Angru Burds

Object oriented programming with C++
ELEC-A7151

Project Documentation

Authors:

Marc Aguado 886732

Roope Kontiainen 707691

Laura Heikkilä 778727

Venla Valve 783974

December 12, 2021

Contents

1 Overview of project

1.1 Game features

Our project is a simple Angry Birds mock-up game, which is implemented using the programming language C++. When started, the game shows the user a simple background with the name of the game, Angru Burds. After the user presses play button, a menu selector with three different levels is presented. The difficulty of levels increases from level to level: more enemies are to be destroyed and they might be better shielded with objects, such as boxes and rocks. Once a level is chosen, the user is taken to the level and the game starts. When a level is started, its timer is triggered and positions of game objects are updated every $\frac{1}{60}$ seconds. The game objects are rigid bodies whose interactions with each other are determined by their e.g. their weight and speed, using a physics library Box2D. For implementing the graphical user interface of the game, the software uses widget toolkit Qt.

The game is played with two different birds, a pink bird and a blue bird, which the user shoots by dragging and releasing the bird as if it were on a slingshot. The blue bird is a special bird that has a speed-boost that can be triggered by clicking the bird, after it has been shot with the drag-action. As the bird is shot, the view follows the bird until it hits the ground and eventually dies, after which the view is taken back to the next bird, which becomes shootable. Moreover, the bird dies if it is shot out of the scene. The enemies are purple pigs that can be killed by colliding with them. The user can either hit the pigs with the bird or make other game objects fall on them. The effect of the collision must be fatal in order for the pig to die. Fatality is determined by measuring the kinetic energy of colliding object and the pig. Other objects besides birds and pigs include wooden boxes and pillars as well as rocks and rock pillars. Wooden boxes and pillars can be destroyed with the bird, if being hit with sufficient speed, but objects made of rock are undestroyable. Moreover, rock objects are twice as heavy as their wooden counterparts. The number of the game objects still present in a level is shown in real time to the user. The current score is also visible and it is updated, when the player kills pigs or destroys obstacles.

The user can win the game by killing all the enemy pigs with the birds made available. The game is lost if all birds are used and there is at least one enemy left. Moreover the game ends if the user exits the level at any given point by clicking the exit button, which takes the user back to the main menu. Clicking the exit button in menu results in closing the application.

1.2 Further improvements and ideas

There are currently three levels, which are constructed from text files containing names of the objects and their scene coordinates. This allows for easy modifications in level design and hypothetically, one could fairly easily implement their own level without understanding of programming. If we were to take this project further, it would be interesting to implement a level constructor, in which such text file would be generated based on user input, for example user dragging a desired object to the scene.

Additionally, if we had time, we would have liked to have multiple special birds and enemies of various sizes. A more ambitious goal would be adding animations, for example having pigs that explode when they are killed. Our project only has the very basic graphics but the groundwork is done, so adding these features should be attainable.

2 Software structure

When main.cpp is run, a new game is created. A game is essentially a QGraphicsView from library Qt, which creates a scene it shows. The game object has different slots for different scenes to be displayed: menu, level selector and the three different levels. It listens to the buttons to determine, which one of these displays should be shown to the user. The game object also has a level loader object that it calls when the user wants to start a level. When a level is selected by pressing the play button corresponding to that level, Game calls its level loader object's Load funtion with the correct file name. The scene is also passed to the level loader when it is created.

Level loader's functionality is to hold the game core, where the game is run and load it with objects. The scene originally created in Game object (view) is passed to the core. Before loading the objects to the scene of the game core, the level loader first initializes the core by calling its Load() function. This triggers the game core to create a new box2d world, a level ground and a timer that is connected to the GameLoop(), which runs the game. The level loader then proceeds to create objects according to the text file it is reading and placing them in the scene of the game core. When these objects are created, their physical bodies are placed in the box2d world created by the core. When loading is done, the timer is started and the level starts running. The core takes care of running the game with GameLoop() function, which updates the positions of the objects by simulating the physics in box2d world and translating the coordinates to screen coordinates.

If a game is either won or lost, the core stops its own timer and clears the screen and box2d world from objects. It then calls the game object to display main menu again. Alternatively, if game detects that exit button is pressed, it calls the game core to start the clearing and exit processes. Essentially, the same game core and level loader are used until the user quits the application. The relationships of the classes are better explained in Figure 1.

