Project plan for “World War III Simulator”

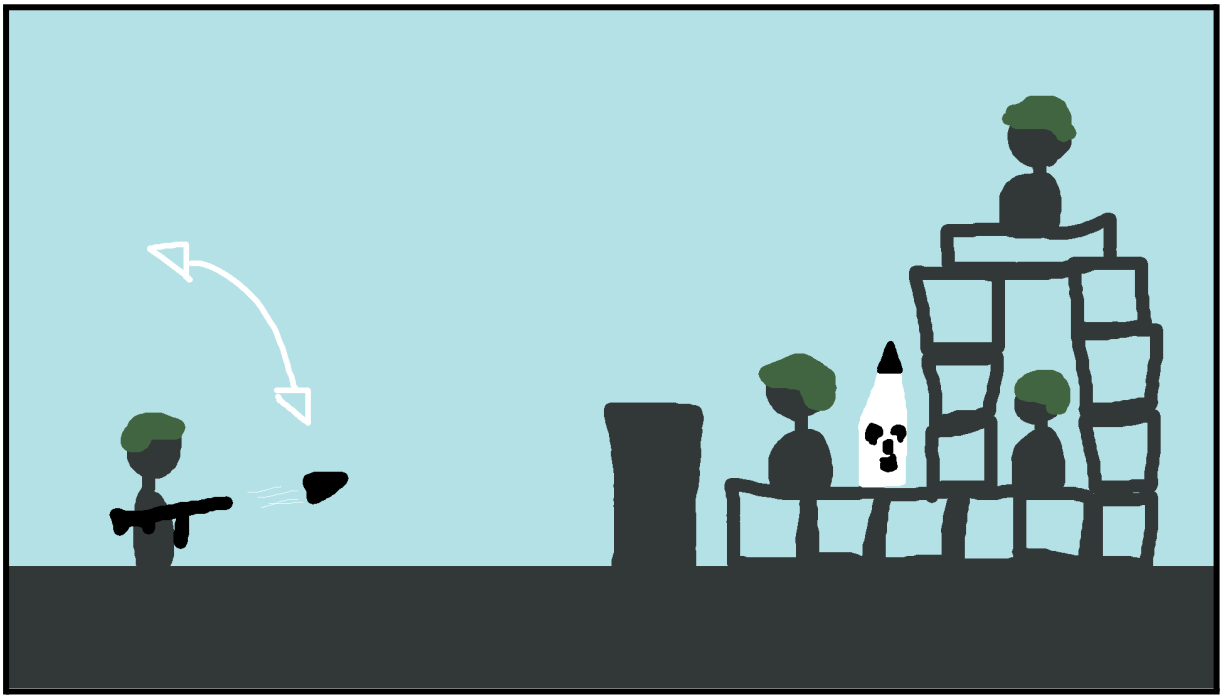


Figure 1: Illustration of the working idea for our project

Features and functionalities

Since our project topic is angry birds, we’ll naturally be implementing all the basic features that come with the topic. Our interpretation of the topic has our “birds” be different types of ammunition that are used in real warfare. We’ll likely start with a regular bullet that works much like the red bird in angry birds. In addition, we’ll likely have mortar ammunition, and perhaps some more exotic “birds” such as drones or grenades. These are the ones that will have special actions on mouse input, for example exploding mid-air or dropping a bomb mid-flight.

As far as we can currently tell, the structures that the player has to destroy will be very similar to those in the original angry birds. Although their physical appearances will maybe be more “realistic” like concrete and glass instead of rock and ice. We’ll likely make it possible to create structures consisting of blocks of different shapes and sizes. For adding them to levels, we’ll possibly use a grid-like class, especially if we decide to implement a level editor in the game. One chance for an extra feature is to make the ground a breakable object as well.

Our game will hopefully start from a simple menu screen, wherein the player can choose the level and probably also see the highscore(s) associated with that level. Here's also where the possible level editor and extra game modes would be selectable.

Speaking of extra game modes, we'd like to include a local multiplayer game mode that would basically just double the size of the level, put the opposing players on either side and have them take turns. This game mode is also very fitting considering that we intend on having the player represent a randomized country that's chosen at the start of the level. This would also make the game graphically more interesting.

Finally, we have some final points regarding the functionality. At least initially, it seems to make sense for the game to function purely with mouse input, as no need for keyboard inputs is immediately apparent and considering only one input type would make the implementation easier and less error-prone. The game will have physics implemented with the Box2D library. File writing and reading will have us come up with a file format suited to our needs. Out of the remaining additional features we plan on implementing at least sound effects and a star rating system.

