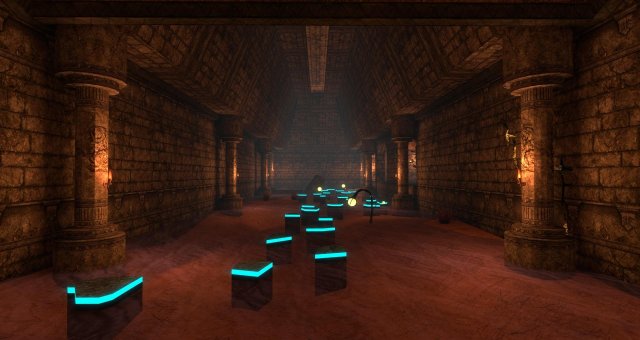
(Sergej Frank)

### Shooting

For shooting there will be a throwing mechanism with a curve. Gravity will do the job to create a former curve.

(Sergej Frank)

### Death Timer

Fitting to the environment of the lower level that is surrounded by lava the player will encounter a death timer that burns the player on the heathen up platforms if he does not manage to ascend fast enough.Picture 1.1 ’A possible look for the lower level of the map considered the ground replaced by lava’

*Source: Appendix 1.1*

(Sergej Frank)

## Design Patterns

Discussing design patterns some common ones are a good way to go, as they are easily implemented and do their job confidently.

Those are:

* Singleton
* State Machine

To get runtime data created by starting, pausing, resuming, stopping or updating the game the *singleton* is a good practice pattern to apply. In this case the **GameData** (Appendix B.1) class carries all information about the game in process to provide shared runtime data between classes.

To handle the input for a characters’ movement – since the input has more than just a few simple commands – a *state machine* pattern was used. Covering all states of players movement, the **Input** class(Appendix B.2/B.6) takes advantage of that pattern.

(Sergej Frank)

# Architecture of Code

Since C++ is an object-oriented language, all classes take advantage of that. Moreover, some classes use common software design patterns as well as Polymorphism.

Generally, the most important class from which the most classes derive is the **GameObject** class (Appendix B.1) that ensures the entities of the game having all the fundamental variables, properties and methods that every game object share.

Also, every game object has a **Transform** (Appendix B.1). That class represents the position, scale and rotation of an object in the 3D world.

(Sergej Frank)

# Functionality

Besides the basic core requirements of a 3D game, there shall also be extra functionality covering advanced use of **shaders, lighting** techniques and **sound**.

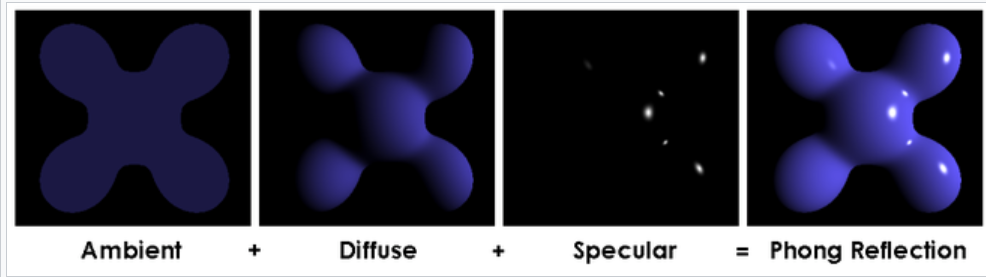
Fitting to the games’ properties a well-polished **SoundEngine** (Appendix B.1) shall be implemented. It shall be responsible for cutting, collecting and dying sounds.

These sounds will be generated with a free to use software called **Audacity** (Appendix A.13)

(Sergej Frank)

## Phong Shading

All light in the game comes from one source, the sun. Therefore, it is a good idea to use a directional light. With the **Phong shading** method this light will be even more identifiable as a real sun. This method allows us to use the pixel shader in a way that transforms the general color input into something more suitable for a 3D world with reflective objects.

