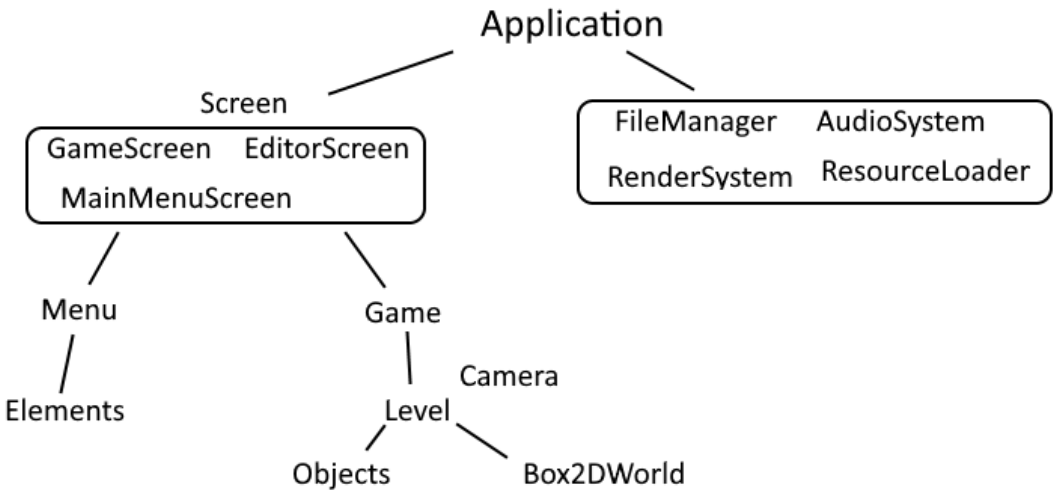
# Project plan Team Angry Birds Group 4

## Overview of the project

The game is an Aalto themed spin-off on the original Angry Birds game, where instead of birds, the player hurls Aalto students from different guilds. The goal of the game is to destroy the fuksis (first-year-students) hiding in the fort, but without destroying any beer bottles that might be in the level.

In addition to the classic game, there are two planned game modes. Time trial, where the player is given a time limit that also affects scoring, and endless mode where the player can recruit new students by hitting them in the level.

## Basic structure




The following diagram is a rough sketch of the basic class structure, and ownership between objects in the program. This also means that the lifetime of objects higher on the tree is guaranteed to be longer than the ones below. The program will use smart pointers and RAII principles in memory management. As such, if an object higher on the tree is destroyed, the whole subtree of objects is also destroyed.

The top level object is an instance of Application, that uses SFML Window module to create the main GUI window for the program, and handle events. It has an active

subclass of Screen (such as GameScreen or MainMenuScreen), which defines what content is currently shown and active. Application also has a loop that is executed at a target framerate (for example, 60 fps) and is used to refresh the active screen. Application also constructs a single instance of a FileManager, AudioSystem, RenderSystem and ResourceLoader, and other similar utility classes. These systems abstract underlying SFML code, and offer functionality such as drawing sprites, playing sound effects etc. AudioSystem and RenderSystem depend on the ResourceLoader, and the ResourceLoader depends on the FileManager.

Here is the basic concept. The program handles resources, mostly sprites and sound effects by preloading a single instance of them from file, that is stored in the ResourceLoader. The physical resource can be accessed by classes like the AudioSystem and RenderSystem through an unique integer ID. Since the resources that the game needs are known at compile time, there can be an enumeration containing all the IDs for all usable sprites and sounds as named constants. When writing game logic, objects such as wooden boxes and such don't directly deal with images, audio clips or filepaths, nor do they have to care about that. They instead use the id, which RenderSystem and AudioSystem can recognize and use to identify the physical loaded resource. This way, resources are never loaded twice, and they are guaranteed to exist, and are never leaked. In the case of a missing image file (or similarly audio etc.), the id will instead point to a missing texture image in the ResourceLoader.