Using the program and behind the scenes implementation

As mentioned earlier, it'd be preferable to have the game take only mouse movement and inputs into consideration, as it would make things simpler, especially when it comes to handling incorrect user inputs. This means that, for example, the menus and the possible level editor will be designed for point-and-click type use.

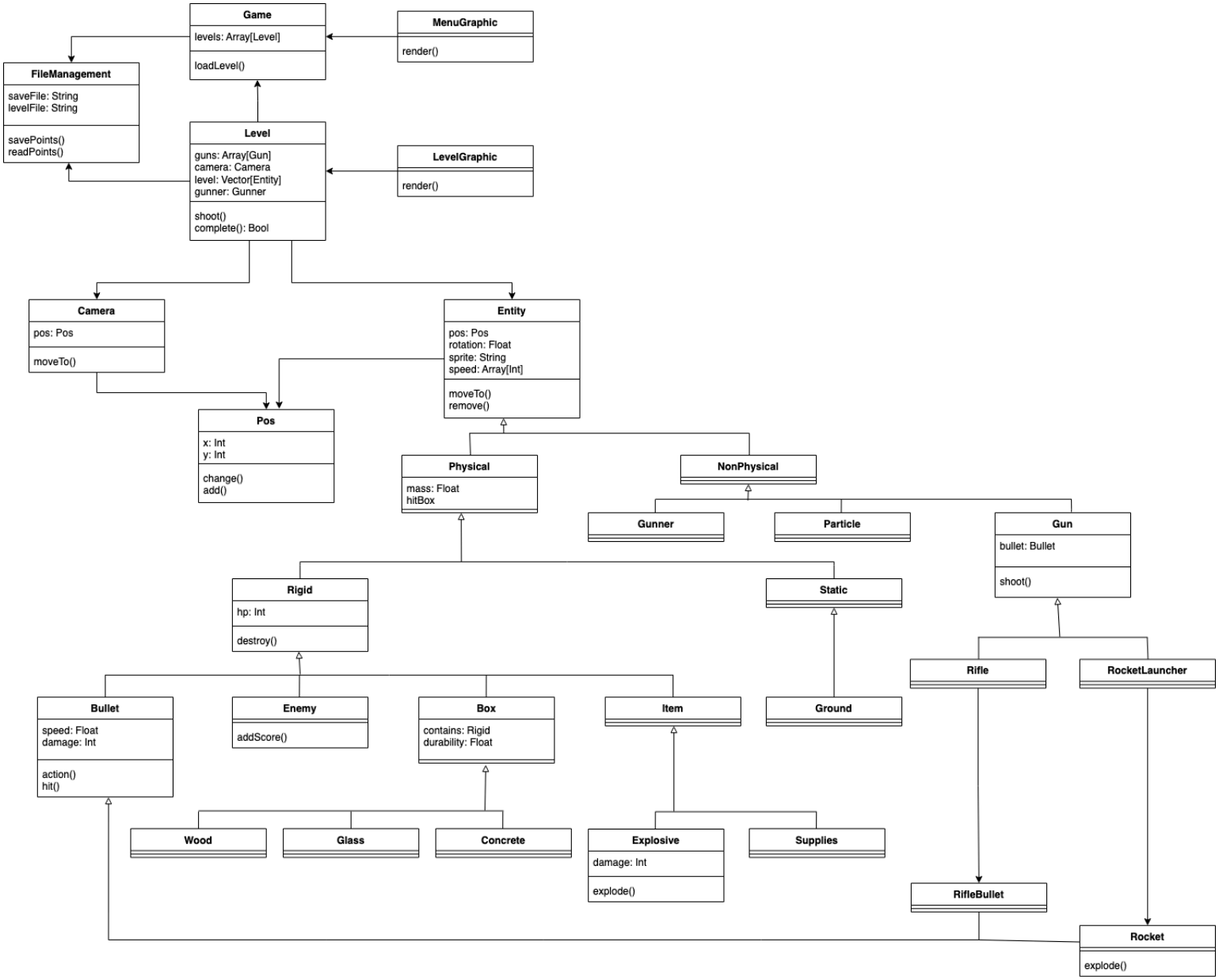
Another consequence is that the probable way of implementing the slingshot-type gameplay is to mark specific coordinates as the slingshot's focal point that the shot "appears" from. Then pressing the coordinates inside some predefined area can be used to perform a function that calculates the relative speed (perhaps by euclidean distance) and direction of the leaving bullet. Alternatively, the function could calculate only the direction and the speed would depend on how long the player holds the mouse button down. In addition, some indicator of the direction of the shot would probably be a nice addition to make the aiming a bit easier.

