



X1337

by

Per-Arne Andersen

Supervisor: Christian Auby

Project report for DAT-220 in Autumn 2013

Faculty of Engineering and Science

University of Agder

Grimstad, 20.12.2013

Status: **FINAL**

Keywords: bullet-hell, sfml, c++, xml, graphics

**Abstract:**

Shooting games have been with us since the early 1960's. The principle of these games is to shoot enemies seen on the screen, and at the end fight a boss. Bullet-hell, a spin-off series of the shooting games, is a variation, which includes a high amount of obstacles. These games often have a storyline and a lore, which follows throughout the gameplay.

Topalan's Batsugun in 1993 is the first variation of the bullet-hell genre, and it was targeted to a more dedicated audience than in earlier shoot 'em up games. It featured immense difficulty and near impossible to complete. This genre grew more popular as the demand for challenge increased, but it never became a popular genre.

The first variations were much alike the usual shoot 'em up. However, as better hardware became available, developers had the possibility to create more intense encounters in their games. The genre has had many variations by different indie developers, and to date is the latest called Akai Katana Shin released in 2012.

Report Version Control

Version¹	Status²	Date³	Change⁴	Author⁵
0.1	DRAFT	2013-05-12	Initial Structuring	P-A
0.2	DRAFT	2013-06-12	Appending text, structuring	P-A
0.3	DRAFT	2013-07-12	Work on 2.4	P-A
0.4	DRAFT	2013-09-12	Work on everything	P-A
0.5	DRAFT	2013-10-12	Structuring and writing report	P-A
0.6	DRAFT	2013-10-12	More writing	P-A
0.7	DRAFT	2013-10-12	Finished first draft, Missing 7 and 9	P-A
0.8	REVIEW	2013-11-12	Review phase	P-A
0.8	REVIEW	2013-12-12	= =	P-A
0.9	REVIEW	2013-13-12	= =	P-A
1.0	FINAL	2013-19-12	Finished last review of the report	PA

1 **Version** indicates the version number starting at 0.1 for the first draft and 1.0 for the first review version.

2 **Status** is DRAFT, REVIEW or FINAL

3 **Date** is given in ISO format: yyyy-mm-dd

4 **Change** describes the changes carried out since the previous version

5 **Author** is the one who did the change

Preface

“X1337” is a project in the course DAT-220 at University of Grimstad in the ICT Department. The project work was done at home and at the university, the work is fully documented and logged via JIRA and STASH.

I would like to thank supervisor Christian Auby for all the guidance throughout the project period. In addition, a special thanks to MSC student Rune Borge Kalleberg, Associate Professor Morten Goodwin and the unofficial SFML IRC channel at [irc.boxbox.org:6667](irc:boxbox.org:6667) for feedback and suggestions throughout the project.

Thanks to Associate Professor Morten Goodwin, Professor Andreas Prinz and Assistant professor Astrid Stifoss-Hansen for the DAT300 bachelor report template.

Grimstad

December 2013

Per-Arne Andersen

Contents

1	Introduction.....	7
1.1	Background.....	7
1.2	Problem Statement	7
1.2.1	Priority Overview.....	8
1.2.2	Priority Specifications.....	9
1.2.2.1	Priority A	9
1.2.2.2	Priority B	10
1.2.2.3	Priority C	11
1.2.2.4	Priority D	11
1.2.3	Expected results	12
1.3	Literature review	12
1.3.1	Google	12
1.3.2	Stack Overflow.....	12
1.3.3	SFML Community	12
1.3.4	Cplusplus.com	12
1.3.5	Youtube.com.....	12
1.4	Problem solution	13
1.4.1	Project roles.....	13
1.4.2	General Approach.....	13
1.4.3	Tools and Software	13
1.4.3.1	JIRA.....	14
1.4.3.2	GIT.....	14
1.4.3.3	Visual Studio.....	14
1.4.3.4	Lucid Chart	14
1.4.3.5	Google Docs & Microsoft Office	14
1.4.3.6	Dropbox	14
1.4.3.7	SFML	15
1.4.3.8	Gantt chart.....	15
1.4.3.9	C++	15
1.4.3.10	PUGIXML.....	15
1.4.3.11	JAVA.....	15
1.5	Project Plan.....	16
2	Game Design.....	17
2.1	Main Menu	17
2.2	Stage Select.....	18
2.3	In - Game	19
2.4	Pause Menu.....	20
2.5	Enemies and Player	21
2.6	Summary	21
3	Technical requirements	22
3.1	Licencing.....	22
3.2	Class diagram	22
3.3	Game Flow.....	23
3.3.1	General Game Flow	23
3.3.2	World Lifetime Flow.....	24
4	Implementation	25
4.1	Game Shapes	25
4.2	Resource Handler	26
4.3	World	27
4.4	Background.....	27
4.5	Bullets and Bullet factory.....	28

4.6	Game Engine	28
4.7	Enemies and Power Ups.....	29
4.8	Player.....	30
4.9	Menu.....	31
4.10	Script System	32
4.10.1	Implementation	32
4.10.2	Stage editor	33
5	Validation and Testing	34
6	Results	35
6.1	Priority A	35
6.1.1	Player & Bullets.....	35
6.1.2	Enemies.....	36
6.1.3	Stages.....	36
6.2	Priority B	37
6.2.1	Difficulty Settings	37
6.2.2	Menu.....	37
6.2.3	Audio	37
6.2.4	Background Image.....	37
6.2.5	High score.....	37
6.3	Priority C	38
6.3.1	Power Ups	38
6.4	<i>Health Pack</i> :.....	38
6.4.1.1	Pulse Gun:.....	38
6.4.2	Stage Select	38
6.4.3	Special Effects	38
6.5	Priority D	39
6.6	Summary	39
6.7	Future Work	39
7	Group Evaluation.....	40
7.1	Discussion.....	40
7.2	Conclusion	40
8	References	41
9	Appendices.....	42
	External dependencies.....	42

1 Introduction

1.1 Background

The purpose of this project period was to develop, document and present a computer application with relevance to the DAT-220 course. We decided to develop a game based on the stress genre bullet-hell. The game should be fully functional with the ability to move, shoot and to destroy enemies in succeed the game. A script system with a wide set of functionality should be implemented for enemy interaction. There would be multiple types of enemies with unique set of attacks.

Focus of this release would be to give a technology review of the work accomplished under the project period. To succeed in such a project, a planning phase would be important. A planning phase, which focused on the main aspects of the bullet-hell genre. In addition, technologies used in the game should be taken account when planning the development process.

In the approach of solving such a task, there would have to be a strict set of rules while developing and implementing new features. A system would be in order for the development to go smooth. This system would be a link between the documentation flow and the code flow in the version control. It would then be easy to document work done on a specific task, which would increase the productivity when new team members emerge.

The game should not require end-user to install or modify existing software or hardware when using this product, neither should it pose a threat to the stability of the operating system.

1.2 Problem Statement

There was multiple requirements set in place to provide a good documentation framework and development framework of our product. A Gantt chart and a Priority list was made. These two components would be one of the primary frames for our project planning process. A secondary frame was also set up, which contained requirements regarding software and libraries, these requirements was not static and could be tuned if necessary. The priority list was considered the most important requirement, and piece of pre-project documentation this project had. It explains roughly how X1337 looks like, and what to expect in the first upcoming release.

This section will introduce the basis tasks for this project and give an overview of the features our game includes.

1.2.1 Priority Overview

The priority framework goes as follows:

A – Implementation, which is **vital** for the game to function

B – Implementation, which is **important** for the game to function

C – Implementation, which is **nice** but not required for the game to function

D – Implementation, which is not planned for first release, but possible to implement

Priority A

Player

Enemies

Bullets

Stages

Priority B

Difficulty Setting

Menu

Audio

Background image

Multiple enemy textures

High score

Priority C

Stage select

Power ups

Special Effects

Priority D

Online Multiplayer

Online High score

Multiple player ships

Achievements

Discrepancies

A discrepancy defined as “*differs from*” or “*missing*” and used as such throughout the report.

The game will possibly have discrepancies, which may or may not be in a further implementation, or a feature of the game.

