Angry-birds-2

Leo Blomqvist 100684892

Oskari Hiedanpää 100761692

Oskari Kokkonen 101656816

Väinö Ristimäki 997430

Overview

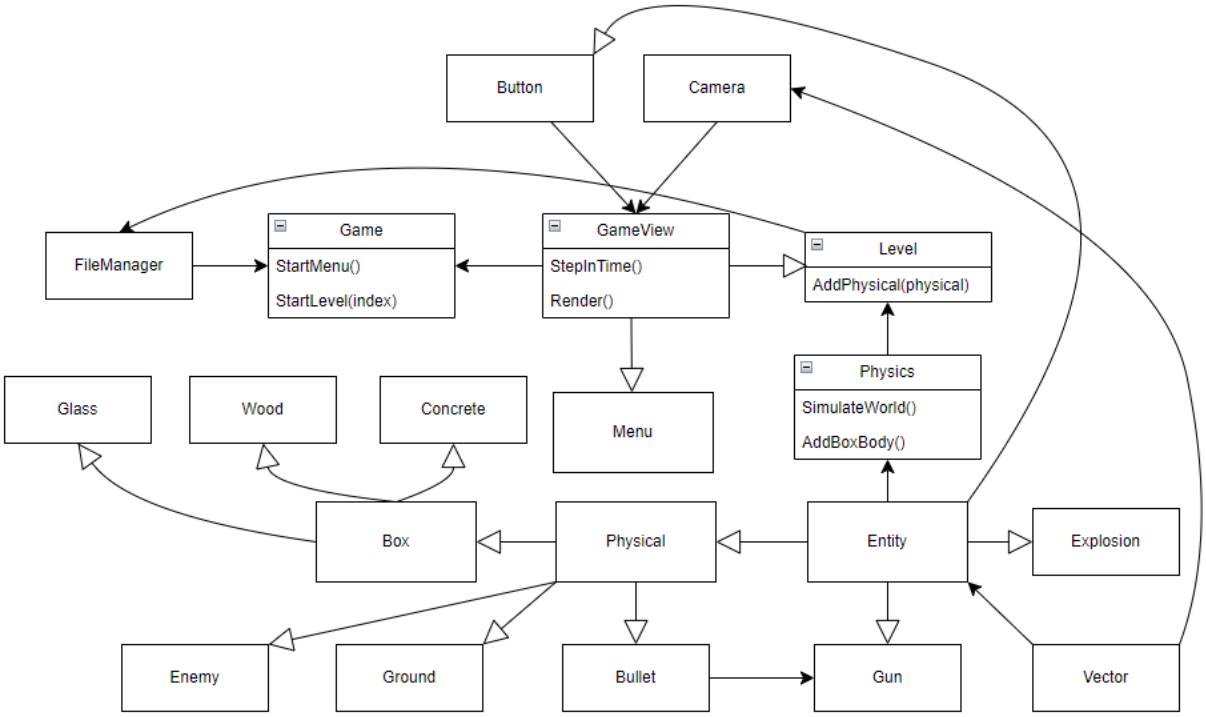
An Angry Birds –inspired game made in C++, with a topical twist. The player plays as modern country leaders trying to destroy their opposition. This game takes no sides, because depending on the level, you play as different leaders.

The game has a main menu wherein you can choose a level from three options of increasing difficulty. There is also an option for multiplayer which will be discussed later. Your task is to shoot guns at the structures in front of you, just like in Angry Birds. Destroying all enemies results in winning the game. The less weapons you use and the more structures you destroy, the more points you get. With more points you get more stars with 3 being the maximum. There are 3 kinds of blocks, concrete, wood and glass, with each being weaker than the previous. The enemies and structures are animated so that when they get on low hp, the textures will be updated to show that.

If you press the button with two people on the top-right corner of the window, you can toggle on multiplayer. Multiplayer basically mirrors the structures and places different leaders behind them. This way you can play locally with your friends. The one to destroy opponent's men first, wins.

We also planned to implement a level editor, scoreboard, more birds and possibly explosives in the structures, but didn't have time to do those. The score count works but is not saved anywhere and levels can currently be added with files that follow a specific format.

Structure



Main-function creates a Game-object and then updates game state and renders the new game state every frame until the window is closed. Game has functions for starting the menu or a level. When the game is first opened, it will automatically start the menu. From the menu you can then choose a level which calls the StartLevel function. All the visuals are done in GameView which extends Level and Menu. When starting a level, the Game-

object first calls FileManager with the chosen level index. FileManager then returns a level object with the structures, players and guns read from the corresponding file. Every frame the Main-function calls GameView's StepInTime and Render –functions. When in menu, the StepInTime –function checks whether buttons are pressed and when in level, it updates physics, gun angle and checks for button clicks. Due to issues with bullets clipping, physics are updated 4 times in 1 frame. Entity is the highest class for physics objects. All inheritances and other relations can be seen in the diagram above.

As for external libraries, we used SFML for visuals and sound and Box2D for physics. SFML is included in Entity-class for sprites, FileManager for sound effects and music and GameView for rendering. Box2D is used in Physics for collisions and gravity.

