Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Mobile application for teaching programming

Bachelor thesis

Zdeněk David

Bachelor specialization: Computer Games and Graphics
Supervisor: RNDr. Ingrid Nagyová, Ph.D.

Prague, May 2021

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **David** Jméno: **Zdeněk** Osobní číslo: **483781**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačové hry a grafika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mobilní aplikace pro výuku základů programování

Název bakalářské práce anglicky:

Mobile application for teaching programming

Pokyny pro vypracování:

Seznam doporučené literatury:

1. Papert, S. Mindstorms: children, computers, and powerful ideas. Basic Books, Inc., New York, NY, USA, 1980. 2. Resnick, M. et al. Scratch: Programming for Everyone. Commun. ACM 52, 11, November 2009, pp. 60-67.
3. O'Kelly, J. and Gibson, J. P. RoboCode & Problem-Based Learning: A Non-Prescriptive Approach to Teaching Programming. Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education. 38, 3, 2006, pp. 217-221.
4. Stolee, K. T. and Fristoe, T. Expressing Computer Science Concepts through Kodu Game Lab. Proceedings of the 42nd ACM Technical Symposium on Computer Science Education. 2011. pp. 99-104.
5. Harper, R. Co-Evolving Robocode Tanks. Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. 2011. pp. 1443–1450.
6. Robocode. https://robocode.sourceforge.io/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ingrid Nagyová, Ph.D., kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **07.04.2021** Termín odevzdání bakalářské práce: **13.08.2021**

Platnost zadání bakalářské práce: **19.02.2023**

RNDr. Ingrid Nagyová, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Declaration

I hereby declare I have written this bachelor thesis independently and quoted all the sources of information used in accordance with methodological instructions on ethical principles for writing an academic thesis. Moreover, I state that this thesis has neither been submitted nor accepted for any other degree.

In Prague, May 2021

.....
Zdeněk David

Abstract

This work aims to create an application that teaches the basic programming and algorithmization principles in a fun and visually appealing way and design a system for programming tanks for Robocode.

Structured interviews were conducted with 5 participants regarding optimal size, shape, and method of joining of blocks, which were then used when implementing the application. This thesis presents the application design and describes the creation of block editor in Android Studio. The editor was implemented in Java and tested by six users.

The work output is an application, which in particular allows creating a basic Robocode battle tank.

Keywords: Android application, Java, robocode, block based programming.

Abstrakt

Tato práce si klade za cíl vytvořit aplikaci, která zábavnou a vizuálně atraktivní formou naučí základní principy programování a algoritmizace, a navrhnout systém pro programování tanků pro Robocode.

Byly provedeny testovací rozhovory s 5 účastníky ohledně optimální velikosti, tvaru a způsobu spojování bloků, které byly následně použity při implementaci aplikace. Tato práce představuje návrh aplikace a popisuje vytvoření editoru bloků v Android Studiu. Editor byl implementován v Javě a testován šesti uživateli.

Výstupem práce je aplikace, která umožňuje zejména vytvoření základního bitevního tanku Robocode.

Klíčové slova: Android aplikace, Java, robocode, blokové programování.

Acknowledgements

I would like to show gratitude to my bachelor thesis supervisor, RNDr. Ingrid Nagyová, Ph.D., for her time, valuable comments and suggestions in the creation of this work.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

With the rising numbers of smartphones, everyone has access to one. Most products that teach programming aim at PC, rightly so, because a smartphone is not a device an average person would be looking to program on. However, if the goal is to teach, all essential parts of the code could be replaced with blocks. Those would then be translated to native code. Luckily, smartphones offer the opportunity to drag and drop, which is ideal for precisely this type of use.

While there have been some applications that allow for translating blocks (e.g. Lego Mindstorms, Baltie) and even such an application exists for mobile users (ScratchJr), none of them has focused specifically on designing behaviour of a robot, improving it over time and testing it against other users.

Firstly, we research similar block programming languages that are being used now and compare them. Then, the Robocode library, what it is, how a Robocode tank looks, and how to create a Robocode tank. In chapter four, there is a list of all blocks that are available in the application and their description. In the next section, the application design is shown. It is followed by a chapter about implementing the application, its architecture and describing external libraries used. In chapter 7, we discuss possible improvements to the application. Then, the test that took part at the end of the implementation is described and lastly, a rundown of a tournament that happened after the program was finished.

This thesis focuses on creating a studio where users can design a robot with its own unique behavior.

Chapter 2

An overview of existing solutions

This chapter briefly describes existing block-based programming software and compares them with pre-defined parameters. After this, the reasoning behind using logic blocks instead of code is explained. Lastly, the chapter ends with user testing aimed at improving UI.

2.1 Blockly

The Blockly library contains an editor that portrays coding principles as interconnecting blocks. It generates syntactically valid code in the preferred programming language. To link it to an application, custom design blocks can be added. However, JavaScript, Python, PHP, Lua and Dart are the only available programming languages. [5]

2.2 Scratch

Scratch is a visual programming environment that lets users (usually ages eight to sixteen) study computer programming whilst operating on personal projects, including animated stories and video games. [17]

2.3 Baltie (Baltík)

Baltie is a programming language and development environment designed primarily for teaching programming in primary and secondary schools, i.e., mainly for children and youth. Unlike other programming languages, the Baltic family tools are not programmed using text commands, but icons replace text commands.

2.4 Kodu Game Lab

Kodu Game Lab is a programming integrated development environment for creating video games. It can be used to teach programming also with the use of blocks.

2.5 Lego Mindstorms

One of the brand new trends is the use of LEGO MINDSTORMS kits in numerous computing publications. These kits allow a wide variety of physical fashions to be constructed, a number of which can be programmed via the RCX processor included in them. Using its standard firmware, the RCX tool can be programmed through numerous extraordinary expert languages. However, the additional availability of bytecode-well-matched alternative firmware for the RCX makes using Java as the programming language a desirable method. [2]

2.6 What can we learn from this?

It is obvious that using categories makes the UI more visually appealing. Baltie with 0 categories only got a four score in the user interface comparison as it very chaotic compared to others.

Only Scratch is available online and has a mobile version, which makes the others miss on quite a big part of the target user audience, as almost every person nowadays has a mobile phone.

	Block UI (1-10)	**# of Categories**	**Target audience**	**Available online**	**Mobile version**
Blockly	9	8	Developers	N/A	N/A
Scratch	8	9	8-16 years old	✓	✓
Baltie	4	0	Pupils	✗	✗
Kodu Game Lab	7	12	Young children	✗	✗
Lego Mindstorms	8	6	10-14 years olds	✗	✗

Table 2.1: Comparison of selected block based programming languages.

2.7 Mobile version

Making a mobile version comes with the advantage of a large target audience but has some downsides as well. The biggest one of them is the smaller screen, which makes creating user-friendly UI way harder.

2.8 Wrapping prefabricated routine modules in logic bricks

With the previous being said, I have opted for blocks instead of actual code. Writing on a smartphone is nowhere near as easy as it is on a PC as well. Using blocks is way faster, and it also makes the application more visually appealing. However, this comes with fewer options for the user to customize his robot than text code.

Blocks are divided into several categories, each category representing a set of available skills/moves for the robot. This is the same principle that is used in existing solutions.

2.9 User testing of block sizes

This user testing aimed to find an optimal size, shape, and method of joining of blocks. The feedback creates an essential part in application design and future implementation of the application.

2.9.1 Test process

First, the testers were instructed to install ScratchJr on their smartphone devices and then apply several instructions. Participants were observed during the process. In the end, all participants were interviewed to get feedback.

1. Create a new project
2. Add the start program block
3. Make the initial animated sprite (only fox onwards) say "hi".
4. Then, make the fox go right ten times
5. Run the program

The expected result can be seen in Figure 2.1, although it is possible to achieve this using different blocks.

