
Angry Birds Project

***Project documentation for the Angry Birds game
development (Project ID: angry-birds-2019-5)***

Technical Design Document

Issue 1

TABLE OF CONTENTS

1	OVERVIEW	1
1.1	Purpose.....	1
1.2	Scope	1
1.3	Work breakdown structure.....	3
1.4	WBS dictionary.....	3
1.5	Document Structure.....	6
2	SOFTWARE STRUCTURE.....	7
2.1	System Characteristics.....	7
2.2	System Architecture.....	7
2.3	Class relationship	10
3	INSTRUCTIONS FOR BUILDING AND USING	12
3.1	How to build.....	12
3.2	How to use	13
4	TESTING.....	15
4.1	Test map and UI	15
4.2	Test birds class.....	15
4.3	Test enemies class	16
4.4	Test material class.....	16
4.5	Test collision class	16
	WORK LOG	17

1 OVERVIEW

The software developed corresponds to the final project of the course **ELEC-A7151 - Object oriented programming with C++**. In this project we implemented our own version of the world-popular classic game **Angry Birds** using C++ standard library with additional libraries to implement the graphics and the physics of the game. In this implementation, as in the original game, the goal is to destroy certain targets located in different positions of the map by throwing a limited number of birds to accomplish this goal. The game was designed with an easy-to-use interface where almost all the interaction with the software is conducted by using the mouse pointer and a few keys in the keyboard to navigate through the scenery, restart the current game level and move back to the main menu.

1.1 PURPOSE

The purpose of this document is to provide a clear explanation of the project by presenting the functionalities implemented in the game, as well as a detailed software structure explanation with a comprehensive description of the classes defined for the game in the form a class hierarchy diagram, a basic user guide is described in section 3.2 showing how to play the game and a section showing how the classes and features were tested during the developing process can be found in section 4. In addition, there is an annex section containing the Work Log of the project where the reader can see the workflow, the division of the work and the weekly time investment allocation for each part of the game.

1.2 SCOPE

The project timeline is 6 weeks. During this period, different parts of the game were built in phases with the following high-level description:

Period	Target
Week 45 ~ Week 46	Base classes implemented
Week 47 ~ Week 48	Have a working base level with one “throwable”
Week 49	Implementing more “throwables” and levels
Week 50	Finalizing graphics and physics
Dec 12	Final commit to git

Table 1. Initial time baseline for the project

The game implements the following features:

- Basic features:
 - Basic graphics: There is a scenery with a nice background picture where the game develops. The birds, enemies and objects are rendered with `png` pictures and they interact each other according to the physics engine and the user interaction.
 - Birds: the birds are the main character of the game and are the objects thrown across the scenery. There are 4 types of birds, 2 of them have special features like increasing its velocity and increasing its size. The other 2 birds are standard and don't have any special power, other than being different in size.
 - Controls: the entire game is controlled by the mouse pointer, including the special feature activation for the birds. There are additional keyboard controls to move the camera horizontally, restart the game level or back to main menu.
 - Levels: the game levels are defined in `csv` files where the specifications for a particular level are defined. These game level files are read by the program and loaded on the scenery accordingly. Three game levels are designed for this game.
 - UI: the user interface is intuitive, and it provides small legends for the user to follow up the game and to easily understand how to use the program.
 - Physics: all the birds, enemies and objects interactions/movements are governed by a physics simulation engine which determines the different physics parameters of the game according to every situation.
- Additional features:
 - Materials: There are multiple material types, and its resistance varies depending on the type. The weakest material is ice, the second weakest is wood and the strongest is stone. These materials can be destroyed depending on the force applied to them.
 - High score: a high score system is in place where the player can record its points and see how well it performed compared to other players.
 - Rates: there is a rating system that assign stars according to the performance of the player in a given level.

1.3 WORK BREAKDOWN STRUCTURE

The project is divided in two levels of activities defined in the following Work Breakdown Structure:

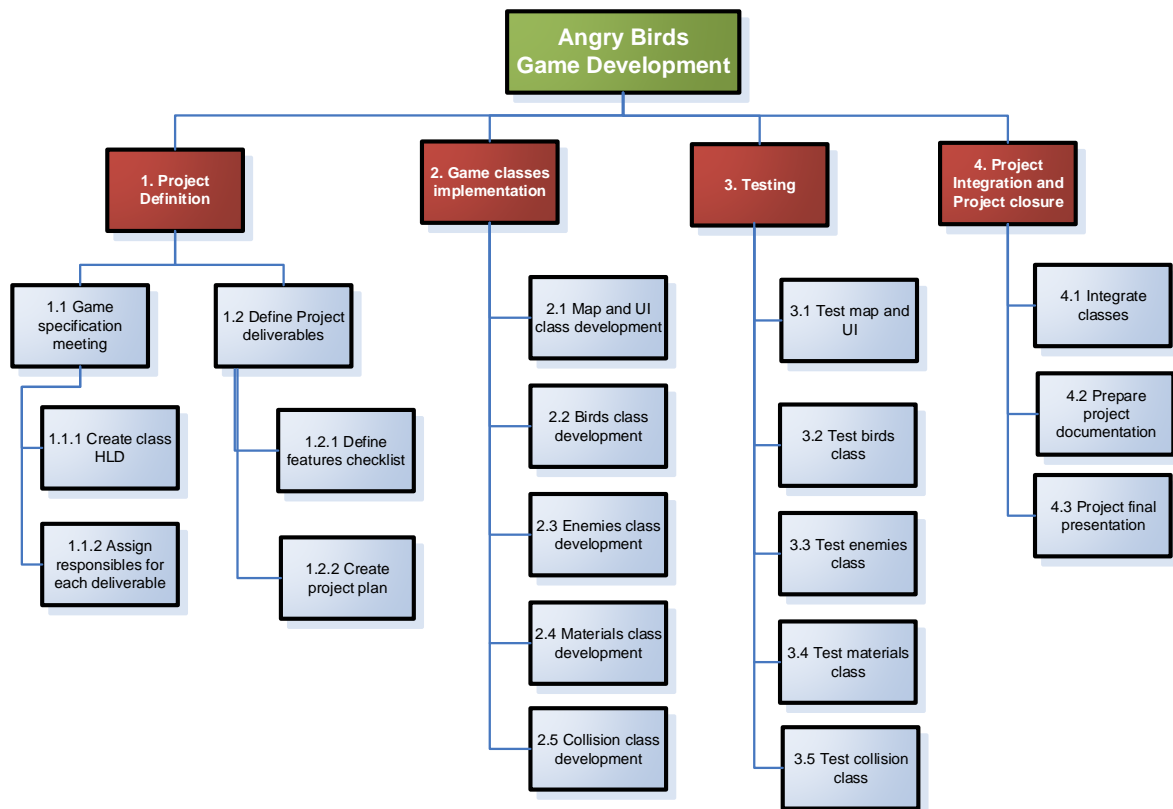


Figure 1. Work Breakdown Structure

1.4 WBS DICTIONARY

1. Project Definition: This activity defines the general scope and expected result of the game. During this activity the project kick-starts and the work is defined.

1.1 Game specification meeting: In this meeting a brainstorm session is conducted in order to gather ideas on the implementation of the project. Also, some features are proposed.

1.1.1 Create class HLD: This activity defines the high-level design of the classes and possible structure and hierarchy of them. This is the first definition for the classes structure of the project and because of this, it is prone to be updated in the future as the project evolves.

1.1.2 Assign responsible: This activity divides the work in tasks and assign a task to a team member, who will oversee the task development and completion. This is not an exclusive

assignment because all team members collaborate each other in case of any trouble. Also, all team members suggest and provide feedback on the implementation.

1.2 Define project deliverables: Here the most important deliverables of the project are formalized and defined in order to create a clear scope of what is going to be done and what is not included in the project.

1.2.1 Define features checklist: This activity defines the detailed list of features that each deliverable and class will implement. The list of features is also updated as the project evolves.

1.2.2 Create project plan: This activity creates the project plan where the timeline, roles, scope, initial features and high-level design are defined.

2. Game classes implementation: This activity implements and develop the all the classes and game features specified before in the project plan, features checklist and other related deliverables.

2.1 Map and UI: this activity implements the classes and methods required for the game to function and how the maps are loaded during the game development. It implements the user interaction with the game and defines how the different screens and menus will be shown to the player. It also creates the general `GameObject` class, which will be the base parent class for the birds, enemies and materials classes. In addition, this activity will define the window size, background image, ground texture and other visual effects for the game menus and maps.

2.2 Birds class development: this activity implements the classes and methods required for the bird's logic. In this deliverable, the four different birds employed in the game are defined, as well as the method elements, variables and related components that enables the birds to interact in the game. In addition, this activity implements the birds throwing functionality, develops the features on the birds that have special features and implements how the features will be activated according to the player during the game.

2.3 Enemies class development: this activity implements the classes and methods required for the enemy's logic. In this deliverable, the different enemies employed in the game are defined, as well as the method elements, variables and related components that enables the enemies to interact in the game. In addition, this activity implements the enemy's callbacks and the logic behind the materials interaction, where the enemies and objects react based on how strong the bird is thrown and how hard an enemy or material is hit.

