NVS Project 5th Class

Robocode

1

Submitted By :
Joel Klimont

Contents

1 Introduction

1.1 Robocode

Robocode is a multiplayer game in which players don't directly compete against their opponents but instead they have to program a robot which fights for them. The goal is to program a robot that can defeat all opponents and survive until the end of the game.

Robots

Each robot has a starting energy of 100%. In order to shoot a bullet at an enemy the robot has a gun that can be rotated 360° degrees. The gun also acts as a scanner. If a robot intersects with the range of the gun the program of the player will receive the position and energy of the scanned robot. What the player chooses to do with that information is entirely up to him. He could for example, shoot at the player or store his position in a vector and try to predict his speed/ position using the last points as reference.

2 Technologies

2.1 Asio

Asio is a Boost C++ library that enables a developer to send and receive messages over the network. The library is open source and was accepted into the Boost library on 30 December 2005.

2.2 Google protobuf and grpc

Protocol Buffers

Protobuf is a language neutral way of defining messages that can be sent over the network. The following code is an example from the project:

```
message Position {
    double y = 1;
    double x = 2;
}
```

```

message Robot {
    int32  id = 1;
    double energy_left = 2;
    Position pos = 3;
}
```

Listing 1: Protobuf example

There are two different messages defined here. The first one is a position which has two attributes an ”y” and a ”x” coordinate. The second message uses the first one and has two more additional attributes. The ”id” is a unique identifier given to every robot a the start of the game. The ”energy_left” value describes the energy of the transmitted robot.

gRPC

GRPC stand for ”Google remote procedure calls”. It enables developers to send and receive protobuf messages over the network.

2.3 Clipp

Clipp is used for parsing all command line arguments from the user.

2.4 Json

The program can also be configured using the ”config.json” file.

```
{
  "port" : 5000,
  "streaming_port": 5010,
  "username" : "",
  "server_ip" : "",
  "background_robot": true ,
  "no_menu_host": false ,
  "log_level": "info"
}
```

Listing 2: Json config

port
Specifies the port used by the game server and the game client will connect to.
streaming_port
Specifies the port used by the streaming server and the streaming client will connect

to.
username
Sets the username of the player.
server_ip
Sets the server IP the client will connect to.
background_robot
Enable or disable the background robot rendered in the menu.
no_menu_host
Disable curses menu and host server.
log_level
Set the the log level. Can be: "debug", "critical", "err", "trace", "warn" or "off".

2.5 Curses

In order to display the game the C-Library "Ncurses" is used. The robots are rendered as 4x7 boxes and posses a gun displayed as two "#". Also a menu is displayed to the user when he starts the program. Its possible to connect or host a server using the options provided by the menu.

3 Software Architecture

In the following section the software architecture of the program will be described. The program consists of five different parts namely: the curses drawable objects, the game objects, the game server and game client, the streaming server and streaming client and the players program.

3.1 Drawable Objects

There are only two different drawable game objects, the "Bullet" and "Robot". They can be directly drawn to the terminal by calling their "draw()" function.

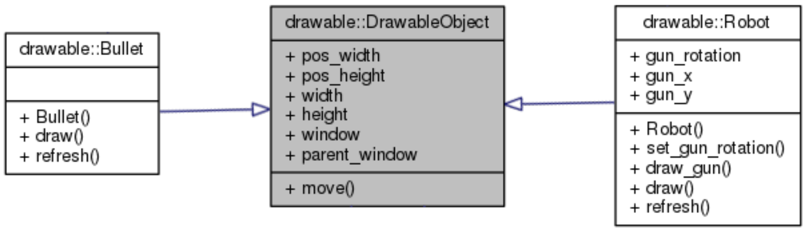

Figure 1: Drawable Objects UML Diagram

3.2 Game Objects

The game objects are being used by the game server to calculate the path of bullets, robots etc. and send the data back to the players. The class"Robot" and "Bullet" each contain a corresponding drawable object and can be rendered by calling the "draw()" function. All game objects inherit their base functionality from the class "GameObject". It contains the position, width, height and speed of the object. It also implements a "tick()" function, which applies the current speed to the position.