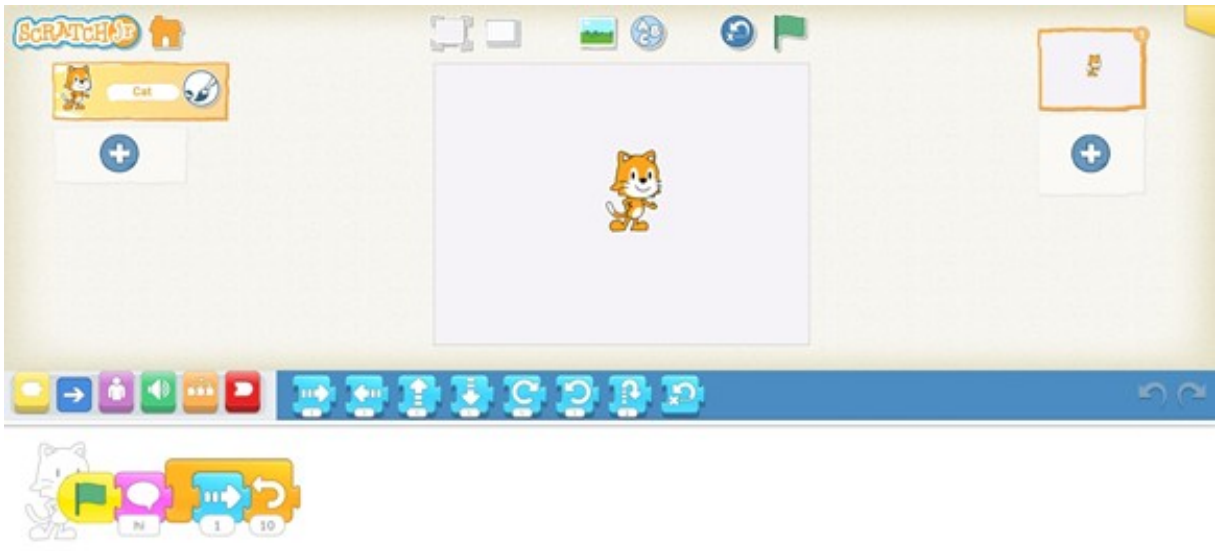

Figure 2.1: The expected result from testing.[15]

2.9.2 The results

Overall, five people participated in the testing (two women, three men) with ages ranging from 11 to 49. All of the users had no experience with programming or with similar applications.

None of the testers encountered obstacles that would prevent them from finishing all the steps. The feedback suggests using slightly bigger blocks than in ScratchJr. The majority of testers found connecting the blocks very intuitive.

Chapter 3

Robocode

Robocode is a programming game (and development environment) that teaches Java in a fun and rewarding way. It was developed and distributed by IBM in 2001. A miniature rectangular battlefield populated by robot tanks serves as the game's setting. These graphical robot tanks are skillfully programmed to drive, search, and destroy other robot tanks on screen. In the class Robot, the skeleton of a robot tank is presented, as well as sample robot tanks that perform a specific type of action.

Students can examine how a robot tank works and read the Java code that goes with it. They can combine the activities of several robot tanks and test the new robot tank's performance in the fighting arena right away. Robocode, according to Bonakdarian and White, promotes "Learning by Subversive Means," in which students must learn to "play" and are motivated to engage. Loadholtes and Hartness promote Robocode as a tool for teaching Artificial Intelligence (AI). Students must examine survival methods such that their robot tanks behave and respond "intelligently" to their environment, even though learning about AI is not a learning outcome of the CS program. [19]

3.1 Robot components

There are three robot parts – a body, a gun, and a radar. Figure 3.1 explains where the parts are and how they look like on the tank.

3.1.1 Radar

An essential part of every robot as it makes targeting other robots and moving safely in the environment possible. Figure 3.2 indicates how the radar cone looks like.

3.1.2 Movement

Robocode uses the Cartesian coordinate system, which means that that the (0, 0) coordinate is located at the bottom-left corner of the battlefield and for direction uses a clockwise direction convention where 0° —& 360° is north, 90° is east, 180° is south, and 270° is west.

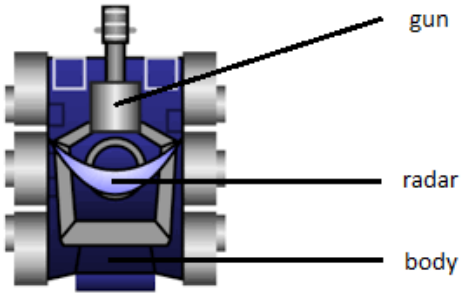

Figure 3.1: The robot components. [9]

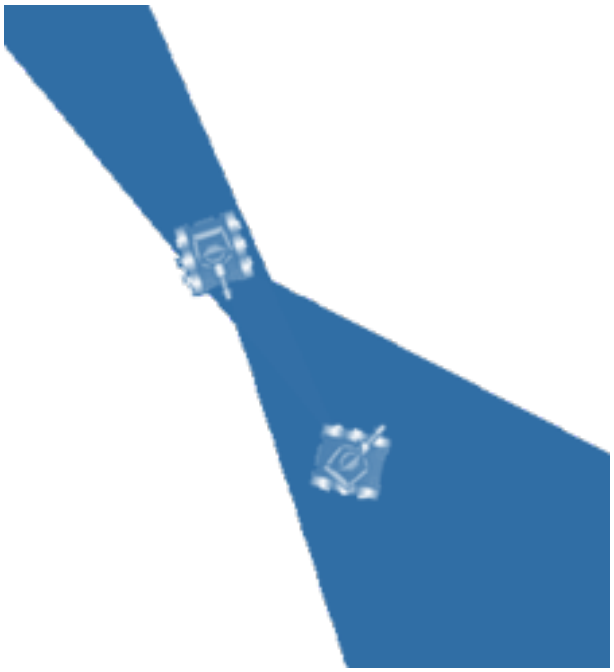

Figure 3.2: The robot's radar. [18]

Getting hit by bullets decreases the robots energy, so dodging enemy bullets increases its chance to win significantly. There are many strategies on how and, very significantly, where to move. Maximizing distance from enemies is reasonable but may cause getting a robot cornered. One very successful strategy, Wave Surfing can dodge enemy bullets very effectively. An example of a more straightforward approach is Stop And Go strategy (move a bit when the enemy fires, then stop before it fires again), which has a high success rate against simple robots.

3.1.3 Targeting

Hitting opponents depletes their energy and increases the robots' energy. Therefore it is vital to choose a good strategy to hit opponents more consistently.

The most common strategies are:

- Head-On Targeting – Shoot at the last known location of the opponent. It is useless against

anything that mostly goes in one direction. It, on the other hand, works surprisingly effectively against oscillating and random movement.

- Circular Targeting – Assume that the enemy will continue to move and turn at the same speed/turn rate as he is currently, and shoot there.
- Pattern Matching with Play It Forward – Use the enemy's most recent movements as input, look for a period in the past when his relative motions were comparable, and predict that he will act in the same way as he did in the most similar previous situation.
- Traditional GuessFactor Targeting (Segmentated Visit Count Stats with GuessFactors) – Select a few attributes to represent firing conditions, segment them into a multi-dimensional array covering all potential scenarios, and add up the GuessFactors in each situation (like a multivariate histogram). Fire at the most often visited GuessFactor for the segment detailing the present circumstance when it is time to fire.
- Dynamic Clustering with GuessFactors – Select a few attributes to describe firing circumstances, keep track of them, and use GuessFactor for each situation. Find the densest area of the graph of the associated GuessFactors at fire time by selecting a number (k) of identical scenarios from the log. [23]

3.2 A basic robot in Robocode

This example shows the most basic functions. This robot will: move forward 100 pixels, turn his gun right by 360 degrees, move backward by 100 pixels, and turn his gun right by 360 degrees again. He will continue doing this in an endless cycle. When the radar scans a robot, our robot fires with power 1.

```
package pkg;

import robocode.*;

public class BasicRobot extends Robot {
    public void run() {
        while (true) {
            ahead(100);
            turnGunRight(360);
            back(100);
            turnGunRight(360);
        }
    }

    public void onScannedRobot(ScannedRobotEvent e) {
```

```
            fire(1);
        }
}
```

It is obvious that creating a robot is relatively straightforward. However, creating a robot that is able to win is a much more challenging task and requires the use of more advanced methods.

Chapter 4

Blocks

Based on similar functionality, several methods from the Robot class from Robocode API were grouped into 4 categories. These methods were chosen based on their frequency of use in basic robots. The categories are events, movement, weapons, and radar.

Each category connects blocks that have similar functionality and has a unique color. This is a good way of sorting blocks and providing the user with another perception of the blocks. The icon next to the name shows what the block looks like in the application.

4.1 Events

Events blocks correspond to several triggers (e.g., onHitWall) or control flow statements (e.g., while).

- run ▶

The main method in every robot. Edit this to implement the robot's behavior. This method takes no parameters and gives the user no fields to work with.

- onScannedRobot

Allows the user to react to other robots in proximity. This method takes no parameters.

- onHitWall

Allows the user to react to the environment. This method takes no parameters.

- while

Creates an endless loop.

4.2 Movement

Movement blocks manage the movement of the robot, mainly to the position and heading of the robot.

- ahead ↑

This moves the robot ahead according to the parameter. The parameter is a number bigger than 0 and represents the number of pixels to move forward.

- back ↓

Moves the robot ahead according to the parameter. The parameter is a number bigger than 0 and represents the number of pixels to move backward.

