Final Report
Blocks World for Teams

BW4T Context Project

Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

FINAL REPORT

BLOCKS WORLD FOR TEAMS

by

BW4T Context Project

Search and Rescue Team:	Daniel Swaab	4237455
	Valentine Mairet	4141784
	Xander Zonneveld	1509608
	Ruben Starmans	4141792
	Calvin Wong Loi Sing	4076699
	Martin Jorn Rogalla	4173635
	Jan Giesenberg	4174720
	Wendy Bolier	4133633
	Joost Rothweiler	4246551
	Joop Aué	4139534
	Katia Asmoredjo	4091760
	Sander Liebens	4207750
	Sille Kamoen	1534866
	Shirley de Wit	4249259
	Tom Peeters	4176510
	Tim van Rossum	4246306
	Arun Malhoe	4148703
	Seu Man To	4064976
	Nick Feddes	4229770

TU Delft Delft University of Technology

ABSTRACT

In this Context Project we have taken the original Blocks World For Teams (BW4T) software and developed a new version that is more maintainable, more extensive in its features set, and easier to use. We have restructured the original code and added different capabilities, including additions that allow the user to interact with the system in various ways. The result is that the software can simulate a wider range of real world problems.

Contents

1

INTRODUCTION

1.1. PROBLEM DESCRIPTION

In the field of search and rescue there are many advantages to using robots over human beings. A rescue team can use a specialised robot to reach areas which would otherwise be hazardous to humans. As useful as a single robot may be, a team of these robots is needed in countless real world situations; these teams need strategies in order to efficiently collaborate. To develop and evaluate a particular strategy, simulation software is required. The main challenge in developing such software is how to ensure realism.

1.2. BLOCKS WORLD FOR TEAMS

Blocks World For Teams (BW4T) [1], is a simulation environment for search and rescue robot teams. The simulation world contains various elements that can be tailored to represent numerous real world scenarios. The main elements are the rooms and the blocks contained in them; these coloured blocks represent injured victims. To test a particular strategy, a sequence of colours can be given in order to specifiy which victims have priority. Another key element in the BW4T simulation environment are the robots. Using the multi-agent programming language GOAL, strategies can be tested that require robots to communicate and share the workload efficiently. The combination of this language support and a fully fledged graphical user interface, make BW4T a truly powerful simulation tool.

1.3. PROBLEM DESCRIPTION

In order for BW4T to mirror reality accurately, new features have to be added to support more advanced simulations. One of the main extensions is the support for varying types of robots; the current version only supports a single default type. Robots need to be able to vary in size, speed, and capabilities. Another vital improvement to the simulation environment is the addition of new zone types including charging zones and blockade zones. These main extensions, along with several other essential improvements, are what the BW4T system needs to continue serving the user's needs.

1.4. END-USER'S REQUIREMENTS

To extend the capabilities of BW4T, the client gave a set of requirements. The main concern of the client is that the system is unmaintable in its current state; the coupling between modules is far too high. The second requirement of the client is that the simulation robots need to be customisable. Bots need to have adjustable sizes and speeds, as well as a new functionality set including color blindness and size overloading. The client also asked for a graphical user interface overhaul that would allow users to edit maps and gain a clear overview of the simulation environment. Finally, the client required that users should be able to control robots manually to facilitate simulations of mixed teams containing both humans and robots.

2

Overview of the developed and implemented software

The BW4T environment is a useful tool to test team strategies but is very limited in its current condition. To give the user more freedom in creating diverse scenarios we added in a number of new features. In this chapter we give an overview of the developed software for the new features.

2.1. Logger

The logger was an already existing feature that allowed a user to see what exactly happened during a simulation after it has happened. The log files it generated were however quite unclear and also couldn't handle our additional features yet. So we made sure that details about the bot (e.g. being colorblind) are also logged.

2.2. Scenario Editor

The Scenario Editor is a GUI that can be used to generate configuration files that can be passed on to the BW4T client. These configuration files contain the IP address and port number that the server binds to and the client connects with. It's also possible to specify a map file that the server has to use, whether or not to launch a GUI for each agent and options to enable path visualization and collisions between robots.

2.3. Bot store

As a part of the Scenario Editor we made a panel that handles the creation of customized bots. This panel is the bot store. In here users can specify the attributes of the currently selected robot from a list of robots. These attributes are the handicaps described in the next section of this chapter.

2.4. Handicaps

A large feature that we have added in this new version of BW4T is that the robots can have certain handicaps allowing the user to create different robots suited for different tasks. As an example, one of these handicaps makes the robot colorblind, causing all blocks to appear grey.

2.5. E-Partner

Another addition to BW4T is the E-Partner. This is a small, tablet-like thing that allows an agent that has the E-Partner to communicate with other agents that also have an E-Partner. It can be found in the corridors and looks like a yellow triangle, turning green when it is picked up. The E-Partner was a special request by one of the customers to allow for communication between agents for more than just GOAL-messages [2]. It can also track if a robot is going in the right direction and it checks when it is dropped that it is not forgotten.

2.6. ENVIRONMENT STORE

In the new Environment Store we can create a map that is much more extensive in features than the maps we were able to create before in the map editor. The user can now create different types of zones at any cell in the grid that we use, enabling us to create much more versatile environments for the bots since we are no longer bounded by the old rows and columns structure. Also we have added in different types of zones including blockades and charging zones that allows for the user to create much more interesting maps.

2.7. HUMAN PLAYER GUI

A feature that we have extended is the Human Player GUI. The extra functionality that is added are the menus. We changed the menus so that the options are different for every bot according to their capabilities. For example, a bot without a gripper won't see the option to pick up a block. Additionally, all the agent handicaps are implemented in the interface too, e.g. a colorblind agent sees all blocks as grey blocks. Furthermore, an E-Partner chat session has been added, where the agent can send messages to the E-Partner and the E-Partner can send messages to the agent.

2.8. COLLISION DETECTION

A whole new feature is the collision detection. Bots can now collide with each other if collision detection is enabled. When they have collided, a new option is available: navigate obstacles. With this option, the bot will take another path to his destination. Also new is the option to visualize the traversed path and there is a new pathfinder.

2.9. REFACTOR

While not really a feature, an important part of the project was to improve the existing codebase. The code needed to become easier to maintain and extend. This was something that with the current heading of the project became harder and more complicated to do. The code has been restructured into different projects and complexity of several classes and methods has been greatly reduced.

3

DESCRIPTION OF THE DEVELOPED FUNCTIONALITIES

In order to expand the use of the BW4T environment, new functionalities were added. These functionalities will make it possible to create more diverse simulations for testing team strategies. In this chapter we describe how the new functionalities are implemented.

3.1. LOGGER

We extended an already existing functionality, namely the Logger. Originally the name of the log files were not very clear, for example 'BW4T748554815099389365.txt'. These names are not giving the users any information about what they've logged. Now the log files get the name of the date and time when the server is started. The format equals 'BW4T-year-month-day-hour.minute.second.log', an example of a name of a log file: 'BW4T-2014-06-20-10.47.54.log'.

When pushing the reset button a new sequence will appear and the client needs to be restarted. The already existing log file will get a .1 after .log, when the reset button is pushed again, this .1 will turn into a .2. So when having multiple log files with same timestamp, the oldest log files will have the highest numbers. Within the log file we added a timestamp as well. Every time something is logged, the time of logging will be added.