1.2.2 Priority Specifications

1.2.2.1 Priority A

Player

The game will focused around a player character. Introducing player as a lone spaceship lost in space. Hostile spaceships quickly attack the player, and is now destined to fight his way through space. After a player succeed in fighting off the fleet of an unknown entity, will then be met by the fleet leader and forced to neutralize the threat. Player uses a gun to shoot and destroy enemies, and are rewarded with score points. These score points will then be compared to the high score list when a game has ended. Death occurs if the player's health reaches zero.

Enemies

There will be enemies, which function much like the player. They will move by themselves, and not interact with the player. Enemies is a fully scripted event, which follows a determined pattern. Following this pattern, the enemy will shoot at the player. An enemy might have different properties based on its type, and introduce different bullet patterns for the player to avoid. When hitting an enemy, health will decrease with the damage value of the specified bullet. Enemies may also be represented as bosses; these will usually have a great amount of health. Bosses have unique set of weapons that defines its bullet pattern.

When a boss dies the stage ends.

Bullets

Without this feature, it would not be much of a bullet-hell game. Either enemies or the player shoots bullets. It should be implemented additional bullets types for features like special attacks. Since the game will have a high number of bullets on the screen, a bullet factory should be implemented to reduce strain on the computer. A bullet factory will preload bullets and gives a scalable framework to work with. The bullet and bullet factory feature may very well be one of the most important implementations in the game.

Stages

Implementation of stages was important in this project. Stages, defined as "a unique game with different challenges". This definition implies that a stage should have a unique set of enemy paths. Stages should be in a script language or an information container. This container must have the ability to include all necessary data for the stage to work. Scripts is loaded into the game via a parser and processed for execution. There should be possibility to create new scripts and easily add this to the game. Implementation of new stages should require no work in game code after implementation. New enemy types or bullet types is not included in previous statement.

1.2.2.2 Priority B

Difficulty Settings

There should be implemented a difficulty setting. This setting will include “easy” and “hard” difficulty.

In easy mode, the player starts with some kind of handicap, while in hard mode the player will have no handicap. Health is measured in “hearts” and when depleted, ends the game in defeat.

Menu

When the player first starts the game, the menu will show up. Menu will be general for the whole game, meaning the same code should apply for all of the different menus. We can typically split up the menu into two groups: Lobby menu and In-Game menu. The player should be able to read credits, start a game or even exit the game from the menus. There was also plans to include an option menu for sound and resolution, but was later discarded because we found no use in such a menu.

Audio

Implementation of audio and audio effects in the first version of the game. There should be ambient sound while playing the game, and audio effects on the player and enemy. Audio related to the menu is not a necessity, but could be implemented if there is time.

Background Image

Background images should be implemented into the game. This background should use the swimming lane technique where the image repeat itself in a vertical movement. This technique presents a moving world, which increases the gameplay quality. This feature is not required if there is lack of time, but implementing of a static background is then of importance.

High score

The player should have the ability to earn high score if playing the game well. High score should be saved into a game save file. When a player earns the highest high score, the name of this player should be shown inside the game and outside in the high score menu. High score must be stored between game sessions, so the player can continue to earn better score at another playtime. The high score implementation is just at basic stage and does not have any further requirements.

1.2.2.3 Priority C

Power Ups

This feature should give the player the ability to pick up extra ammunition, weapons and health. These power ups appear as object flying through the screen, in vertical direction. Power ups should be implemented as a part of the already existing script standard done for enemies.

Stage Select

Stage select is an extension of the menu. When entering the stage select, a view with available stages should appear. From here, the user can select the desired stage. Stages represents with an image and a stage title. Each of these graphical blocks should be possible to interact with via the keyboard.

Special Effects

We consider special effects for this game graphics. By graphics, we specifically mean textured enemies, bullets and player.

1.2.2.4 Priority D

Online Multiplayer

An online multiplayer is possible to implement in such a bullet-hell genre. The goal is to give the game an online lobby where players can meet and play together. This feature is not considered subject for the first release.

Online High score

Expanding the high score to an online portal is also a goal, which is set as a low priority. This portal should be accessible via the game, but also via the player's standard web browser. The game will then be expanded with a game server, which serves this feature.

Additional player ships

Adding the ability to switch between player ships is something to implement. These ships would have different properties like textures, weapons and health. This feature should include a menu section where the player can preview current ship, and choose between other ones.

Achievements

Achievements, gained whenever an in-game goal is met. These goals is defined by the programmers of the game and will be simple tasks like killing enemies in a specific order, or killing an amount of enemies in a set timeframe.

1.2.3 Expected results

Expecting priority **A**, **B** and major parts of **C** to be implemented in the first version of the game. Goals in **D** sections are considered future work. The game is expected to have all implementation described in the first three priorities. Furthermore, implementation seen fit can be appended to the priority list.

Our product is expected to be fully playable at the end of the project period, and no new features is to be implemented after the set freeze period. This period starts last Monday before the hand-in.

1.3 Literature review

Literature is an important part of the pre-study for this project. There are many ways to retrieve information nowadays, and it is therefore likely that not all of the literature were covered in this chapter.

The only literature used in the project was the internet, below is a list of the most used literature in the project.

1.3.1 Google

Google is the most popular search engine in the world. Google is not considered a literature source, but rather the referrer to the source of the literature. Google provides us with great hits on the page Stack that is regularly used when stuck on a problem.

1.3.2 Stack Overflow

This can be considered the most regularly literature our project used. When an issue arise in the project, Stack overflow was there to help. Stack overflow is a forum where people meet discuss their main field of interest.

1.3.3 SFML Community

The SFML community was very helpful in providing literature to understand problems. They have a well-documented library, which helps beginners to get started with the library. SFML also have an IRC channel, which is open for everyone. Most of the people here are great and gives help if required.

1.3.4 Cplusplus.com

This page was used as a reference to the STL library in C++. This web page widely documented. It gives great information about the library and it would be very hard to learn STL without it.

1.3.5 Youtube.com

Youtube is a webpage, which serves entertainment and instructional videos to the user. This is a great webpage with a lot of information in most areas. There is a lot of information about C++ and SFML here, and is therefore a very good source of information for this project.

1.4 Problem solution

1.4.1 Project roles

Scrum master

Shahab Afsharipour & Per-Arne Andersen

Gitmaster

Shahab Afsharipour

Programmers

Gunnar Abusdal

Per-Arne Andersen

Shahab Afsharipour

1.4.2 General Approach

The project group decided to make a game in the project period. The game would be as mentioned a bullet hell game. There was many aspects to keep track of when planning for this game and it was important that we came to an agreement on how to approach the project period. If all of the requirements were to be done in time, we had to play together and all participate in the project. There had to be a good documentation flow on work done, which was a requirement from the supervisor.

Approaching the project period all team members would have to prepare their computers for development. Group members did also have to get familiar with the software used for development in this project period.

1.4.3 Tools and Software

Following tools and software was used in the project period:

- JIRA in project management
- GIT as version control application
- SourceTree
- Visual Studio 2013, development IDE
- Eclipse JUNO IDE
- Lucid Chart, UML diagram
- Google Docs and Microsoft Office for report documentation
- Dropbox

1.4.3.1 JIRA

A project management tool developed and maintained by atlassian. This tool was used to plan and develop our game. JIRA is a very strong tool development and documentation if used correctly. Kanban is the board of preference, and we can typically categorize this board into following steps:

1. Visualisation of workload
2. Limitation of the project dimension
3. Workflow and delegation of workload
4. Specified and detailed workload
5. Feedback of current issues in progress
6. Shared problem solving.

Our group used JIRA frequently in our project period. We used both the Kanban board and a customized board for a fast overview of the project state (X1337 Dashboard).

1.4.3.2 GIT

GIT was used in the development process of the project. GIT is a version control, which is much like SVN. Our GIT repository is hosted by UIA and we use this alongside atlassian stash. GIT ensures a good overview over the project code and tracks the contribution of each team member closely. In addition, branches will be available to separate code from the compliable code.

1.4.3.3 Visual Studio

Visual studio was used as a development platform. This is an IDE developed by Microsoft and includes C++, and numerous other languages. This was only the preferred IDE, but it is important to note that all team members had the opportunity to use an alternative IDE.

1.4.3.4 Lucid Chart

This is a HTML5 online tool for drawing of UML diagrams, flowcharts, mind maps and chart maps. Some of the team members used this tool heavily in the development process. It is a very strong UML tool when used correctly, and it is certainly one of the better tools used in this project period.

