Energirenovering

P4 Project
Group sw408f16
Software
Aalborg Universitet
Den 26. maj 2016

AALBORG UNIVERSITET
STUDENTERRAPPORT

Titel:

Not known yet.

Projekt:

P4-project

Projektperiode:

February 2016 - May 2016

Projektgruppe:

sw408f16

Deltagere:

Christian Dannesboe
Frederik Børsting Lund
Karrar Al-Sami
Mark Kloch Haurum
Emil Bønnerup
Søren Lyng

Vejledere:

Giovanni Bacci

Oplagstal: X
Sidetal: X
Appendiks: X
Afsluttet 26-05-2016

Synopsis:

Synopsis

Preface

Insert preface here..

Christian Dannesboe	Frederik Børsting Lund	Karrar Al-Sami
Mark Kloch Haurum	Emil Bønnerup	Søren Lyng

Indholdsfortegnelse

Indledning 1

In Danish high schools all kinds of different languages are available for the students, and the programming languages has found their place as well. For beginners, the syntactic rules, type systems and the nature of the language can be hard to comprehend at first, which is why this report will focus on constructing a language for the high school students. The interest for computer games in the 21st century is bigger than ever, and computer games is a fun and educating approach to learning programming languages. Robocode is a game where the player has to code their own robot, giving every player the opportunity to battle each other's robots, making it a competition of coding the 'best' robot. This is mainly coded in Java, which is an object-oriented programming language, and this nature of the language can be hard to understand without having programming experience. There is no popular domain-specific language for Robocode, and therefore the high school students, who is not experienced programmers, will not be likely to code a working robot without studying the language and Robocode in great detail. This is a problem, since Robocode could be a potentially great way of introducing these students for programming languages. If there was a domain-specific language, with less complex type systems and good writeability, students could easily be introduced for a programming language, and afterwards expand their knowledge by applying more complex type rules and syntax. In this report, Robocode will be studied, and the final product should be a domain-specific language for Robocode, compiled to Java. The compilation to Java is done, since Java is the most used language for Robocode programming so far.

Analysis 2

Robocode is an Open Source game project on SourceForge originally started by Mathew A. Nelson in late 2000, who was inspired by RobotBattle from the 1990s. Contributors for the Open Source lead to two new projects, RobocodeNG and Robocode 2006, by Flemming N. Larsen. These two new versions had bug fixes, and new features by the community of Robocode, and in 2006 Flemming merged one of the projects, the Robocode 2006, into an official version 1.1. The Robocode client was introduced in May 2007, which can be used to create the robots for the game. These robots are usually coded in Java, but in the recent years, C# and Scala are popular as well.

In schools and universities, Robocode is introduced for education and research purposes, as it is intended to be fun and easy to understand the core principles: One robot each, with abilities to drive forward, backwards, turn to the sides, and shoot a gun. These core principles can be vastly expanded to more complicated demands, as the robots universe is bigger than it looks at a first glance. [Larsen, Updated 2013]

As mentioned before, the Robocode is usually coded in Java, which leads to this report only examining Java samples. This is to prevent any misleading keywords or misinterpretations. The Robocode client comes with a text editor, and a few code sample robots. In this chapter, some of these sample robots will be examined, and the general setup and main events or methods will be presented.

When creating a new robot in the Robocode text editor, the following methods and events are present:

Listing 2.1. Eksampel of the main loop in Robocode label

```
1  public void run() {
2        while(true) {
3                //Robots behaviour
4        }
5  }
```

This method is the loop for the robot, this loop will determine what the robot does constantly, unless interrupted by an event, which the user can define. The robot behaviour is what the user will code as the AI, along with the robot behaviour in the following code snippets.

Listing 2.2. Eksampel of the onScannedRobot event from Robocode label

```
1  public void onScannedRobot(ScannedRobotEvent e) {
2        //Robots behaviour
3  }
```

The robot's radar will spot enemies when they get within the vision of the radar, which will raise the onScannedRobot event. This event is used to determine how the robot reacts to spotting an enemy, where the ScannedRobotEvent e is the source for information about the enemy robot spotted.

Listing 2.3. Eksampel of the onHitByBullet event event from Robocode label

```
1   public void onHitByBullet(HitByBulletEvent e) {
2           //Robots behaviour
3   }
```

When the robot gets hit by another robot's bullet, the event onHitByBullet will be raised. The robot can then be programmed to act a specific way, change behaviour or carry out a task when the event is raised.