At the end of the file a summary will be logged. Originally, the time that the sequence was finish was logged, but if you wanted to know how much time it took to complete the sequence you had to calculate it yourself. To make this easier, the logger now does this for you. When it took more than a minute to finish the sequence, the total minutes and seconds will be logged. If it took less than a minute the seconds and milliseconds will be logged. Furthermore, the original log files always mentioned that the type of a bot was unknown. Now, the handicaps will be printed or 'none' if the bot has no handicaps. Another issue was that the original logger did not log the amount of room entries per bot correctly. When a bot entered a room, the logger registered that the bot entered that room more than 10 times. When the bot left the room the logger also registered that the bot entered that room. We fixed this and now the number of rooms entered is correct.

3.2. SCENARIO EDITOR

The Scenario Editor (Figure 3.1) is a new feature with which a configuration file can be created, opened, modified, saved and exported as a mas2g working directory. The editor can be divided into two main parts: The configuration panel and the entity panel. In the toolbar there is a button "File". When clicked a drop down menu will appear with the options to open, create, save and export the file. Exiting the editor is also possible here.

To implement the Scenario Editor we made use of the MVC structure.

CONFIGURATION PANEL

Here, the client and server IP address and port number can be set. It can also be specified whether the GUI needs to be launched, whether the paths will be visualized and whether collisions will be enabled. Lastly,

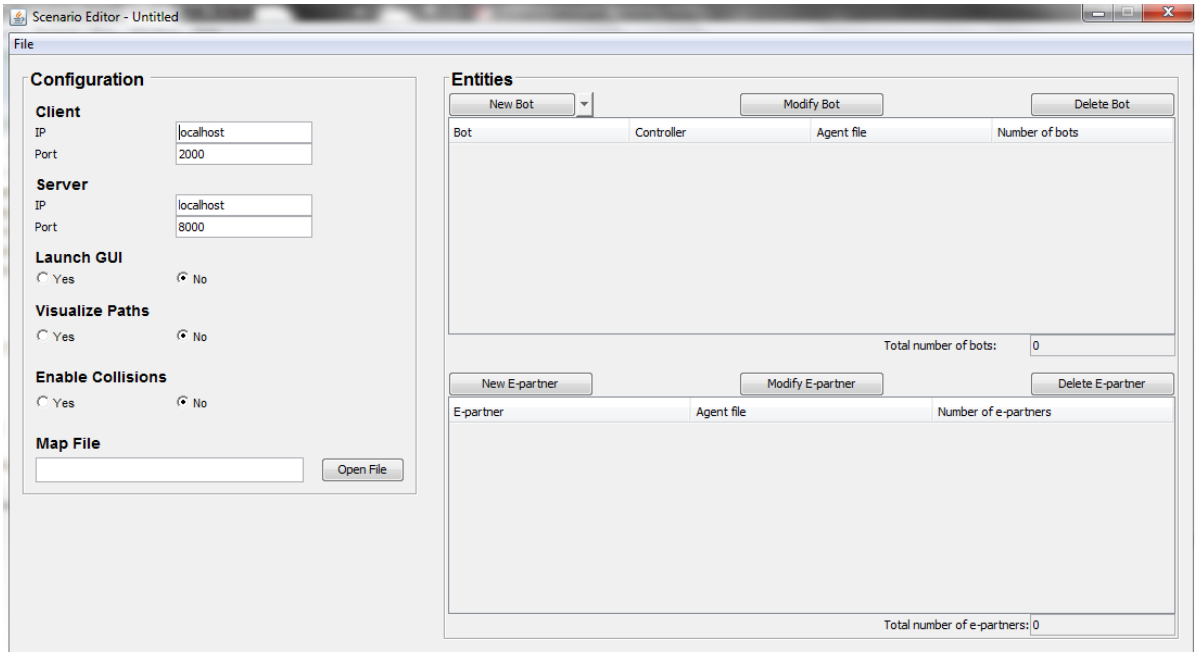


Figure 3.1: Scenario Editor

selecting a map file to use in the configuration is possible.

ENTITY PANEL

The bots can be created, modified and deleted here with the following three buttons: "New Bot", "Modify Bot" and "Delete Bot". Next to the "New Bot" button there is a drop-down button. Here you can add a new default bot to the bot list, which is being displayed below these three buttons. On the right side, under the bot list, the total number of bots is shown.

Below this, there are the three e-partner buttons, followed by the e-partner list and the total amount of e-partners created. It has the same functionalities as the bots part, but then for the e-partner.

By selecting a bot or e-partner and then clicking on the modify bot or e-partner button, the Bot Store or E-Partner Store will open. Here you can further configure your bot or e-partner.

3.3. BOT STORE

In the Bot Store a bot can me modified to the users wishes. A standard bot is the most capable bot, meaning it does not have any handicaps. The handicaps are discussed after this. Several handicaps can be added to a bot. This way the user can adjust what a bot can or cannot do. Besides the handicaps the user can specify how fast a bot can go and what capacity the batter of the bot has. Also bots can now have different sizes. The speed, battery capacity and size can be adjusted with the sliders. When satisfied the save button is clicked and the custom bot settings are saved.

3.4. HANDICAPS

Part of the functionalities that we have developed is that robots in BW4T can now have certain abilities and inabilities, also referred to as handicaps, to broaden the possibilities of types of simulations.

The first handicap that we have developed is the gripper handicap, allowing us to set the number of grippers of a robot in a range from 0 to 5, allowing it to hold up to that number of blocks simultaneously.

The second handicap is the handicap of color blindness, making the robot unable to distinguish colors of different blocks. This handicap still allows to robot to see the blocks but it will make all blocks appear as dark gray blocks, simulating a robot with a black and white camera. Besides adding handicaps to a robot we can now also set different properties of the robot. In the scenario editor we can change the size and speed of the robot, creating a robot that is unable to pass through doors because it is simply too large for the door, or making a really fast bot that can walk through the map at a much higher speed than other bots, possibly

serving as a robot purely meant to scan the map and tell robots with multiple grippers where certain blocks are.

The last feature we have added to the robots is that robots can now carry a battery with them. The original bots can always continue walking around the map, picking up blocks and talking to their team members. However, when the user now chooses to set a battery for a robot (enabling it in the scenario editor) it can set its capacity and the bot will run out of it after moving around the map for a period of time. It can then recharge its battery by finding its way to a charging zone and recharge it by walking over or standing on the charging zone. When the battery capacity is disabled the robot will always be able to keep moving.

3.5. E-PARTNER

A feature that one customer specifically requested us to do was to implement an E-partner. An E-partner is a device similar to a tablet to support the robot.The E-partner is available to a robot in the corridors. It can have two functions. First it can have a gps function. With this function the E-partner checks wether or not the bot is going in the direction that it should. If the bot goes into the wrong direction it gives a notification that the bot should change its course. The second function the E-partner has is that it can detect if a bot puts it down and leaves the room. When this happens another notification is sent to the bot to inform it that it has forgotten the E-partner and it should head back to pick it up again. These two function can be selected in the E-partner store. When the right selections are made the E-partner can be saved with the save button.

3.6. ENVIRONMENT STORE

The old map editor has been completely redesigned to what we now call the Environment Store. In this new Environment Store we are no longer limited to the standard mapping of rows and columns, bordered and separated by corridors with the start and drop zone at the bottom of the map. It offers the user the ability to create much larger maps and place the different types of zones across the map according to a grouping that is preferred by the user.