Lucid chart was used to create all diagrams found in this report, as well as those found in the pre-project report. It is a very handy tool, which made time-consuming tasks into minor tasks.

1.4.3.5 Google Docs & Microsoft Office

Google docs, used for documentation and pre-project report. Reason for this is that Google docs features the ability for document meeting. This is a feature, which allows multiple users to modify a single document at the same time. It also features safe storage, which was a very important feature for us. After a document was finished, we formatted it in Microsoft Office.

1.4.3.6 Dropbox

Dropbox was used to backup for all documentation written in Microsoft Word. In addition, the project source code, images and media was stored onto dropbox.

1.4.3.7 **SFML**

In this project, **Simple and Fast Multimedia Library** is used as the graphical library. This library focus on 2D graphics using C++ or another language with bindings support.

SFML provides a set of tools, which makes hard tasks trivial. It is important to mention that it does not only provide graphics, but also sound and network support.

SFML is hardware accelerated with OpenGL, which is a cross compatible software. We can typically divide SFML up in 6 different modules:

- **System Module:** handles multithreading if this is a desired feature
- **Window Module:** handles the window and user interaction
- **Graphics Module:** defines all shapes in the library. The most relevant module in our project is the graphic module.
- **Audio Module:** sound management
- **Network Module:** Network programming with sockets(TCP) or HTTP protocol

1.4.3.8 **Gantt chart**

All of these modules can be used independently of each other (Window and Graphics module are exceptions), this one of the features that makes SFML a very flexible and good library for development of 2d graphic applications. A Gantt chart was created in to divide the workload throughout the project period

Our Gantt chart made the basis for the project period, and had its purpose to give a framework of tasks to complete to a set date.

1.4.3.9 **C++**

C++ is the preferred language for this project. C++ is low-level language, which is extremely fast in terms of computer power. It is the preferred language when developing application such as games or graphical applications. SFML is included into the C++ project, and use these two together.

1.4.3.10 **PUGIXML**

I introduced pugixml, a lightweight C++ XML processing library. This library features extremely fast parsing of XML. Used to load and save configurations files via the easy to use framework built into the library. XML or “eXtensive Mark-up Language” is a language designed to transport and store data.

In our project, it was crucial to use some kind of standard for the configuration and game script.

1.4.3.11 **JAVA**

Java will also be a part of this project if secondary software is required for the project development.

1.5 Project Plan

It is of importance to have a project plan. This plan must be short and concise so that all group members have a clear line to follow throughout the project period.

In the project, Gantt diagram is the primary documentation for the project period. Important feature is recorded with a start time and an end time.

X1337 Gantt

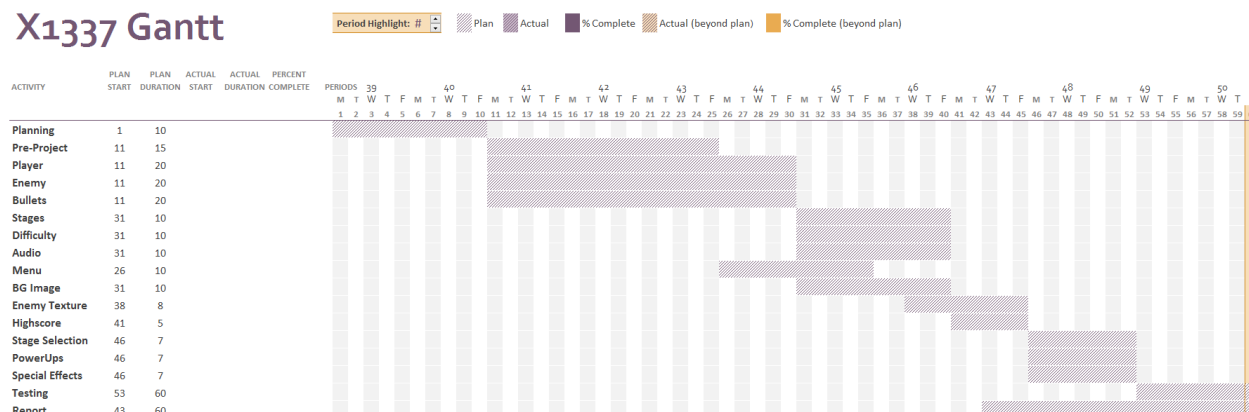


Figure 1 - Gantt chart

The above figure shows the expected timeline of implementation of each of the priorities. This a rough estimation of how our implementation timeline should look like post-project. It is important to note that the figure is only an estimation.

2 Game Design

Design is very important in the planning phase when developing a game [1]. My game implementation did not focus on the front-end, but I think the result presented is acceptable considering the short project-period.

This chapter will cover all of the important game design choices done in the project. Since game design is in general very broad, I will shorten it down by explaining with figures and diagrams. Game design is the most important features for an end-user, and it is therefore important to have a thorough plan.

2.1 Main Menu

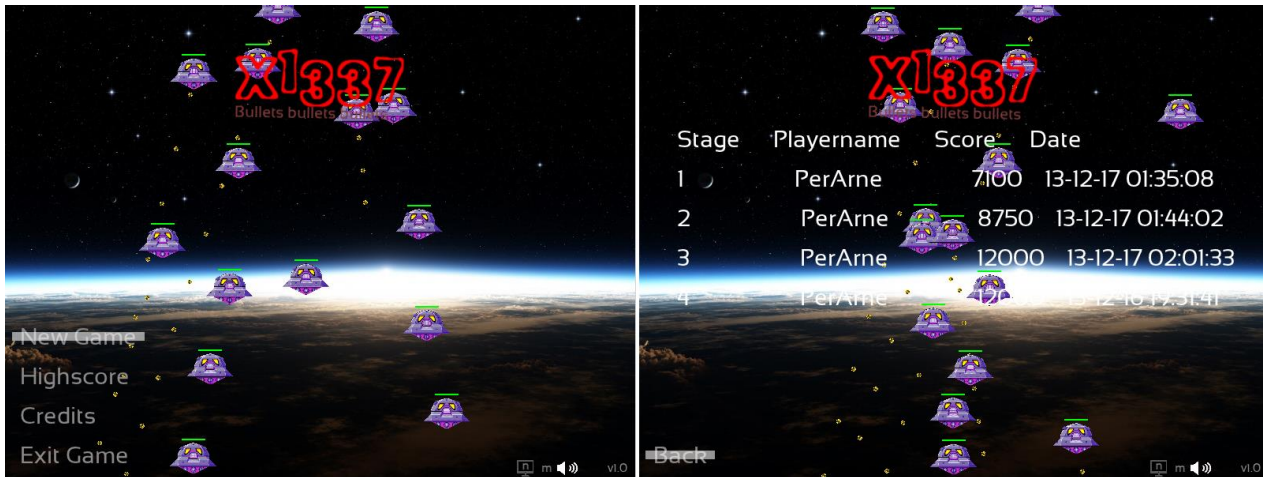


Figure 2 - Main menu and High score

When the player first loads the game, a black screen will appear with “Loading... Please Wait”. This normally takes 5-20 seconds depending on hardware. When the game is loaded, player enters the Main menu (**Figure 2 left**). From here, the player can reach all features the game has:

- Start new Game
- Read High score
- Read Credits
- Exit the game

The background is a star filled image. In addition, enemies are flying around on the monitor acting as a “demo” of the game. The purpose of the menu of the game is to give an easy to use interface, which allows the user to interact with the whole game. This is done successfully with the current system set in place.

When a player select the High score option at Main Menu, a new screen will show with High score table of the game (**Figure 2 right**). This table defines high score for all of the available stages. The player can then see if he is on this table.

2.2 Stage Select



Figure 3 - Stage Select

The player enters the Stage Select via the main menu (**Figure 2**). The page responsively changes to a grid view displaying each of the stages available.

The player can choose between going back and selecting a stage. To navigate between stages the player must use left and right keyboard arrow, and press the return button when selecting a specific stage.

When selecting each of the stages, a soundtrack is played, which represents the boss in that specific stage. Lore is displayed at the lower centre of the screen. The reason for having such features is to create a better atmosphere in the game, which ultimately leads to a better product.

In this example, the player selected stage 3. A new menu will now show up which is used to determine the difficulty settings of the game.