It describes the final color as followed: 

Source: Appendix A.3

(Sergej Frank)

## Timer

A closer look on the timing will give us the idea of how to calculate the time steps between the moments when things happen. Commonly known as **delta time**. Reasonable a necessary part of gameplay, since we want all players to have a consecutive experience of the process, regardless of their hardware or uncalculatable frame drops. If we would not use delta time, the game would be limited by frames per second that could give either advantages to players with better or disadvantages to ones with not so good hardware. Therefore, we need to find a solution for that problem. Nothing hard to implement with this simple equation that describes the time between two frames in theory:

Source: Appendix A.15

However, it is hard to implement it in practice, because we need to keep track of time to always have the current and the previous time between each update. Some pseudocode gives an insight of the complexity (Appendix B.6).

(Sergej Frank)

## DirectSound

For a basic SoundEngine class (Appendix B.1) we will need to initialize a SoundEffect which will hold information about the engine being used (IXAudio2), the file format (WAVEFORMATEX) and the sound data itself (Array<byte>).

(Francis Jones)

## Lighting

(Francis Jones)

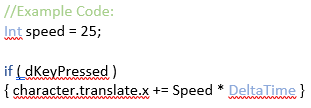
## Tessellation

(Francis Jones)

## Testing

Types of testing the group will be using are Black box, White box and Ad-hoc.

Black box testing summarily is testing an aspect of the program without context of the code I.e. to push the move right key “D” and test that the player moves in the expected direction by the expected distance. This form of testing is a planned test where the user performs an input and tests against an expected result.  This type of test is generally performed by testers rather than developers.

White box testing summarily is testing an aspect of the program with the context of the code known, I.e.     
 

*Figure 1 – Francis Jones ‘18*

In the context of *Figure 1* the user knows exactly that while the “D” key is pressed the character world position should Change by “Speed” along the X axis in the positive, relative to frame rate. When the button is pressed the test is that holding the key for X number of seconds the character should move Y distance precisely.      
This type of test is generally performed by developers only as Code is exposed.

Ad-hoc testing summarily is testing an aspect of the program in a manner that is not planned i.e. during a test run-through of the program where an unexpected bug becomes apparent.   
This type of testing can be carried out by anyone.

Figure 2 – Francis Jones ’18; shows the outline of the groups intended Black box test plan, White box and Ad-hoc will look the same but with White box having an extra column describing a change log of the amended code. White box testing will be carried out pending the results of any Ad-hoc and Black box testing.Page Break

Black box test plan:   
(Note: this test plan is subject to change during development).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Test Description  (Input) | Observed Results  (Output) | Expected Result  (Y/N) | Additional Details |
| 0 | Example:  Directional movement:  when move key is pressed character should move at reasonable speed in the intended direction. | Example:  On key press, the character moved across the screen in the wrong direction and at exaggerated speed. | Example:    N | Example:  Error occurred Lv.1, Moving (intended) Left to right between the first hut and lava pool, no enemies.  player passed through the single obstacle in the scene (Large Rock)  Animation was stuttery. |
| 1 | Game executable:  The .exe file should load in the game. |  |  |  |
| 2 | Menu Buttons:  Clicking menu buttons should activate the relevant scene (game play, options etc.) |  |  |  |
| 3 | Audio Slider (options menu):  Audio slider should affect the sound volume from almost muted to loud. |  |  |  |
| 4 | Mute button:  Should mute/unmute the sound. |  |  |  |
| 5 | Directional movement:  When move key is pressed character should move at reasonable speed in the intended direction. |  |  |  |
| 6 | Death Timer:  Player does not die prematurely (excluding enemy contact) |  |  |  |
| 7 | Death Timer:  Player Dies immediately upon timer expiration |  |  |  |
| 8 | Swing Ability:  When the player jumps toward a hook point the arm should snap to the hook when in range and the character should swing continuously until they jump away |  |  |  |
| 9 | Swing Ability:  Mouse movement should allow the player to swing at slightly increased velocity |  |  |  |
| 10 | Enemy contact:  any contact with the enemy should result in immediate death |  |  |  |
| 11 | Lava Contact:  Contact with lava should cause immediate death.  (You can’t swim in lava) |  |  |  |
| 12 | Upon Death:  the player has 3 lives, upon death the player should lose a life icon and respawn at the last spawn point. If the player has lost all lives, game should reset to the main menu. |  |  |  |

*Figure 2 – Francis Jones ‘18*

# Appendix

## Appendix A

A.1 OneDrive link <https://ssu-my.sharepoint.com/:f:/r/personal/4frans08_solent_ac_uk/Documents/AdvancedGamesProgramming/SolentRoyal?csf=1&e=mUxlAq>