Besides the new functionality that allows us to position the door of a room on any side, we can now also add two new types of zones called blockades and charging zones. The blockades serve to create a map with parts where the robot cannot pass through. This way the user can create a maze that is difficult to navigate or zones that take a long time to reach. We have added charging zones which allow the bots that contain a battery to recharge when passing it. A charging zone is like an open space and does not contain any walls or doors. Both blockades and charging zones are the size of a basic room or corridor and therefore easily fit in the new mapping. In the new Environment Store we allow the user to randomize every aspect of the new map separately. Through the tools menu the user can randomize the rooms through a standard algorithm or randomize the blocks in rooms and sequence according to parameters the user can set himself. This way a complete map can be generated within seconds, even if the map is as large as a 1000 editable zones.

After the user has created a map, or while editing a map, it can at any time show a live preview of the real map by using the ‘Preview Map’ option in the ‘File’ menu. This way the user can see how the changes in the map affect the actual environment as it would run in BW4T. When the user is satisfied with the map he has created, or simply wants to save the created map as a draft to continue with at a later stage, he can save and later open the map through a few simple clicks in the ‘File’ menu.

3.7. HUMAN PLAYER GUI

One of the functionalities we have extended is the Human Player GUI (Figure 3.2). We roughly split the interface up in two sections. The map is being displayed on the left, while the chat sessions are being displayed on the right.

THE MAP

In the map there is the possibility to zoom in and zoom out with the mouse scroll button. There is also a scrollbar, which will appear when the map is larger than the area where the map is being displayed. Upon clicking on the rooms, corridors, blocks and/or e-partners, a drop-down menu will appear with the possible actions.

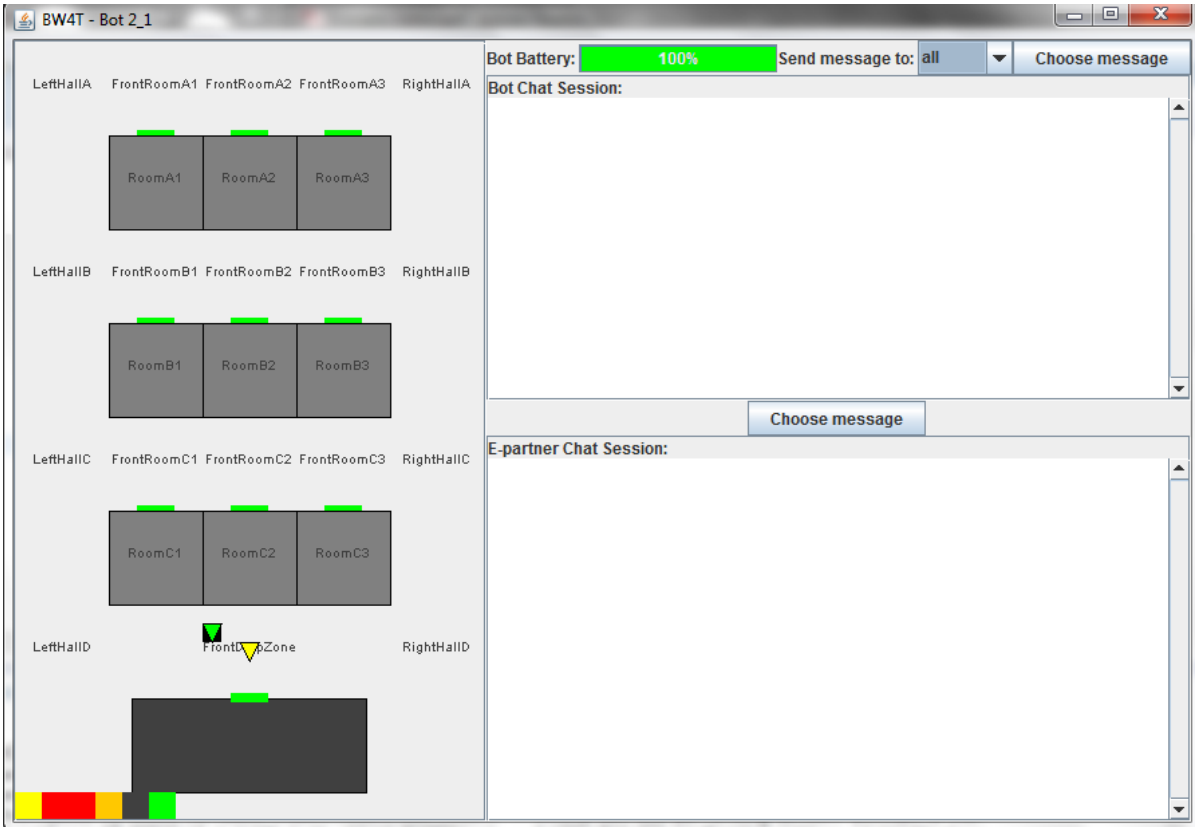


Figure 3.2: Human Player GUI

CHAT SESSIONS

Originally there was only one chat session, but because of the addition of the e-partners, we have decided to split up the chat session into two chat sessions. One for bots to bots and one for bot to e-partner.

The bot battery is being displayed first. Next to it, there is a drop-down box where the user can select to which agent he/she wants to send a message to. There is also a button: 'Send Message', where you can select the message to send. Below this, there is the chat session box of the bots. A drop-down menu with possible answers to questions will appear when clicked in the box.

Some space is reserved for the e-partner chat session and message button below the bot chat sessions. These will only appear when an e-partner is being held by the bot. As soon as the e-partner is dropped, the message button of the e-partner will disappear. The chat box will not disappear though. This way the e-partner can still send messages to the bot, if it has GPS enabled. The message button is only used for sending messages to the e-partner and dropping the e-partner.

3.8. COLLISIONS

FEATURES

With the addition of collisions a few new features are available in BW4T. In this section a list of these features will be presented, along with a short description of said feature.

Feature: Collisions

It is now possible for bots to collide with each other. This option can be toggled on and off.

Percept: bumped

When a collision occurs a bumped percept is sent to the bot who caused the collision. The bumped percept contains the name of the bot being bumped into.

Action: navigateObstacles

When a collision the navigateObstacles/0 action can be used to request a new path from the path planner, taking blocked tiles into account.

Feature: Visualize Paths The paths being traversed by the bots can be made visible on the server display.

Feature: Improved Path-finding There are two pathfinders, one that uses the map zones as navigation points, and the new one which uses the grid points.

JUSTIFICATION

BW4T3 introduces the ability to collide with other bots on the map. Before version 3 there were no collision detection and handling methods, due to this bots were free to move through each other as they traversed the map. Before collisions could be enabled a few key issues had to be solved. In this section these issues will be discussed and the reasoning behind specific design decisions explained.

The main issue faced when planning the implementation of collision was that the path planner inherently could not handle collisions. In fact, the path planner as it is, is the main reason there are collisions. Currently all paths go through the center of each zone. Thus when traversing from A to C, all bots on that route will move through the center of B, thus creating a bottleneck. Furthermore, as the path planner only used the center of each zone, collisions could not be avoided if they were in the way to the center of a zone. Therefore it was decided that a new path planner had to be implemented. This new path planner would not use the center of the map zones as navigation points, rather the points on the grid. To clarify, the map seen when the server is running displays the map zones. These can be rooms, corridors, drop zones, etc. All objects (e.g. zones) present on the map are located on an 'invisible' grid. The new path planner uses the points on this grid as its vertices, with the points on the north, south, east and west as neighboring vertices, connected by an edge with length 1. While in theory vertical neighbors could be supported, they were not added as they are infinitesimally many, and would complicate collision detection.