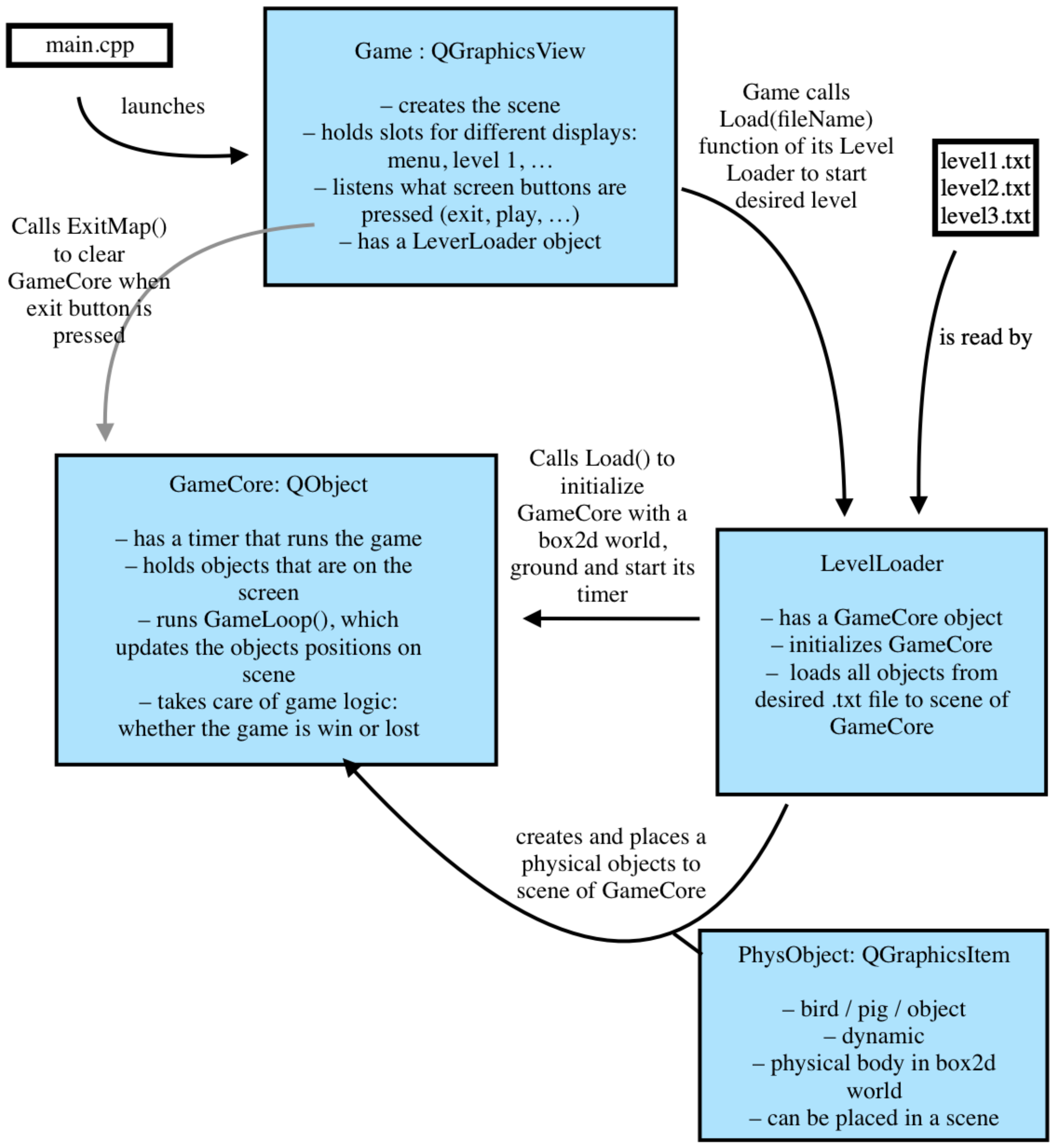

main.cpp
launches
Game : QGraphicsView
– creates the scene
– holds slots for different displays: menu, level 1, ...
– listens what screen buttons are pressed (exit, play, ...)
– has a LeverLoader object
Game calls Load(fileName) function of its Level Loader to start desired level
level1.txt
level2.txt
level3.txt
is read by
Calls ExitMap() to clear GameCore when exit button is pressed
GameCore: QObject
– has a timer that runs the game
– holds objects that are on the screen
– runs GameLoop(), which updates the objects positions on scene
– takes care of game logic: whether the game is win or lost
Calls Load() to initialize GameCore with a box2d world, ground and start its timer
LevelLoader
– has a GameCore object
– initializes GameCore
– loads all objects from desired .txt file to scene of GameCore
creates and places a physical objects to scene of GameCore
PhysObject: QGraphicsItem
– bird / pig / object
– dynamic
– physical body in box2d world
– can be placed in a scene

Figure 1: Software structure logic