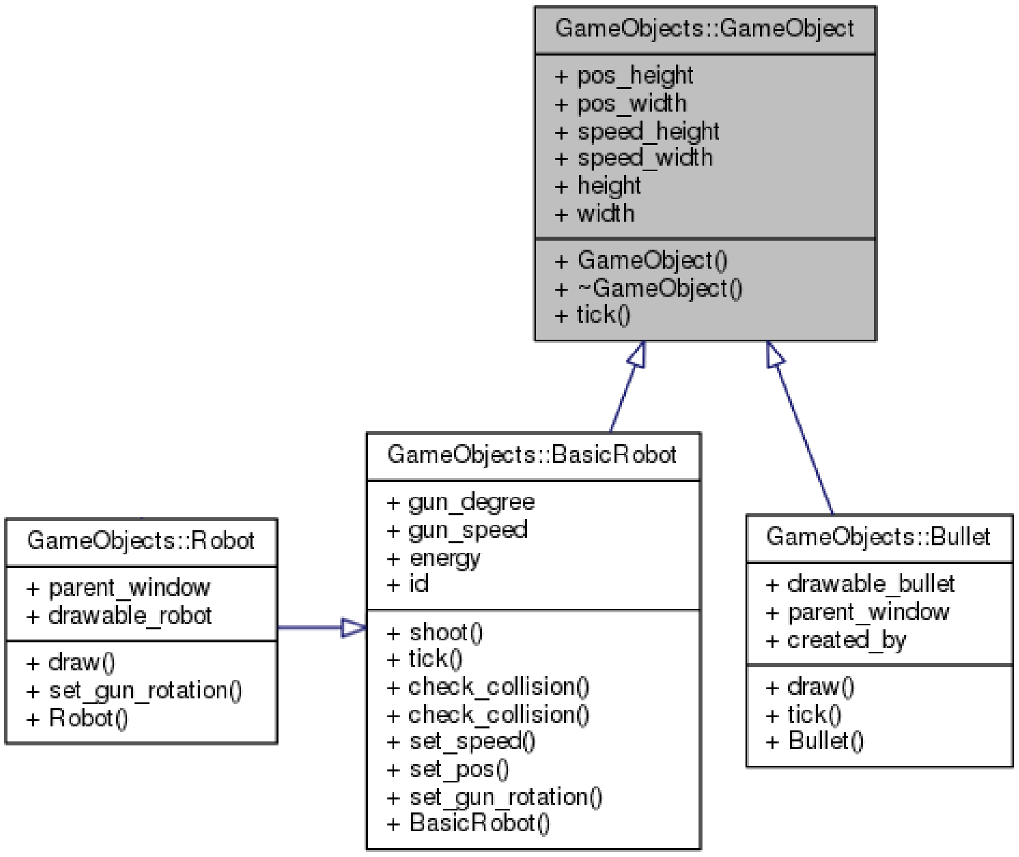

Figure 2: Game Objects UML Diagram

3.3 Game Server and Client

Server

The gameserver consists of two different classes, first the actual game which calculates movements of objects, renders the them and finally sends an update via the server using grpc. The gameserver also prepares the data for the streaming server, which sends the all positions via asio to the player in order for them to see the whole game. The most important part of the gameserver is the "game_loop()" function which covers all the functionalities that are described above.

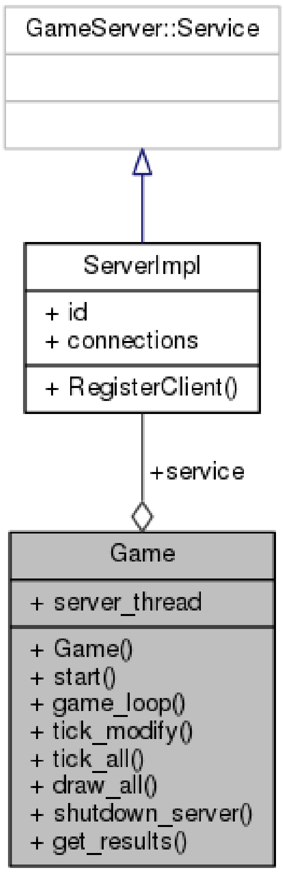

Figure 3: Diagram of the game server

Client

The client not only communicates with the server but also manages the players robot. The class prepares updates, sets the new position, speed etc. for the players convenience.

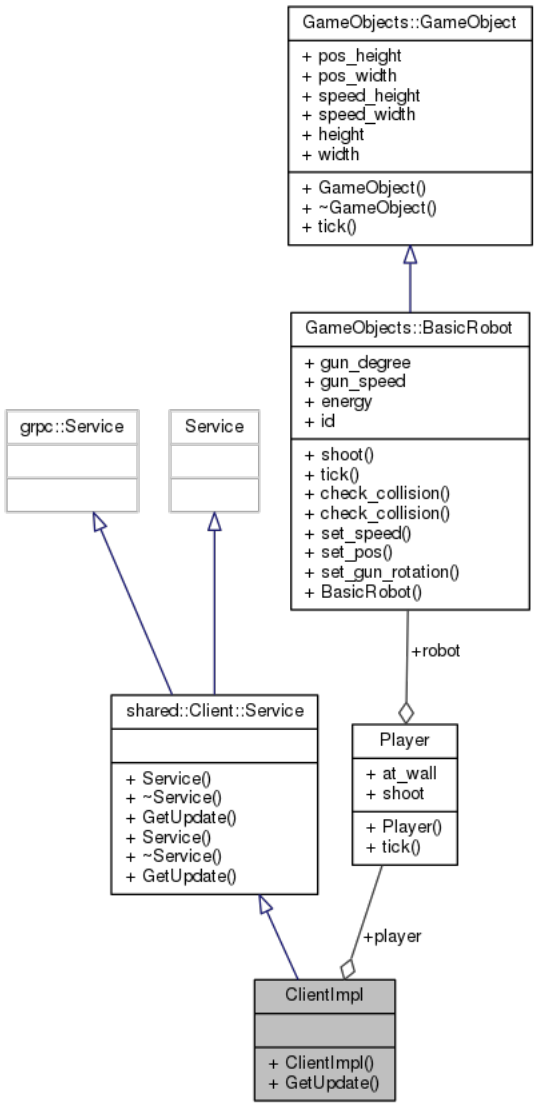

Figure 4: Diagram of the client

3.4 Streaming server and client

The streaming server and client are automatically started when hosting or connecting to a game. The streaming server continually sends an "StreamingUpdate.proto" message which the client can use to display the game state.