With the path planner created the next issue was detecting if a bot was going to collide with another bot or not. The original specifications called for a bumped percept to be called after a bot collided with another bot. Since implementing this would take as much effort as detecting a collision before it happened it was decided to detect collisions before they actually happen. In a way this is more realistic, as generally preventing a collision is the preferable solution.

When a collision is detected the 'navigateObstacles' action can be used to plan a path around the obstacle. This action must be called separately. The action is not automatically called when an obstacle is detected, as it would remove the possibility for the agents to coordinate a plan to go around an obstacle. (Especially if two agents are blocking each others path). Note: When a collision occurs the agent can also use the goTo actions to go to a new location provided the path is free.

During the development of the collision features debugging proved quite complicated as there was no way to visually see the path generated. The solution was to draw the path on the map. This debugging tool proved quite handy during simulations, as it could be used to see how to bot reacted when a collision occurred. While it may not have any scientific value as it is purely visual, it has been included in BW4T3. It could be a helpful debugging aide, especially for 1st year students following the AI course, as it can be used to debug and visualize the actions of the robots.

A key improvement in BW4T3 was the complete and absolute separation of the client and server. In accordance with this it is possible to turn collision detection on/off by setting the appropriate option in the Scenario Editor. However, it is also possible to change this setting on the server (and thus modifying the scenario loaded by the client). While this is in stride with the decision to fully separate the server and the client, it was decided to include the option on the server to make it easier to test environments with collisions enabled. This to avoid having to change the scenario and reload it on the server as this would be extremely cumbersome. By simply checking a box to have collisions be enabled or disabled this process was made significantly easier.

CURRENT CONCERNS

At the moment there is an issue that prevents the startup of bots on top of each other when collisions are enabled. Due to the complexity of writing a procedure that makes sure every bot spawns on an unoccupied location, it had been decided that for all bots collisions will be disabled for as long as, after they have spawned, they share a grid point with another bot. However, when using GOAL, somewhere during the initialization of Repast one of the navigating robots is presumably being cloned. But this happens after the robots are initially loaded into the context. Due to this, when the NavigatingRobot requests its location a null pointer is returned. This only happens when one robot is stepped before the other, with collisions enabled and two bots being on top of each other. Despite meticulously searching for the cause it could not be located. However, a simple workaround exists. To prevent this error from occurring one must, after initializing the environment, run the server for at least one step.

3.9. REFACTOR

We improved the quality of the code and the overall structure of the project by implementing the following:

- **Split up codebase into Client and Server** - The client and server shared the same codebase. This made it hard to keep a proper overview as it wasn't always clear at a glance which class belonged to which subsystem, and some classes were even shared entirely. We split these subsystems into individual projects to make it easier to maintain, extend and test them. New features like the Map Editor and Scenario GUI were also added as separate projects. It turned out we couldn't cut all dependencies between the projects. For that reason, we created a shared library project as well, called the BW4T-core.
- **Convert to maven project** - Maven is a build automation tool. It makes building and testing the project easier because it shields you from many of the details. It replaces the current complicated ANT buildscript, and accounts for dependencies. Maven provides guidelines for best practices, which is useful for large projects which can become very complex easily, like BW4T. Using Maven, it becomes much simpler to maintain a continuous working project.
- **Improve overall code quality** - Using source code analysis tools like Sonar and inFocus, we discovered that there were a lot of code quality issues present in the existing BW4T codebase.
  - There were a lot of very complex methods and classes. We reduced complexity to make the code easier to understand, maintain and extend. Cyclomatic complexity of methods now rarely exceeds 10 with an average of 1.8 per function, while class complexity doesn't exceed 159 with an average of 10.4. This reduction makes the code perform better, allows for easier extension and improves readability.
  - In order to make the code more readable, we applied a well-defined Checkstyle throughout the project. This included the use of curly braces and indentation, in order to prevent silly bugs from being created by missing braces and the likes.
  - Javadoc was occasionally present, but often times contained insufficient information. We not only made sure that Javadoc was present for all relevant methods and classes, but also improved existing Javadoc so one is able to easily figure out what a method or class does without needing to read the actual code.
- **Add sufficient tests** - There were no tests present whatsoever in the old codebase. In order to ensure functionality after changes, a lot of tests have been written. We achieved 50% line coverage for existing code, and 75% line coverage for newly written code.

4

THE HCI MODULE

In this chapter we will discuss the human-computer interaction part of the project. First we discuss the personas and user scenarios created by group 1. Afterwards we discuss the user test results of group 2, followed by group 3.

4.1. INTERACTION DESIGNERS FROM LAST YEAR

These are the names of all the students who already passed the Interaction Design course (TI2600) last year (2013): Ruben Starmans, Martin Rogalla, Joop Aue, Wendy Bolier, Arun Malhoe, Calvin Wong Loi Sing, Seu Man To and Katia Asmoredjo.

4.2. GROUP 1

PERSONAS

As can be seen in the attachments, our personas are all in the range of 18 – 60 years old and make use of the BW4T environment. Obviously, the design of our graphical user interfaces, views and other choices are based upon this range of age. If, for example, we would have been developing this software for children aged 5 to 12, we could have chosen to make it more of a game. To do this, we could have introduced missions, where a bot has to perform several actions in some specific order to progress through the game. When dealing with children we also could have added more graphics and effects to make it more attractive to play. We could have made real buildings, with several stages and effects like burning trees or houses. A shaking screen as a cause of an earthquake, making it harder for the player to complete the game level.

On the other hand, if we had to develop this software for elderly people (age 60+), we probably would not even have thought about gamifying the software. Instead, other parts of the software would have changed. For example buttons would have been bigger so that it would be easier to click on them correctly. Text fonts would have been larger so they can be read more easily and maybe another font would have been chosen if it would turn out to be easier to read.

What we did not consider during the making of BW4T was the phenomenon of colorblindness. One out of 12 men and 1 out of 250 women are colorblind. Although the bots themselves are able to have the handicap colorblindness, there is no option for users to choose a colorblind option. In order to take this into account, blocks could contain numbers instead of having colors. Changing the colors would not have a significant impact, as there exist different kinds of colorblindness.

After some tests with our client (K.V. Hindriks), he told us that he had a hard time seeing the colors of a button during the test. With that in mind, we are still considering a good and practical solution for this problem.

All our personas fall in the same category: they all know how to use GOAL and study artificial intelligence. If we would change the personas to people who would not do any research in or study artificial intelligence, we would probably get back to the “game” as described for children. We adjusted the user interface to a simple

and intuitive user interface. The students and researchers should not have any problems finding features or using the program because of that. The logger gives a clear view of each bot. It says which bot did which action and it logs for each bot the amount of good block drops, bad block drops and especially how long it took the bot to finish the sequence. If our personas did not contain researchers, it would probably not matter much how long it takes a bot to finish the sequence as for the student it is only important that the bot completes the mission.