The class inheritance of objects and their relation to the two external libraries, Box2D and Qt, is shown in Figure 2. In short, all game objects have a Box2D body that is placed in a Box2D world. The bodies have different masses and textures, and the physics library calculates their coordinates based on the forces acting on them. The objects also have a graphical representation so that they can be placed in a Qt scene and shown to the user. As a result, the objects have two sets of coordinates at all times, the ones in Box2D world, which are in meters, and scene coordinates, which are in pixels. The diagram is missing the classes Rock, BoxType2 and RockType2, as their position in the diagram is equivalent to that of class Box.

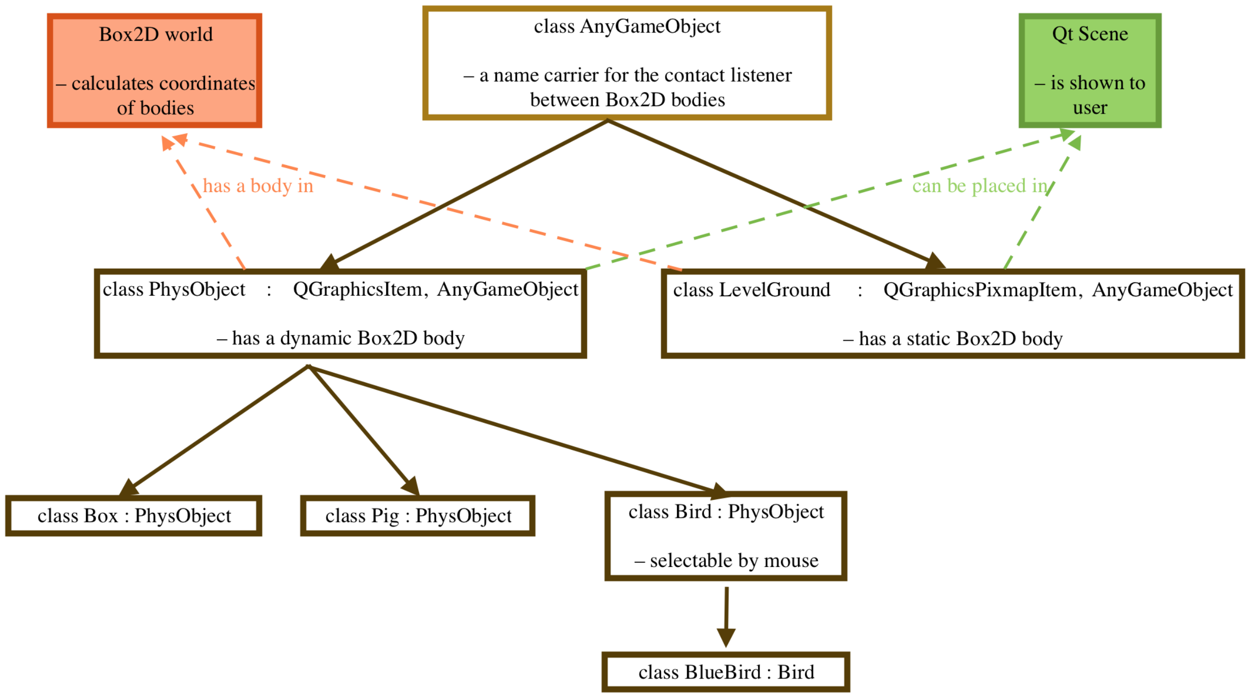

Box2D world
– calculates coordinates of bodies
class AnyGameObject
– a name carrier for the contact listener between Box2D bodies
Qt Scene
– is shown to user
has a body in
can be placed in
class PhysObject : QGraphicsItem, AnyGameObject
– has a dynamic Box2D body
class LevelGround : QGraphicsPixmapItem, AnyGameObject
– has a static Box2D body
class Box : PhysObject
class Pig : PhysObject
class Bird : PhysObject
– selectable by mouse
class BlueBird : Bird

Figure 2: Class inheritance of game objects and their relation to external libraries

3 Building the software

The game should be built using a Linux operating system.