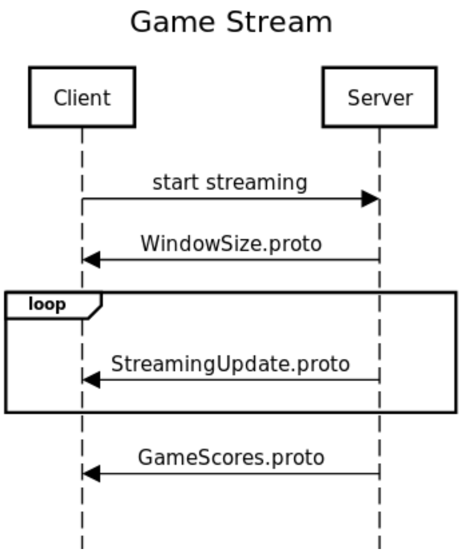

Figure 5: Diagram of the streaming sequence

4 Usage

4.1 Building the game

The game can be build using either Meson or Cmake.

Meson

When building with Meson be sure to change all references in the "meson.options" files accordingly.

Cmake

For cmake to find all required library's the following environment variables have to be set:
ASIO_INCLUDE_PATH
Point to asio standalone include directory.

SPDLOG_INCLUDE_PATH
Point to spdlog/include directory.
CLIPP_INCLUDE_PATH
Point to clipp/include directory.
JSON_INCLUDE_PATH
Point to the directory where the "json.hpp" is located.

4.2 Writing a robot

For writing an own robot simply go to the file "player.h" and overwrite the example bot with your own one. The function "tick()" will be called every time the client receives an update from the server.

```cpp
bool tick(GameObjects::BasicRobot* scanned,
          std::vector<int> hit_wall) {
    bool shot = false;

    // find first wall
    if (at_wall == -1) {
        this->robot.set_speed(1, 0);
        if (! hit_wall.empty()) {
            at_wall = hit_wall.at(0);
            this->robot.set_speed(0, -1);
        }
        return shot;
    }

    // cycle along walls
    if (! hit_wall.empty()) {
        int wall = 0;
        for (int possible_new_hit: hit_wall) {
            if (possible_new_hit != at_wall) {
                wall = possible_new_hit;
            }
        }

        if (wall == 1 && wall != at_wall) {
            at_wall = wall;
            this->robot.set_speed(-1, 0);
        } else if (wall == 2 && wall != at_wall) {
            at_wall = wall;
            this->robot.set_speed(0, 1);
        } else if (wall == 3 && wall != at_wall) {
            at_wall = wall;
            this->robot.set_speed(1, 0);
        } else if (wall == 4 && wall != at_wall) {
            at_wall = wall;
            this->robot.set_speed(0, -1);
        }
    }

    // if a robot is scanned shoot
    if (scanned != nullptr) {
        shot = true;
    }
    this->robot.gun_degree += 10;
    this->robot.gun_degree = std::fmod(this->robot.gun_degree, 360↩
);
    this->robot.gun_speed = 10;

    return shot;
};
```

Listing 3: Player.h

Listing 3 is a simple wall bot example. The player robot circles around the walls while rotating his gun and shooting every time he scanned a robot. The program first searches for the lower wall and from there on travels clockwise next to the them.

4.3 Hosting a game

The game can be hosted by simply calling the built program via `./robocode`. Then navigating to the menu "Host a new game" and "Start getting connections", using the arrow keys and the enter button. Then wait until at least two players have connected to the game and press enter on one of their names.

The game can also be hosted using the command `./robocode -nmh`. "-nhm" stands for "no-menu-host" and means that the server will be started immediately and no menu will be displayed.

The "-nhm" option can also be set in the "config.json" file.

4.4 Connecting to a game

Again the player can simply use the menu to connect to a game. Call `./robocode` again and navigate to "Connect to a game". Then type in an IP address in the "Connect to" field and press enter. The next window will open and requires a "username" to be typed in. When the connection is established the game server send the size of the game window. The terminal then has to be resized to match that width and height.

A connection can also be established using the command `./robocode -c 127.0.0.1 username`.

Again this command can also be set in the configuration file by filling in the "server_ip" and "username" parameter.

5 Additional Documentation

More documentation can be found at "docs/doxygen". The Doxygen documentation can be used by running the command "doxygen robocode" in the folder. This will automatically generate a documentation which can be viewed in the browser.