USER SCENARIO

If we did not have a batchrunner, a small script that runs the program multiple times in a row on the background, it would have taken researchers huge amounts of time to run the program multiple times. Not to mention putting together and analyzing the data afterwards. Without the Scenario Store, users could only select the amount of rows and columns (which would turn out to be the amount of rooms). The user would not be able to specify which rooms would be at what locations or where the drop zone would be.

As this program is an assistance tool for researchers, you don't want that agents just control your bots. In real life situations there are always humans involved, at least, nowadays, and probably in the future too. So there is a Human GUI (Graphical User Interface) which lets the user control a bot in the environment.

Even though we use bots which are probably capable of everything, there are still different kinds of bots. Robots come in all colors and sizes and therefore we want our bots to have some handicaps. In the Bot Store users are able to choose handicaps on robots, e.g. not being able to pick up a block or not being able to fit through small doors. This is very helpful for researchers as well, because they can analyze the difference in completion time when using different kinds of robots.

4.3. GROUP 2

We decided to do some user tests to find out how user friendly our system really is. We based these tests on the user stories and personas described above.

PREPARATIONS AND EXPECTATIONS

As we did not have much time for these user tests, we decided to keep it short. But in order to still get representative results, we decided to gather three test users, each representing a different persona. Fortunately, we found three test users willing to participate in our user tests: a first year student, a professor and a researcher.

There is a lot of documentation that comes with the system so, in real life, the users can consult these documents to learn more about the system and how to work with it. However, these documents are too much reading material for a simple user test. That is why we decided not to give these documents to the test users. Also, we expected that the test users would not need documentation or manuals because we believed our GUI's were clear enough.

First we gave the participants an informed consent form (see attachment *Informed Consent*). The participants had to sign this document before we commenced the experiment. During the user tests, our participants received a short roadmap (see attachment *Manual*), with a few simple tasks they had to complete. We used the "Think Aloud" protocol, so we knew what our test users were thinking when performing these tasks. We wanted to know whether they immediately understood what they needed to do and how they needed to do it, or whether they needed a bit of time to comprehend such tasks. We expected the experiment to go smoothly, and that the user would not have too much trouble, but some elements that were obvious for us (because we made the system) were maybe not simple for our test users.

Furthermore all test users were given a questionnaire (see attachment *Questionnaire*) with seven questions in total. In addition to this questionnaire, we also asked some questions in person, depending on how the evaluation was going.

TEST RESULTS

Our test users were very positive about the changes we brought to the system. They had (some) experience with the old BW4T and they were happy with the new functionalities we had added. However, the system was not as clear and self-explanatory as we believed.

All the test users had problems with:

- Finding out how to add blocks to the rooms
- Opening a file

Opening a file went wrong because the Environment Store begins with a dialog named StartDialog. In this dialog the user can input the map's number of rows and columns, and then continue to the editor screen. Only in this screen is it possible for the user to open a file. It is only natural that this seemed contradictory to our test users: they had to set a number of rows and columns in order to open a map that already existed.

Moreover, a charge zone would not always appear when randomizing zones in the map. The users' opinion was that there should always be one charge zone, no matter what.

Furthermore, the name attributed to the randomize zones command used to be 'randomize rooms'; one of the users pointed out that this was incorrect, because not only did it randomize rooms, but it also randomized other of the map's elements.

Finally, our users suggested various improvements for next versions of the system:

- A one-click-random: it should be possible to randomize everything (zones, blocks and sequence) with one button. The result should be a solvable map, so the randomize functions should take each others' results into account. For example: there cannot be a color in the sequence which does not exist anywhere else on the map.
- Setting up more zones at once: it should be possible to select multiple zones by dragging the mouse over them, and then changing them all at once into rooms, blockades etc. This would be rather useful especially when dealing with large maps.
- Being able to save a map without a start- and/or drop-zone.

WHAT WE DID WITH THE RESULTS

Unfortunately, we did not have much time to adjust the system to the test users' suggestions. So we only focused primarily on modifications that were easy to apply. For the few things we did not have time to correct, we listed them in our final report, so that future teams that will work on this project have an idea of what needs to be done.

What we changed:

- Added a more detailed explanation in the Environment Store. It tells the user how to edit zones and how to add blocks to rooms.
- Made sure at least one charge zone is generated when randomizing zones.
- Changed the name of 'Randomize rooms' to 'Randomize zones'.

We have not yet fixed the 'open file' problem. Unfortunately, we did not have the time to do this properly, though we described this problem in our documentation for the next project group that will work on this.

4.4. GROUP 3

During the last week of the project a usability evaluation was scheduled to find out how different users interact with our newly added features.

PREPARATIONS AND EXPECTATIONS

In order to keep the user test as accurate as possible, a first-year student, a researcher and a professor were asked to participate in the evaluation. Each of these participants had a different level of experience with the old system, ranging from none at all to a lot. Also, each of the participants had used the system before to preform different tasks ranging from programming a simple bot to studying the way in which a large group of bots interacts while trying to achieve a shared task. These handful of participants were chosen to best represent the personas that we had thought out and which can be found in the attachments.

To be able to actually test the intuitiveness of the newly added features it was decided to not explain anything about the features and only give them detailed tasks which forced them to use these features. These

tasks can be found in the attachments (Usability Evaluation - Manual, see the group 3 section). Because the addition of the Scenario Editor enables almost everyone to set up an environment, build a scenario and start the simulation, it was decided that each user test should include these three crucial parts.

TEST RESULTS

During the tests it became very clear that each of the participants had not only a completely different view of the system, but also interacted with the system in a different way. On the one hand there was the participant that started reading lines of text included in the Graphical User Interface while on the other hand there was also a participant that started clicking right away and discovered most features by accidentally bumping into them.
Overall there were only a few features that every participant had problems with:

- What parts of the program have to be started - as in which jar files have to be run -W to preform certain actions
- It was not clear that the table in the Scenario Editor was editable
- It was not clear which disability slider belonged to which robot disability
- It was not clear what the export to MAS function did

As it was decided on to not give a any thorough explanation of how the system worked, all participants needed help starting the right jar files in order to run the right parts of the system. Both the researcher and the professor told us that it was currently better than how it used to be, but were not able to start everything without explanation. This did not come as a real surprise, since we had only made the jars available but had not thought about a logical folder to place them in.
The Scenario Editor has two tables in which the added agents and e-partners are visualized. To speed up the process of setting up an environment this table is editable so that a user doesn't have to open the associated store and change the basic values there. Unfortunately it wasn't clear that this was possible and most participants wouldn't have known without being told. This wasn't really what we had expected, since we thought this feature would be easy to find. But after all this is just a speed-up feature for which alternative methods exist to achieve the same result.
Most disabilities have a slider next to them in order to indicate the severity of the disability (size and battery capacity of the robots for example). Due to the crowded layout of the BotStore some participants had trouble seeing which slider belonged to what disability. As a result of this, a new task was created for the team that consisted of improving the layout so that a slider is next to its disability check-box. We didn't deem this necessary at first, since a slider would only light up when the corresponding check-box was selected.
In order to speed up the process of creating a simulation, an export to MAS (Multi-Agent System) function has been added. This function exports the current configured Scenario and builds a ready to launch mas2g (MAS to GOAL) project. To do this a lot of files have to be created and copied in the background, something that isn't very visual and therefor caused some confusion for the participants.