A.2 Github repository <https://github.com/radlog/SwingingGame>

A.3 Phong Shading <https://www.3dgep.com/texturing-lighting-directx-11/>

A.4 Class diagrams <https://www.draw.io/>

A.5 Flow diagrams <http://logicnet.dk/meesoft/DiagramDesigner/>

A.6 Harvard referencing tool <https://refworks.proquest.com>

A.7 Sound <https://docs.microsoft.com/en-us/windows/desktop/xaudio2/xaudio2-introduction>

A.8 LongJump <https://www.youtube.com/watch?v=TkxtdC2L3MQ>

A.9 Kreedz Community <https://xtreme-jumps.eu/kreedz_tutorial.php>

A.10 Github AirSyncing <http://flafla2.github.io/2015/02/14/bunnyhop.html>

A.11 Swinging Algorithm <https://en.wikipedia.org/wiki/Pendulum_%28mathematics%29>

A.12 CS 1.6 reference <https://gameplay.tips/guides/695-kreedz-climbing.html>

A.13 Audio tool <https://www.audacityteam.org/>

A.14 Game reference <http://igg-games.com/energy-hook-free-download.html>

A.15 Time BELLOT, G., 2017. Keeping Track of Time  
. In: [*https://bell0bytes.eu*](https://bell0bytes.eu/)*.*Jul Available from: <https://bell0bytes.eu/game-programming/>

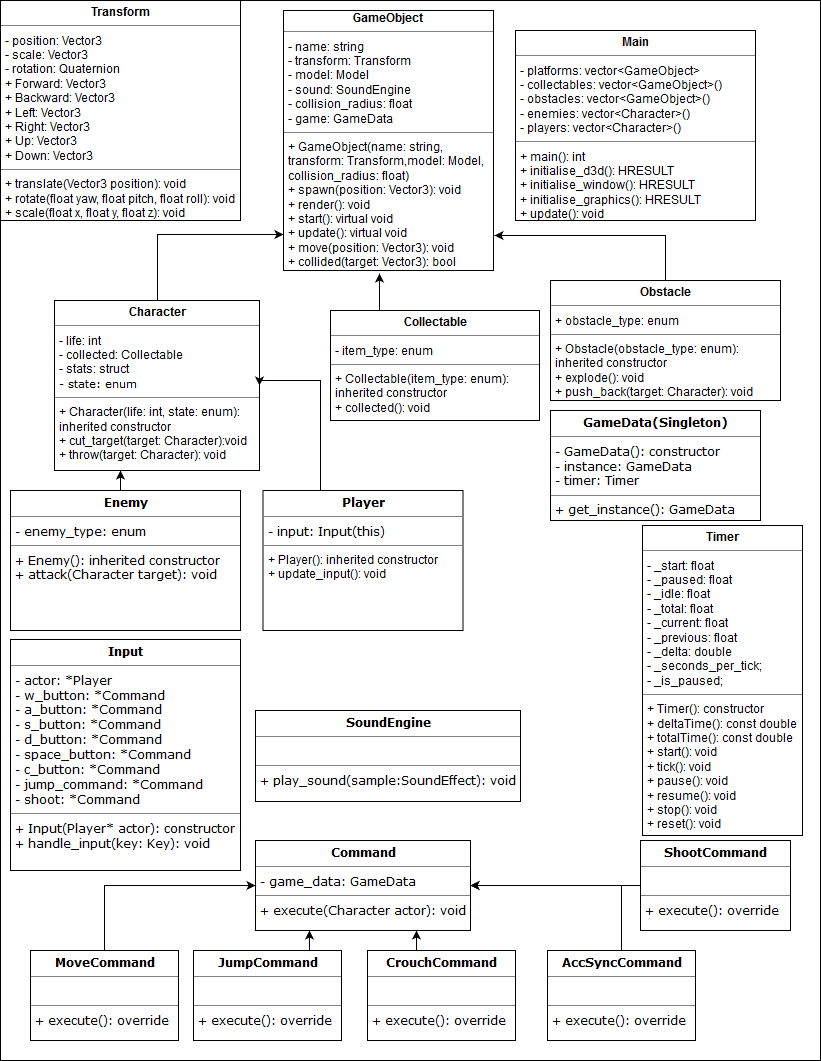
A.16 Patterns NYSTROM, R., 2014. *Game programming patterns.*S.l.: Genever Benning