Screen is an abstract class, and subclasses of it define what content is shown when the screen is active. This content can come in two categories, Menu content and Game content. A Menu consists of Elements, such as buttons, textboxes, labels and such. These elements have a rendering method that takes a const reference to the RenderSystem as a parameter, and they use the provided functionality to draw themselves accordingly. Their position and size is stored as a percentage, in comparison to the total size of the window.

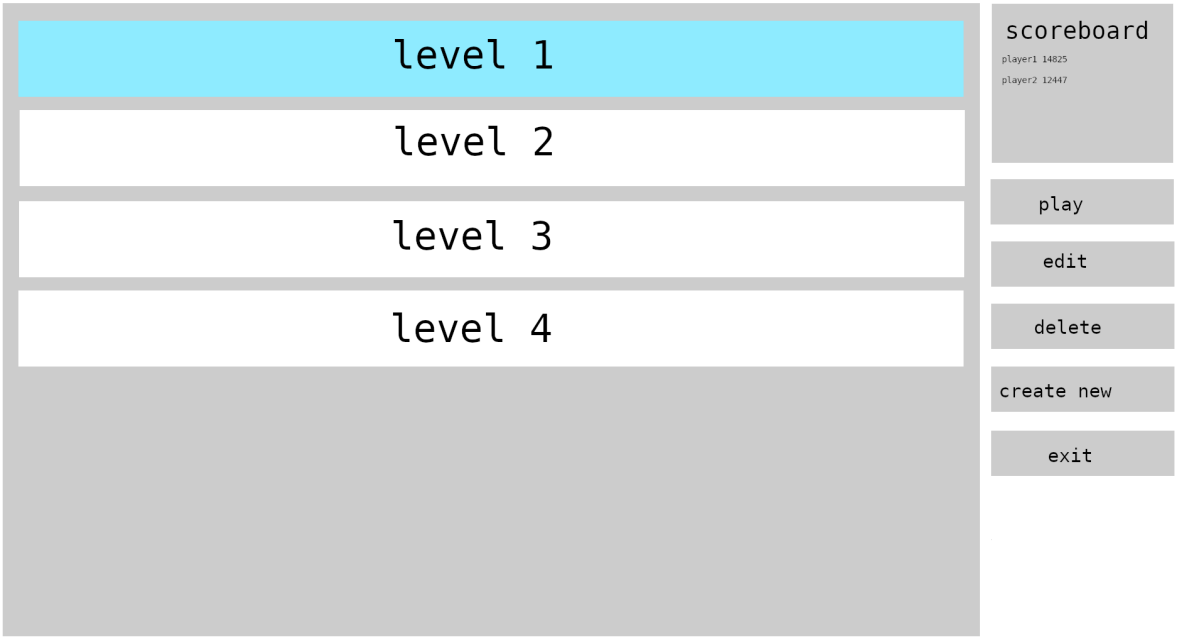
Game on the other hand will construct a Level with data loaded through FileManager, and manage Objects within that Level. It has a Camera that can be moved and zoomed according to things that happen in the Level, and can be used to transform objects' positions in Box2D's world space to their corresponding positions in screen space.

Elements and Objects can respond to input, and get updated regurarly, because Application will refresh the active Screen every iteration of its loop, and pass events such as keyinputs to the Screen. When a Screen is refreshed, it will update Game and Menu based on elapsed time since the last update, and ask both to render their

content with the provided RenderSystem. Game will update the phycics and call game logic, and render each Object in their z-order. After this, Menu will ask all of it's components to render.