- turnLeft ←

Turns the robot left by the specified amount. The parameter is a number bigger than 0 and represents the number of degrees to turn left.

- turnRight →

Turns the robot right by the specified amount. The parameter is a number bigger than 0 and represents the number of degrees to turn right.

4.3 Weapons

Weapons blocks are associated with the gun robot component. The functionalities are changing the heading of the gun and firing it.

- turnGunRight

Turns the robots gun right by the specified amount. The parameter is a number bigger than 0 and represents the number of degrees to turn the gun right.

- turnGunLeft

Turns the robots gun left by the specified amount. The parameter is a number bigger than 0 and represents the number of degrees to turn the gun left.

- fire

Fires a bullet from the gun in the direction the gun is pointing with the power specified by the parameter. Robot's energy will be decreased by the specified number. The bullet deals 4 * parameter damage to other robots and deals additional 2 * (parameter - 1) damage if the parameter is bigger than 1. If the robot hits another robot, he gets 3* parameter power back. The parameter is a number between 0.1 and 3 according to the Robocode rules [16].

4.4 Radar

Radar blocks are related to the radar robot component. The functionality of those blocks focuses on managing the radar rotation.

- turnRadarLeft

Turns the radar to the left by the specified amount. The parameter is a number bigger than 0 and represents the number of degrees to turn the radar left.

- turnRadarRight

Turns the radar to the right by the specified amount. The parameter is a number bigger than 0 and represents the number of degrees to turn the radar right.

- setAdjustRadarForGunTurn

Makes the radar turn independently from the gun.

- setAdjustRadarForRobotTurn 

Makes the radar turn independently from the robot.

Chapter 5

Application design

5.1 Client

5.1.1 Design of user interface

The design's goal was to meet a number of basic development principles that lead to optimal user experience.

- Simplicity
- Effectivity
- Manageability
- The similarity to existing solutions

The design is inspired by other similar solutions (Scratch etc.), which is why the application is easy to use. The application consist of the block editor screen, a screen to show the battles and a home screen with a list of all the users' robots.

The image below shows how the final block editor screen should look, with the block categories on the bottom of the screen. When clicked, they expand to show a list of blocks available in the category. The plus button will be used to create a new robot. RobotName indicates the name the user chose for the robot. The play button will run the code.

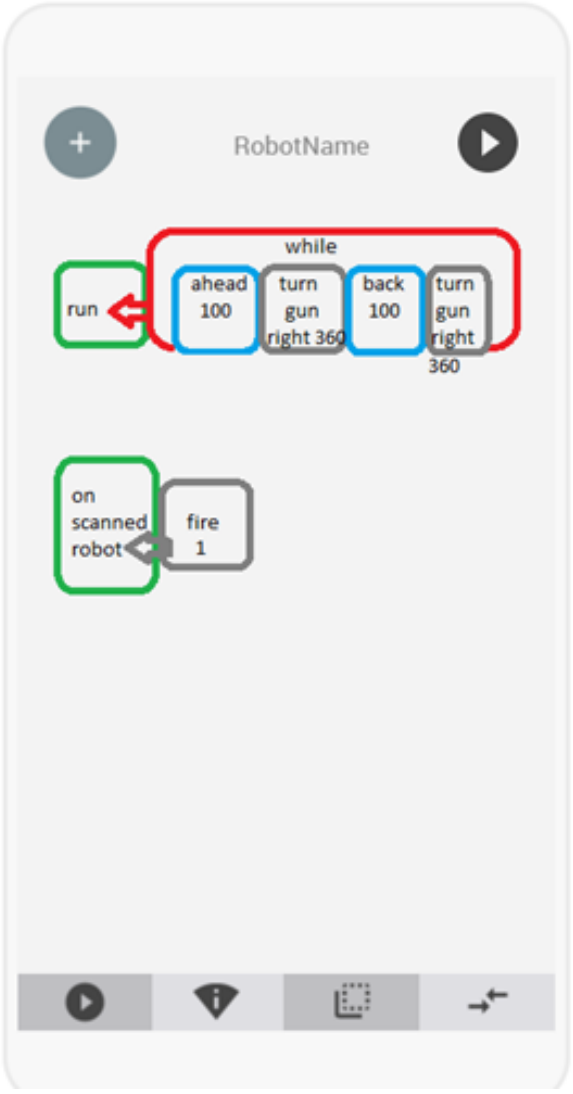

Figure 5.1: Design of the block editor screen.

The code Figure 5.1 corresponds to the example in section 3.2. Green and red blocks are event blocks, blue are movement blocks and dark gray are weapons blocks.

5.2 Block design

First of all, the important part of the block design is that even an inexperienced user should be able to distinguish between blocks with different categories. To achieve that, each category has a unique color. The colors were chosen so that white text could be clearly seen on them. This can be clearly seen in Figure 5.2.

Given the information from the previous testing, my goal was to create the blocks slightly more prominent than how big they are in ScratchJr. Therefore, the size was set to 128 pixels. Other than that, the blocks are very similar to other solutions – a small icon indicates their function, and if there is a parameter, it can be edited in a small input field within the block.

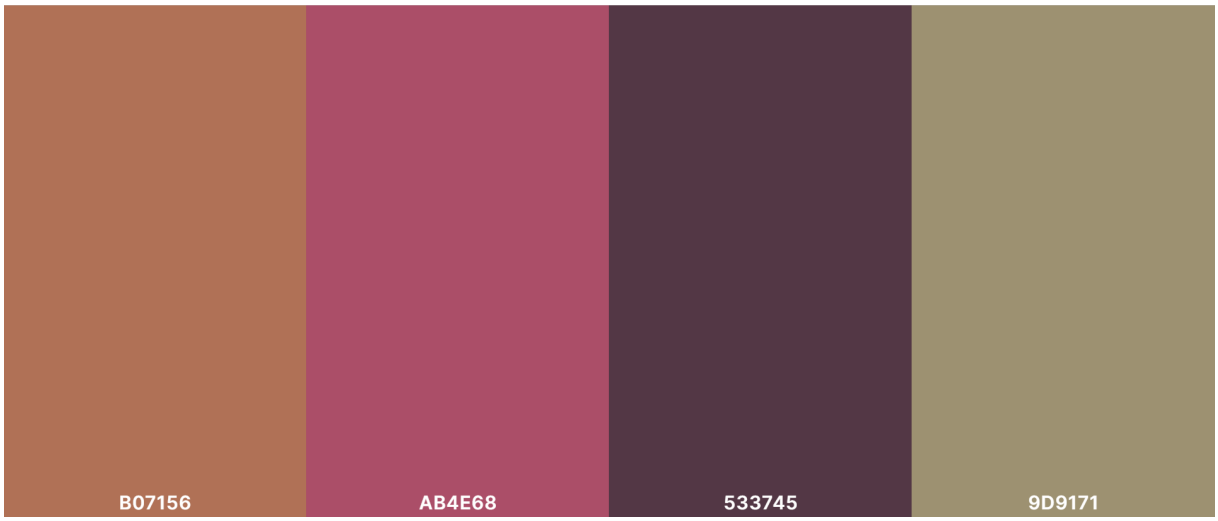

Figure 5.2: Color scheme for the block backgrounds.

Chapter 6

Implementation

Let's take a look at the development of the application. This chapter will explain the project architecture, used frameworks and used software to develop the application.

While working on the project, I followed principles from the book Effective Java[3] by Joshua Bloch. The goal was to make the code readable and easy to edit.

6.1 Architecture

The project consists of these packages:

- **Model** - model classes that represent each one of the blocks. The block editor instantiates and further works with these classes. This package is further divided into four categories - **weapons, movement, radar** and **events**. Figure 6.1 best explains how the model package is structured - blocks from movement, weapons and radar extend either ParametrizedBlock or Block, while events blocks use the more advanced ComboBlock.