2.3 In - Game

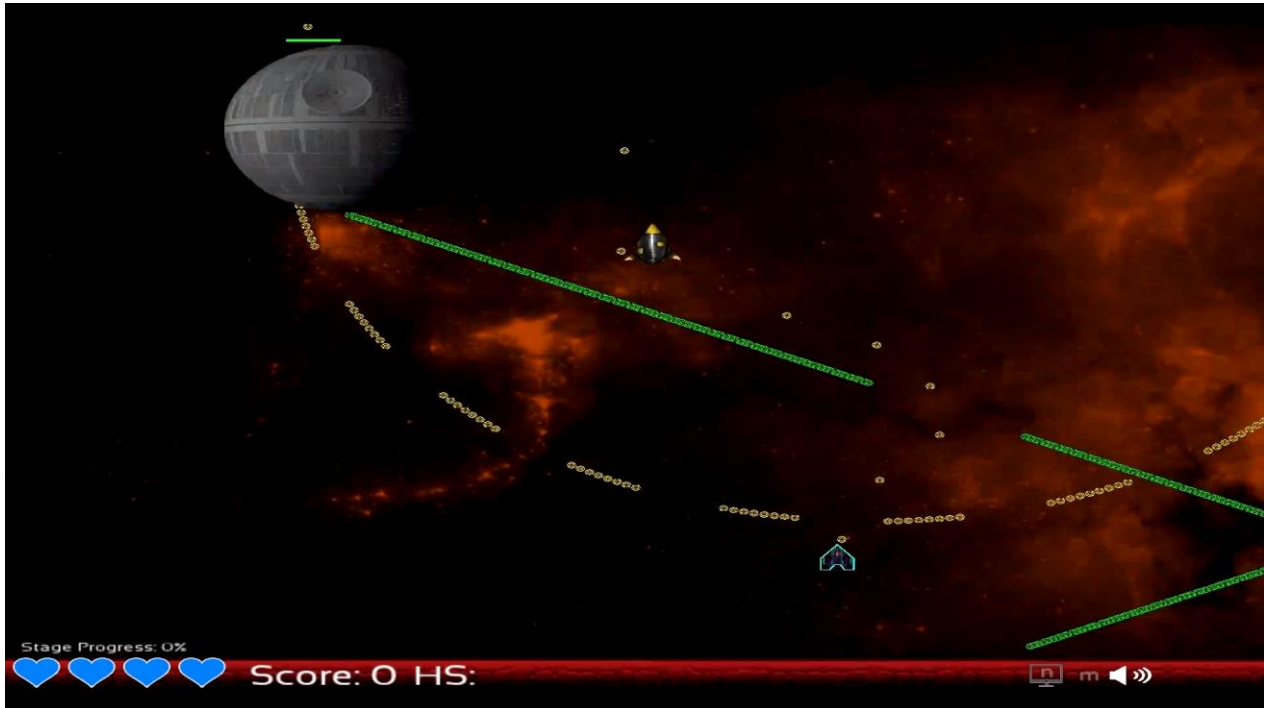


Figure 4 - In Game

This is an illustration of the in game encounter “Deathstar”, a character from Star Wars. The game have many technicalities that makes relatively advanced patterns, easy to develop. In terms of design choices, I decided to have the player menu at the lower section of the screen. This increases the visibility of information flow contra having this menu at for example the top or right side of the screen.

The background is an originally white space image, which is blended with colours. The background also moves in a stable pace throughout the game. This makes the game move alive, which is important for the player. Some bosses also do emotes when reaching a specific amount of health, or does a special attack.

The purpose of current in-game feature is to tempt the player to continue playing. Each of the stages available have a learning curve, and must be mastered before a stage can be completed.

Focus of the in – game, is to give an epic feel of the gaming environment and I think I have achieved this especially on the “Deathstar” encounter.

2.4 Pause Menu



Figure 5 - In-game pause and Game over

A pause menu is important for when a player wants to have a break from the game, it was therefore of importance that I implemented such menu into the game.

Figure 5 illustrates how the game is in pause and in game-over mode. These modes are much alike, but they have some differences. In the pause menu, we have the option “Continue” instead of “Restart Game” which is in the game-over menu. A black overlay is added to set the pause mood. It also acts to increase the game’s mood when resuming.

A third pause menu was also added when a player wins a stage, this menu is the same as Game-over, but includes a “Next Stage” option.

2.5 Enemies and Player

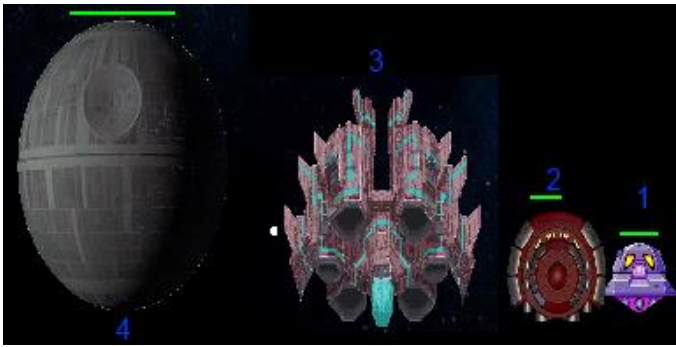


Figure 6 - Enemies



Figure 7 - Player and bullets

There is currently four types of enemy combatants in the game (**Figure 6**). The first two is regular mobs named “Regular (1)” and “Chubby (2)”. Number three “Boss (3)” represents the boss figure for stage one and two. Number four (4) is called the “Deathstar” is a space station from the popular movies of Star Wars. Deathstar is the final boss in my game, and is the stage that had a final polish before game release.

Figure 7 illustrates the player and the corresponding bullets implemented into the game. The player itself is a small ship encapsulated in a cyan outlining. The player also have two types of bullets. The first bullet illustrates the regular bullet, while the second bullet is the player’s special ability.

2.6 Summary

Game Design was a big step in the planning phase for this game. The game design was later revised to give generally better game and gameplay experience.

My game design focus on simplicity, as well as being good to look at. Bullet-hell games have a long history, so there was a huge focus to keep the retro aspects intact in the final product.

Considering that, there was no graphic developer in my team, the resulting graphics better than expected.

This release focuses on the gameplay of the Deathstar encounter. This encounter should easily show how expandable this game has become, all because of the dynamic design of my game.

3 Technical requirements

3.1 Licencing

When choosing a library for a project it is important to have knowledge about licences. In my project, two external libraries are used from which is both legally used without crediting the author of the library.

SFML is licenced under zlib/png [2] licence. This licence is generally just a formality, which protects the SFML author against any legal actions if the software were to harm the computer. SFML have additional libraries as dependency. All of these licences are under the BSD and/or LGPL. Both of these licences is just a “free to use” licence.

My last external library is PUGIXML, licenced under the MIT licence. MIT licence is free to use. However, its only restriction is that the copyright notice is intact [3]

3.2 Class diagram

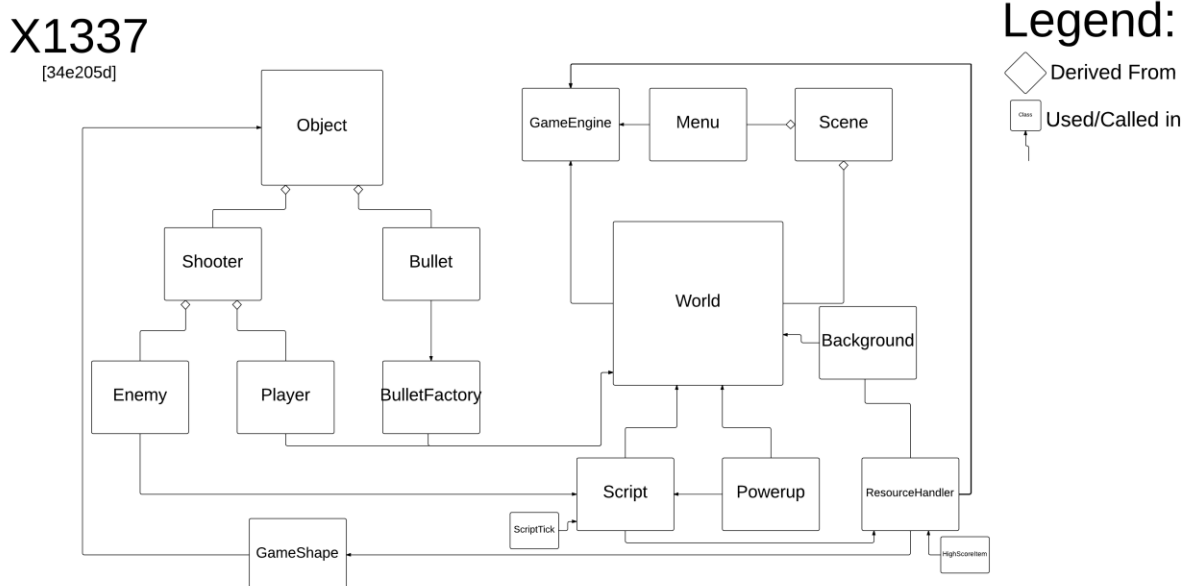


Figure 8 – Class Diagram

This is the final code structure of my game. All initialization in my game origins from the GameEngine class. This class contains the game loop, which is the main loop for the entire game. GameEngine initializes the menu and the World. The menu object is alive throughout the game, while world is recreated depending current state (described in detail later). World initializes all game objects needed in order for the game to work.