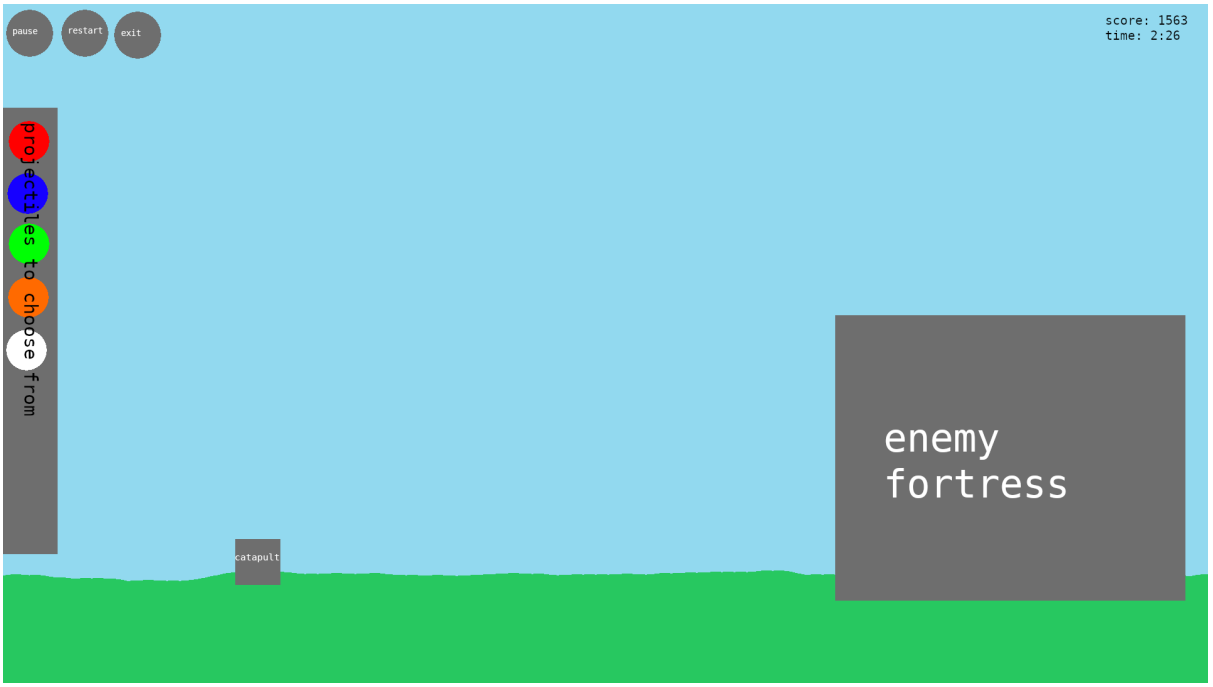
## Features to implement

**Main menu**




List of all levels is shown in the main menu. A User can select a level from the list by clicking it and the selected level gets highlighted. The scoreboard of the selected level is shown in the top right corner and below it there are buttons that allow the user to play, edit, or delete the selected level. If no level is selected the scoreboard is empty and play, edit, and delete buttons are dimmed to indicate that they are inactive. There are also buttons for creating a new level and exiting the game. If all levels won't fit into the view a scroll bar is added.

**Game view**




In game view the level is rendered on the background and all the UI elements are rendered on top of it. In the top left corner there are buttons for pausing, restarting, and exiting the level. The panel on the left hand side show all the projectiles the player can use. The projectiles are selected by clicking them. This allows the player to choose in which order the projectiles are used. Player can see the current score and time in the top right corner. Both the score and time are only shown when they are meaningful for the gamemode. When the level is completed the final score is calculated and one to three stars are given for the player. The player is also prompted to give a nickname to show on the scoreboard.

**Level editor**


In level editor most of the settings are adjusted from the panel on the right. First there is an input field for level name and below it there is a drop down selection for gamemode. The selected gamemode will affect the settings shown below it.

Below the gamemode specific settings there is a list of building blocks. These building blocks can be added by selecting them from the list, and then clicking in the level. The rotation of a building block can be adjusted with left and right arrow keys. Additionally, placed objects can be dragged around in the game world. During dragging the element being dragged will be colored red when it is overlapping with another element. If the element is released in an invalid position it is returned to its previous position. A building block can be deleted from the level by right clicking it.

A list of projectiles is shown below the building blocks. When one of these projectiles is clicked it is added to the panel on the left indicating that it is available for the player in the level. Projectiles can be removed from the level by clicking them in the left hand side panel.

In the top left corner there are three buttons. The first button allows the user to save the level. The second button can be used to start the physics simulation. When the physics simulation is running this same button allows the user to stop the simulation and resetting the level to its original state. The third button allows user to exit the level editor.