3.1 Step-by-step guide

1. Install Qt.

   Download Qt online installer: Linux users can use the file 'qt-unified-linux-x64-4.2.0-online.run' inside this directory. Otherwise, download the appropriate installer from: https://www.qt.io/download-qt-installer

   Run the installer as adminstrator:

   ```
   $ sudo ./qt-unified-linux-x64-4.2.0-online.run
   ```

   In case of an error, check section Troubleshooting, error #1.

   You are asked to create a Qt account to login. After a few steps you are asked to select the directory for the Qt installation. Which ever directory you choose, we will refer to it as: <your Qt install dir>.

   Below that, check the box: Custom installation

   On the next page, select the following components (And all that was selected by default): Qt 6.2.2/Desktop gcc 64-bit Qt 6.2.2/Qt 5 Combatibility Module

   Install.

2. Cofigure Qt installation directory:

   Edit the only line in config.txt as follows: Qt install directory:<your Qt install dir>

   This step may not be necessary. But it is if you have conflicting Qt installations.

3. Install Angry birds.

   Run the build script:

   ```
   $ ./build.sh
   ```

4. The game is ready.

   Start the game:

   ```
   $ ./'Angry Birds'
   ```

   In case of an error, check section Troubleshooting, error #2.

3.2 Troubleshooting

Error #1:

error while loading shared libraries: libxkbcommon-x11.so.0: cannot open shared object file: No such file or directory

Fix:

```
$ apt install libxkbcommon-x11-0
```

Error #2:

Could not load the Qt platform plugin "xcb" in "" even though it was found.

Fix:

Make sure the following libraries are installed:
libxcb-icccm4
libxcb-image0
libxcb-keysyms1
libxcb-render-util0
libxcb-xkb1
libxkbcommon-x11-0
libxcb-randr0

4 Simple user guide for playing the game

This section provides instructions on how to launch a level, shoot the birds and possibly win the game.

Starting the game: Once the application is launced, the user is presented with a starting scene with a play button, illustrated in Figure 3. Pressing the orange play button takes the user to level menu, as seen in Figure 4. The desired level is started by pressing the corresponding play button.