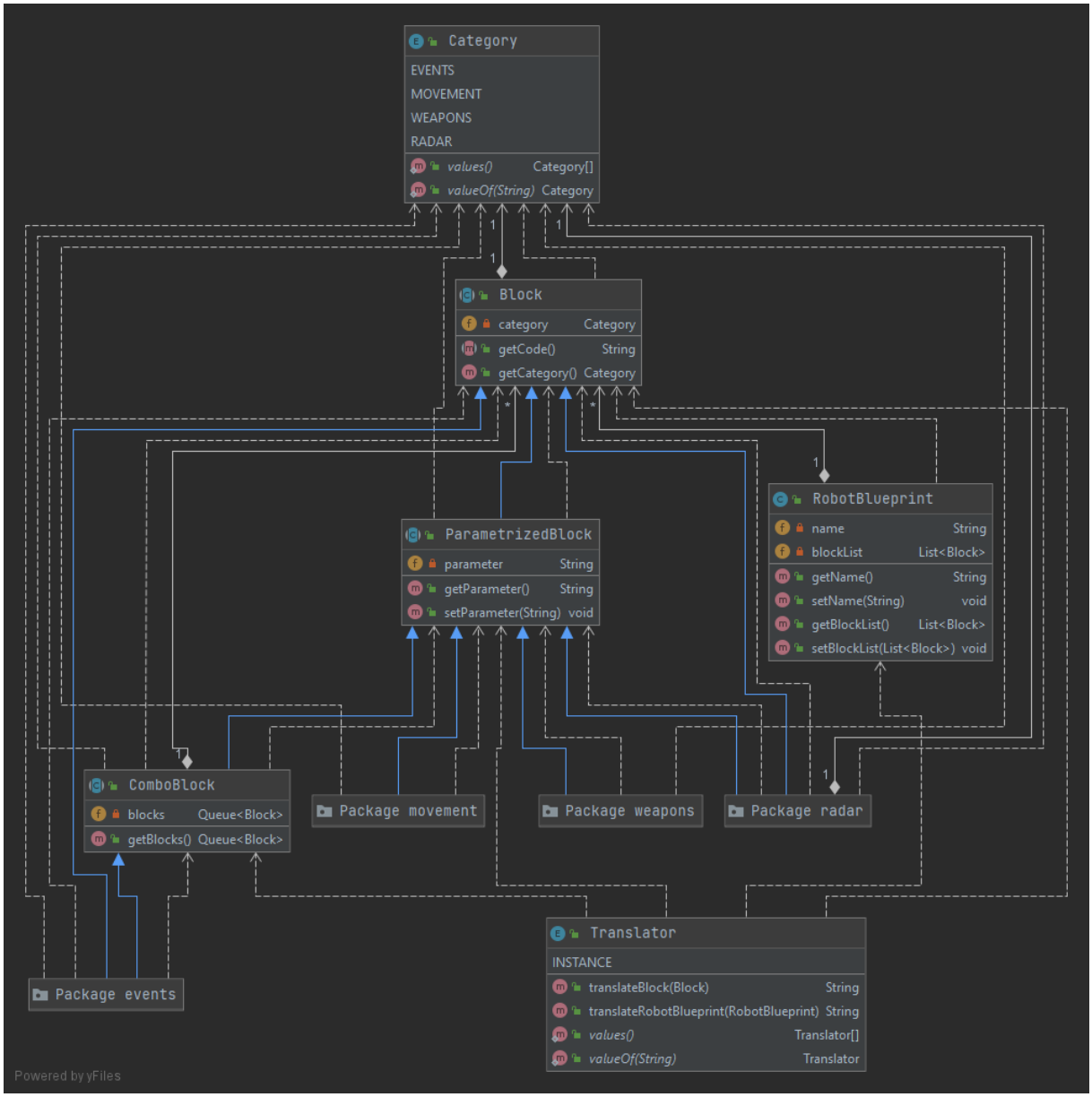

Figure 6.1: Model class diagram.

- **Support** - helper classes.
- **View** - user-interface related classes.

6.2 External libraries

To save time by "not reinventing the wheel", I used 2 third-party libraries. They are listed in table 6.1. FontAwesome[20] is used throughout the whole application, and all block icons are sourced from this library as well as most UI button icons. Jackson[8] is used as a serialization/deserialization tool.

Name	Description
Jackson	Converting Java objects to JSON and vice versa.
FontAwesome	Icon toolkit.

Table 6.1: List of libraries used by the application.

6.3 Development

I have used IntelliJ IDEA to develop the model. Then I moved into Android studio to create the frontend of the application. I used Gradle, the default Android Studio build toolkit to package my app. I was using git as a version control system during the development. Next follows a brief description of each tool that I used.

IntelliJ IDEA

IntelliJ IDEA is an Integrated Development Environment (IDE) developed by JetBrains for JVM languages. It does the routine and repetitive tasks for the user by providing clever code completion, static code analysis, and refactorings. [14]

Android Studio

Android Studio is Android's official IDE. Android Studio offers build automation, dependency management, and customizable build configurations.Android Studio provides a unified environment where the user can build apps for Android phones, tablets, Android Wear, Android TV, and Android Auto. [12]

Git and GitHub

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. [22] GitHub is a cloud-based Git repository and essentially simplifies working with Git. [11]

Gradle

Gradle is an open-source build automation tool that is designed to be flexible enough to build almost any type of software. Gradle runs on the JVM and the user must have a Java Development Kit (JDK) installed to use it. [13]

6.3.1 Model

First, I started working on the model. I implemented 3 key abstract classes: **Block, ParametrizedBlock** and **ComboBlock**. All the blocks extend one of them.

Block, being the most basic one, only contains the information about the Category and text corresponding to the Java code. SetAdjustRadarForGunTurn is a perfect example of a class that extends it.

```
public class SetAdjustRadarForGunTurn extends Block {

    private String code = "setAdjustRadarForGunTurn(true);";

    protected SetAdjustRadarForGunTurn() {
        super(Category.RADAR);
    }

    @Override
    public String getCode() {
        return code;
    }
}
```

Adjusting radar for gun turn is turned off by default, so this block is only needed when it needs to be turned on.

ParametrizedBlock is a slightly more advanced block, with the opportunity to apply parameter to the code's body.

ComboBlock, the most advanced of the three, allows for a parameter as well as containing other blocks. This is important for blocks like Run or While.

Another important class is RobotBlueprint. It contains the name of the robot and stores the blocks that the robot consists of. The data structure of multiple connected blocks is actually a generic tree as shown in Figure 6.2 - each block (given it is a ComboBlock) can have an arbitrary number of children. We want the tree to contain no duplicate elements and to have a predictable iteration order. In order to achieve that, we are using Set collection and especially LinkedHashSet to keep the order in which elements were inserted into the set.

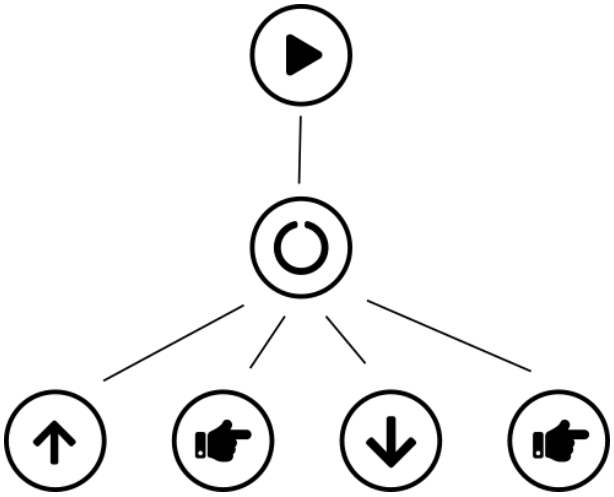

Figure 6.2: Generic tree of the Run block from BasicRobot from section 3.2

To get the actual Java code of a robot, I have implemented Translator and the method

translateRobotBlueprint, which takes RobotBlueprint as an argument. It first appends the name and other miscellaneous code, then proceeds to process the blocks. For that, the method translateBlock is called, in case of ComboBlock recursively until all the blocks are translated.

6.3.2 Saving robots to internal storage

Saving the robot to internal storage is needed to preserve user robots if there is a need to close the application. It is managed by the RobotDataManager class. This class uses Jackson library to store a RobotBlueprint instance as a JSON file and vice versa. The filename is created from the robot's name and has no extension.

6.3.3 Frontend

Landing page

On the frontend part, I have first created MainActivity. It is a simple homepage, which leads to either robot editor, match history, or lets the person quit the application. In Figure 6.3 is illustrated how the page looks like.

Inventory of robots

After clicking on the robot editor page, the user will be taken to RobotEditorInventoryActivity, which shows a list of all previously user created robots or allows them to create a new one. The list of robots is implemented using RecyclerView. Upon clicking on an existing robot or creating a new one, the RobotEditorActivity is started. This is where the user can edit the robot's name and arrange the blocks to create a meaningful (or not) code. The page with multiple saved robots is shown in Figure 6.4.

Robot editor

This is the part of the application where the user can create/edit a robot's behavior. To be able to do that, we need a visual representation of our blocks and drag and drop framework to allow users to move the blocks around the screen. For representing the blocks, the BasicBlockView class extends the AppCompatTextView class. Instead of showing text as expected from TextView, icons from the FontAwesome library are shown according to the specific Block this class represents. The color of the background changes according to the category as well.