WHAT WE DID WITH THE RESULTS

In order to improve the visual aspect of the BotStore each disability has gotten a little more space, thus making it easier to see which slider belongs to what disability.
All other problems encountered by the participants have afterwards been explained in the manuals for the users to come. The added features were designed to be as self explanatory as possible, but there is always that little bit of explanation that is necessary in order to fully understand the system. So in the end the usability evaluation was quite useful and has resulted in changes to our system and detailed explanations in the manual.

5

EVALUATION

In this chapter we discuss backward compatibility and analysis of the functional modules and the product. The first section of this chapter is about how we handled backwards compatibility. The user test of the functional modules follows in the next section. The analysis of the product is described in the last section.

5.1. BACKWARD COMPATIBILITY

During the testing of the system, we also ran special tests to ensure that BW4T version 3 was compatible with the GOAL code used for the assignments for first year students. These assignments were created during the Logic Based AI course.

Immediately, the first assignment did not run as expected. The bots would go into a room an see a block of the color it was looking for. It would then try to go to the block, however it would not receive the "atBlock()" percept, so it went back to the centre of the room. These actions repeated endlessly. After improving the logic of the percept, the assignment code worked.

Initially, we found a problem that was caused by a change brought to the "holding()" percept. Originally, only the ID of the block that was being held was contained within this percept, but because of modifications, it then included the position of the Block as well. This, of course, was unnecessary information, as the block was in the same spot as the robot holding it. After removing the data the exercise the second assignment ran without any additional problems.

Some of the old goal agents will not work on new maps, as they use the hard-coded name "FrontDropZone" for the zone that is in front of the "DropZone". The new maps no longer assign a special name to this zone. We decided not to implement this, because we thought a robot should not need to be told which place is connected to the "DropZone", as it can derive that information from its knowledge of the map.

There was another problem with backwards compatibility: we noticed robots started behaving "strangely" when the amount of tics per second was increased (approximately over 75 tps). After deeper analysis of the situation, we came to the conclusion that this problem originated from the fact that GOAL could not keep up with the system. Percepts came in too fast for the bot to make a decision, so its "mind" went overboard. As we could not really fix this issue, we decided to set up a default speed (50 tps), and leave a warning to the users who would want to increase it.

5.2. FUNCTIONAL MODULES

ENVIRONMENT STORE

The customer wanted to be able to build and search through various environment types. The user should be able to build/play/vary the physical realism of the map. An environment is a map with specification of the physical realism. This has been satisfied for the most part according to the customers needs. A user can create a custom map from scratch thus a user can create an environment resembling a real life situation.The user should also be able to add charge zones where the bot can recharge its battery which i also possible now. Other features such as color blindness and other percept failures or bots not being able to pass through each other are handled in the bot store. The features requested by the customer that we did not satisfy are the ability to make bots not be able to see other bots under certain circumstances, limited communication

range, and bot failures. We did not add these features because they had low priority on the backlog.

BOT STORE

- The body store:
  - The E-Partner: satisfied, as it grants a communication potential to other robots in the environment, and bots can perceive it.
  - Humanoid bots: didn't satisfy, at this moment it doesn't make any difference whether the bot is a wheeled bot or a humanoid bot, because of the abstract environment.
  - Wheeled bots: didn't satisfy, for the same reason as the humanoid bots.
- Bots can crash into each other: partially satisfied, the bots can crash into each other which causes them to stand still if they are not commanded or programmed to recalculate it, but nothing else happens (no damaging of the bots, for example). This is because the functionality had a very low priority and was only developed very late in the project, when we didn't have a lot of time to implement further consequences for a collision.
- Equipment store: satisfied, all the functionalities specified in the document are implemented except an alternate form of perception (which doesn't really matter at this stage, as BW4T is still very abstract). Some functionalities, however, are implemented differently than specified, such as the charger which became a charge zone that the bot had to walk over.
- Combination logic: didn't satisfy, the BW4T environment is still too abstract for this to have any result. All bots can have all handicaps activated at the same time, which allows for at least as many possible configurations that combination logic would also allow.
- Accessibility: satisfied, the bot store is directly accessible from the environment store and has to be this way, as the bot store edits the bot currently selected in the environment store.

SCENARIO EDITOR

- Bot type: didn't satisfy, see the body store part of the bot store.
- Capabilities: satisfied, these are the possible handicaps which were satisfied when making the bot store.
- Control: partly satisfied, as it is only possible to specify whether a human or an agent controls the bot. It is not possible to specify partial control.

HUMAN PLAYER GUI

- Appropriate GUI: satisfied, there are no constraints on what a bot cannot do when it is controlled by a player that the bot could do otherwise.
- E-Partner: partially satisfied, this is already managed in the standard human player GUI.

USER TESTS

The functional modules, those being the environment store, the bot store, the scenario editor and the EPartner store are analyzed and evaluated here. A small user test with a researcher has actually been executed with the environment store. This gave the following results: it is hard for a user that has not developed the system to actually understand how to add blocks to a certain room. There is no hint that the white box at the bottom side of the room has to be clicked to add blocks to the room. A solution would be to add a small label or tooltip explaining what the white box at the bottom of the room or map is for. The task to create your own map and save it, however, went very well. As the researcher was very experienced with the BW4T environment, she was impressed by what is possible now using the environment store. There was an issue with there not always being a charging zone in the map at the time, but this has since been fixed. The other issues are discussed in the 'Outlook' part of the report.

DEMO FEEDBACK

There was some feedback on the environment store during demos where the customer could use the software. The original warnings for saving a map that wasn't solvable were simple and did not evaluate on what was the unsolvable part of the map. This was fixed later and the warnings are far more descriptive. They still don't show where in the map the problem is, because that was quite hard to do.

5.3. PRODUCT

As the functional modules are mainly evaluated through usability, the product in general is going to be evaluated through structure and architecture of the components. Also, our part focuses on the new feature aspects of the software, so only the structure and architecture of the code forming that part of the system will be discussed here. The analysis is as follows:

DECORATOR PATTERN

Unlike most of the other groups, we started with pretty much no starting code at all. There was no previous support for robots with handicaps, let alone a bot store where one could create a robot with certain handicaps. The handicap functionality makes use of a decorator pattern to be able to put multiple handicaps on a single robot. At first, the decorator pattern was not correct, so the code was rewritten to be able to function as a decorator pattern. This required some minor changes to the standard robot code. The decorator pattern is correctly implemented now, as stated by the project supervisors, and also allows for a single robot to have multiple handicaps active at the same time. The code used is high quality code now.

MVC PATTERN

The model view controller architecture was used in the Scenario Editor, the Environment Store and the Human Player GUI. This was useful since different team members could work on their own piece of code without being dependent on others and without getting in the way of each other.

6

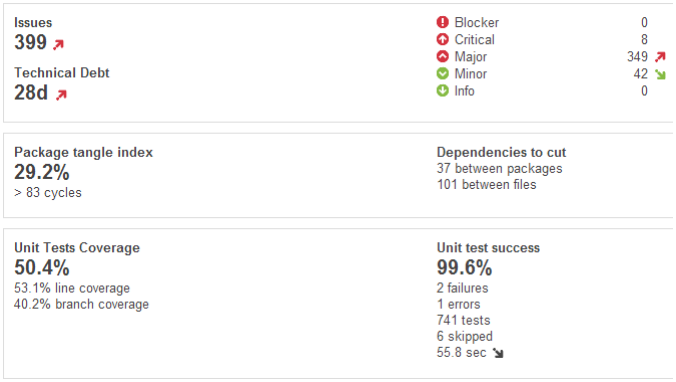
OUTLOOK

6.1. REFACTORING

We see many possibilities and improvements for in the future. We have achieved a lot over the past weeks, and it can be considered as a good start in the right direction. However, a lot more can be done to improve BW4T and make it even easier to maintain and extend its functionality.