A.17 Maths ERIC, L., 2016. *Foundations of Game Engine Development.*1st ed. Terathon Software LLC

## Appendix B

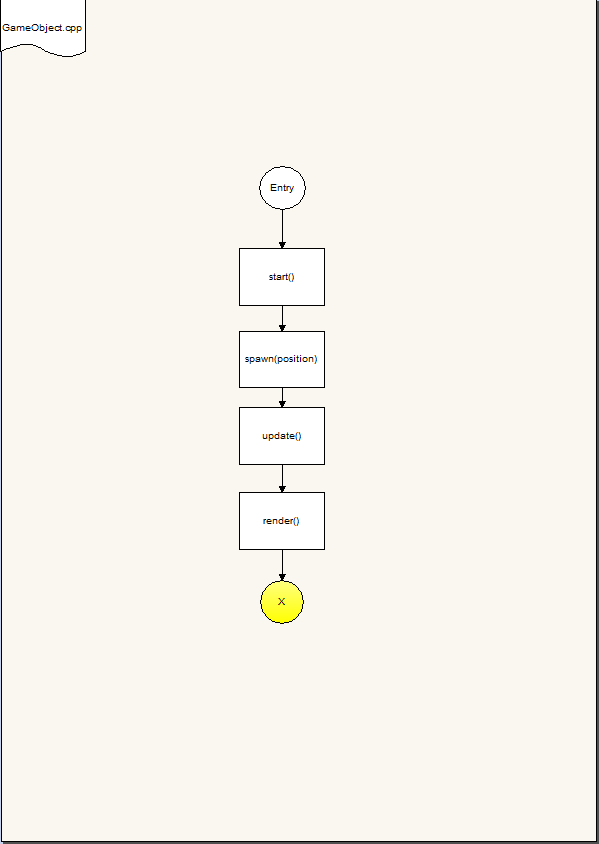
(Sergej Frank)