To allow for drag and drop movement, the official Android drag/drop framework [6] was implemented. Upon creation, each BasicBlockView gets assigned an inner class that implements View.OnDragListener as well as an anonymous listener object that implements the OnLongClickListener interface, which starts the drag on registering a long click on the view. The DragListeners allows to respond to several types of DragEvents, which is an event that is sent out by the system at various times during a drag and drop operation [7]. The DragListener was implemented in such a way that ACTION_DROP occurs when a BasicBlockView was dropped

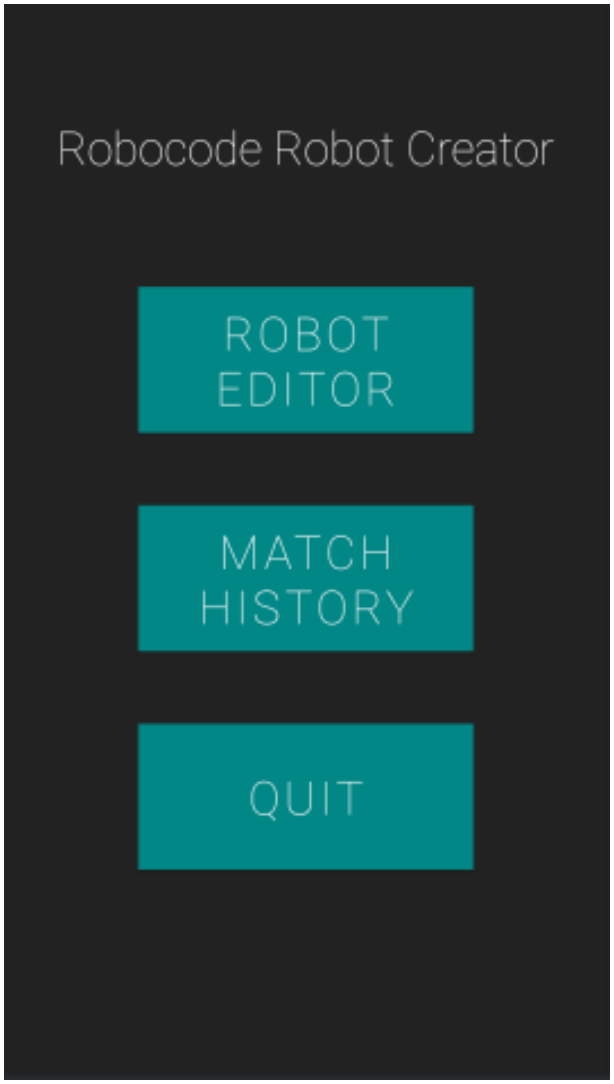

Figure 6.3: MainActivity

on another BasicBlockView. This is very useful, as we can easily assign the dragged block to the block it was dropped onto, assuming that it was a ComboBlock. We have to increase the block in size to indicate that one block is a part of the other. The method adaptBlockDimensions takes care of that. When an ACTION_DRAG_ENDED occurs, this means that the dragged BasicBlockView was dropped elsewhere - essentially means it was dropped on the FrameLayout canvas (referred to as canvas onwards), which contains all the block views.

While dragging a block, a bin icon appears in the bottom right corner. Upon dragging the block on it, it gets deleted. This is achieved by setting the same DragListener to the bin icon and then comparing the tags of the dragged views.

The activity has a text input field and two buttons in the upper part of the screen. The text input field is used to rename the robot. The left icon is used to save the robot to internal storage. The right icon shows how blocks on the screen translate to code. For long codes to be readable, the popup view is a ScrollView. In the bottom part of the screen are multiple popup menus, each representing a single category, the items are all blocks. On clicking on an item, a

Figure 6.4: RobotInventoryActivity

BasicBlockView referencing the block is added to the canvas.

The robot code from section 3.2 visualized by blocks can be seen in Figure 6.5. Also worth noting the implementation is relatively close to design, which can be seen in Figure 5.1.

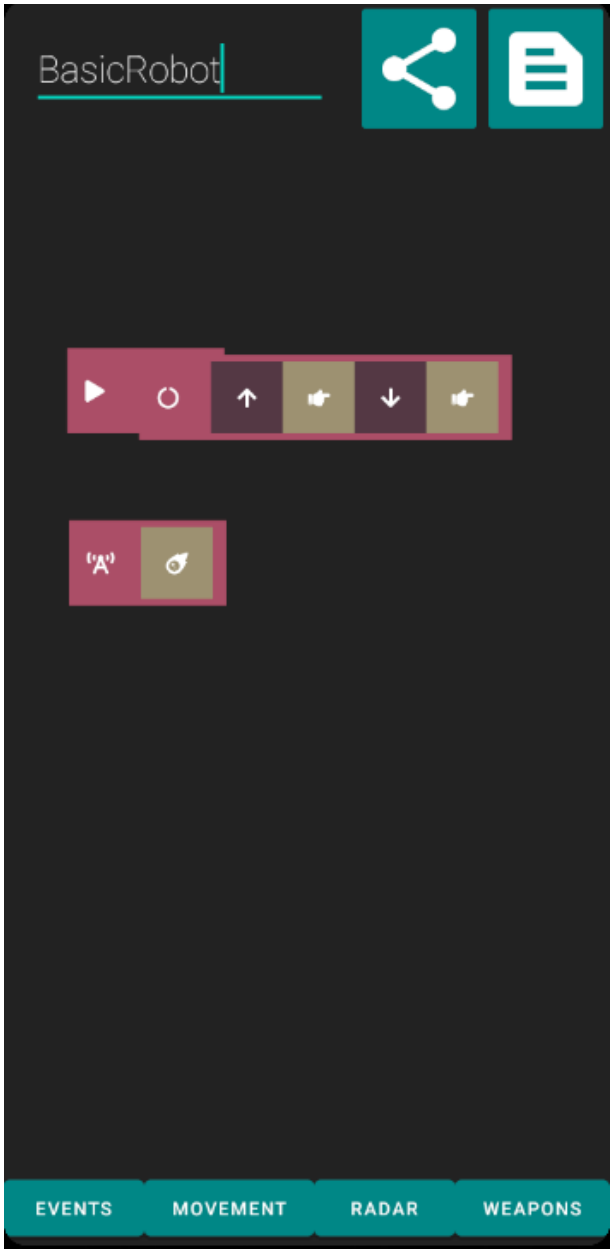

Figure 6.5: Basic robot created in robot editor

Chapter 7

Further development

7.1 Robocode server

For starters, a massive upgrade would be to have a self-sufficient Robocode server, which could run multiple independent robot battles. This would open up a whole lot of possibilities - automatic matchmaking with other robots, individual robot or player rating (which could be similar to chess or competetive videogames Elo [1]) and live viewing battles as well as match history. Although the application itself is operational, it would be needed to connect it to the previously mentioned Robocode server for it to reach maximum potential.

7.2 Improvements to existing features

The selection of blocks is sufficient enough to create a robot. However, to be able to create a more advanced robot, some additional blocks would be needed, e.g., arrays and a whole new category consisting of primitive variables.

As of now, you can only fit around twenty-five blocks to an average size screen. Collapsing and expanding blocks would allow fitting much more and would benefit clarity in projects. This could be working similarly to Blender node collapsing - illustrated in figures 7.1 and 7.2.

The view that shows current code is not very user friendly as it is hard to read due to limited space on the screen. It would be best to highlight syntax, improve code formatting and allow

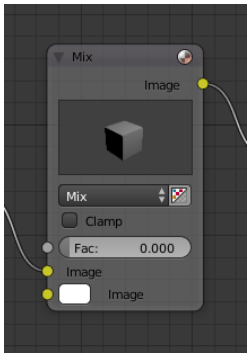

Figure 7.1: Expanded node[10]

Figure 7.2: Collapsed node[10]

the user to turn the screen to create more space.

7.3 Robot statistics

Some players would undoubtedly appreciate the opportunity to see an overview of each of their robots' abilities, e.g., percentage of shots hit, average placement (in more prominent lobbies), ramming damage and so forth. These three properties are shown in Figure 7.3.