The game can typically be split into three types:

- Object Classes
- Scene Classes
- Independent Classes

Each of the first two base classes contains about 70 percent of the game, while the last 30 percent is independent classes in the game.

3.3 Game Flow

3.3.1 General Game Flow

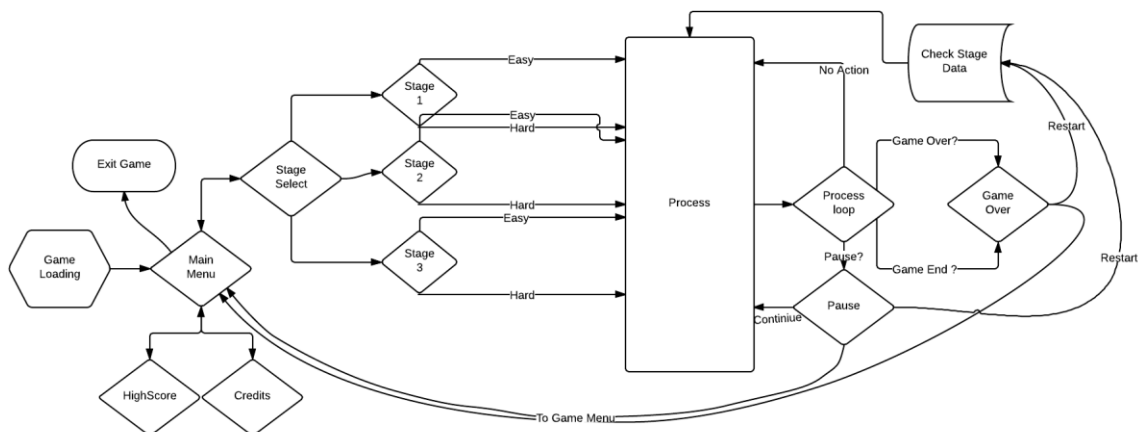


Figure 9 - General Game Flow

The illustration above (**Figure 9**) describes how the game flow is in the game. Consider this the “life” of each of the states. Starting from left to right, loading resources into the game. When this is done, Main Menu state is set. From here, the user can either go into High score, Credits, Stage select or exit the game. All of these choices will give the opportunity to go back to the Main menu except for the exit game.

Whenever a stage is selected, the player can no longer go back. The player is required to select a difficulty setting and start the stage. Now the game loop checks for player input. For each game loop iteration, there is polled for Pause State, Game Over state or Game End state. If one of these states happen, action is executed accordingly.

The game does this loop until an action sends the player back to the main menu.

When player is back on the main menu, he can do the same steps again, or exit the game.

3.3.2 World Lifetime Flow

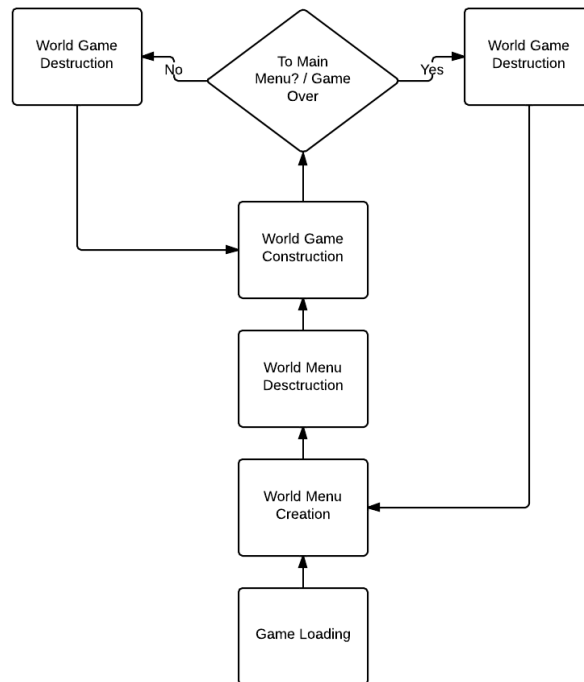


Figure 10 - World lifetime flow

It is of importance to design a proper World system when developing a game. There are multiple approaches to the problem of resetting used variables or recycling old data. I felt there were two options to go with in my implementation.

1. Create a reset handler for World. This would practically reset all objects within the game.
2. Destroy all object within the world, and then destroy the world. At last, construct the world and its appropriate objects.

I went with the second implementation, and it is arguably the better choice for my game.

Consider the fact that a bullet factory creates many bullets. These bullets are recycled throughout the stage and returned to the factory. I believe that such recycling may have the risk of leaking memory, even though it is unlikely in this case. It is safer to destruct the world and reconstruct it again to ensure that everything is in its original state. There is a slight performance hit when going for this model however. This hit is for the player, highly unnoticeable, as it is only about 500 millisecond of delay between construction and game start.

Figure 10 shows how the system works.

First, the game loads, the player gets into the main menu, and this is the first World load the game has. The player selects a stage, and world is deconstructed/destroyed. A new World is constructed with flags specifying the type of the World. Depending on which choices the user take throughout the game, there will be a new World construction either with **Game** flag set, or with **Menu** flag set.

4 Implementation

In this chapter I will be going over all of the implementation if the project. This being how and why I went for the discussed implementation. There will be example code and usage suggestion for each of the subjects. Implementation discussion will closely explain game behaviour earlier discussed in the Priority list.

4.1 Game Shapes

I used *sf::CircleShape* and *sf::RectangleShape* in the earlier stages of the game implementation. This was quickly discarded when hit detection became a problem. There was multiple problems where the bounding box would stretch textures on all of the shapes. This was not acceptable so an alternative implementation was developed. I named this implementation the *GameShape*. The *GameShape* implementation consists of a collection of different convex shapes (and concaves with the property to be correctly hit detected). The game currently has six defined shapes. These shapes are as follows:

```
const enum ShapeType{
    CIRCLE,
    TRIANGLE,
    STARSHIP,
    BOSS,
    PLAYER_SHIP,
    SQUARE,
    SHAPECOUNT
};
```

Figure 11 - Shape Enum

- Circle works much like *sf::CircleShape*, but is now defined as a convex polygon
- Triangle is a three point shape
- Boss is a relatively complex shape. It does not have the convex property, but splitting the shape gives us multiple convex shapes.
- Square is an alternative implementation of *sf::RectangleShape*. This is based upon scaling a 1^2 sized square.

The class itself consist of **three** different constructors

```
GameShape(ShapeType shapeType); // Other
GameShape(ShapeType shapeType, float size); // Triangle
GameShape(ShapeType shapeType, int radius, int pointCount = 5); // Circle
```

Figure 12 - GameShape Constructors

Each of these constructors defines different shapes. It is not required to create a new constructor when a shape is implemented. These constructors are rather help to the programmer when implementing new shapes.

```
new GameShape (GameShape::ShapeType::CIRCLE, 40, 30)

new GameShape (GameShape::ShapeType::STARSHIP)
```

Figure 13 - GameShape object

When using a GameShape you create a new object of the shape. Looking at **Figure 13**, we see that it is very easy to declare shapes. These two lines describes respectively the *DEATHSTAR* enemy and *STARSHIP (Regular)* enemy.

