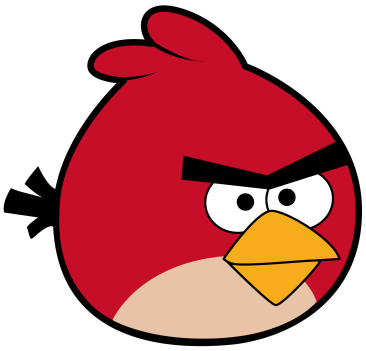
Aalto-universitetet

School of Electrical Engineering

Project Documentation - Angry Birds

ELEC-A7151 Object-oriented Programming with C++



Casper Tillander
Linnea Haapio
Julius Halmela
Matilda Stendahl

Contents

1 Overview

This project is developed to function as a game that resembles and works in the same way as Angry Birds (*Angry Birds* 2023). The objective of the game is that the user launches and shoots birds at a fortress and tries to destroy the enemies, which in this case are the pigs.

The game uses a graphical user interface, from where the user can choose between birds, change settings (sound + game mode), choose levels, and play the game. The game includes three distinct levels with increasing difficulty, which are loaded from CSV files. The game can be played in 2 modes, normal mode and no gravity mode. While playing, the user interface displays the number of throwable birds left and after completing a level, it will display the final score, measured in stars (1-3). In each level, there are three available birds to use, and the user gets the number of stars relative to how many birds were used. A level is completed when all pigs have been destroyed or all birds have been used.

The user can also create their own levels with the level editor. This is done by choosing a background and then placing different objects in desired positions in the world. The created level will not be saved to the level selection screen, but the user will get a score for the level and the opportunity to try again. The user can also login to the game using a desired username, which enables saving scores for played levels. These stars are saved even though the program is closed since they are saved to an external file.

The gameplay is centered around the mouse. The user clicks and drags the mouse on the bird, which displays a flight path, and then shoots the bird by releasing it. If the special bird is used, a secondary click after releasing the bird will activate the special action, which in this case shoots the bird to where the user is clicking the second time. The pig will be destroyed if it is hit directly by a bird, or if it is hit several times by other objects.

2 Software structure

There are 3 main modules in the project. The modules are for the graphical user interface, the gameplay, and the game objects.

The graphical user interface module contains the classes GUI, readfile, levelleditor, and player. The GUI class handles all graphics of the game. It displays the home screen and the game screen, shows the physical simulation of the bird flying, and displays different statistics during and after completing the game. This class also downloads sounds and background images, and it includes the actual method that runs the game and its events through the graphical user interface. The readfile class reads CSV files containing the information of the levels, i.e., the initial positions of the different game objects. This information will then be passed to other classes that will create the level. The levelleditor class creates new levels specified by the user. It reads input about the positions of different objects and creates a CSV file based on the information. The player class represents a player who has logged in to the game. It updates the scores of the levels played by the user and stores them in an external file.

The gameplay module includes classes that create the game and handle the physical simulation and game logic. The game class creates the instance of the game, which triggers the process of all other events to create and run the game. The level class constructs the levels according to the information in the CSV files. It creates the simulated game world and places the game objects in it. It also processes the actions of the game, including checking the number of birds used. In addition, it checks the completion of the level and calculates the score. The collision detection class handles the collision logic when two objects collide in the game world. It calculates the impact force of the collision to determine the damage to the pig. In this way, full-on collisions with the pig allow for bigger damage compared to for example collisions with boxes or walls. After taking enough damage, the pig eventually is destroyed.

The game object module contains classes for all different objects in the game. The bird class defines the properties of the normal bird. It handles the user input through the mouse that decides how and with what speed the bird flies. After that, it will launch the bird and update its position and rotation. The special bird class inherits the bird class and does the same tasks for the special bird. In addition, it accounts for the special action of the bird. The pig class constructs the pigs, which are the enemies to be destroyed. It updates the pig's position and rotation and renders it if the pig is still alive, as well as keeps track of the damage to the bird to destroy it if its health reaches zero. The box and wall classes construct and update the position and rotation of the box and wall objects.

In this project, we have used both SFML and Box2d libraries. We have utilized SMFL for

the whole graphical user interface and its aspects, such as the sound. SMFL is also used to handle events that are initialized with the mouse. The physics implementation is done with Box2d, including the gravitational game world and the flying, and moving objects. Figure 1 shows the relationships between the classes.