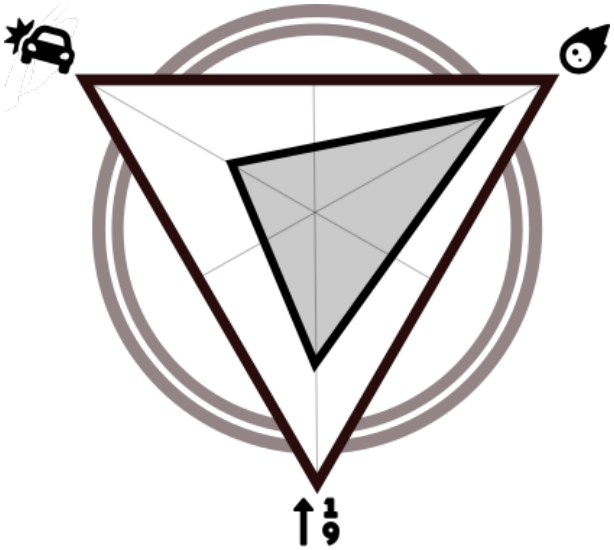

Figure 7.3: Robot statistics design

7.4 Expansion of game mechanics

The battlefield is very simple as of now - no obstacles, just a rectangle with walls all around. Pickable power-ups could be added as well - more speed for whoever picks it up, make all opponents slower, make shots hurt more, make the robot invincible, etc. Furthermore, the robot tank components could be edited either to gain performance or as a visual upgrade. Nowadays, purchasing virtual items to make some cosmetic change for a low price is a prevalent business model known as microtransactions [21]. All changes listed here would require editing the source code of the Robocode library.

7.5 More intuitive user interface

Firstly, graphic effects in Robocode are considerably limited. If the Robocode library were to be kept to create a game for commercial purposes, it would be needed to enhance the visual part. The textures and animations are sketchy as the library was created in 2001. A number of the block icons are not very user-friendly, as FontAwesome does not provide enough fitting icons. This could be solved by using custom-made icons. Additionally, using a game engine could prove helpful as there would be no need to rely on Android UI views to represent the blocks.

Chapter 8

Testing

There was a test once the application was nearly finished. The goal of the test was to detect bugs and see if the application was usable enough to create functional and better robots than the most basic ones.

Six people participated in this test (two women, four men) with ages ranging from 16 to 49. One of the testers was a programmer and two had some experience with programming. Other than that, others had zero to no experience with programming, but all of them were familiar with computers and mobile phones. Only one of the users had experience with similar applications - the educational programming language Karel.

8.1 Test process

The testers were instructed to explore the application and try to create their version of a battle robot. Throughout the procedure, participants were observed. After that, all participants were asked to provide feedback. The testing was performed on Xiaomi Redmi Note 6 Pro with Qualcomm Snapdragon 636 processor and 4096 MB RAM.

8.2 Test results

Four participants found the blocks to be too small. All participants did not want to spend too much time reading the Robocode documentation and advised it would be better to provide some sort of tutorial. Some people had trouble seeing input parameters clearly when adding big values. One participant had problems recognizing the differences between the blocks, but as mentioned in section 7.5, there is little to be done without paying for more comprehensive icon sets or designing and creating a dedicated icon set.

After the testing, a robot battle with all tanks created by participants was held. The results were very random, as most of the robots were not very functional. Not all tester-created tanks were compilable at first, but with some help, they were able to create tanks that could be compiled.

8.3 Test summary

In response to testing, block size was increased from 128 pixels to 192 pixels. Seeing how many people request a tutorial, it will be included in next version. As an answer to the bug, where the parameter could not be seen, maximum lines on the editText were set to 1. The question of how much the user should be limited by the editor while creating the tank remains. Additional checks were set, but it is still possible to create an uncompilable tank.

There were a few bugs; however, the players understood the purpose of the application and said that the experience was generally enjoyable.

Chapter 9

Tournament

Part of this thesis was supposed to be a tournament with several battle tanks. The tournament was organized with the same six people from testing so that the participants would have at least some experience with creating the robots.

Tournament settings

The rules for the tournament were as follows:

- Number of Rounds: 100
- Gun Cooling Rate: 0.1
- Inactivity Time: 450
- Sentry Border Size: 100
- Battlefield size: 600x600

The number of rounds was set to a hundred to decrease random score spikes caused by random spawns.

9.1 Tournament participants

Czendych

This participant had only a little knowledge of programming but had good computer knowledge. He used a broad spectrum of blocks. On scanning another robot, it fires a shot with the power of two. At the start of the battle, it allows the robot's base, gun, and radar to rotate independently. Then repeats the following action in an endless loop: go forward fifty pixels, turn radar and gun 360 degrees. Upon hitting a wall, it goes back twenty five pixels and then turns 120 degrees left.

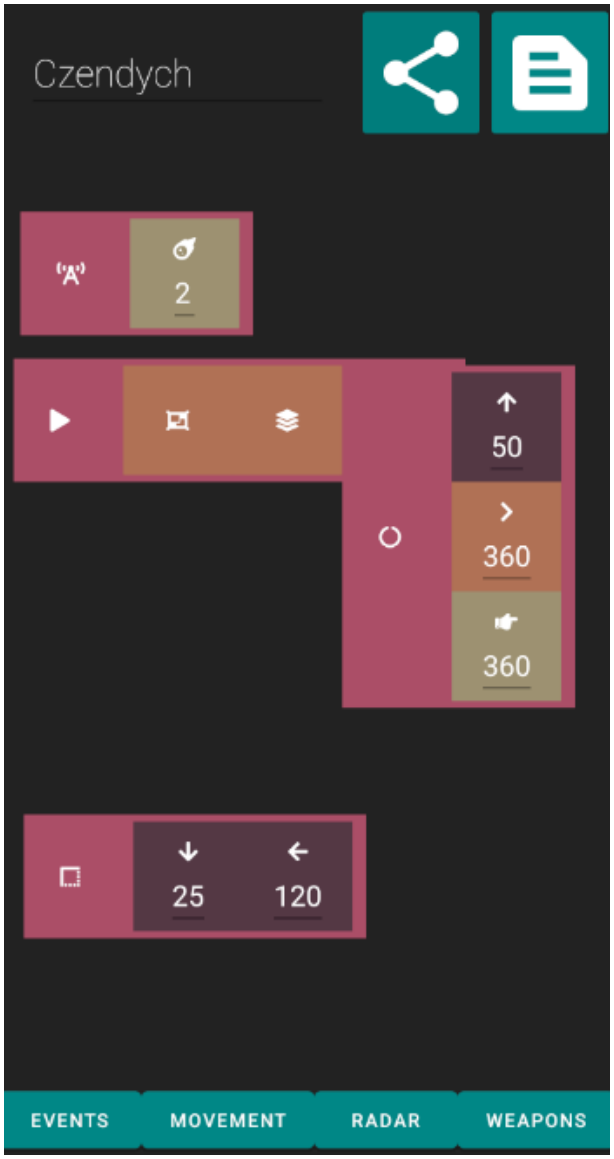

Figure 9.1: CzendychBot block representation

IrenaBot

This participant had no knowledge of programming and little computer knowledge. So naturally, this robot is pretty simple. On scanning another robot, it fires a shot with the power of 0.5. After the start, it repeats the following action in an endless loop: go forward a hundred pixels, rotate radar 360 degrees. Upon hitting a wall, it turns 180 degrees right.

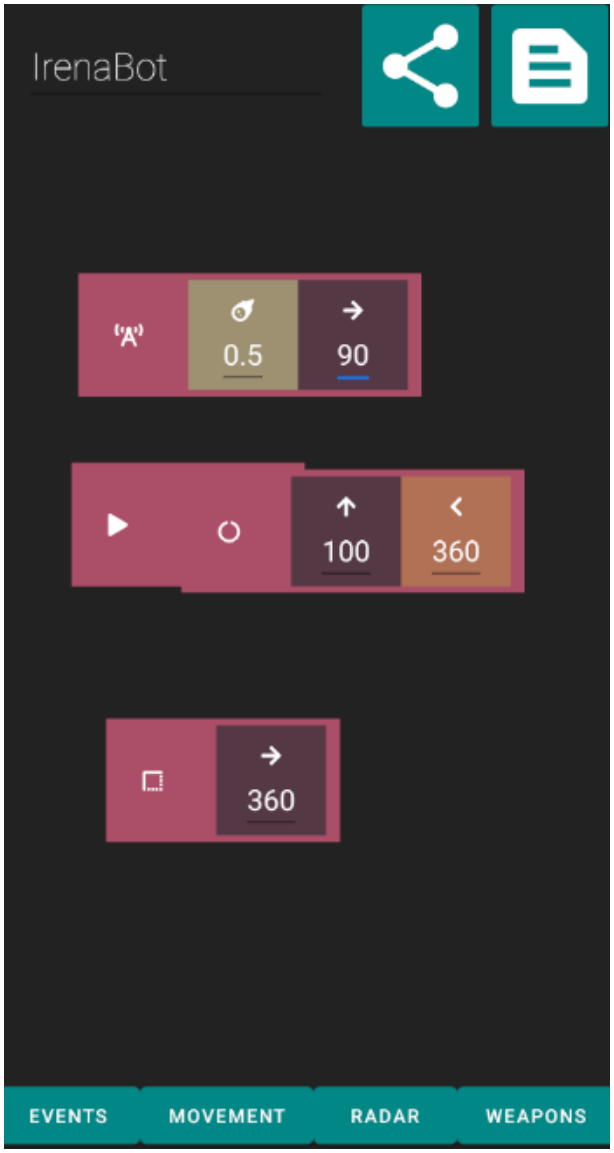

Figure 9.2: IrenaBot block representation

Lucinka

This participant had some experience in programming and good computer knowledge. On scanning another robot, it fires a shot with the power of 1.5, then turns 90 degrees left and moves forward 100 pixels. After the start, it repeats the following action in an endless loop: go forward ten pixels, rotate the gun 360 degrees. Upon hitting a wall, it turns 90 degrees left.