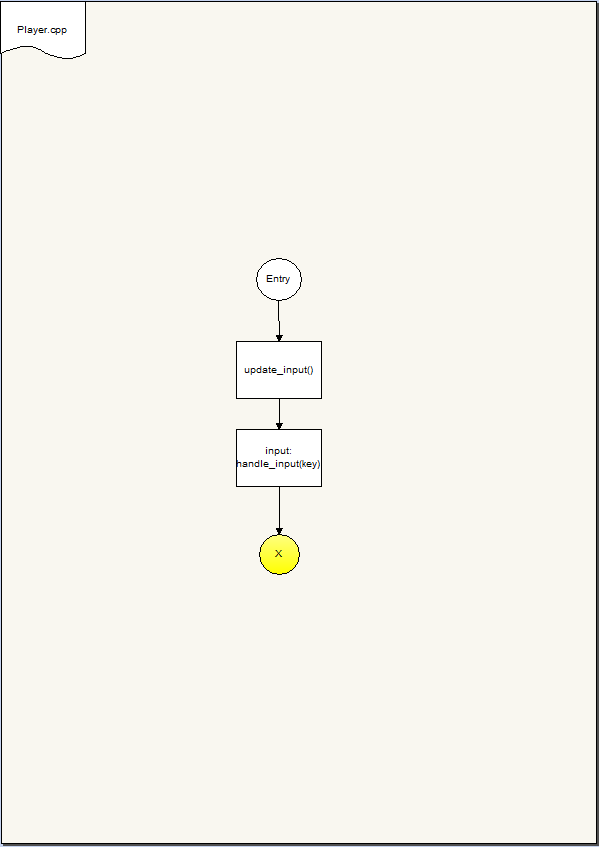
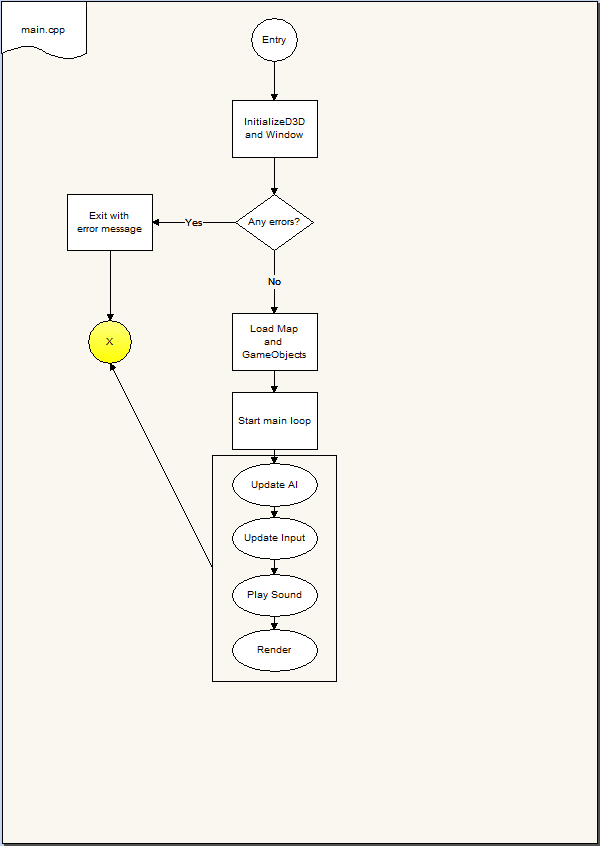
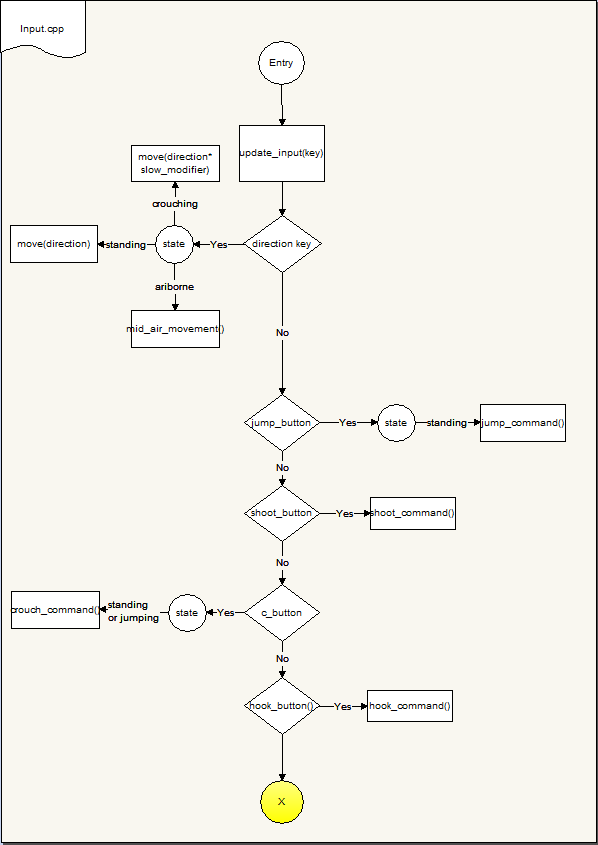
## B.1 Class Diagrams



(Sergej Frank)