2.4 Materials class development: this activity implements the classes and methods required for the materials definition logic. In this deliverable, the different materials employed in the game are defined, as well as the method elements, variables and related components that enables the materials to interact in the game. In addition, this activity implements the materials resistance and its picture.

2.5 Collision class development: this activity implements a sub-class of the contact listener class. In this deliverable all the collisions happened between any two Box2D bodies where handled through implementing the callback function at the beginning of the contact and at the end of the contact, this activity also adds for every object in the game the total impulse received which is used to calculate the score and make objects disappear after some threshold.

3. Testing: This activity tests the functionalities and methods developed for each class, ensuring that the features and game logic work as expected.

3.1 Test map and UI: here, all the methods and classes for the map loading, UI and game functionalities are tested.

3.2 Test birds class: this activity tests the classes and methods defined for the birds. It tests if the throwable functionality for the birds works, the special features and the birds appearance.

3.3 Test Enemies class: here, the enemies' classes and methods are tested, as well as the different callbacks and physics interaction with them.

3.4 Test Materials class: this activity tests the classes and methods created for the materials. It tests that the different type of materials works properly, and its resistance corresponds to the material definition.

3.5 Test Collision class: this activity tests the collision management and the objects disappearance/change in image. It tests deleting objects that were allocated heap memory without causing crashes to the game.

4. Project integration and closure: This activity integrates all the classes defined in the activity 2 by merging the branches into the `master` branch, in order to ensure that the whole game works properly. In addition, the final

project documentation is created, and the game is presented during the corresponding presentation session.

4.1 Integrate classes: this is a critical activity and one of the most important because here the whole classes are merged together, and the full game is tested. In this stage if there is any bug or misfunction the deliverables on the section 2 and 3 are revisited/reviewed.

4.2 Prepare project documentation: this activity creates the final project documentation and closes all the other deliverables.

4.3 Project final presentation: here, the final project is presented, and the game logic, UI and functionalities are showed in a demo session.

1.5 DOCUMENT STRUCTURE

Section 1: Provides an overview of the project and a description of the activities, works and features implemented in the game.

Section 2: Provides an overall architecture including a class hierarchy diagram, detailing the external libraries used in the project.

Section 3: Describes how to install, build and launch the game. In addition, this section includes a user guide.

Section 4: Describes how the project classes and functionalities were tested.

Work Log: provides a detailed description of the work and the deliverables assigned to each member of the team. In addition, there is a weekly report containing the time in hours allocated by each member for the tasks and deliverables.

2 SOFTWARE STRUCTURE

The Angry Birds project is a game developed using a hierarchical classes approach where the game is launched from a main file and the windows and views change according to the user interaction. In addition, the birds, enemies and objects in the game are derived from a main class and the characteristics and particular features of every object are defined depending on the parameters assigned to the objects. Also, the game levels are read from files where the level setup is defined. The selected architecture brings several benefits to the implementation of the game such as: modular development where it is easy to add or remove objects, birds, features, etc., the testing of the classes can be done gradually facilitating the bugs and misbehaviors detection and finally it helps the project documentation.

2.1 SYSTEM CHARACTERISTICS

The game was developed with the following characteristics:

- The game can operate in Ubuntu 18.04/16.04 Linux distributions and Windows 10.
- The graphics and multimedia library utilized is SFML v2.4/SDL v2.0
- The physics library engine is Box2D v2.3.1
- UI is all graphical and the interactions with the game are performed with mouse pointer and few keyboard keys.
- The software is highly resilient since all the user interactions are controlled and pre-defined, avoiding the wrong user input data.
- The game is scalable for future releases since the levels are read from csv files and the birds, enemies and objects can be easily expanded.

2.2 SYSTEM ARCHITECTURE

The game runs locally on the computer and has the SFML and Box2D libraries compiled with the software. It is launched by running the corresponding executable file in Ubuntu or Windows. After launching the game, the program executes the following software sequence diagram:

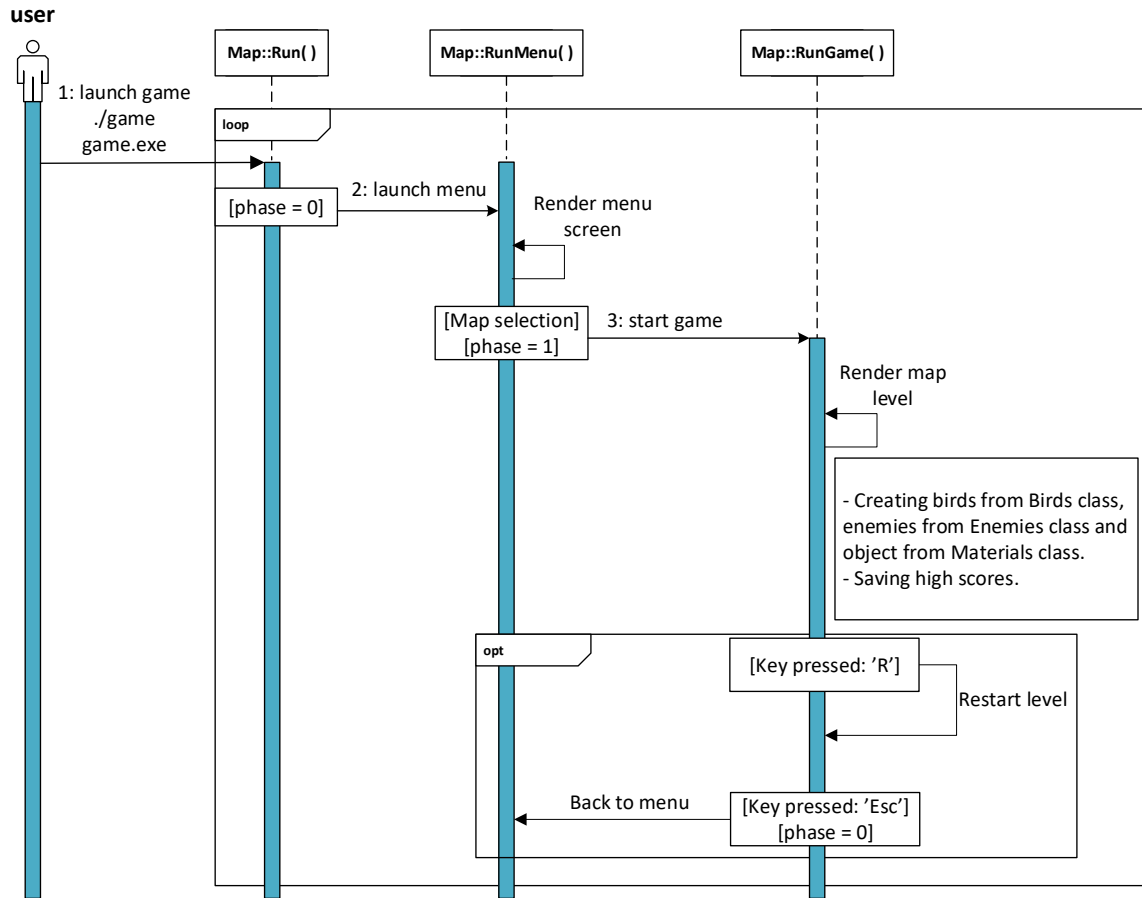


Figure 2. Software Sequence Diagram

The diagram depicted in the figure 2 shows the software sequence diagram of the game from the moment that the user launches the executable file that runs the game. The program works according to the following sequence:

- 1) The user executes the file `./game` (Ubuntu) or `game.exe` (Windows) to launch the game.
- 2) The program executes the method `Map::Run()` which starts a loop where one method out of two can be executed according of the phase variable.
- 3) By default, when the program starts the first time, the `phase` variable has a value of 0, which shows the main menu of the game by executing the method `Map::Run()`.
- 4) The method `Map::Run()` will render the main menu screen, from this screen the player can select the level to play.
- 5) When a certain level is selected, the `phase` variable changes its value to 1 and then the `Map::RunGame()` method is executed.
- 6) In the `Map::RunGame()` function, the game will read the corresponding level map from the csv file and will render the level on the screen showing

the birds, enemies and objects according to the level specifications. In this method there is also the logic implemented to interact with the birds and throw them into the scenery. Also, the interactions with the objects and enemies are implemented based on the physics of the game simulated by Box2D. During the execution of this method the user can restart the level by pressing the 'R' key or go back to the main menu by pressing the 'Esc' key. In addition, if the user is able to complete the level successfully, then the score obtained is recorded.

- 7) The loop continues indefinitely until the window is closed or the game is terminated by the user.

2.3 CLASS RELATIONSHIP

GameObject class:

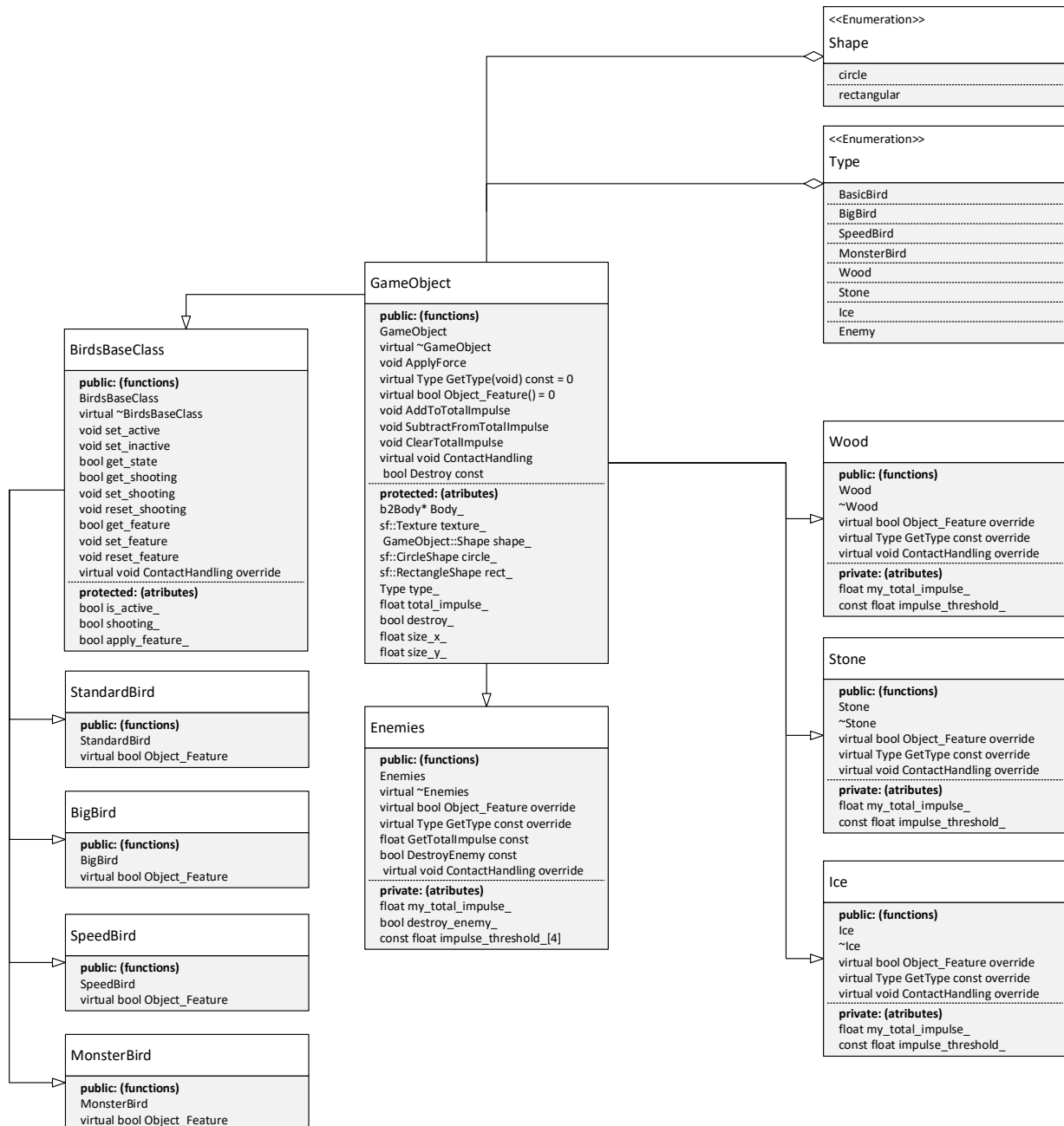


Figure 3. GameObject Class

GameObjectContanctListener:

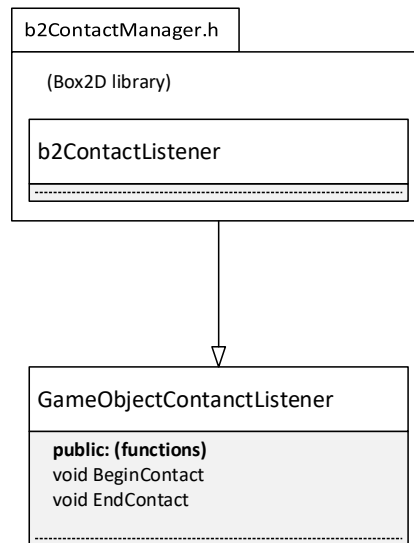


Figure 4. *GameObjectContactListener Class*

Map:

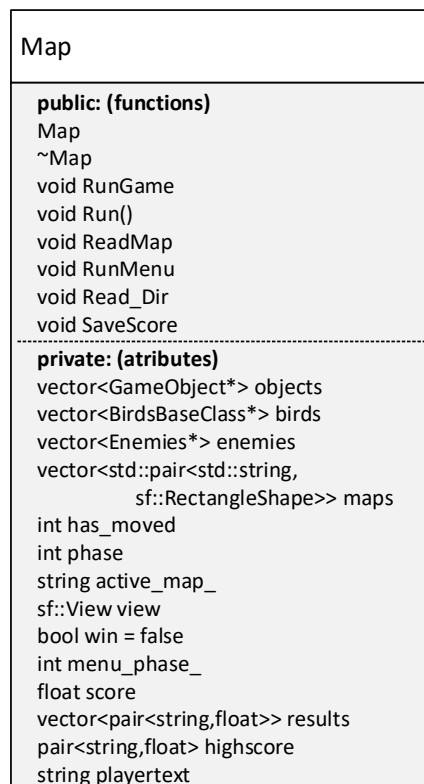


Figure 5. *Map Class*

3 INSTRUCTIONS FOR BUILDING AND USING

3.1 HOW TO BUILD

Pre-requisites:

- OS: Linux Ubuntu 18.04/16.04
- Make sure you have the Box2D and SFML libraries in the standard location. In case you don't have the libraries installed on the standard location or you are not sure, please run the following:

```
#Install Box2D library
sudo apt install libbox2d-dev

#Install SFML library
sudo apt install libsFML-dev
```

- Once you have the libraries installed, clone the game from git repository:

```
git clone git@courses-git.comnet.aalto.fi:CPlusPlus/angry-birds-2019-5.git
```

- Compile and build by running the following commands:

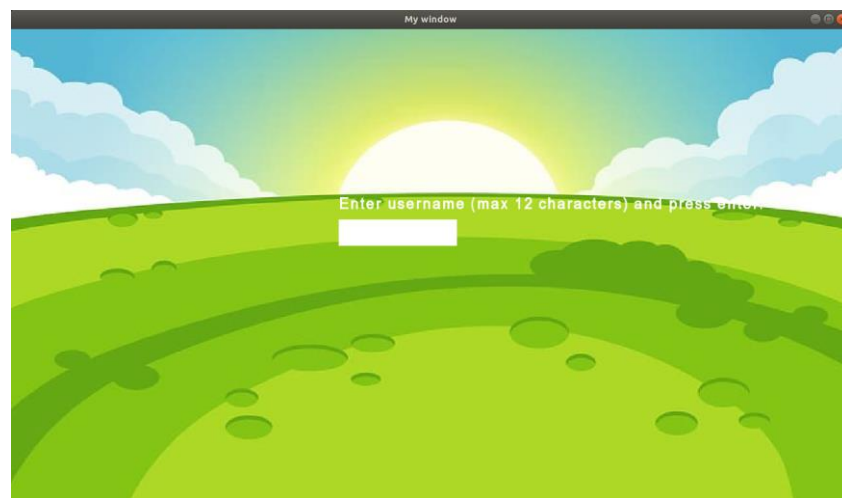
```
cd src
mkdir build
cp Fonts/arial.ttf build
cd build
cmake ..
make
```

- Finally, run the game by executing:

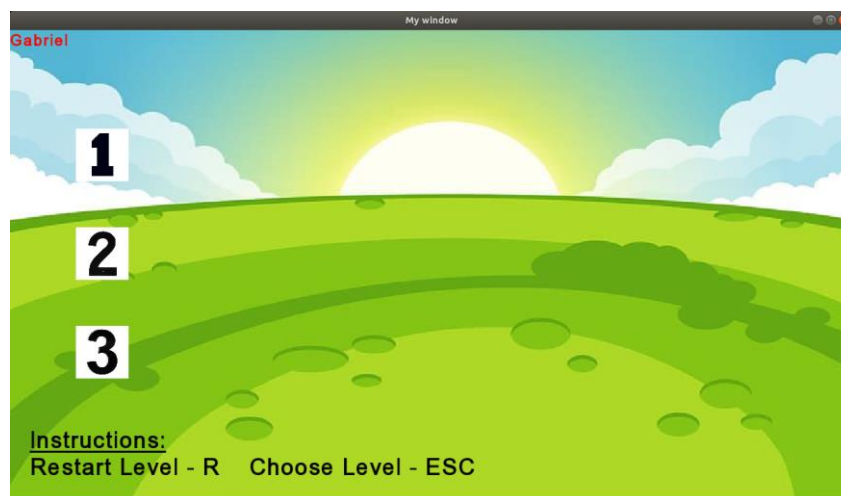
```
./game
```

3.2 HOW TO USE

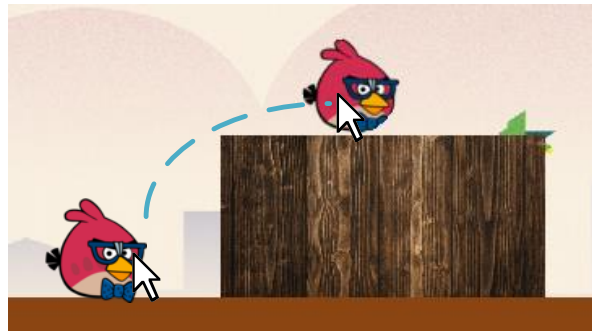
- 1) When running the game, you will be presented with a home screen where you have to input your name or nickname:



- 2) After entering your user name, you will access to the menu screen, where you can select among three different levels, with increasing difficulty (1: easiest – 3: hardest):







- 3) When you select a level you will have a platform with the birds. In order to throw the birds, simply click on the bird and drag the mouse until the desired point, release the mouse button and the bird will be thrown:



- 4) Depending on the interaction with the pigs and materials, these will be destroyed and you will earn points for that movements.
- 5) In addition, there are four types of birds, two of them posses special features:

(Note: the way you activate special features of the birds is by clicking anywhere in the screen after the bird is thrown and before it hits any object)

	This is a standard bird, it does not have any special feature, use it in cases where the pigs are easy targets!
	This is also a standard bird that does not have special features, however, its size is bigger and can cause more damage!
	This bird is a speed bird, when thrown it can increase its velocity tremendously.
	This bird looks small and harmless at first sight, but it can increase its size and cause big damage!

4 TESTING

4.1 TEST MAP AND UI

The map and UI are purely graphical so most of the testing was trial and error. There are also very good tutorials on SFML. We tried different types of pictures, shapes and so on to find the ones that work best for our game. Then the part when we had to the menu was made the same way. Trying different shapes to see what suits for the type of menu we implemented. When making the csv files for the different maps we only put a piece to see where it goes and then used that as a reference for every other piece. We started with building a basic `RunGame()` function in the master branch and made a base-map that we could use for testing and also as a base for further development such as birds and enemies. We had a friend of ours to test the basic UI to get some feedback regarding for an example how user-friendly it was.

4.2 TEST BIRDS CLASS

The birds class consists of a base class called `BirdsBaseClass` and every new bird we want to implement inherits from this class. In addition, the `BirdsBaseClass` inherits from `GameObject` class where we define the `Box2D` and `SFML` associated objects for the entities interacting in the actual game. We created a `birds` branch in our git repository to test there the new birds created, features, functionalities and so on. The way we tested the `Birds` classes and functionalities was by first creating a standard bird with no special feature and adding a particular texture to it, this bird should appear on the map in the desired location, i.e. on the shooting platform, and it can be thrown in the scenery. At this point the collisions with the materials and enemies is not tested, instead, the focus was on implementing all the visual and physics functionalities of the birds. In addition, during the testing phase we added several checkpoints to be printed in the standard output to make sure the bird was working properly. The list of parameters printed on the standard output during the testing and debug were the following:

- Position of the bird in 'x' axis.
- Position of the bird in the 'y' axis.
- Print 1/0 when the mouse clicked on a bird or outside a bird.
- Print 1/0 when a special feature is applied to the bird.
- Printing the position of the bird when it is thrown.
- All the methods in the birds class were `void` or `bool`, for this reason printing them in the standard output helped to track that the class was working fine.

4.3 TEST ENEMIES CLASS

Testing the enemies class was divided to two main parts. Firstly we had to make sure each enemy is affected by the objects impact and that the game is winnable, which means that at the end the user can destroy all the enemies, so the enemies densities had to be chosen correctly for that cause. Also we had to make sure that all three phases of the enemies pain (going from happy to beaten) are working correctly and smoothly, so the impulses thresholds for these phases had to be chosen taking into account the enemies density.

4.4 TEST MATERIAL CLASS

The first step with testing the materials was to sync the SFML(Graphic) object with the Box2D(Physics) object. This required a lot of trial and error testing before we managed to make a working base-material. Another thing that also required a lot of testing was the interaction between birds and materials. We had to find a suitable combination of densities, restitutions etc. to make the materials and birds interact as we wanted them to. Then when the health decreasing from the collisions were working, we started to experiment with the thresholds of the materials. We tested with different thresholds to find a balance when the materials were sturdy enough to make the difficulty of the level suitable.

4.5 TEST COLLISION CLASS

The GameObjectContactListener class is made of two methods the first method is called at the very first moment of any contact while the second method is called after the contact is resolved. For testing, first we made sure that callbacks are working. Then we had to make sure that the two objects that are colliding together are identifiable, so we had to set the user data in the Box2D objects to be the object itself, so that one can identify which objects are colliding. After that we had to implement ContactHandle function for materials and enemies which was being called at the end of the contact in order to take some action. For each of these we were using prints console to make sure it works as desired. Next thing which needed to be added is the collision impulse calculation, so at the beginning of the impulse of the object is assumed to be 0 to be able to find the change in the momentum for each object and based on that objects were deleted from the screen. In order to delete we had to first disable contact listening to avoid segmentation fault and core dump. Lastly we asked our friends to play the game and tell us their feedback to improve the user experience.