3 Instructions for building and using the software

3.1 How to compile the program

For building the software, it is required to install the SFML package using the provided packet manager. After installation, CMake will then find the package and build the software. Box2d is implemented as a subdirectory, so no installations are needed for it. We tried doing this with SFML as well but did not manage to make it work. The repository should then be cloned, and the user should navigate to the main "Project" directory. To compile the program, the user should first create the build directory using the command "mkdir build", then go to the directory using command "cd build", and after that compile using the command "make". The executable will be in the build directory under the name "AngryBirds", and it can be run using the command "./AngryBirds". More detailed instructions can be found in the README.md.the

3.2 How to use the software

When running the program, the user can choose to change the sound and game mode from the settings, change the bird between the normal and the special bird, and then play one of the three available levels. To play the game, the user clicks on the bird, drags the mouse to create a flight path to the bird, and then launches the bird toward the enemies. The special bird can be launched as normal and then shot at a target whose position is given by a second click of the mouse. If all enemies are destroyed before having used up all birds, the level is completed. Depending on the number of birds used, the user will get a score displayed on the screen. The user can also log in to the game to save scores and create their own levels.

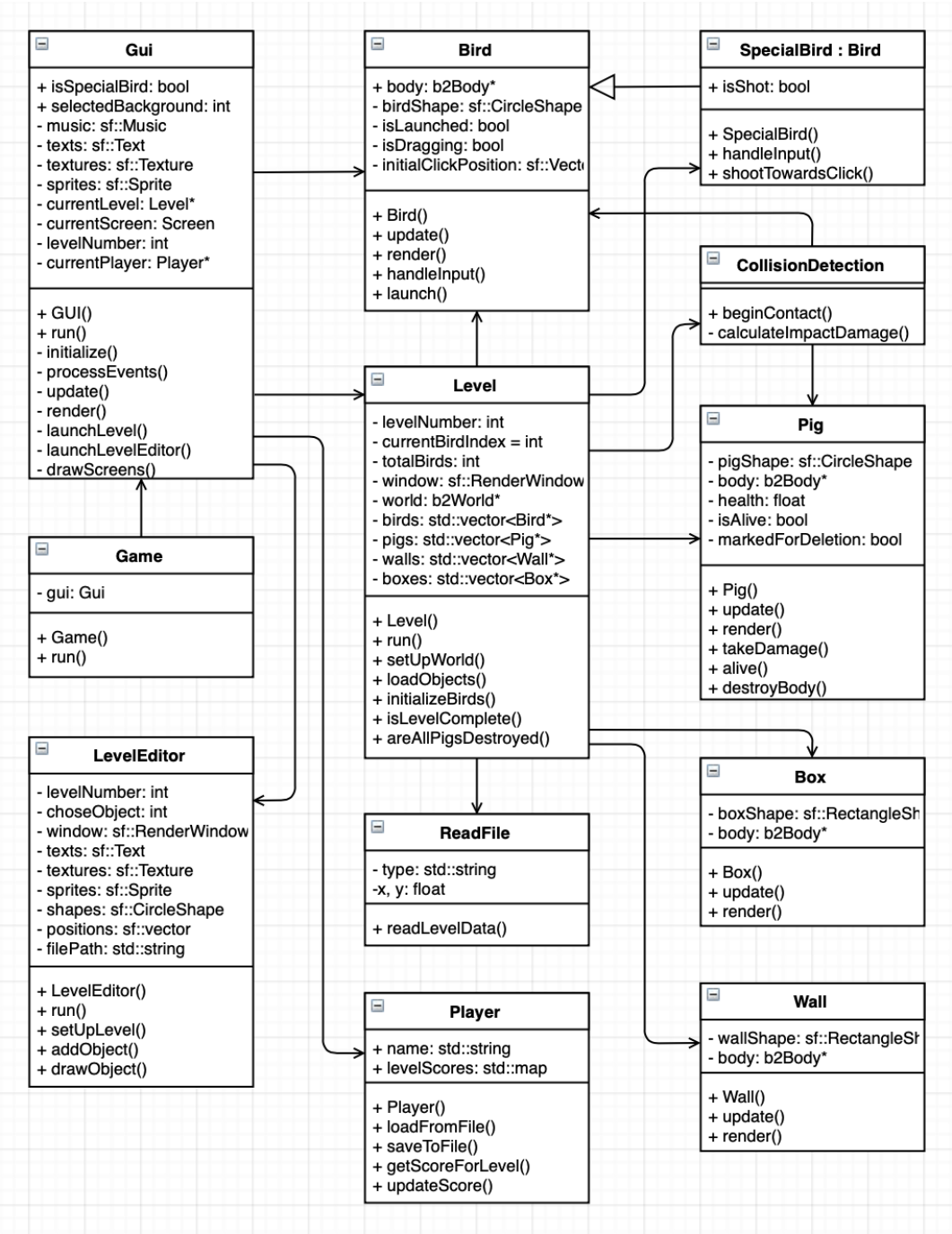


Figure 1: Class diagram of the project

4 Testing

The testing for this project aims at unit testing the most fundamental classes in the project, hence ensuring the functionality of the basic components. The graphical or more advanced classes can more effortlessly be tested through the gameplay itself.