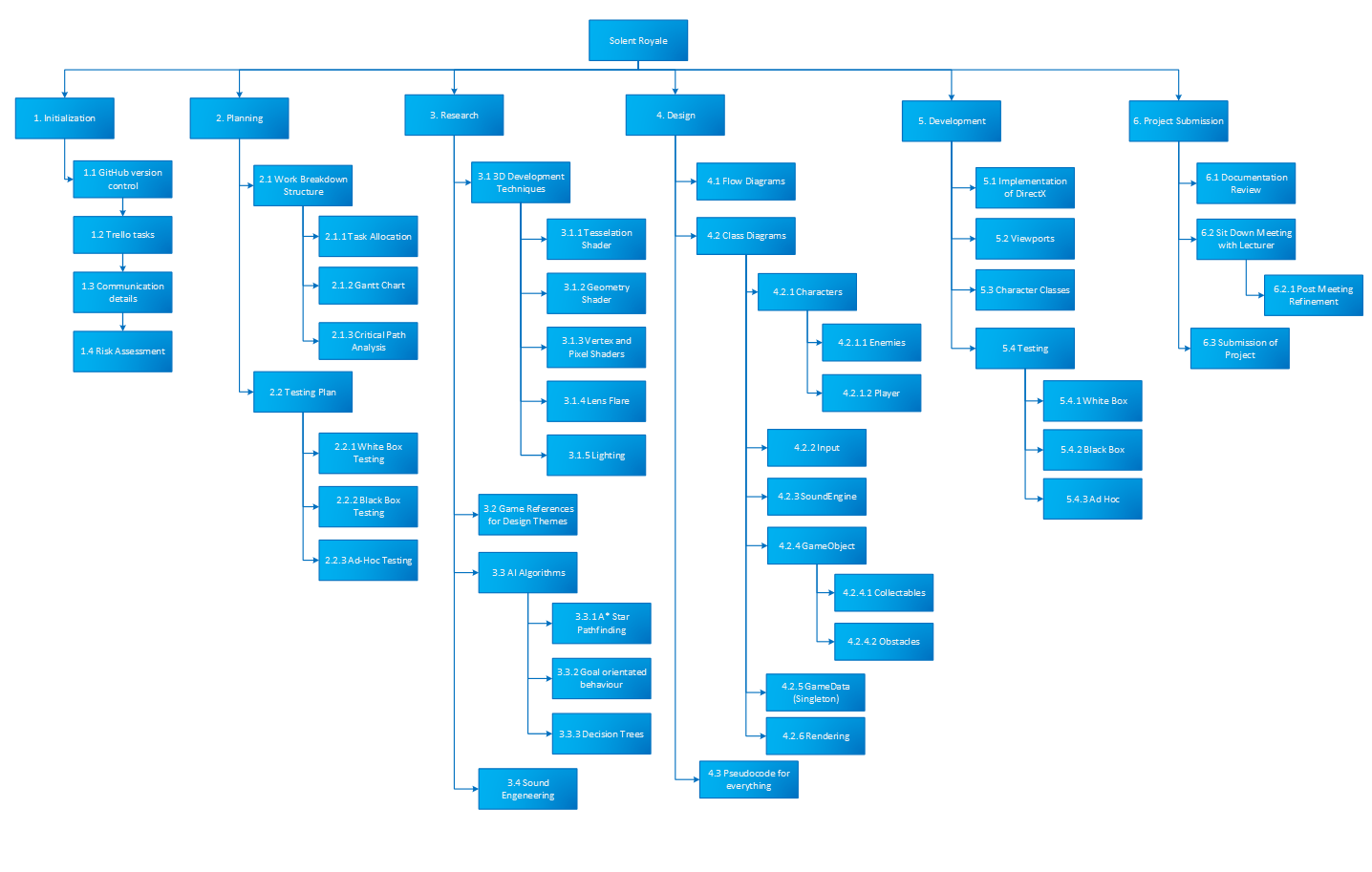
## B.2 Flow Diagrams





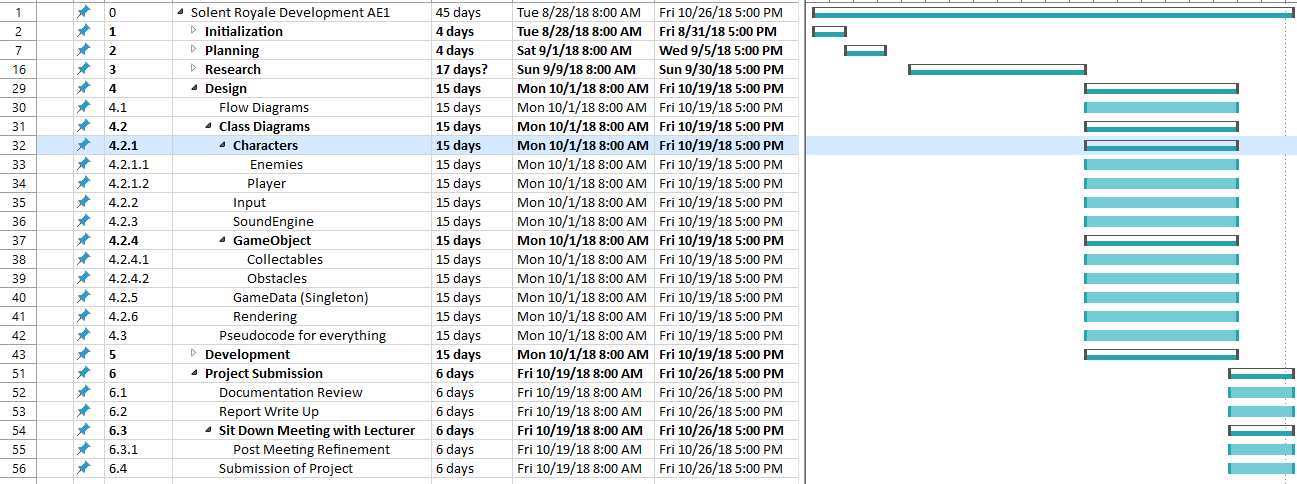
(Francis Jones and Sergej Frank)