GameShape is not described in the priority list, but is categories under **(A)** Player, **(A)** bullets and **(A)** Enemies.

4.2 Resource Handler

The resource handler is easy to use, highly expandable and customizable when required. It is based on Christian Auby's Resource handler from the pong example available in git. My implementation is however much different from the original resource handler. The main difference is the usage of `std::map` for each of the available resources.

```
// Fonts
{
fontList[Fonts::COMICATE] =
"assets/fonts/COMICATE.ttf";
fontList[Fonts::SANSATION]=
"assets/fonts/sansation.ttf";
}
```

Figure 14 – Font declaration

The variable name is not to be confused with `std::list` as this is a map, which maps enumerable values the font file path. It is important to note that I keep two maps, where the first is only the file **path** information, and the second map is the data container for that specified resource.

My solution on the resource handler gives us the wished flexibility and the scalability required in the game. Appending a new resource is simply to append a new path record in the well organized `void ResourceHandler::init()` function. After this, you simply load the resource with ether of the getters assigned for that resource (i.e.: `sf::Font& ResourceHandler::getFont(ResourceHandler::Fonts query)`).

It is worth to mention that all of the resources is kept in their container throughout the game; they are never reloaded in any way. Total memory usage is approximately 300 megabytes. This is an acceptable usage when preloading all resources before the game starts.

The implementation also allows for direct usage of a resource. In example, I use sounds directly with the `getSound(ResourceHandler::Sound::X).play()`. I can then retrieve the same sound reference with the exact same line of code if additional manipulation of the sound is required.

The resource handler gives a highly scalable intermediary between disk resources and game resources. Without such a system, there would be risk of choppy gameplay when loading new resources.

4.3 World

World implementation binds together all of the objects in the game. Bullet, Powerup, Enemy or Player is defined as objects, and hereby referenced to as such.

“*World*” inherits properties from the Scene class, which forces the World to implement following functions:

```
virtual void draw() = 0;
virtual void process() = 0;
virtual void input(sf::Event&) = 0;
```

Figure 15 - Pure virtual Scene.h

These three function is the foundation for our game world.

Draw takes care of drawing all of the objects onto the screen. It does not do anything besides drawing. It is an iterative function with the complexity of $O(n)$

Process takes care of all data processing in the world. It is important to note that because of the structural nature of the game, all object have a process function. This makes it very easy to create generic solutions for processing.

Processing takes place in a for-loop for each of the object types. In addition, I have a Script processor running, which takes care of spawning Enemies and Powerups.

4.4 Background

This is one of the smaller implementation in the game.

Background works like this:

- **Primary** image is set to position 0, 0 and stretches to `window.width`, `window.height`
- **Secondary** image is set to “0 - `window.width`” and “0 - `window height`”.
- Both images is translated 1 pixel in y direction every frame
- If one of the images hit `window.height` as its Y position its reset to 0 - `window.height`
- This process is repeated

This is generally all there is to the animated background. Note that it is fully possible to deactivate the scrolling function of the background. This is done with sending a *Boolean* in the second parameter of *addBackground* function

4.5 Bullets and Bullet factory

Bullet implementation is small but a very central part of the game. The bullet inherits `process()` from the Object class. Bullet also uses `draw()` but this resides in the object class (not pure virtual).

Bullet types are based on a on an Enumerable which decides the bullet shape and damage. This is set in the constructor with a simple if-else statement.

A bullet has flags determining its owner. This owner will go in correspondence with the hit detection. This is used to reduce the processing load on hit detection (do not calculate hit detection for same enemy types).

Current bullets support two types of movement. Rotational and linear movement. Rotational movement is defined in degrees. A flag called *deg*, which if set to -1 sets the specified movement to linear.

It is important to remember that bullets is merely a subclass of the bullet factory, as it is the bullet factory who owns all of the bullets (correctly speaking World).

Whenever a bullet is put back into the bullet factory, it calls the reset function, which resets the bullet to factory settings.

Bullet factory is as mentioned the superior class to bullets. It constructs bullets and keep them safely contained inside the bullet list

The available functions in the bullet factory is as follows:

```
void returnObject(std::unique_ptr<Bullet>&);
std::list<std::unique_ptr<Bullet>> requestBatch(int, Bullet::Type);
std::unique_ptr<Bullet> requestObject(Bullet::Type);
```

Figure 16 – Bullet Factory Central Functions

These few functions is the foundation of the bullet factory. The first function returns an already used bullet back to the factory. The latter two functions is getters of bullets. I made a batch function if there ever was need to spawn multiple bullets in a single frame. Moreover, I found this function very useful when implementing the *Deathstar* boss.

4.6 Game Engine

Game engine is based on states with three main functions respectively named `void input()`, `void process()` and `void draw()`. These are all functions called in the game Loop function. My game loop uses a fixed time Step, which is set to 240 frames per second. This loop reduces load drastically when comparing it to a time-based loop without this time step limitation.

The time step loop works like this:

- First, start the game clock
Check if the game clock has exceeded the set time step
- Do all game processing
- Subtract a time Step from the elapsed time (Because a frame takes one timeStep)
- Then draw the game.
- Repeat until game exit

With this method, we ensure smooth gameplay even on very poor computers.

Our draw, input and process function is based on state. Depending on the state the player is currently in, the appropriate processes will run. My implementation of this problem could have been better as it tends to be messy with all of the states. I do however think that our implementation provides a good framework to build the application on, even if it lacks the wished structural result for my game.

4.7 Enemies and Power Ups

Enemies and PowerUps is closely related classes, and in an earlier implementation the same class. I felt after a while that having PowerUps combined with the enemy was generally a bad idea, so I separated it into two classes. To start with enemies, these have two elementary functions in order to work, these are:

```
void Enemy::shoot(int shoot)
void Enemy::process()
```

Figure 17 - Enemy elemental functions

The shoot function takes an integer, which defines what type of shoot pattern it is going to use for the upcoming (current) frame.

It goes into and through a series of if-statements in order to produce the correct pattern for the specified enemy type.

This function take use of the bullet factory, and uses heavily the “request” functions from the factory.

After the `request` function is executed, the enemy will have bullets. These bullets must be correctly configured before added to the bullet list. Information about colour, and start position is set by the enemy and it is then added to the list for processing.

PowerUps is the Enemy class stripped down to fit the requirements of a PowerUp. A PowerUp does not use a list of paths, but instead only a single start point, which is, uses to project a path down the Y-Axis.

This is generally everything these classes do. They have no further computations except for their own Movement.

4.8 Player

The most central implementation is the player class. This class defines the general interface between the game and the person behind the keyboard. It was therefore a high priority to develop a robust implementation of the player.

In the constructor in the player class, I define the shape and texture of the player. Furthermore, I have an additional input handler to handle the mouse movements.

I decided to give the movement an alternative approach that normal. Generally, SFML does not support locking the mouse into the game [4], but we made an alternative system, which can act as the desired feature.

Implementation of the input handler goes as follows (ref: `void Player::input(sf::Event&)`)

- Get the coordinate of the mouse
- Check the difference between this point and the centre point of the screen
- Calculate movement based on this difference
- Reset mouse to the middle

This is generally all it is to the input handler.

Furthermore, I have a handler for left and right mouse click. If the player clicks on one of these two buttons, a bullet will be pushed to the bullet list with its directional vector in the positive Y-axis.

The texture and shape of the bullet depends on the button clicked.

I also define special draw cases in the player class, specifically called `void`

`Player::drawStats(std::list<std::shared_ptr<HighScoreItem>>&)`.

This function tunnels via the World class in order to reach the state handler in the game engine. Reason for this being a requirement is that stats require a different view, which I felt that was easier and structural wise to implement that way.

The player is managed in the same list as Enemies. Reason for this being possible is that the Shooter class has the following enumeration:

```
const enum ShooterType
{
    PLAYER,
    ENEMY
};
```

Figure 18 - Shooter Type enumeration

This enumeration defines if it is a player or an enemy. I could if required cast the shooter to a player when processing, but that was not required in current implementation.

The general implementation of the player was carefully planned and should perform very well in all game situations.

4.9 Menu

The menu implementation is not the best work done in the project. Reason for this is the messy implementation of states. Even though it have its design flaws, I think that the menu works very well, and the user will not notice the disappointed the developers have. However, it is clear that some parts of the menu is a bit verbose.

