Angry Birds - Documentation
Group 2

Kankaanpää Kalle 793456 Juusti Mikko 793469

Hintsala Jaakko 886790 Porio Sami 793540

ELEC-A7151
December 2021

1 Overview

We sought out to do an Angry Birds like game in C++. The game plays like the hit game from Rovio. It has an appealing main menu from which the player is able access a level selector. After selecting a level the game starts. The goal of the game is to shoot different kinds of birds from a slingshot across a field and hit evil pigs. After hitting each pig the level is cleared. The catch is that the player has a limited amount of birds to achieve this and clearing the level without using all of the available birds grants extra points.

The game has three levels from which to choose. The game does not have a support for custom levels. Also a moving camera helps recon the evil pig fortress. The gameplay is kept fresh with different kinds of birds and increasingly difficult levels. Adding to the competitive aspects the game allows players to record their high scores for each of the levels with their own custom nicknames.

2 Software structure

The structure of the game has changed a bit from the original plan, but not too much.

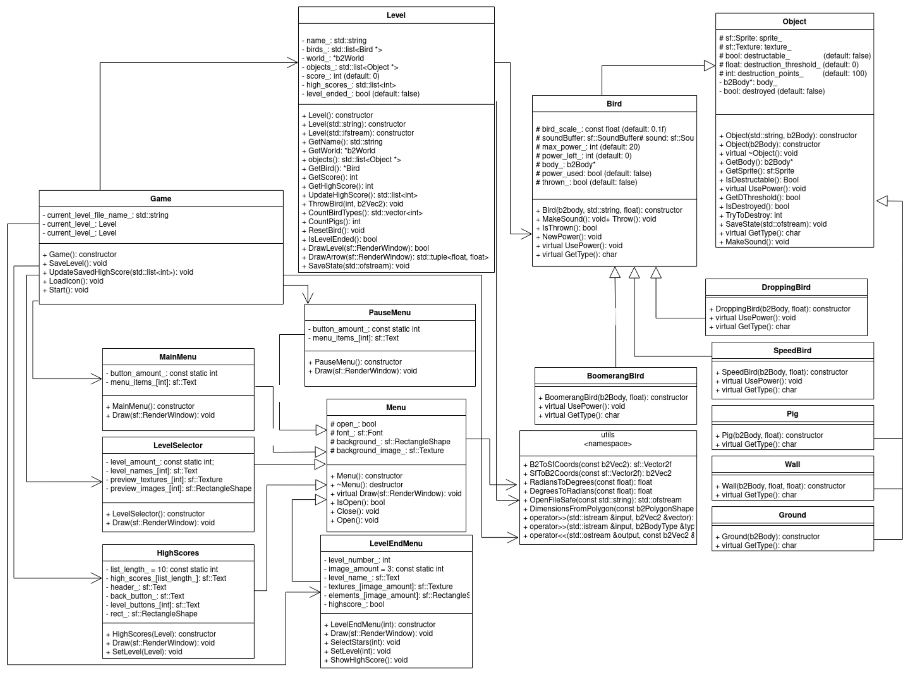
Level
- name_: std::string
- birds_: std::list<Bird *>
- world_: *b2World
- objects_: std::list<Object *>
- score_: int (default: 0)
- high_scores_: std::list<int>
- level_ended_: bool (default: false)
+ Level(): constructor
+ Level(std::string): constructor
+ Level(std::ifstream): constructor
+ GetName(): std::string
+ GetWorld: *b2World
+ objects(): std::list<Object *>
+ GetBird(): *Bird
+ GetScore(): int
+ GetHighScore(): int
+ UpdateHighScore(): std::list<int>
+ ThrowBird(int, b2Vec2): void
+ CountBirdTypes(): std::vector<int>
+ CountPigs(): int
+ ResetBird(): void
+ IsLevelEnded(): bool
+ DrawLevel(sf::RenderWindow): bool
+ DrawArrow(sf::RenderWindow): std::tuple<float, float>
+ SaveState(std::ofstream): void
Object
# sf::Sprite: sprite_
# sf::Texture: texture_
# bool: destructable_ (default: false)
# float: destruction_threshold_ (default: 0)
# int: destruction_points_ (default: 100)
- b2Body*: body_
- bool: destroyed (default: false)
+ Object(std::string, b2Body): constructor
+ Object(b2Body): constructor
+ virtual ~Object(): void
+ GetBody(): b2Body*
+ GetSprite(): sf:Sprite
+ IsDestructable(): Bool
+ virtual UsePower(): void
+ GetDThreshold(): bool
+ IsDestroyed(): bool
+ TryToDestroy: int
+ SaveState(std::ofstream): void
+ virtual GetType(): char
+ MakeSound(): void
Bird
# bird_scale_: const float (default: 0.1f)
# soundBuffer: sf::SoundBuffer# sound: sf::Sou
# max_power_: int (default: 20)
# power_left_: int (default: 0)
# body_: b2Body*
# power_used: bool (default: false)
# thrown_: bool (default: false)
+ Bird(b2body, std::string, float): constructor
+ MakeSound(): void+ Throw(): void
+ IsThrown(): bool
+ NewPower(): void
+ virtual UsePower(): void
+ virtual GetType(): char
Game
- current_level_file_name_: std::string