Listing 2.4. Eksampel of the onHitWall event event from Robocode label

```
1   public void onHitWall(HitWallEvent e) {
2           //Robots behaviour
3   }
```

When the robot drives into the wall, the event onHitWall will be raised, and the robot can be programmed to do a specific task when this occurs.

The above mentioned is only a few of the events that can occur in RoboCode. In the while loop, events and functions the user can use many build-in methods from the robot class, which is moving and controlling the robot, controlling the radar and the gun and getting information about the battlefield, the user's robot, other robots and many other things.

In this section the robocode concept will be described. It will include the different functions the system contains.

2.1   Robocode

Robocode was originally developed by Matthew A. Nelson, back in 2000. During 2005 Robocode was brought to SourgeForge as Open Source. The development on Robocode had more or less stopped, and therefore it encouraged the Robocode-community to further develop it, so that they could fix bugs and other issues. Mid 2006 Flemming N. Larsen took over the Robocode project as administrator and since then the game have been updated fairly regularly with the help of the community. Robocode is best described, as they describe it on their website: "Robocode is a programming game, where the goal is to develop a robot battle tank to battle against other tanks in Java or .NET. The robot battles are running in real-time and on-screen." [RoboWiki, 2013]

The way this game works, is by writing code in one of their supported programming languages and then setting it into battle with other people's robots. There are some sample robot, when the game is downloaded, in order to give the users/players a chance to see how it's supposed to be written and from there, it's up to the single individual to make the "best" robot. http://robowiki.net/wiki/Robocode/My_First_Robot

There are held tournaments around the world, where people from around the globe compete. It varies in size, some tournaments are only country based, while others are worldwide, some have leagues and the options are more or less limitless. [Nelson, 2015]

2.1.1   Robot

The robot is the core of the game. The robot can be coded to act differently in various of encounters. There are some events in Robocode that the user can use to program their strategies, one of the events could be onHitWall which basically tells the user that if the robot hits the wall, then the robot will execute the code matches the event that occurred. The robots also have a gun. The gun is used to damage other robots, that the robot will be fighting in the arena. The gun is always the same gun, but the players have the opportunity to code different types of bullets. For example, one play could use small and fast bullets, they won't necessarily hurt very much, but it allows the player to shoot more frequent, compared to larger slower bullets.

Finally the robot also have a radar. With the radar, it allows the robot to scan for robots for example or scan for walls etc. This could once again, affect the strategies of the robots. There's a sample robot called Rambot, this robot will just simply target a robot and run towards it very fast while shooting at it.

2.1.2   Battlefield

The battlefield is the arena where the robots will fight each other. It's also the visible field on screen when the game is running. When coding, the battlefield can be used for different reasons, for example, to get the number of enemies that are alive. A robot might be programmed to act differently if there's less than three robots left. All this depends on the way the player decided to program his robot. Some of the other examples that the battlefield can be used for could be the field size, time or the current round number.

2.1.3   Energy

Robots have energy, which is the spendable resource when shooting bullets, it is the 'health' of a tank, since being hit by a wall or another robots bullet causes a robot to lose energy. But if one robot hits another one, it regains energy. All robots start at 100 energy at the start of a fight, but can exceed this amount, by hitting other robots to regain energy, without losing it. The amount of energy gained when hitting a robot is (3 * bulletpower), which is three times the power you spend shooting it. By being hit by a bullet, the robot lose (4 * bulletpower), and hitting a wall with an AdvancedRobot extended robot will cause the robot to lose energy as well.

If a robot shoots a bullet which uses the last energy that particular robot has, it will be disabled. A disabled robot will not be able to move or shoot. The last shot that robot took, has a chance to restore the robot, if it hits an enemy and thereby regaining energy.

Energy for the robots are both the health and the spendable resource for attacking, which makes every decision of maneuvering and shooting count.

2.1.4   Scoring

Winning in robocode is not about being the sole survivor, not even in the RoboRumble gamemode, which is the "every man for himself" type of gamemode. It is all about scoring,

and there are different methods for scoring points. The various types of scoring are as following:

- Survival score, every time a robot dies, all remaining robots get 50 points.
- Last survivor bonus, the last robot alive scores 10 points for every other robot that died before it.
- Bullet damage, robots scores 1 point for every point of damage that robot deals to other robots.
- Bullet damage bonus, if a robot kills another robot with a shot, it will gain 20% of all the damage it did to that robot as points. Ram damage, any robot that rams another robot gains 2 points for each damage they cause through ramming.
- Ram damage bonus, every time a robot kills another robot by ramming, it scores an additional 30% of all the damage it did to that robot as points.