## Game logic

The game will have multiple game modes, so multiple variations of game logic needs to be implemented. In all game modes, the physics and student abilities will be the same but the calculation of score and winning goals change.

In classic mode the player slings students in order to destroy all fuksis in a level. Here the winning goal is obviously destroying all fuksis. The score is calculated based on the pieces of level destroyed and the amount of students used.

In time trial player's goals stay the same, but in score calculation time is also taken account of.

In endless mode, the player's goal is to keep playing as long as possible. Player has a limited amount of students in the start, but he can recruit new students by hitting these in levels. When all fuksis in a level are destroyed, a new level will be loaded and the game goes on as long as the player has students to use.

**Game objects**

Game objects can be divided in four categories based on rendering order. First we render the background, which can vary from level to level. The second category contains all the building blocks used in a level and the slingshot used to sling students. The third category of objects is gameplay objects like the students and fuksis, and beer bottles. The fourth category of objects is effects and animations, that are drawn on top of the other objects. This can be things like explosions, student special abilities, and smoke particles and debree.

Most objects with the exception of the fourth category will have collision and physics applied to them, and collide with everything in the level. Box2D collision filtering can be used to disable some collisions if needed. These objects also have health, and can take damage when colliding with other objects. When an object takes too much damage, it is destroyed and can provide points (like building blocks and fuksis) or give negative points like beer bottles.

Building blocks come in few different shapes/sizes and have materials which affect the mass and amount of hp.

**Students**

As projectiles, the game has different types of students from the guilds in Aalto. Students and fuksis are composed of a head, torso, arms and legs that are connected together through joints to form a ragdoll. The students and fuksis can have randomly selected faces from a collection of heads. Fuksis are visually similar to students, but don't have the black and white teekkari cap.

Here is a list of planned guilds, and their special abilities:

- Physics student changes the direction of gravity for himself.
- Computer science student causes a “bug” and teleports to the right.
- Bioinformation technology student drops a cow with a huge mass.
- Industrial engineering student swings his hands causing destruction.
- Electrical engineering student sends a lightning strike to the nearest metal element. The electricity spreads into nearby metal elements destroying them.
- Mechanical engineering student throws 3 wrenches
- Professor integrates over the map and destroys all objects in his path

Different students can be more effective against certain types of building blocks. For example, the wrenches thrown by the mechanical engineer deal more damage to wood, while the electrical engineer is clearly the best for destroying metal. The industrial engineer is good against glass and small blocks, because his hands destroy the blocks without slowing him down.

## Used tools and libraries

The program depends on two main libraries, SFML and Box2D. SFML is used for drawing windows, UI and graphics of the game. Box2D is mainly used for physics simulation. CMake is used in compilation. If the UI code requires it, Qt might be added for simpler creation of UI elements.

## Division of work and range of responsibilities

The implementation of the application is divided into several modules and not all modules can be implemented simultaneously. For instance, the level editor can be fully implemented after the level class and its functionality is fully designed. We will start by creating the basic structure of the application and then building the resource loader/manager and the basic functionality of the game simultaneously. A part of the features will be added later in the development such as the level editor.

The work will be divided flexibly in the group during the entire development. All members will be responsible for unit tests. The range of major responsibilities of the group members are listed in the table below.

| Group member | Range of responsibilities |
|---|---|
| Tommi korpelainen | Artwork and Sound, overall structure and framework |
| Ilari Tulkki | UI design and implementation |
| Joonas Palmulaakso | Physics and rendering |
| Aleksi Rintanen | Physics and rendering |

## Project schedule

The project is scheduled based on several milestones as follows:

| Date | Milestones |
|---|---|
| 5.11. | Project plan completed and committed to git |
| 14.11. | Basic structure of the application created (classes designed and their methods named) |
| 21.11 | Fundamental functionalities implemented and tested |
| 28.11 | Level editor completed |
| 5.11 | All features implemented |
| 8.12 | Testing and validation completed |
| 12.12 | Documentation fully finished |
| 12.12. | Final deadline to commit to git |
| 17.12. | Project evaluation deadline |