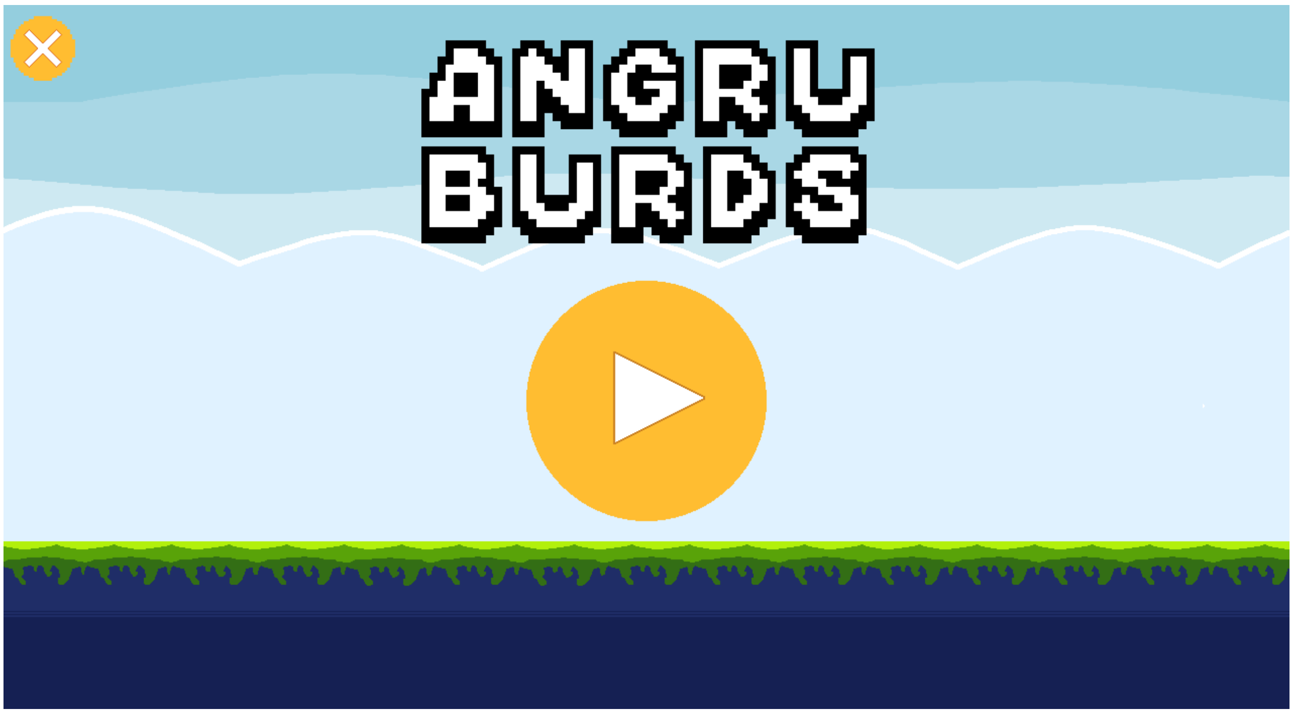


Figure 3: Starting scene

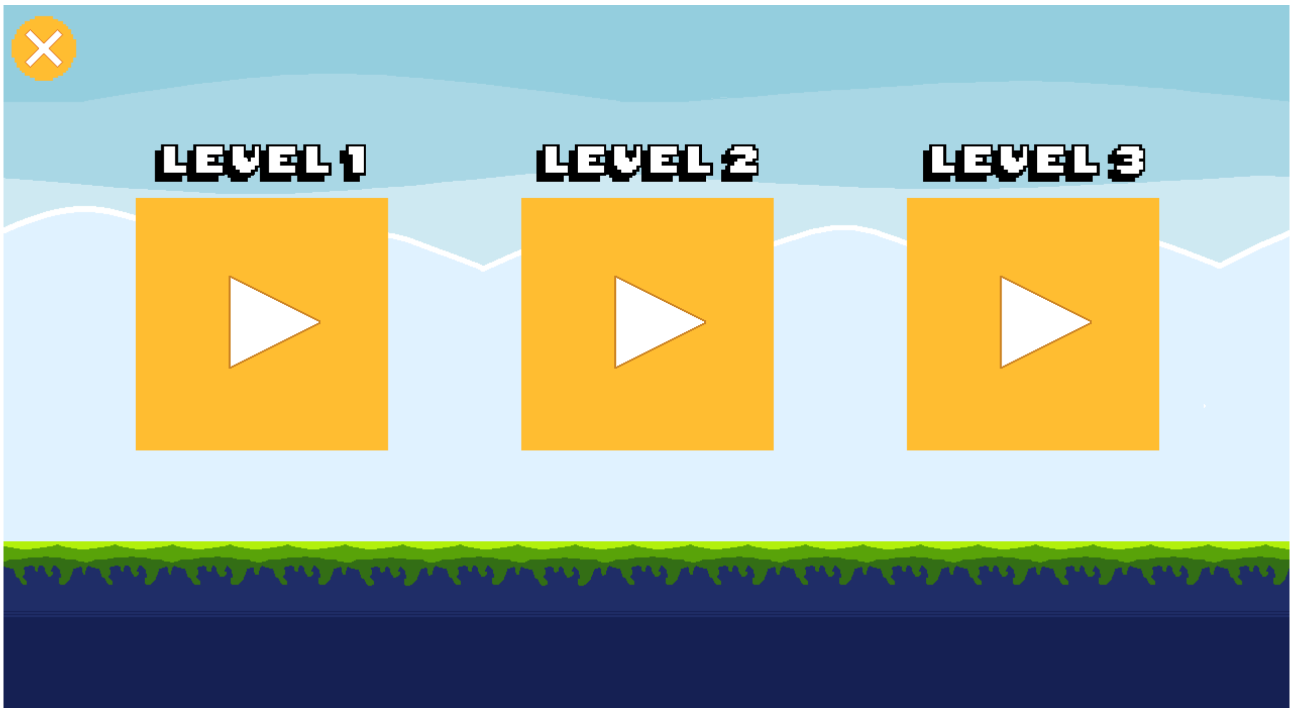
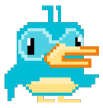
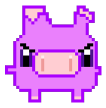