POSSIBILITIES

The image below shows a part of the Sonar interface. It shows a number of metrics about the project. Based on these metrics, we can list and explain some improvements.



- Testing: At the start of the projects, there were no tests written whatsoever. We managed to achieve a 50.4% coverage. While this is a great step forward, there are still a lot of classes and methods untested, or tests that could be improved. Sometimes it was impossible for us to check a complete class within the scope of the project. This had 2 reasons. Firstly, some methods interact with the *environment*. It wasn't feasible to set up an entire environment to test a simple method. Secondly, PowerMocking was not possible.

  PowerMocking is a system that enables testing of static/ final/private methods. It's a extension of *Mockito* that uses an older version of *java*. While we use the newest version we aren't able to use PowerMocks because of compatibility issues. This makes it impossible to test a lot of the code.

- Issues & Technical Debt: At this moment there are 399 issues and we have a debt of 28 days. This might sound like a lot, but many of the issues are minor issues, or issues that wrongly marked as an issue.

- Decoupling: There are still too many dependencies in the project. There are 37 between packages and 101 between files. This also results in a Package tangle index of 27,6% (83 cycles). This is pretty bad and could definitely be improved further.

STRATEGY

- Testing: Start with looking for a way to create an simple testing *environment* to test the big classes like *server* and *client*. Also try to get PowerMock running (by waiting until it becomes compatible, or finding another method). Make sure to continue testing!
- Issues & Technical Debt: Issues are divided in Blocker, Critical, Major, Minor and Info. Make sure there are no Blocker and Critical issues and and resolve Major ones that can be fixed. This way you'll not waste too much time fixing issues compared to extending the code.
- Decoupling: With *Sonar* and *Stan* you can visualize the dependencies. When you have an overview you can decide whether to change the class/package and try to cut dependencies. Most likely, restructuring will be needed.

6.2. ENVIRONMENT STORE

We think that the software we created can be easily extended in general to include more features, because of the massive effort to restructure the code and to create a framework for features. Aside from the functionalities which we didn't implement, ideas for a new version (which would be BW4T4) could be: more zones (such as repair zones for robots that are damaged by a collision), more hazards in the map that do different things (such as a supercharge zone, which overcharges robots that use a battery and damages them because of the overcharge, but robots without a battery would be unaffected), et cetera. A usability feature would be a button to randomize both the rooms and the block sequence at once, so that the map is still completable, because during the user tests there was a comment that randomizing both the rooms and the blocks/sequence was a little bit of a hassle. This would add more customization to the BW4T environment, and make it more user friendly, allowing it to be a better platform to test out GOAL agents that are going to be used in real life situations, the goal for developing the BW4T environment further.

We also made some errors with developing the software. For example, the option to load a map is not in the size dialog but in the map editor. This causes problems because the size of the map is already defined by then, and loading a map of a different size causes the obvious problems, especially when loading a map that is too big. It is also not logical to define the size of the map again before loading the map in the editor. Also, the bot store part of the system does not use the MVC design pattern anymore. It actually did after we refactored its code, but because group 3 was building their parts of the system around the old, non-MVC version of the bot store and we discovered this a day before the deadline, we decided that we would reset the bot store to the non-MVC compliant version. A true shame, but it was necessary because it was only shortly before the deadline and we had way too little time to fix and test everything. Better communication could have helped us here tremendously, but alas. A design issue for a new development group is therefore to recreate the bot store to an MVC compliant part of the system, and also rebuild the environment store to work with this new version.

WISHES FROM TEST USERS

These are the wishes our participants expressed during the user tests.

- A one-click-random. It should be possible to randomize everything (zones, blocks and sequence) with one button. The result should be a solvable map, so the randomize functions should take each others' results into account. For example: there cannot be a color in the sequence which does not exist somewhere in a room.
- Setting more zones at once. It should be possible to select multiple zones by dragging your mouse over them and then changing them all at once into rooms, blockades etc. This would be especially useful when having maps with a lot of zones.
- Being able to save a map without it having a start- and drop-zone.
- Being able to add parameters to the randomization of both the blocks and map. Example: different odds for reclassifying rooms as blockades/charge zones/corridors, a new probability that a corridor will be reclassified as a charge zone, and a different ratio (relatively) for every possible block color (the probabilities for the map randomization are hardcoded, as well as the ratios for each possible block color).

6.3. HUMANPLAYER GUI

In the current setup of the BW4T environment blocks can be dropped outside a room, but will then disappear from the screen. The environment doesn't provide functions for showing, picking up and communicating about blocks outside rooms. Keeping the visual aspect of the blocks outside a room is a fairly easy fix, implementing the needed functionality to be able to pick blocks back up and communicate about these blocks asks for bigger changes throughout BW4T.

The messaging system of the current BW4T environment only allows robots to communicate the most basic commands like the color of a block or the location of a robot. This also means robots are only able to carry out basic tasks. Adding more advanced messages opens up a wide variety of possibilities for more advanced tasks for bots.

At the moment e-partners are displayed as yellow triangles. These could be replaced with an image that represents the function the e-partner has, as this would make it easier for the human player to identify the function of an e-partner.

6.4. PATH PLANNING

In the BW4T environment robots plan their path by navigating through the zones on the map. The navigation algorithm creates a graph consisting of all zones, and calculates the shortest path from the start zone to the destination zone. Each zone has a location on the map, however, this location specifies where the center of the zone is. As a result, when paths are calculated using the zones location, the path always goes through the center of the zone.

The new version of BW4T includes collision detection and obstacle avoidance. When navigating around obstacles a new path planner is used, which unlike the usual path planner does not navigate over zones, but points (coordinates) on the map. Due to the higher precision the robot can easily navigate around obstacles in its way. The downside of this path planner is that the algorithm is computationally expensive. A graph with N vertices and up to N^4 has to be generated, with N being the product of the width and height of the map (in points). As such the path planner can not be used often as it would significantly slow down the simulation.

Since the paths by the original path planner all go through the center of zones, the chance is fairly high that a robot will collide with another bot. While this can be safely navigated, it does not realistically simulate a corridor where more than one bot can concurrently pass through. Due to time constraints this problem remains unsolved, but three possible solutions will be presented.

The first solution is to, when planning a path through the zones, return a random point in the zone as opposed to the center of the zone. Because each bot will (most likely) have a different path, collisions will be avoided to an extent. The disadvantage of this method is that the paths will not be elegant. A bot may have to make like a snake to go to a location when a straight line would have been more adequate. Especially with the addition of the battery, which is used when the bot is moving, this may result in an inaccurate simulation of battery use since the bot would be using more energy than is strictly needed.

The second solution, albeit more complicated to implement, is to select a random point in the zone, and make sure that the points selected in all subsequent zones have (if possible) either the same X coordinate or Y coordinate. This would result in straighter lines, lowering the battery use and reducing the chance of collisions when criss-crossing between zones.