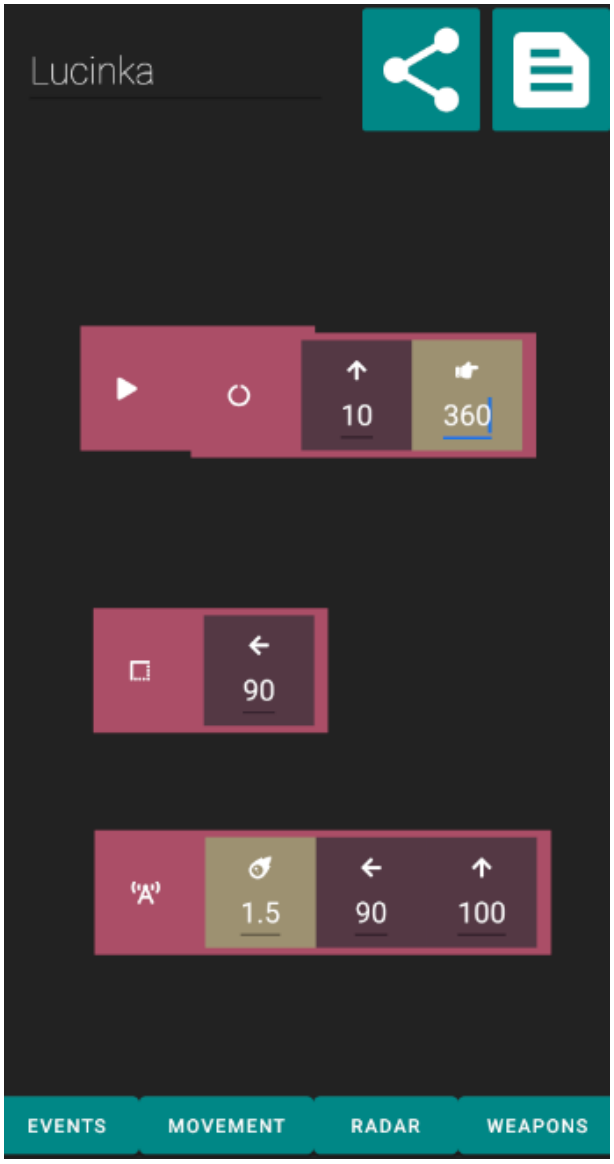

Figure 9.3: Lucinka block representation

BreakingDad

This participant had little experience in programming and good computer knowledge. On scanning another robot, it fires a shot with the power of two, moves forward ten pixels, fires a shot with the power of one, and turns the gun 45 degrees right. After the start, it repeats the following action in an endless loop: go forward five pixels, turn 45 degrees right, go forward ten pixels, turn 90 degrees right and finally rotate gun 360 degrees. Upon hitting a wall, it goes backward fifty pixels and then turns 90 degrees right.

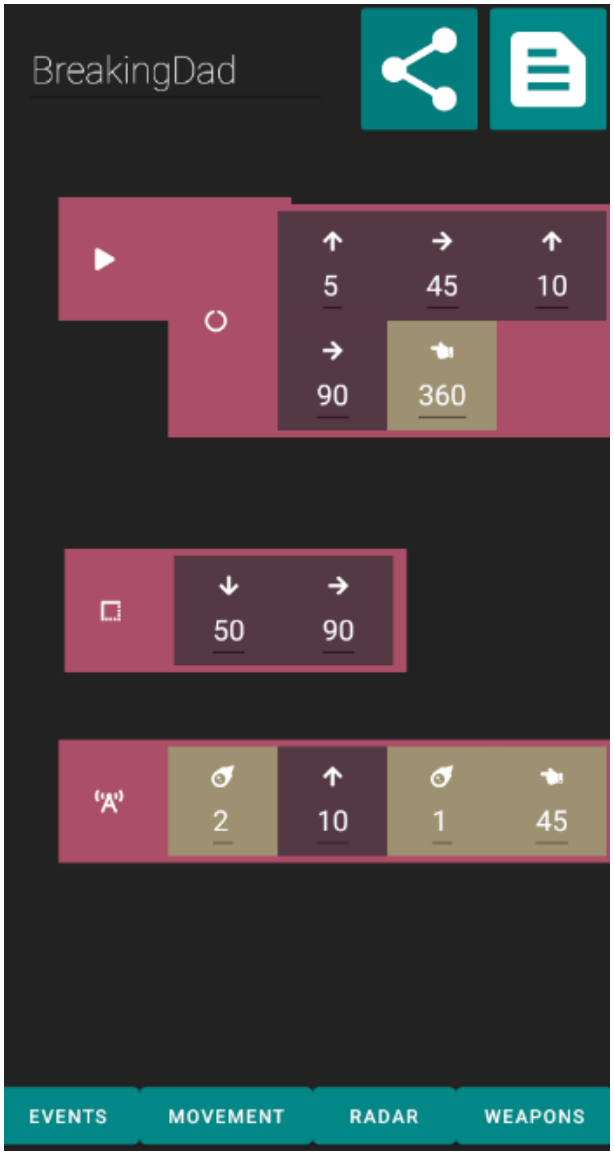

Figure 9.4: BreakingDad block representation

Pirat

This participant had no experience with programming and good computer knowledge. On scanning another robot, it fires a shot with the power of three, moves forward thirty pixels. After the start, it repeats the following action in an endless loop: rotate gun 360 degrees, go forward twenty pixels, rotate gun 360 degrees, turn 45 degrees right, go forward thirty pixels, rotate gun 360 degrees and then turn 135 degrees right. Upon hitting a wall, it goes backward fifty pixels, rotates gun 360 degrees, and turns 90 degrees right.

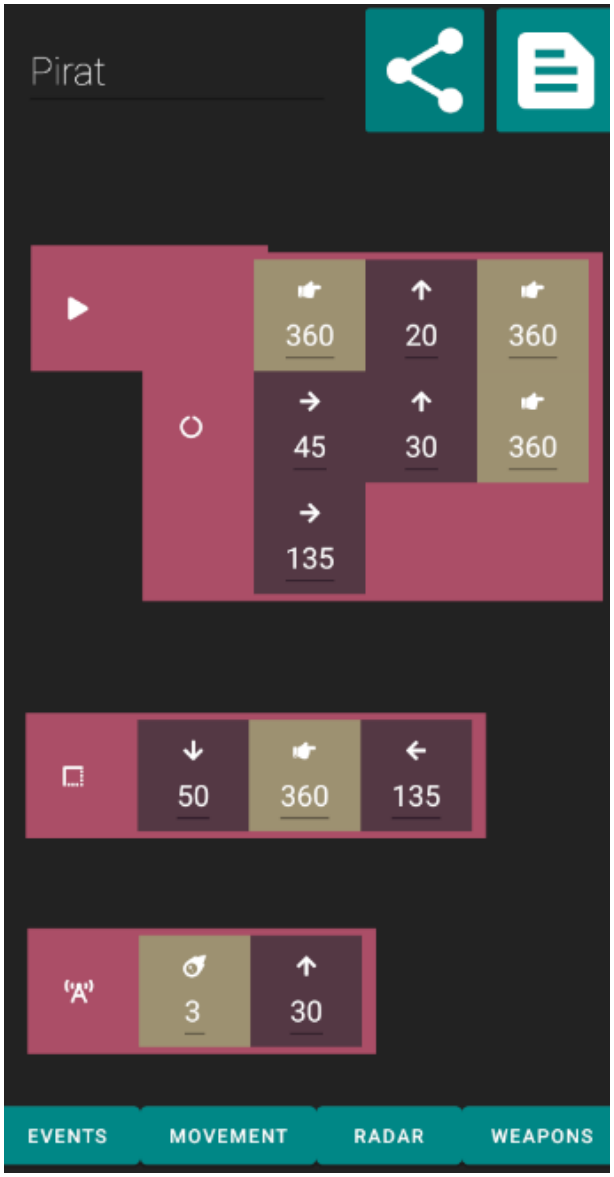

Figure 9.5: Pirat block representation

MELOUN

This participant had no experience with programming and good computer knowledge. On scanning another robot, it fires a shot with the power of 0.5, moves forward 100 pixels and fires a shot with the power of 3. After the start, it repeats the following action in an endless loop: turn 360 degrees right. Upon hitting a wall, it goes backward fifty pixels and turns 45 degrees right.