One of the better aspects of our menu is the `void Menu::loadMenuOptions()` which can take in any menu and draw it. This is as general as it gets, and I are very satisfied with this implementation.

The code found in the menu is highly customizable. It based on switch cases that checks which case the user are in and changes process the menu accordingly.

Our input handler is also something that is recognizably one of the better code in the menu implementation. It is clear and concise, and have all the necessities to work properly.

4.10 Script System

4.10.1 Implementation

Script system was a big part of my game implementation. Since our game is highly based on enemies, which shoots bullets at the player, I had to implement a stable script system for the game. The first implementation of the script system was made in a custom defined script language. The syntax on this was:

- Move(50,50)
- Shoot(1)
- Move(10,10)
- Shoot(0)

I quickly realized that this was not sufficient for the needs of the application, so I had to go back to the workbench. After a week of thoughts, the new system could be implemented. The new system was based on XML where the structure is as follows:

```
<Script>
  <Name encounter="1">STAGENAME</Name>
  <Repeat>BOOL</Repeat>
  <Enemies>
    <Enemy>
      <Type>ENEMYTYPE</Type>
      <Delay>FLOAT</Delay>
      <Path>
        <_0 shoot="0" x="-12" y="94"/>
        <_1 shoot="1" x="240" y="51"/>
      </Path>
    </Enemy>
  </Enemies>
  <PowerUps>
    <PowerUp>
      <Type>-11</Type>
      <Delay>2000</Delay>
      <Repeat>0</Repeat>
      <Path x="450" y="0">
    </Path>
    </PowerUp>
  </PowerUps>
</Script>
```

Figure 19 - Script format

This is parts of the *DEATHSTAR* encounter and it is practically a fully function script.

We define script into two sub nodes Enemies and PowerUps. Each of these nodes may contain multiple Enemy or Powerup nodes.

I think the system developed worked very well. It was fast to parse with the PUGIXML library. XML files is parsed in the *ResourceHandler* via the `loadScripts()` function.

When a script is loaded into memory, it is processed via an process loop which is hooked to `World::process()`; The script stops to process and ultimately the World when the queue is empty.

4.10.2 Stage editor

In addition to the flexible script system developed inside my game, I felt it was necessary with a script editor for creating stages. However, this was developed and released in Java.

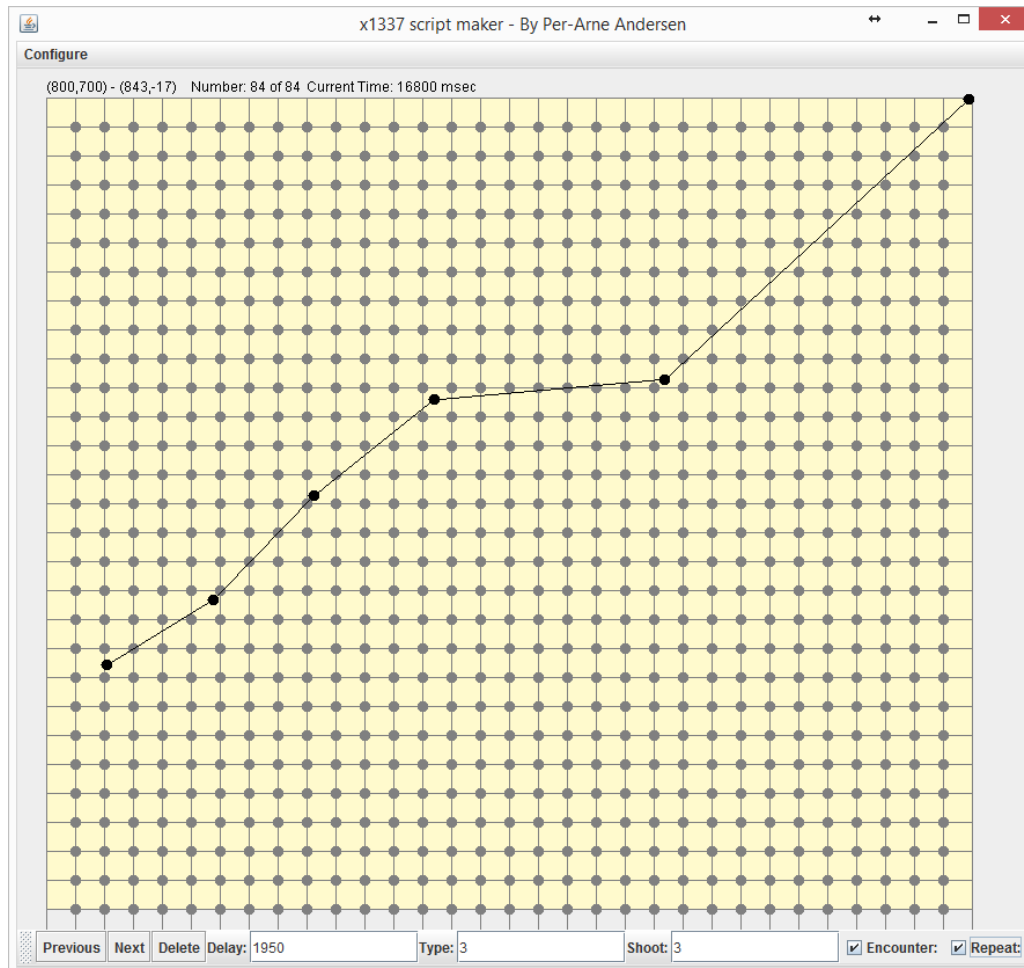


Figure 20 - Script maker 1.1

The stage editor is simple to work with. You add a set of points and press enter; you will then have a path for your enemy. The upper right corner has the following information (left to right):

- Window dimensions and current mouse position
- Current slide / Total Slide
- Current time slice

Since the script is based on delays, we can also measure the time a script takes to execute.

In the bottom left, we can navigate through each of the defined slides. The first textbox defines the delay between current slide and the previous; this defines the stress factor of the game. Type defines the enemy type, which can typically be *DEATHSTAR*, *CHUBBY* or *BOSS* (**1, 2, 3**)

Lastly we have which type of shoot pattern the enemy should use in **CURRENT** path (Between the last placed point, and the one you are planning to place). For the checkboxes, we define if it is a demo stage or an encounter and if the enemies should repeat their Paths or not. The latter is normally true as this increases the stress factor of a stage. To export a stage press SPACE.

This application is very bad structural wise. There was no intention to plan for this application. Expect bugs in this script editor!

5 Validation and Testing

An application should be thoroughly tested before a release. My game was tested a lot before handing in the product. Even though I did not get to use the boost unit test framework in the application, I have a very good idea on how to implement this later.



Figure 21 – Jenkins Logo

If I were to use boost unit testing framework, it would be in correspondence with Jenkins, a Hudson fork. Jenkins provide continuous integration services aimed at software development. Continuous integration can be summed up to automatic testing and building of software.

I think it is bad that this was not implemented, but it was also never covered in the initial planning phase, which is the probable cause for exclusion.

I still wants to stress that the game did not have any noticeable bugs in the first release. Taking this into consideration, I can assume that Validation and Testing was a success in the project, despite the lack of a unit test framework.

Testing was done by setting and trying different values in the game. If some of these manual tests failed, I looked into the issue. The game was also stress tested with stages having a very high density of bullets. Testing was done in many ways a unit-test would do it.

The below tests was done before hand-in:

Test	Status
Check player border collision	PASS
Check player shoot timer's, (Try to mess up timers)	PASS
Enemy Dies with the correct health	PASS
Player Dies with the correct health	PASS
Audio plays when supposed to	PASS
Enemy movement is correct	PASS
Highscore is saved correctly with correct data	PASS
Stage select works fine, without any bugs	PASS
Stage Quits when enemies are dead	PASS
Deathstar emotes is correct	PASS
Next stage navigates correctly	PASS
No memory leaks is to be found	PASS
All graphics shows correctly	PASS (Not on RDP)

6 Results

A list of priorities was set up as a part of the pre-project report. This chapter can be read as a checklist over the results seen in the final project. This checklist will comment on discrepancies in the final product and if any, list the reason for the discrepancy. A summary of what is implemented will also take place in each of the checklist points. Note that this chapter only scratches the surface of the resulted implementation. This is required so this chapter does not lose its purpose. It is expected that the reader have knowledge of each of the priorities from **chapter 1.2.2 Priority Specifications** (redundancy)

6.1 Priority A

6.1.1 Player & Bullets

I successfully implemented player without any problems. The player receives a set amount of health before the game starts. The player appears as a small spaceship. It is important for the player to have a clear view of the player object, so a cyan outline was added to the sprite. Player's movement is done with the mouse. If a wall is hit, the movement stops and the player cannot move further in that direction.