Memory management

Throughout the source code unique pointers are widely used to manage memory for dynamically stored objects. Move semantics are used to pass them to methods and containers. Only one raw pointer is present, namely the currentBullet_ variable of the Level class. Infact, it points to an object that is at the same time stored inside a unique pointer in the physicals_ vector. This is not an optimal design solution, but the raw pointer has the very specific use of tracking the movement of the flying bullets with the camera and the pointer's use is monitored carefully. There were problems with getting proper feedback from valgrind because it produced countless errors related to Nvidia graphics drivers, wsl and SFML (the SFML errors stemmed from inside the library itself). However, due to the use of unique pointers practically everywhere the game should have minimal issues with dynamic memory management. The Box2D library also handles memory management internally and it's only required to delete the simulation world when the program is done using it. This is handled in the destructor of the Physics class.

Building the software

The game can be built using CMake, and requires version 3.22 of CMake or higher due to the requirements of the Box2D library, one of the external libraries the project uses. The other external library is SFML the dependencies of which are listed in the readme.md of the "libs" folder; they need to be downloaded so that the game can be compiled. Both of the

libraries themselves are fetched during building by CMake so they don't need to be explicitly downloaded. In addition, CMake fetches the header files for a C++ wrapper for Box2D although the wrapper is not currently used in the code.

Using the software

All the buttons in the menu can be seen in the image above. Levels in the middle and multiplayer in the top right corner. Pressing "esc" will quit the game.

Upon opening a level, the camera pans from the buildings to the player. On the top left corner, you can see your remaining weapons and score. When the mouse is pressed a load

number is shown next to the weapons. This indicates how far the bullet will fly. Pressing “esc” will also quit the level.

In multiplayer your own troops are in front of you and enemy’s behind them. It is advised not to shoot at your own, as it will increase your chances of losing although no sanctions follow.

This might happen if you shoot your own troops.

Testing

As our project is a graphical game the most logical way to test the functionality of it was by using its graphical user interface. With the UI we tested that the physics and graphics worked, and all of the menu buttons did what they were supposed to do.

Most of the values, like the impact speed wasn't constant and was generated with user input, so they were impossible to test with predefined test functions. So, while developing the game we used mostly the vs code's debugger and print functions to check that the output values were correct while running the game.

However, some of the classes like level had a lot of functions that mutated values that didn't depend on user input. Due to that, it was one of the only classes that we were able to test the unit tests. With these tests we made sure that the functions did what they were supposed to do.

Work log

First meeting 24.10.

- Blomqvist, Hiedanpää and Kokkonen
    - Studied the project assignment
- Ristimäki
    - Studied the project assignment and created plan document

30.10.

Planned:

- Blomqvist
    - Section "Using the program" in the plan
- Hiedanpää
    - Section "Sprint planning" in the plan
- Kokkonen
    - Section "External Libraries" in the plan
- Ristimäki

    - Section “Modules and Classes” in the plan

Realized progress:

- Blomqvist, Hiedanpää, Kokkonen and Ristimäki
    - Worked on the plan

6.11.

Plan:

- Blomqvist
    - CMake
- Hiedanpää
    - Initialize Game, Level and Pos
- Kokkonen
    - Study the physics library
- Ristimäki
    - Graphics

Realized progress:

- Blomqvist
    - Initial version of CMake, so the project could be built
- Hiedanpää
    - Was sick
- Kokkonen
    - Studied the physics library and implemented initial physics
- Ristimäki
    - Graphics and essential elements for some classes

13.11.

Plan:

- Blomqvist
    - Continues with CMake and begins implementing Bullet-class
- Hiedanpää
    - File management
- Kokkonen
    - Physics
- Ristimäki

    - Makes rendering entities possible

Realized progress:

- Blomqvist
    - CMake is ready, worked on bullet
- Hiedanpää
    - Worked on file management
- Kokkonen
    - Tried working on physics, but they fought back
- Ristimäki
    - Worked on graphics and made objects spin

20.11.

Plan:

- Blomqvist
    - Try to solve the problem with physics
- Hiedanpää
    - Finalize file manager and start working on enemies
- Kokkonen
    - Also work on physics
- Ristimäki
    - Graphics

Realized progress:

- Blomqvist
    - C++ wrapper for Box2D and implemented the “slingshot”
- Hiedanpää
    - Enemy and file manager ready for the moment
- Kokkonen
    - Boxes take and deal damage and bullets deal damage
- Ristimäki
    - Graphics, texture loader, menu

27.11.

Plan:

- Blomqvist

    - Makes “slingshot” ready and fixes bullets
- Hiedanpää
    - Levels
- Kokkonen
    - Makes enemies die and boxes break
- Ristimäki
    - Rocket launcher and explosions

Realized progress:

- Blomqvist
    - Created gun queue logic, refactored main, bullet works
- Hiedanpää
    - Created levels
- Kokkonen
    - Worked on physics, box damage is realistic and fun
- Ristimäki
    - Implemented explosions and rocket launcher

9.12.

Plan:

- Blomqvist
    - Implements multiplayer
- Hiedanpää
    - Valgrind and levels
- Kokkonen
    - Makes boxes disappear and fixes damage
- Ristimäki
    - Makes UI ready and rockets explode from click or from timer

Realized progress:

- Blomqvist
    - Implemented unique_ptrs to fix valgrind issues, lots of refactoring, multiplayer
- Hiedanpää
    - Created new levels
- Kokkonen

    - Box2D bug fixing
- Ristimäki
    - Graphics, UI and animations

13.12.

Final things:

- Blomqvist
    - Final refactoring, sounds and bug fixing here, there and everywhere
- Hiedanpää
    - Sound effects and documentation
- Kokkonen
    - Tests and final physics debugging
- Ristimäki
    - Comments, sound effects, bug fixing