## B.3 Work Breakdown Structure



(Francis Jones)

## B.4 Gantt-Chart



(Francis Jones and Sergej Frank)

## B.5 User Stories



(Sergej Frank)

# B.6 Pseudocode

(Sergej Frank)

## Main

function main()

spawn\_entities()

while(running)

update()

function update()

update AI, Input, Sound, Graphics. In that order!

(Sergej Frank)

## GameObject

function spawn (position)

set visible to true

set transform position to position

function render()

if GameObject is visible

draw()

virtual function start()

virtual function update()

function move(position)

position of gameobject = position

function collided(target)

if distance to gameobject is less than target collision\_radius + gameobject collision\_radius

(Sergej Frank)

## Collectable

function collected()

item is collected by a character and destroyed

(Sergej Frank)

## Obstacle

function collided(target)

if obstacle\_type is bomb

explode()

if obstacle\_type is bouncer

push\_back()

(Sergej Frank)

## Character

function cut\_target()

if hook of target is in range

play cutting animation

change state of target to falling

function throw()

if target is in range

throw projectile

if projectile hit is a character

character:get\_damage()

(Sergej Frank)

## Enemy

function attack()

if enemy\_type is grounder

throw()

else if enemy\_type is swinger

cut\_target()

(Sergej Frank)

## Player

function update()

input:handle\_input(key)

(Sergej Frank)

## Input

function handle\_input(key)

switch key:state n-m

case direction key

if player:state is crouching

move in direction with slow modifier

if player:state is standing

move in direction

if player:state is airborne

mid-air movement

case jump\_button

if player:state is standing

execute jump command

case shoot

execute shoot command

case c\_button

if character-state is standing or jumping

execute crouch command

case hook\_button:

execute hook command

(Sergej Frank)

## SoundEngine

function play\_sound(sample)

sample:play()

(Sergej Frank)

## GameData (Singleton)

function get\_instance()

if instance not null

return instance

else

create new instance of GameData

(Sergej Frank)

## Transform

function translate(position)

position = position;

function scale(x, y, z)

scale = vector3 with x, y, z

function rotate(yaw, pitch, roll)

rotation = quaternion with yaw, pitch, roll

(Sergej Frank)

## Command

virtual function execute(character)

modify character

(Sergej Frank)

## MoveCommand

function execute(character)

direction as vector3

switch key:state

case w

add transform:forward

to direction

case a

add transform:left

to direction

case s

add transform:backward

to direction

case d

add transform:right

to direction

add direction to character:transform

(Sergej Frank)

## JumpCommand

function execute(character)

add transform:up with multiplier to character:transform

(Sergej Frank)

## CrouchCommand

function execute(character)

play character:crouching\_animation

(Sergej Frank)

## AccSyncCommand

function execute(character)

(Sergej Frank)

## ShootCommand

function execute(character)

character:throw(target)

(Sergej Frank)

## Timer

Function deltaTime()

Return \_delta

Function totalTime()

Return \_total

Function start()

QueryPerformanceCounter(\_start)

Set \_current to \_start

Set \_previous to \_current

Set \_paused to 0

Set \_is\_paused to false

Function tick()

If \_is\_paused

Set \_delta to 0

Else

QueryPerformanceCounter(\_current)

Set \_delta to \_current - \_previous \* \_seconds\_per\_tick;

Set \_previous to \_current

If \_delta < 0

Set \_detla to 0

Set \_total to \_current - \_start - \_idle) \* \_seconds\_per\_tick