Figure 9.6: MELOUN block representation

9.2 Tournament results

The tournament results can be seen in Figure 9.7. The first three rounds were recorded and are available in this YouTube video[4]. A positive sign is that more advanced robots seem to achieve better results. An interesting fact is that despite using a broad range of blocks, battle tank Czendych ended last. A screenshot for illustration can be seen in Figure 9.8.

Results for 100 rounds

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	roboteditor.Pirat* (5)	54050 (38%)	21000	3400	24026	3374	2015	234	69	9	10
2nd	roboteditor.BreakingDad* ...	27666 (19%)	13350	800	12342	747	340	87	16	28	14
3rd	roboteditor.Lucinka* (3)	24572 (17%)	14850	700	8268	385	368	0	14	33	22
4th	roboteditor.MELOUN* (6)	14711 (10%)	7150	0	6246	278	1007	30	0	4	13
5th	roboteditor.IrenaBot* (2)	11409 (8%)	10700	0	677	3	29	0	1	24	20
6th	roboteditor.Czendych* (1)	11005 (8%)	7850	50	2754	159	184	9	1	1	21

Figure 9.7: Tournament results

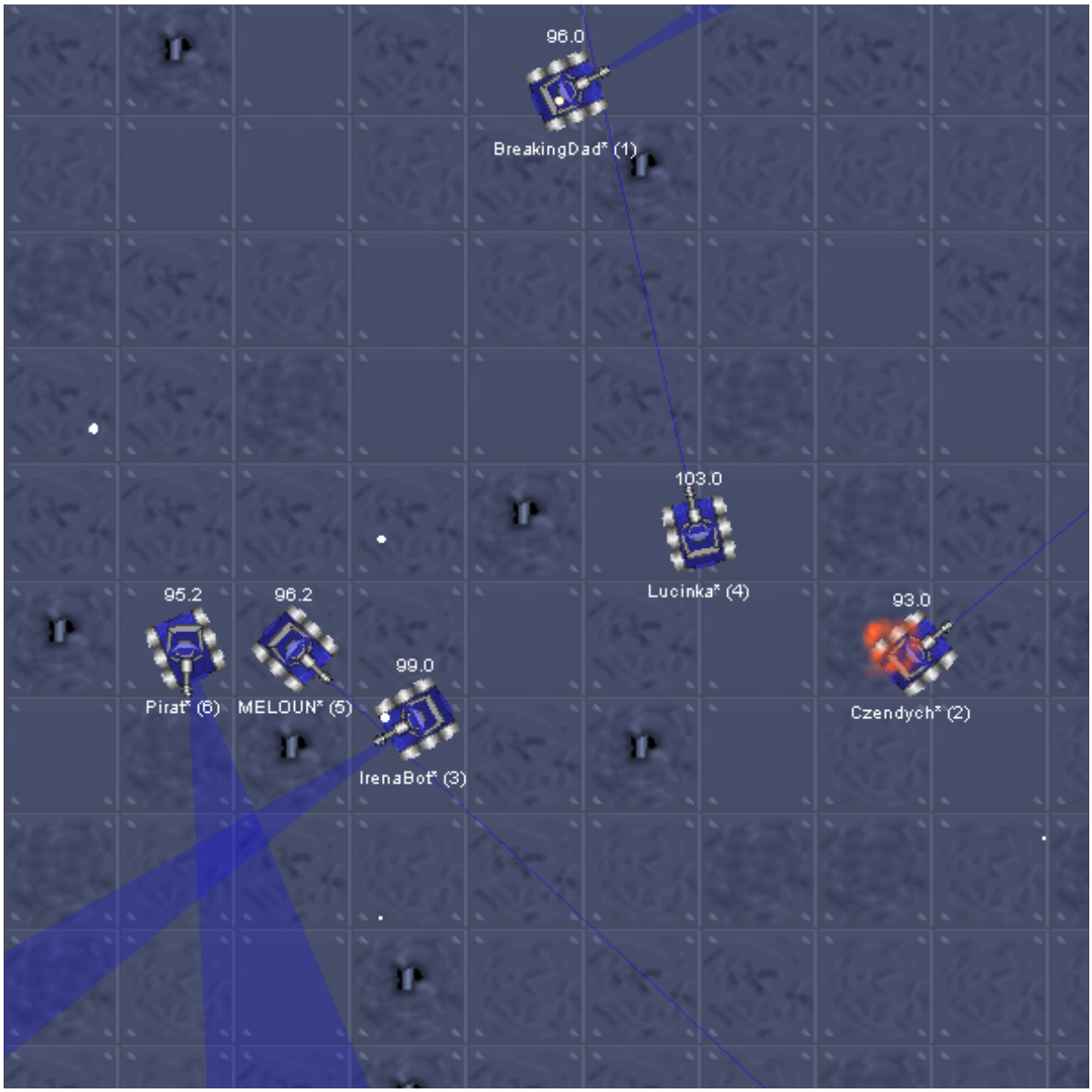

Figure 9.8: A screenshot from Robocode battle tournament

Chapter 10

Conclusion

This thesis aimed to create a studio where users can design a robot with its unique behavior. With the exception of the integration of the mobile application's communication with the server, this goal was met.

To accomplish this assignment, we looked into the concepts behind block programming languages and other comparable tools. As a result of this investigation, essential parts of the Robocode library were chosen to be represented as blocks and the block editor was created.

Finally, after the application was completed, we tested the editor with six users and organized a play-off competition.

Bibliography

[1] Paul CH Albers and Han de Vries. "Elo-rating as a tool in the sequential estimation of dominance strengths". In: *Animal Behaviour* (2001), pp. 489–495.

[2] David J Barnes. "Teaching introductory Java through LEGO MINDSTORMS models". In: *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*. 2002, pp. 147–151.

[3] Joshua Bloch. *Effective Java*. Addison-Wesley, 2018.

[4] Zdeněk David. *Robocode tournament*. URL: `https://youtu.be/FqoMaRc40IU`.

[5] Google Developers. *Blockly*. URL: `https://developers.google.com/blockly` (visited on 11/08/2021).

[6] Google Developers. *Drag and drop*. URL: `https://developer.android.com/guide/topics/ui/drag-drop` (visited on 10/08/2021).

[7] Google Developers. *DragEvent*. URL: `https://developer.android.com/reference/android/view/DragEvent` (visited on 10/08/2021).

[8] LLC FasterXML. *Jackson*. URL: `https://github.com/FasterXML/jackson`.

[9] Fatec Jundiaí – Deputado Ary Fossen. *Robocode robot components*. URL: `https://www.fatecjd.edu.br/portal/2018/05/15/oficinas-e-copa-robocode/`.

[10] Blender Foundation. *Node Parts*. URL: `https://docs.blender.org/manual/en/2.79/editors/node_editor/nodes/parts.html`.

[11] Inc. GitHub. *GitHub*. URL: `https://github.com/`.

[12] JetBrains Google. *Android Studio*. URL: `https://developer.android.com/studio/features`.

[13] Inc. Gradle. *Gradle*. URL: `https://docs.gradle.org/current/userguide/what_is_gradle.html`.

[14] JetBrains. *IntelliJ IDEA*. URL: `https://www.jetbrains.com/help/idea/discover-intellij-idea.html`.

[15] MIT Media Lab. *ScratchJr*. URL: `https://www.scratchjr.org/`.

[16] Flemming N. Larsen. *Robocode rules*. URL: `https://robocode.sourceforge.io/docs/robocode/robocode/Rules.html#MIN_BULLET_POWER` (visited on 12/08/2021).

[17] John Maloney et al. "The scratch programming language and environment". In: *ACM Transactions on Computing Education (TOCE)* 10.4 (2010), pp. 1–15.

[18] MultiplyByZer0. *Robocode radar*. URL: `https://robowiki.net/wiki/File:Radar.png`.

[19] Jackie O'Kelly and J Paul Gibson. "RoboCode & problem-based learning: a non-prescriptive approach to teaching programming". In: *Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*. 2006, pp. 217–221.

[20] Font Awesome Team. *FontAwesome*. URL: `https://fontawesome.com/` (visited on 12/08/2021).

[21] Nenad Tomić. "Effects of micro transactions on video games industry". In: *Megatrend revija* 14.3 (2017), pp. 239–257.

[22] Linus Torvalds. *Git*. URL: `https://git-scm.com/`.

[23] Voidious. *Robocode: Targeting*. URL: `https://robowiki.net/wiki/Targeting`.