4.1 Unit Tests

Bird and SpecialBird Classes

Validation of the proper initialization, launch functionality, and special ability execution for the birds.

Pig Class

Verification of the Pig class's correct instantiation, damage-taking mechanism, and the impact of destroying its body on state and damage.

Box and Wall Classes

Testing the correct initialization and position updating of Box and Wall objects.

Readfile Class

Verification that the Readfile class correctly creates objects from a file and that it handles empty files appropriately.

Level Class

Checking the functionality of the Level class, specifically confirming that all birds are used up in each scenario.

Player Class

Ensuring that the Player class displays correct messages and updates the score accurately.

4.2 Interactive Testing

The more advanced and graphical aspects of the game, including the graphical user interface, level editor functionality, and the physics simulation, are tested through interactive gameplay. This involves playing the game with different conditions and in this way assessing the overall user interface. This combination of unit tests and interactive testing ensures that the individual classes work correctly and also work correctly together. All of the unit tests were tried until they passed and found bugs were taken care of.

5 Work log

Week 43, 23-29.10

Casper

- Created the repository.
- Worked on the project plan as a team.

Time spent on the project this week: 2 h

Linnea

- Worked on the project plan as a team.

Time spent on the project this week: 2 h

Julius

- Worked on the project plan as a team.

Time spent on the project this week: 2 h

Matilda

- Worked on the project plan as a team.

Time spent on the project this week: 2 h

Week 46, 13-19.11

Casper

- Added Box2D and SFML to the project, integrated them using cMakeLists.txt.
- Created a test file to confirm library functionality.

Time spent on the project this week: 13 h

Linnea

- Added SFML to the project.

Time spent on the project this week: 10 h

Week 47, 20-26.11

Casper

- Implemented the basic GUI, including the home and levels screen.
- Added several object classes and started implementing level and bird logic.
- Implemented the pig class and launch logic for the bird.
- Implemented the box and wall object classes; box is dynamic, wall is static.
- Added functionality for creating levels from CSV files.

- Added three simple levels.
- Implemented a damage system for pigs using a collision detection class.
- Integrated doxygen into the project, generated documentation, and fixed scaling issues.
- Implemented 3 birds/level and game completion logic.
- Added a settings screen and background music as well as the functionality for turning it off.

Time spent on the project this week: 30 h

Linnea

- Added necessary files to gitignore to solve merge issues.

Time spent on the project this week: 1 h

Julius

- Fixed bug in screen scaling after level completion.

Time spent on the project this week: 3 h

Week 48, 27.11-3.12

Casper

- Implemented special bird and the ability to switch birds.
- Implemented a point system using stars.
- Added trajectory dots for the birds.

Time spent on the project this week: 15 h

Linnea

- Started working on the level editor and added background selection feature.

Time spent on the project this week: 10 h

Julius

- Created a no-gravity game mode and updated pig damage logic.
- Included statistics in level GUI.

Time spent on the project this week: 8 h

Matilda

- Added new backgrounds with slingshots.

Time spent on the project this week: 5 h

Week 49, 4-10.12

Casper

- Translated meeting notes to markdown, fixed some valgrind errors.
- Updated level graphics, completed readme and documentation.
- Implemented system to save player stars in external files.
- Fixed a bug where the level editor and the level launcher got mixed up.
- Wrote or updated readme files for a lot of the folders.
- Worked on a smaller part of the documentation and moved it into a latex document.

Time spent on the project this week: 10 h

Linnea

- Completed the level editor.
- Fixed a small bug with the window.
- Wrote a majority of the documentation and made the UML diagram.

Time spent on the project this week: 5 h

Julius

- Wrote all of the tests.
- Made the documentation for the tests.

Time spent on the project this week: 10 h

Matilda

- Made the CSV files for the levels.
- Wrote Doxygen-compatible comments for the code and organized the code.

Time spent on the project this week: 2 h

References

Angry Birds (2023). Rovio Entertainment Oyj. URL: https://www.angrybirds.com (visited on 03/12/2023).