The third solution is to change change the way robots navigate through the map. By only planning a small part of the path at a time and taking other robots into consideration while planning its path a robot could find the shortest possible path to its destination. This however comes with a downside as it could consume a lot of computation power and slow down the simulation.

6.5. FINAL WORD

All in all, there are few things that the customer really wanted but we didn't implement because they had a lower priority than other features. Also, the time given for the project was too short to implement all these features. The things that we didn't implement are the communication handicap (being able to communicate with another bot or not, or having a set chance that a certain message is going to be delivered) because this was too hard to do given the communication of GOAL-programmed agents, and the path finding and collisions of robots (robots can or cannot walk over each other in a path), because this was hard to do and

we had time constraints. These are functionalities that can be added by new teams further developing the software. Also, the way we created handicaps allows for easy extension to more handicaps, although some old code had to be rewritten in order to pass the data to the data object and actually be able to see the handicaps in action.

BIBLIOGRAPHY

[1] M. Johnson, C. Jonker, B. van Riemsdijk, P. J. Feltovich, and J. M. Bradshaw, *Joint Activity Testbed: Blocks World for Teams (BW4T),* (2009).

[2] K. V. Hindriks and W. Pasman, *Goal user manual,* (2012).

ATTACHMENTS

PERSONAS

AD —- RESEARCHER

Ad holds a masters degree in Computer Science and is a researcher in Multi-Agent systems and Artificial Intelligence. Together with his research group, he does research from the Technical University of Delft to explore the possibilities and restrictions in the field of agents and robots. His research focuses on the analysis, modeling, and development of agent technology that integrates different aspects of intelligence such as reasoning, decision-making, planning, learning and interaction but also integrates aspects such as emotional intelligence. This multi-agent technology has been applied, among others, in micro-simulation of domains such as traffic, logistics and supply chain management, serious gaming, negotiation, socio-cognitive robotics, and user modeling. As a researcher, Ad would like to use software that is complex in its possibilities but easy to use. He would like to run different simulations multiple times and track results easily.

BOB —- PHD STUDENT

Bob is 27 years old en just finished with studying. He now owns a PhD in artificial intelligence programming. He lives alone in the city of Amsterdam and does not have a job yet. There is a competition coming up which he finds very interesting. In this competition you have to program an artificial intelligence for deep sea exploration. He really likes the idea of deep sea exploration and decides to take part in the competition. For the programming and testing of his AI Bob decides to use the programming language goal and the BW4T environment. He decides on this because he has used BW4T a lot during his studies and is well aware of the environment. Bob can use BW4T to simulate and if he has to even control his agents. He can also change the environment to scenarios he desires. He can change lots of things like: the speed to the bots, how far they can see and even the distance they can communicate with each other. He hopes to simulate deep sea conditions with these features.

JASPER — STUDENT

Jasper is 18 years old, and is a first year student Computer Science at the TU Delft. He participates in the course Logic Based AI, and has to work with the GOAL programming language in the BW4T environment. The course is worth more points than it was last year (the previous course, Multi Agent Systems, was worth 3 points, while this new course is worth 5 points), so as an extra requirement to complete this part of the course, the bots all need to have a handicap, causing them to be unable to complete the problem individually but also causing them to be able to complete it as a team. To specify the required handicaps, Jasper uses the bot store built into the BW4T environment. He knows how to work with GOAL and the program itself, so that isn't the problem. It needs to be clear for him how the handicaps are to be specified and what the handicaps do in terms of disabling the bots.

PAUL —- PROFESSOR

Paul is a professor in Artificial Intelligence at the Technical University of Eindhoven. He teaches students in Computer Science who are in their second your of their bachelor program the course Engineering Multi-Agent Systems and together with his students he participates in different Multi-Agent and Robotics competitions because he thinks it is most important that the Technical University of Eindhoven spends much research in this field. This way he tries to exploit the possibilities of agents and artificial intelligence together with his students. To make Multi-Agent programming accessible to his students, Paul would like to use a system that is easy to use for the students but has limitless possibilities in ways of programming the agents. Furthermore, Paul would like to use the system to do research on Human-Computer Interaction and communication between different agents and would he like the simulations to be as realistic as possible concerning the different scenarios but does not value fancy graphics and animations. Paul is good with computers and assumes that his students are as well.

INFORMED CONSENT FORM

Name: .....

Date: .....

You have been asked to be part of a usability evaluation of our new release from BW4T, which we made during our Contextproject. During this evaluation we will ask you the following things:

- Fill in a short questionnaire before and after the assignments
- Complete a few assignments we give you, while 'thinking out loud'.

This evaluation is to test the system and not to test the user. So don't worry when you can't finish an assignment, there is no right or wrong.

The software is new and untested, so problems may arise.

Your participation in the evaluation is completely voluntary. You do not have to answer any questions you do not want to and you can stop at any time just by saying you want to stop.

Your test results will be kept completely confidential. Your answers will not be linked to your name and will only be used for this evaluation.

If you have any questions left, feel free to ask us. If everything is clear and you still want to participate in our evaluation, please mark the box below and fill in your signature.

☐ I read this informed consent and agree with participating in the usability evaluation.

Signature of test user:

USABILITY EVALUATION - MANUAL

Dear test user,

Welcome to our usability evaluation. This evaluation will consist of 2 parts:

In the first part you will test our EnvironmentStore functionality, in the second part you will test our ScenarioEditor and run the just created Scenario in the HumanPlayer GUI.

While making the assignments, please try to 'think aloud'. That way we have a better understanding what's going on in your head and why you make the choices you make.

PART 1 (GROUP 2)

Create your own map and save it

1. Create a map with 7 rows and 8 columns.
2. Make sure the bots will have a place to start, to charge and to drop their blocks.
3. Create at least 1 blockade and 2 rooms.
4. Place some blocks into the rooms.
5. Create a color sequence which the bots will have to find.
6. Preview your map.
7. When finished, save your map.

Open your own map

1. Open your own map.
2. Add another room.
3. Save it again.

Create a random map

1. Open a new map.
2. Randomize the rooms.
3. Put random blocks into the rooms,
   we want to maximize the possible number of blocks per room and no blue ones.
4. Create a random sequence.

PART 2 (GROUP 3)

Create your own scenario and save it

1. Add two Human controlled bots.
2. Give them the colorblind handicap and give them a battery with capacity 50.
3. Add the mapfile you've just created to the Scenario.
4. Viisualize the paths and set collisions disabled.
5. Save the Scenario you've just created.

Open a saved scenario and export it

1. Open the scenario you've saved.
2. Add a Human controlled bot.
3. Give the bot the gripper handicap.
4. Save the Scenario.
5. Use the MAS export to generate a mas2g project.

Run your scenario

1. Find the first block of the sequence with bot1.
2. Tell bot2 were to find the block.
3. Let bot2 pick up the first block and bring it to the Dropzone.

This is the end of the assignments, if you have any questions left, feel free to ask us.

USABILITY EVALUATION - QUESTIONNAIRE

Name: .....

Date: .....

Please fill in the first 2 questions before you start with the assignments and the last 5 questions after finishing the assignments.

BEFORE

1. How much experience do you have with BW4T?

2. Are you familiar with the changes we made during our project?

AFTER

1. Was everything clear? If not, what was unclear?

2. What part of the system did you like the most?

3. Was there something about the system you didn't like? If so, tell us what you didn't like and why.

4. If you could change something, what would you change?

5. Do you have any other comments/improvements for us?