When all the above scoring points for all robots in a battle has been added up, the robot with the most points wins the game.

2.2   Choice of parser generator

When choosing a parser generator, one also has to choose a lexer generator, for the lexical analysis. The choice of not building a parser for this project without a generator tool, was due to the fact that the ANTLR4 tool had a build-in lexical analyzer, and a plugin for Eclipse/IntelliJ, generating abstract syntax trees while writing the grammar. Other parser generators were discussed before making a final decision, such as CUP, but the lack of lexical analyzers and abstract syntax tree builders, and at the same time the ease of installing the ANTLR4 parser generator, stated that the ANTLR4 tool was the choice of generator for this project. For the IntelliJ IDE there was a single plugin the user had to install, but for Eclipse, and the abstract syntax tree builder for Eclipse, it required a few plugins, and a bit of experience with the tool, to fully understand how to operate with the tree builder window. Both Eclipse and IntelliJ has been considered as the IDE to use. IntelliJ is preferred because of the ease of installation and use of the ANTLR4 plugin.

2.3   ANTLR4 Parser Generator

In this project, the ANTLR4 parser generator was chosen, as the tool generated both the parser and the lexical analyzer. This tool, as a plugin for both IntelliJ and Eclipse, could also build abstract syntax trees (AST), which are the trees representing abstract syntactic structures, language correct (syntactically correct) sentences (source code) in a computer language. These trees have a top representing the program, and nodes representing terminals and non-terminals. The roots of the tree are terminals, which is the syntactically correct sentence.

The AST tree builder for the ANTLR4 Eclipse plugin would show the trees in an Eclipse window, where the user would be able to write a sentence, and the window would show the AST for that particular sentence, if it was syntactically correct, corresponding to the grammar described in a .g4 file in the Eclipse project. ANTLR4 parses through an LL(*) algorithm, which means it can process any LL(x) grammar, where 'x' is the amount of

lookahead needed for parsing, and the LL means it parses from left to right, with leftmost derivation. This makes it a top-down parser. The grammar input to this tool should be a CFG (Context-free grammar) in EBNF (Extended Backus-Naur form), which is a formal description of a formal language, including programming languages. This parser generator can parse to four different languages, where the interesting one for this project would be Java. This tool can also generate a C# output parser, but as this project narrows Robocode to only be written in Java, this wasn't in consideration for choosing the parser generator. Robocode can, as earlier stated, also be written in C#, which would enforce the choice made, if this project also included the C# source code for Robocode.

The ANTLR4 tool for Eclipse required a few other plugins to make the AST window work correctly, and it is a little bugged. When the user defines a grammar, the user then generates an ANTLR4 recognizer, if the .g4 file is then saved, the user then has to edit the document to make it a not-saved file to operate in the AST window. If the user has saved the document, without editing afterwards, the window would be unresponsive.

In IntelliJ the ANTLR4 plugin requires very little work to get started. To generate the ANTLR4 files the user has to rightclick and select the generate options of the plugin and the IDE does all the work. Similarly to generate the AST all one has to do is rightclick and choose to test the grammar.

Language design 3

In this chapter the design decisions made during the process of creating the language will be described here. There are three criteria for the development of the language, *readability, writability* and *reliability.* The decisions made to accommodate these will be described in detail in the first section of the chapter. In the next section, the MoSCoW method and the application of it in the design process will be described.

3.1 Language criteria

In this section the three main criteria for designing the language will be discussed with focus on the implementation of these in the language. The criteria are based on theory from the book *Concepts of Programming Languages*[Sebesta, 2009].

3.1.1 Readability

Readability is referring to the ease of reading and understanding a programming language. The language in this report should be very simple, since the programming language is, as earlier mentioned, targeted for high school students with little or no programming experience. Therefore the only the necessary features for a beginner in both programming and RoboCode should be implemented.

Since the language is targeted for beginners, one of the criterias for the language would be to make the syntax as simple as possible, but still have it generally look like Java. The language should also have a high level of orthogonality, which also will help make the language simpler.

3.1.2 Writability

The general purpose for a DSL language is a language is to be able to make solutions for a specific problem, therefore the writability is important in this project, since the purpose of this project is to make a DSL language for RoboCode. As mentioned in the section above, the language should have a high level of orthogonality, which will also help on the writability of the language. Reliability

3.1.3 Reliability

REMEMBER TO INSERT!