WORK LOG

The tasks are divided and assigned in the following responsibility matrix:

ID	Task	Owner			
		Simon Nordström	Kim Koski	Wesam Koraim	Gabriel Ly
1	Project Definition				
1.1	Game specification meeting	R	R	R	R
1.1.1	Create class HLD	R	R	R	R
1.1.2	Assign responsible	R	R	R	R
1.2	Define project deliverables	R	R	R	R
1.2.1	Define features checklist	R	R	R	R
1.2.2	Create project plan	R	R	R	R
2	Game classes implementation				
2.1	Map and UI	R	R	S	S
2.2	Birds class development	S	S	S	R
2.3	Enemies class development	S	S	R	S
2.4	Materials class development	R	R	S	S
2.5	Collision class development	S	S	R	S
3	Testing				
3.1	Test map and UI	R	R	S	S
3.2	Test birds class	S	S	S	R
3.3	Test Enemies class	S	S	R	S
3.4	Test Materials class	R	R	S	S
3.5	Test Collision class	S	S	R	S
4	Project integration and closure				
4.1	Integrate classes	R	R	R	R
4.2	Prepare project documentation	R	R	R	R
4.3	Project final presentation	R	R	R	R

R = Responsible

S = Support

Following the detailed working hours and tasks completed per week:

Team Member		WEEKS					
		W45	W46	W47	W48	W49	W50
Simon Nordström	Hours	9	7	10	12	11	6
	Work Completed	1) Project Definition: Create project plan, HLD, features, responsibility matrix. Figuring out cmake	GameObject class development Map class development Making a drawable window, background and ground	Implementing a ball with certain speed for testing. Materials class development. Syncing with SFML and Box2D	Merging with birds branch Debugging birds throwing feature. Managing more than 1 bird(birds list). Camera follows bird.	Game menu development. ReadDir() function implementation. Saving score to csv file. Reset button added. Merging with enemies branch.	4) Project integration and closure 4.1 Integrate classes 4.2 Prepare project documentation 4.3 Project final presentation
Kim Koski	Hours	3	7	10	13	9.5	5.5
	Work Completed	Project Definition: Create project plan, HLD, features, responsibility matrix.	GameObject class development Map class development Making a drawable window, background and ground	Implementing a ball with certain speed for testing. Materials class development. Syncing with SFML and Box2D	Managing more than 1 bird(birds list). Camera follows bird. Camera moves with buttons. Readfromfile() function development.	Game menu development. Reset button added. Added 2 more types of material. Check state of bird. Merging with enemies branch.	4) Project integration and closure 4.1 Integrate classes 4.2 Prepare project documentation 4.3 Project final presentation
Wesam Koraim	Hours	3	6	8	13	15	5
	Work Completed	Project Definition: Create project plan, HLD, features, responsibility matrix.	Learning more about Box2D and SFML, and Enemies class structure design and development. Enemies class is a sub-class from GameObject	Learning more about Box2D collisions and b2ContactListener class, and testing throwing balls at the enemies and detecting collisions	Implementation of GameObjectContactListener class which inherits from the b2ContactListener class and setting up the callbacks for the enemies class. Adding impulse calculation to handle enemies pictures changing.	Handling materials disappearing. Adding score calculation to materials and enemies. Merging with the birds and master branches. Handling the score counter to the map class.	4) Project integration and closure 4.1 Integrate classes 4.2 Prepare project documentation 4.3 Project final presentation
Gabriel Ly	Hours	6	8	10	12	12	7
	Work Completed	1) Project Definition: Create project plan, HLD, features, responsibility matrix.	2.2 Birds class development: - Create <code>birdBaseClass.hpp</code> and <code>birds.hpp</code> - Inherits from <code>GameObject</code> class	2.2 Birds class development: - Implement one bird with SFML texture and assign physics behavior. 3.2 Test birds class: - Test the implemented bird. - Check that the bird appears in the map.	2.2 Birds class development: - Implement an additional bird with special feature that increases its velocity. - Implement the birds throwable functionality. 3.2 Test birds class: - Test the throwable functionality. - Test the special feature of the bird. - Test that the birds work according to the features and the physics.	2.2 Birds class development: - Implement two new birds, one of them with new functionality that increases its size. - Complete 4 birds in total. - Set a limit area where the bird can be thrown. 3.2 Test birds class: - Test the throwable functionality. - Test the special feature of the new birds. - Test that the birds can be thrown according to limitation.	4) Project integration and closure 4.1 Integrate classes 4.2 Prepare project documentation 4.3 Project final presentation