- current_level_: Level
- current_level_: Level
+ Game(): constructor
+ SaveLevel(): void
+ UpdateSavedHighScore(std::list<int>): void
+ LoadIcon(): void
+ Start(): void
PauseMenu
- button_amount_: const static int
- menu_items_[int]: sf::Text
+ PauseMenu(): constructor
+ Draw(sf::RenderWindow): void
DroppingBird
+ DroppingBird(b2Body, float): constructor
+ virtual UsePower(): void
+ virtual GetType(): char
MainMenu
- button_amount_: const static int
- menu_items_[int]: sf::Text
+ MainMenu(): constructor
+ Draw(sf::RenderWindow): void
SpeedBird
+ SpeedBird(b2Body, float): constructor
+ virtual UsePower(): void
+ virtual GetType(): char
BoomerangBird
+ BoomerangBird(b2Body, float): constructor
+ virtual UsePower(): void
+ virtual GetType(): char
Menu
# open_: bool
# font_: sf::Font
# background_: sf::RectangleShape
# background_image_: sf::Texture
+ Menu(): constructor
+ ~Menu(): destructor
+ virtual Draw(sf::RenderWindow): void
+ IsOpen(): bool
+ Close(): void
+ Open(): void
Pig
+ Pig(b2Body, float): constructor
+ virtual GetType(): char
LevelSelector
- level_amount_: const static int;
- level_names_[int]: sf::Text
- preview_textures_[int]: sf::Texture
- preview_images_[int]: sf::RectangleShape
+ LevelSelector(): constructor
+ Draw(sf::RenderWindow): void
utils
<namespace>
+ B2ToSfCoords(const b2Vec2): sf::Vector2f
+ SfToB2Coords(const sf::Vector2f): b2Vec2
+ RadiansToDegrees(const float): float
+ DegreesToRadians(const float): float
+ OpenFileSafe(const std::string): std::ofstream
+ DimensionsFromPolygon(const b2PolygonShape
+ operator>>(std::istream &input, b2Vec2 &vector):
+ operator>>(std::istream &input, b2BodyType &typ
+ operator<<(std::ostream &output, const b2Vec2 &
Wall
+ Wall(b2Body, float, float): constructor
+ virtual GetType(): char
Ground
+ Ground(b2Body): constructor
+ virtual GetType(): char
HighScores
- list_length_ = 10: const static int
- high_scores_[list_length_]: sf::Text
- header_: sf::Text
- back_button_: sf::Text
- level_buttons_[int]: sf::Text
- rect_: sf::RectangleShape
+ HighScores(Level): constructor
+ Draw(sf::RenderWindow): void
+ SetLevel(Level): void
LevelEndMenu
- level_number_: int
- image_amount = 3: const static int
- level_name_: sf::Text
- textures_[image_amount]: sf::Texture
- elements_[image_amount]: sf::RectangleS
- highscore_: bool
+ LevelEndMenu(int): constructor
+ Draw(sf::RenderWindow): void
+ SelectStars(int): void
+ SetLevel(int): void
+ ShowHighScore(): void

Figure 1: UML-diagram of the software structure. Presented as an object oriented UML model.

Listing and more details about the largest classes.

2.1 Game class

We have all governing Game-class which controls the menus, switching between them and levels. It also handles the calls for level saving and loading from files.

2.2 Level class

The Game-class has an instance of the Level-class. This class has the information about the SFML graphics world and also the Box2d world. It is also responsible for drawing the game and running the physics engine.