3.2   MoSCoW analysis

INTRO

Must have

- Primitive types and variables (assignment)
- While loop
- Reserved calls
- Robot naming
- If/Else/Elseif statements)
- Arithmetic expressions and operators
- Logical expressions and operators

Should have

- Events
- Void and type methods
- Cos, Sin & Tan

Could have

- For loops
- Arrays
- Strings
- Print statements
- Comments
- Setup block

Want to have, but can't right now

- Random number generator
- Other robot types
- Other RoboCode gamemodes

Language Description 4

This chapter is focused on describing the technical details of the language. The usage of the language will be described with a walkthrough of some features, a context-free grammar will be showed describing the syntax of the language in detail.

4.1 Grammar

4.1.1 Lexicon

The definition of what input are allowed for each lexical in the grammar is defined by regular expressions. This will be described here with a table of terminals with matching regex. A stream of characters is read by the scanner of the compiler and then turned into a lexical defined by the regex.

Terminal	Regular expressions
ID	[a-z] ([a-z] | [A-Z])*
TYPE	Num | Bool | Text

Tabel 4.1. Table with terminals and matching regular expressions.

Conclusion 5

Discussion 6

Future work 7

Litteratur

Larsen, **Updated 2013**. Flemming N. Larsen. *ReadMe for Robocode.* http://robocode.sourceforge.net/docs/ReadMe.html, Updated 2013. Accessed: 03/03-2016.

Nelson, **2015**. Mathew A. Nelson. *Robocode.* http://robocode.sourceforge.net, 2015. Accessed: 03/03-2016.

RoboWiki, **2013**. RoboWiki. *Robocode.* http://robowiki.net/wiki/Robocode, 2013. Accessed: 03/03-2016.

Sebesta, **2009**. Robert W. Sebesta. *Concepts of Programming Languages.* ISBN 978-0-13-139531-2. Pearson, 2009.

Rettelser

Appendiks name A

Reserved calls	Our language	RoboCode
Tank.	forward(num distance)	ahead(double distance)
	backward(num distance)	back(double distance)
	doNothing()	doNothing()
	energy()	getEnergy()
	heading()	getHeading()
	height()	getHeight()
	width()	getWidth()
	velocity()	getVelocity()
	xCoord()	getX()
	yCoord	getY()
	stop()	stop()
	resume()	resume()
	turn(num degrees)	turnLeft(double degrees)
Gun.	shoot(num power)	fire(double power)
	coolingRate()	getGunCoolingRate()
	heading()	getGunHeading()
	heat()	getGunHeat()
	turn(num ddegrees)	turnGunLeft(double degrees)
	adjustGunForRobotTurn(boolean)	adjustGunForRobotTurn(boolean independant)
Radar.	heading()	getRadarHeading()
	scan()	scan()
	adjustRadarForGunTurn(boolean)	adjustRadarForGunTurn(boolean independant)
	adjustRadarForRobotTurn(boolean)	adjustRadarForRobotTurn(boolean independant)
	turn(num degrees)	turnRadarLeft(double degrees)
Battlefield.	height()	getBattleFieldHeight()
	width()	getBattleFieldWidth()
	numOfRounds()	getNumRounds()
	enemies()	getOthers()
	roundNume()	getRoundNum()
	time()	getTime()

OL / Robocode Event	OL Event Information	Robocode Event Information
BulletHit / onBulletHit	bullet()	getBullet()
	energy()	getEnergy()
BulletHitBullet / onBullet-HitBullet	bullet()	getBullet()
HitByBullet / onHitBy-Bullet	bearing()	getBearing()
	bearingDegrees()	getBearingDegrees
	bullet()	getBullet()
	heading()	getHeading()
	headingDegrees()	getHeadingDegrees()
	power()	getPower()
	velocity()	getVelocity()
HitRobot / onHitRobot	bearing()	getBearing()
	bearingDegrees()	getBearingDegrees
	energy()	getEnergy()
	myFault()	isMyFault()
Death / onDeath	time()	getTime()
HitWall / onHitWall	bearing()	getBearing()
	bearingDegrees()	getBearingDegrees()
EnemyDeath / onRobot-Death	time()	getTime()
RoundEnded / onRoun-dEnded	round()	getRound()
	totalTurns()	getTotalTurns()
	turns()	getTurns()
ScannedRobot / onScan-nedRobot	bearing()	getBearing()
	distance()	getDistance()
	energy()	getEnergy()
	heading()	getHeading()
	velocity()	getVelocity()
Status / onStatus	status()	getStatus()
Win / onWin	time()	getTime()