Figure 4: Level menu

Shooting a bird: The available birds are lined up on the ground, and the leftmost birds is shootable. Clicking the other birds in line has no effect yet. The first bird is shot by pressing the bird and dragging the mouse away from the pigs, as if shooting a slingshot. To release the bird in desired direction, simply release the mouse to send the bird flying. There are two different birds: pink bird in Figure 5a and blue bird in Figure 5b. The enemy is illustrated in Figure 5c. The blue bird has a speed boost action, which is triggered by clicking the bird, once it has been shot with the slingshot. Note that besides the speed boost, the user cannot control the birds after they have been shot. The birds die once they have hit the ground for 1.5 seconds. After the bird dies, the view focuses on the next bird if there are any left, who then jumps a little to indicate that it is now shootable. Figure 6 illustrates a scene from the game: a blue bird is flying towards the enemy.

(a) A pink bird


(b) A blue bird


(c) A pig

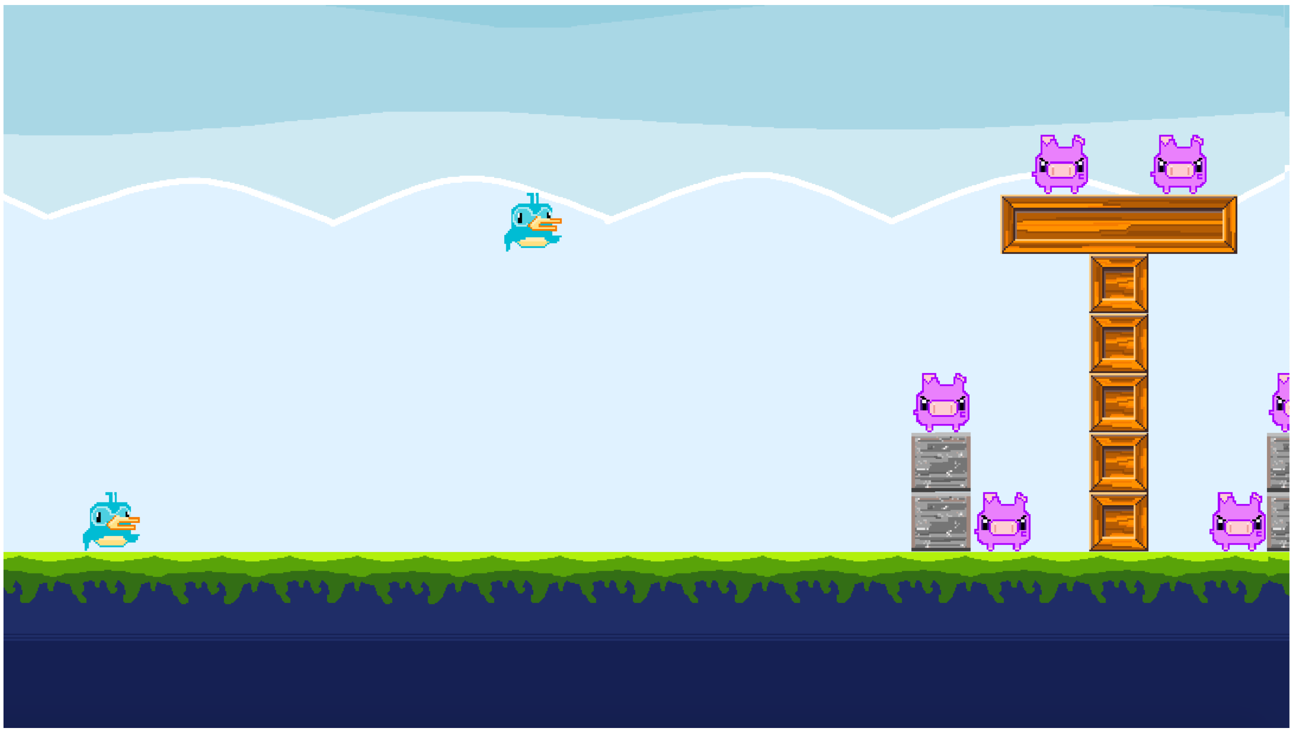


Figure 6: Level play

Winning the game: The goal of the game is to kill all the pigs in a level by shooting birds at them. A pig dies if it collides hard enough with a bird or another game object, such as box or another pig. If the player runs out of birds before all pigs have died, the game is lost.

5 Testing

We tested the physical object classes (Bird, Pig, ...) with a unit test to see that the physical Box2D body coincides with the graphical image shown on the screen. The test created a GameCore and added the object in the scene. The object was then made to rotate against the level ground so that any issues in the behaviour of the object could be detected when the scene is set visible.

The test was motivated by our difficulties to understand the two different coordinate systems the game is using: meters in box2d world for the physical body of the object and pixels on the scene for the graphical body. The origin of the physical body is in the center of the object, whereas the origin of the graphical image is on the right top corner of the object. The unit test helped us to verify that our conversion methods were working correctly. We also detected that the graphical image was being rotated around its top right corner, which caused unwanted behaviour. We fixed the problem by manually setting the Qt origin point to the center of the object.

The test was also used to verify that the GameCore class was functional and its Load() function created a Box2D world and added a level ground to both the world and the scene. Moreover, its AddPig, AddPig and AddObstacle functions were working as expected.