2.3 Object class

The object class is a super class for all the objects in the game world. Bird, wall, ground, pig all inherit the object class. All of the objects have a physics body in Box2d and a sprite image for SFML. All objects keep track of their own position and orientation. One notable mention is that the bird class has multiple children, these are all the different kind of birds and no instance of Bird itself is used.

2.4 Utils

We created a name space for some useful utility tools to be used pretty much everywhere in the program. These tools include the graphic engine SFML coordinate mapping to the physics engine, Box2d, coordinates and vice versa, the file streaming tools and also some generally useful functions. All global constants are included from the converters header file.

3 Instructions

The program has been designed from the start to be easy to use across different desktop platforms. After cloning the git repository on has to only run the build script provided in the root folder of the project.

After that, if the cmake compilation ran without problems the executable named 'angry_birds' has been created in the build/ folder. Executing this will open the program and set the user to the main menu. In the main menu, there are a few options to choose from. Play button will open the level selector from which the game itself can be accessed. The high scores button will show high scores of different players in different levels. The exit button will exit the game, throw the user back to desktop and close the program execution.

4 How to compile the program

Compiling is made easy by utilizing CMake, its lists, git submodules and allready build scripts. By running the platform specific build.sh or build.ps1 script on Linux or Windows respectively it will make the program and build all necessary file structures for the build of the program. However because external libraries and cmakelists are utilized, using only the 'make' command on the root folder is not sufficient. One note about Linux has to be made, if the script.sh cannot be run with the command ./build.sh it probably has wrong access rights, namely it cannot be executed. This can be fixed by using the command chmod +x build.sh.

5 How to use the software

Playing the game itself is designed to be very intuitive. After choosing a level the player is inserted into the selected level. On the top of the screen is shown a pause button, amount and types of still available birds, the remaining number of evil pigs, and current score along with the high score of the level.

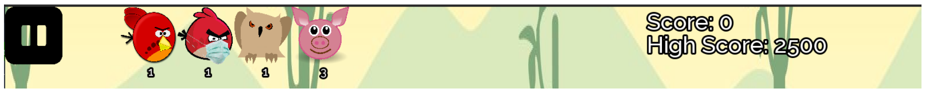

Figure 2: Example status of the current level hud.

The player can shoot a bird by using their mouse on the left hand side of the bird sitting in the slingshot.

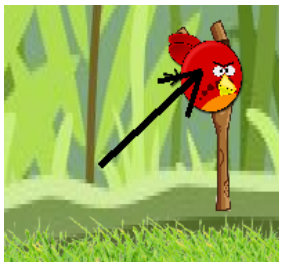

Figure 3: The arrow shows the launch angle and the power. The longer the arrow is the more power the bird will have.

By clicking the left mouse button while the arrow is visible the bird is hurled towards the evil pigs. When the bird is in flight, pressing the left mouse button again will activate the bird's power which is different for every kind of bird. All birds have some power, testing them out will teach the player what each bird does.

To complete the level the player has to clear it of pigs. Points are awarded for each destroyed pig, structure and also for each bird left in the arsenal. The level can also be completed without getting all of the pigs but this will never result in a three star performance.

The player is able to move the camera view when they have not launched a bird and the world is not moving (all objects are still) by using the arrow buttons. This helps the player to decide where to launch the bird. By pressing the space bar the player is able to reset the view back to bird so they can shoot it.

6 Testing

Our systematic testing methods have been employed in the weekly meetings we have had. Before merging the feature branches we tested the software independently and also after code implementation. Adding to this, we created unit tests for most of the utility functions. We chose to write unit test for the functions in utils namespace since they are pretty easy to test because they don't have any side effects. It's also important that the utility functions work as intended and testing allowed us to be sure of that. Some utility functions also have algorithms which would have been hard to test without the unit tests. We were fortunate with the tests and actually found a bug with the initial implementation of DimensionsFromShape function while writing the tests. Furthermore, along the technical testing we have done a notable amount of play testing so the game itself would play as intended.

7 Work log

We had a rough start as we started building the program on top of the Qt graphics library. This idea was totally abandoned after three weeks of building and debugging. After disbanding Qt library and qmake totally we started the game development with SFML and Box2d only. We had relatively regularly a meeting to which all of us attended. The average time of these meetings were 3-6 hours. Each week we checked how the parts designated to each member of the group were progressing. We merged the complete parts to master from their own feature branches. We also helped each other out if someone had gotten really stuck or had something common to ask. Also after all this we decided what to do next week allocating some new job or feature implementation to each member somewhat according to the schedule we made in the project plan.