Figure 22 - Player and Bullets

There is no discrepancies in the player implementation. There is however, some additional features added which was not in the original priority.

Additional features is as follows:

- Player outline
- Texture
- Abilities (This is closely linked to power ups)

6.1.2 Enemies

Enemies was implemented as described in the pre-project report.

Multiple enemies was introduced to the game. These enemies have unique abilities and the implementation of *Deathstar (4)* led to the implementation of expanded shoot handling in the *Script handler*. All of the enemies follow a specific pattern specified by the flexible script system implemented in XML.



Figure 23 - Enemy Types

6.1.3 Stages

Stages was successfully implemented as described in the pre-project report, Stages defines enemies patterns, which in becomes the stage. I am very satisfied with the outcome of the stage system. Even though I had trouble developing “the perfect” script system, I think that the current system does the work very well.

I am very satisfied with XML as the language for the script system. This way, I were able to generalize all of the configurations in the game.

The system behind the scenes is very scalable, flexible and easily expandable in the future.

6.2 Priority B

6.2.1 Difficulty Settings

When a player selects a stage, a question prompts for which difficulty the player wishes to play on. Currently implemented difficulty settings is hard and easy.

Difficulty is reflected over how much health the player starts with.

It is safe to assume that the difficulty setting was properly implemented and that there is none missing features from this implementation.

6.2.2 Menu

This implementation turned up to be the most verbose code in the entire game. The menu system works great and it was implemented after the specifications given in the pre-project report. As an extra neat function, an in-game demo was appended to the main menu. There is no discrepancies in the menu system for the user, it could however, been implemented in a more orderly fashion.

6.2.3 Audio

I added space theme sounds to the game. This means ship death, shooting sound and background ambience. Sound is loaded via the flexible resource handler. Sound is then easily played via the `getSound(ResourceHandler::Sound)` function. I consider audio fully implemented in the game.

6.2.4 Background Image

This feature was originally implemented early on in the project, but an idea in the planning phase was adopted into the priority list. The swimming lane technique was introduced in the late project phase.

In addition, the background image changes background colour. This makes the graphical aspects of the game even better.

6.2.5 High score

The high score feature was implemented without any problems, it is based on XML which is accepted by the project standards. Chapter 1.2.2 defines this problem as “This high score is saved into the game save file”. It is therefore safe to assume that the high score does not lack any functionality beyond the scope of the pre-project definition.

6.3 Priority C

6.3.1 Power Ups

Implementation of this feature happened at the end of project period. Because of the scalable script system, this was easily implemented with the existing functionality in game.

Two types of power ups was introduced to the game, respectively Health Pack and Pulse gun,

6.4 Health Pack:



Health packs appear randomly throughout a stage. By randomly it implies that it is scripted, but will seem random to the user. These health packs will increment player health by one if picked up. If the player fails to pick up a health pack, it will despawn depending on the status given by the script (respawn = 0)

6.4.1.1 Pulse Gun:



Pulse gun is a gun, which gives the player temporary new abilities. This ability pulsates bullets from the player ship. This is an extremely valuable upgrade in tight situations.

To summarize this implementation, it is safe to say that this is a fully implemented feature.

6.4.2 Stage Select

This priority is closely related to the menu priority in the **B** section. It was implemented in the already existing code of the menu. Each of the stages have an image of the end boss. In addition, sound is played when selecting a stage. This sound represents the end bosses voice.

There was also added lore to the stage select. Every stage has its own lore that goes in correspondence with the stage. It does not have any deep meaning, but it is lore nevertheless.

6.4.3 Special Effects

Special Effects were defined as textures and sound effects in our pre-project report. I have implemented this via the scalable resource-handler the game has. Whenever a new feature is added, special effects are easily appended to the implantation.

The game have some special effects in the Deathstar encounter. I consider different type of enemy bullets and emotes the boss does as special effects.

6.5 Priority D

Proceeding the priority D I have decided to not comment any results on these priorities. This is due the lack of implementation on any of the D priorities.

I never intended to implemented any of these priorities, but rather have them as a backup if there was lack of work before the project period was over. I did however never have this problem.

These priorities is however possible to implement in a later release.

6.6 Summary

Summing up my discoveries in the results, I can say by certainty that our product implementation was a success.

With few discrepancies, and the additional features added to the original game plan, makes my product better than first expected.

I did not focus on getting **D** priorities implemented into the game. The reason for this is that I found these priorities to cause very big challenges in restructuring for the use of networking.

I am generally satisfied with the result produced throughout the project-period.

Documentation was also something that was important in the produced result. The game was documented with Doxygen, and this can be found at <http://persoft.no/ftp> and in the appendices. Also the doxyfile is included in the project files.

6.7 Future Work

For future work, I will start by implementing all of the D priorities described in the report.

An interesting idea I have been thinking of is to develop a Game mode, which is a “mmo” extension to the game.

This mode would have a game server, which was updated in real-time. It could hold up to 512 player's and would scale accordingly by the amount of players online. The game would consist of a much bigger world than in current implementation. After a while, a world boss would spawn. All players would then have to contribute in order to kill the boss.

Another thing I wish implemented is more special effects. Specifically particle effects and explosions when enemy ships die. I would use the **Thor** [5] library made by the SFML community to implement these special effects.

The quad tree implementation is also something I would like to implement into the game. Since this was discarded for now, it is one of the things I see as likely to be one of the first things to be implemented after the project period.

7 Group Evaluation

I will in this chapter discuss and conclude the project period. There will be a discussion regarding the earlier group and myself as an individual group in the last weeks of the project period.

7.1 Discussion

The group started out as three-man group, but was split up by request from me in the last couple of weeks. This was due to the uneven workload done by each of the team-members.

This is something that had its effect on the product, and I am certain that the product could have been even better with a group with evenly distributed workload.

The learning curve have been steep, but it is arguably something that I think was worth my time.

There is many things that could have been better, for instance the planning phase. A planning phase would increase the productivity and arguably give the project a better finish. This must not be misunderstood, because the product became very satisfactory, but there is always things that could have gone better.

Generally very satisfied with my own work in this project period. Even though the group did not work out as planned. This is something, which I think everyone have learned from, and it will then be something, hopefully neither of us will experience again.

7.2 Conclusion

At the very end, I conclude with the project being a success, considering that everything in the A to C priority were implemented. The final product is very good documented, and I consider this being a plus because it allows new developers to understand the code.

Summing the project period up I think it has been a very educational period. Most parts of the project have been a very interesting and fun to participate in.

8 References

- [1] I. ARTIFICE, "INTELLIGENT ARTIFICE," [Online]. Available: <http://www.intelligent-artifice.com/2007/10/why-game-design-is-important.html>. [Accessed 19.12.2013 19.12.2013].
- [2] SFML, "SFML Home Page," 10 12 2013. [Online]. Available: <http://www.sfml-dev.org/license.php>.
- [3] O. S. Org, "Open Source Initiative," [Online]. Available: <http://opensource.org/licenses/MIT>.
- [4] Mario, "Github," [Online]. Available: <https://github.com/LaurentGomila/SFML/pull/396>.
- [5] SFML. [Online]. Available: <http://en.sfml-dev.org/forums/index.php?topic=4646.0>.

9 Appendices

External dependencies

- Pre-report - Forprosjektrapport DAT220-X1337 v2.pdf
- Time Sheets from JIRA - time-sheet1.png && time-sheet2.htm
- Meeting Minutes - /meeting minutes
- Power Point presentation - X1337-PP.pptx
- Git log - git-log.log
- Jira issuelist - Filter for DAT220 2013 - X1337 (JIRA).htm
- Class Diagram - class-diagram-small.png && class diagram.png
- Game Flow Diagram - game-flo.png
- World Scope Diagram - world-scope.png
- Doxygen – <http://persoft.no/ftp> && /doxygen