The preceding paragraph then basically already describes what the core gameplay loop will look like. Aim and shoot, watch the result unfold, and repeat. The viewpoint of the player is of course the camera, so proper implementation for it is very important. For example, we don't want the camera to switch away from the action too early. This could be implemented by somehow checking if any objects in the level are moving. One possibility is to simply add a boolean variable indicating movement to dynamic (rigid) objects.

Naturally, an intuitive and informative user interface is also important from the perspective of the player. Some of the more abstract elements such as score could be simply added to a corner of the screen, for example to the top left corner in Figure 1. Others like the remaining "birds" can be placed behind the shooter in a natural way, much like how the birds queue behind the slingshot in Angry Birds. Also similarly to Angry Birds, we want to have pictures of the blocks at different stages of damage to indicate progress to the player. The blocks will most likely have a certain amount of health in the beginning and the different images will appear depending on the relative amount of health left.

Much of the internal logic and graphics will be handled by the Box2D and SFML libraries respectively, so we won't go into deeper detail regarding, for example, the underlying physics calculations or rendering methods that will be used in the game. Although we have reasoned that using a different set of coordinates for game logic and graphics is beneficial. The rendering methods will then just get the logic-based coordinates as input and output elements on the screen in correct positions. The main motivation for this is that the camera will be dynamic, not showing everything all the time, so we don't want to render elements outside the view of the camera so that the game is computationally more efficient.

Class Structure



The current understanding of the classes is this: The game has two most important classes: Game and Level. Game makes the game go on and has the game loop. Level does the physics calculations as well as keeps track of everything happening during a level.

Entity is also an important class, since it has a lot of subclasses. Entity is basically everything rendered on the screen. Entity has a sprite texture and position in the world. Entity's Physical subclass represents everything that has a hitbox. The members of this class are Box2D objects. Rigid objects have also physics enabled, and static have not. NonPhysical objects are

not affected by physics and do not have a hitbox. This includes for example the gunner and the gun.

The graphical objects MenuGraphics and LevelGraphics use SFML library to render the game state on screen. These objects are completely separate from the game logic. It is Game's responsibility to call these objects' render() -methods.

File writing and reading is also its own object. This has a few methods, because it reads and writes not only points, but also level data.

On the bottom of the UML diagram are the final subclasses of the Entity Class. These include for example different box types and different guns. Due to this class structure, it is easy to implement new subclasses like this. For example, if we ever want to include a third weapon, it is done by just adding that as a subclass of a gun and we are ready to go already.

The class structure depicted by the UML diagram is not final and will probably change when we implement new features (the planned additional features are not included in the diagram). It however gives a good basic structure for the game and it is easy to start developing using it. Any adjustments are easy to do on the fly, however properly documenting them is highly important.

External Libraries

In our project we need some external Libraries. Firstly we'll use the SFML library to render the graphics and output the sounds to the user. The library has multiple modules in it from which we'll mainly use the audio, graphics, and window modules.

The window module opens OpenGl window in which we can render different shapes and images from the graphics module. That way we can render the graphics of the classes that inherit the entity class. The window module also provides tools for getting the mouse location and keyboard presses and the audio module provides tools for playing audio to the user

Depending on how we are going to save the gamefiles, the system module can become useful due to its file writing classes.

Another library we're going to use is the Box2D library that will provide the game physics for our game. The library provides collision physics which we'll need for our game when launching the bullets to the structures and want the structures to break realistically. With this library we'll apply physics to the classes that inherit the physical class.

Division of work and Sprint planning

The division of work will approximately look like this:
Ristimäki: Graphics
Kokkonen: Physics
Blomqvist and Hiedanpää: Game logic

Here's how to project timeline is planned to look like:
Between 24.10.-30.10.

- Planning

Between 30.10.-6.11.

- Class outlines for Game, Level, Pos, Ground, Box
- Studying physics libraries
- CMake

Between 6.11.-13.11.

- Physics, Bullets, other classes

Between 13.11.-20.11.

- Game logic should be ready

Between 20.11.-27.11.

- Menu
- Levels
- Points
- File management

Between 27.11.-4.12.

- Testing
- Extra things if time

Between 4.12.-11.12.

- Polishing the game
- Final testing