The rest of the game was developed without unit tests by compiling and running the whole application. Errors were detected by simply playing the game. Although this might not be the ideal approach in large projects, it was very time efficient. Moreover, in the beginning of the project, the master branch contained a very simple first application, which the team members cloned to their own branches. Each member then developed and tested their functionalities on the very basic first version and the branches were merged in a meeting. The merging was unexpectedly time consuming and the group will most likely avoid this approach in future projects.

6 Worklog

6.1 Marc

Week	Tasks completed
43	Project plan
44	Studying Qt
45	Level loader module
46	Produced a system for loading the levels from text files Started working on background
47	Finished background graphics Started working on level menu
48	Level menu and level selector graphics and implementation Merged the groups work in a meeting Continued working on level loading Fixed some bugs Graphics for level ground
49	Finalized level loading from files Implemented two new blocks Designed two levels Valgrind testing More advanced graphics for level menu Fixed some more bugs

6.2 Roope

Week	Tasks completed
43	Project plan
44	Studying Qt Makefile
45	Simple first program to create a box and a platform, where the box could "jump" and collide. Basic structure for some of the classes needed for the project
46	Started implementing a game logic system to keep track of the birds etc.
47	Contact listener for collision detection
48	Split LevelLoader to GameCore and LevelLoader Merged the groups work in a meeting Coordinate system fixed
49	Updated the building system: build.sh, Makefile, and instructions Improved the software structure Wrote unit tests

6.3 Venla

Week	**Tasks completed**
43	Project plan
44	Studying Qt
45	Graphics for two different kind of birds
46	Implemented pig and obstacle classes based on Roope's bird class Graphics for pig and obstacle
47	Implemented rock class and did graphics for it Implemented speed boost for the bird Fixed angle bug
48	Merged the groups work in a meeting
49	Camera follows the bird Stopped game from crashing when exiting Only one bird at a time is shootable Graphics for stone and wooden pillar Real time update of items left shown on screen Source code comment blocks Wrote documentation

6.4 Laura

Week	**Tasks completed**
43	Project plan
44	Studying Box2D
45	
46	Made it possible to select a bird by clicking it with a mouse. Also each Bird can only be thrown once
47	Slingshot-like shooting
48	Merged the groups work in a meeting
49	

7 Weekly hours used

Week	Roope	Marc	Venla	Laura
43	12	3	3	3
44	28	10	3	5
45	9	4	3	2
46	18	6	4	4
47	11	8	7	4
48	7	21	4	4
49	52	30	42	3
Sum	**135**	**82**	**66**	**25**