BW4Tv3 Manual

Blocks World for Teams

BW4T Context Project

Delft University of Technology

TU Delft Delft University of Technology

Challenge the future

BW4Tv3 Manual

Blocks World for Teams

by

BW4T Context Project

BW4Tv3 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

Contents

1

INTRODUCTION

This document is the user manual for the latest version of the Blocks Worlds for Teams (BW4Tv3) environment. In this manual you can find all the new changes and features in BW4T. In Chapter 2 we discuss the structure of BW4Tv3. Here the new structure is explained and we discuss the changes made to the server, why we chose for these changes and some concerns we have. We also discuss the changes between the old and new maps and the benefits and disadvantages these changes have and the backwards compatibility. In Chapter 3 we discuss the new features added to BW4Tv3 and how to use these features. Here it is explained how a user can create a new map, how the user can create scenarios and how the user can control a bot. In Chapter 4 we discuss the transferability of the BW4T environment. We first explain the refactoring that was done to the code, after which we discuss the new features.

2

STRUCTURE

2.1. NEW STRUCTURE

The BW4Tv3 system is composed of the following components:

- The server project, containing all files to run a server instance of BW4T.
- The client project, containing all files to run a client (to control one agent) on a running server.
- The core project, containing all shared resources and classes used by more than one project. The core project is not runnable on its own.
- The map editor project, containing all files needed to run the map editor as a standalone application.
- The scenario editor, containing all files for the scenario editor.

All of these systems can run independently, except for the core project. All subsystems need the core project files to be able to run. The client and server project are both part of a Client-Server architecture. All projects (apart from the core project) are based on the classical Model-View-Controller architecture. All classes are in packages belonging to one of these 3 sub-components.

2.2. CHANGES MADE TO THE SERVER

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 in 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 Pathfinding** There are two pathfinders, one that uses the map zones as navigation points, and the new one which uses the grid points.

2.2.1. JUSTIFICATION

BW4Tv3 introduces the ability to collide into other bots on the map. Before v3 there were no collision detection and handling methods. Due to that 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 will be 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 via B, 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, but rather the points on the grid. To clarify, the map seen when the server 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 infinitely many, and would complicate collision detection.

With the path planner created, the next issue was detecting if a bot was going to collide into another bot or not. The original specifications called for a bumped percept to be called after a bot collided into another bot. Since implementing this would take as much effort as detecting a collision after 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 feature, 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 BW4Tv3. It could be a helpful debugging aide, especially for 1st year students doing following the AI course, as it can be used to debug and visualize the actions of the robots.

A key improvement in BW4Tv3 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 (thus modifying the scenario loaded by the client). While this is in stride with the decision to fully seperate 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. To have to change the scenario and re-reload it on the server is extremely cumbersome. By simply checking a box collisions can be enabled, with the reverse being true as well.

2.2.2. CURRENT CONCERNS

At the moment there is an issue that prevents the start-up of bots on top of each other when collisions are enabled. Due to the complexity of writing a procedure that makes sure that every bot spawns on an unoccupied location, it had been decided that the bot collisions will be disabled as long as they share a grid point with another bot after they have spawned. However, when using GOAL, somewhere during the initialization of Repast one of the navigating robots is presumably being cloned. 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 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.

2.3. OLD MAPS VS NEW MAPS

The old map editor has been completely redesigned to what we now call the Environment Store. It offers the user the ability to create much bigger maps and place the different types of zones across the map according to a grouping that is preferred for a simulation by the user.

2.3.1. DIFFERENCES OLD AND NEW MAPS

The main difference in the maps that are generated by the new Environment Store is that the distribution of rooms and other zones on the map is no longer set through a standard algorithm but can be set by the user themselves. Through a grid of equally sized zones you can set each of these zones to being either a corridor, room, start zone, drop zone, blockade or charging zone. This way we allow the user to have a lot more variety in the maps they create and allow for the possibility to create a map that can more accurately simulate a real environment.

The old maps always used to have the start- and drop zone at the bottom, this is no longer a must in the new maps. Start- and drop zones, as well as any other type of room can be placed anywhere in the map. This is because the robot's starting position is not necessarily the place where the blocks are supposed to be dropped.

Also, you can now add two new types of zones called blockades and charging zones. The blockades serve to create a map with parts where the robot should not be able to pass. This way the user can create a maze or zones that should be difficult or 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.

2.3.2. EDITING SAVED MAPS

In the old map editor the user was never able to modify a previously created map, while in the new Environment Store they can. This allows the user to create a map, save different versions of it and edit it in a later stage when an error was spotted after the map had been saved.

The user can also open maps that were created using the old map editor. When the user does this, the map is automatically converted to be ready to edit in the Environment Store. This does mean that the large drop zone, which in these maps is located at the bottom of the map, is set to the standard size of the grid. Also it creates corridors in the zones left and right from the old drop zone which used to be non-navigable empty zones. Besides these minor changes the map will be the exact same as after saving it in the old map editor.

2.3.3. BENEFITS

The benefits of the new maps, that can be created in the Environment Store, have been listed below:

- The user is no longer bounded to a standard mapping of rooms.
- The user can now create a map that is up to 4 times the size of the largest possible map created in the old editor.
- The user can add blockades to the map.
- The user can add charging zones to the map.
- The user can save and open previously created maps.
- The user can randomize the positions of rooms, the blocks in a room and the target sequence separately using extra options.
- The number of entities that can spawn in a map is no longer bound in the new maps.

2.3.4. DISADVANTAGES

Although we have tried to improve all aspects, there are still minor disadvantages to the new Environment Store. Those are listed below:

- It generally takes more time to create a map in the new Environment Store.
- To set the number of bots that can spawn on a map to infinite, the bots will spawn on top of each other which could go unnoticed by the user.

2.3.5. CONSIDERATIONS

One thing we have considered is how we were going to spawn the bots in the new maps. We came to the conclusion that when physical realism has been turned on in the Scenario Editor, we would still like to be able to spawn as many bots as the user wants in the map. This is why we have chosen to create 4 spawn points per start zone, allowing us to spawn up to 4 bots of the largest size without letting them collide. A fifth bot would simply spawn on the same place as the first one after that one has moved. This should not be possible when physical realism has been turned on, therefore we have set the functionality that as long as

they spawn on top of each other they can move, but as soon as they have moved away from each other they should not be able to move over each other again. By doing this, the number of bots that can spawn in a map is not bounded by the number of start zones.

2.4. INIT PARAMETERS

The following MAS init parameters are available (the default values are shown inside the brackets)

- CLIENTIP("localhost"): Our IP. Passed to server so server is able to find us with this IP.
- CLIENTPORT("2000"): The port that we use.
- SERVERIP("localhost"): The IP address of the server.
- SERVERPORT("8000"): The Port used by the server.
- AGENTCOUNT("0"): Number of agents that we have.
- LAUNCHGUI("true"): Launch GUI for entities
- HUMANCOUNT("1"): Number of human bots
- AGENTCLASS("nl.tudelft.bw4t.client.agent.BW4TAgent"): The java agent class to load when new entities appear.
- GOAL("true"): are we connected with GOAL? This param should be auto detected, it will be set to false if the program is started from commandline.
- KILL(""): The key we should try to use to kill the remote server.
- CONFIGFILE(""): The file from which the client reads the configuration.
- GOALHUMAN("false"): Forces the use of the human GUI with an GOAL agent to translate the commands
- MAP(""): Name or filename of the map to use
- LOG("OFF"): The log4j log level to be used. Available values: OFF, FATAL, ERROR, WARN, INFODEBUG and ALL.

3

NEW FEATURES

3.1. ENVIRONMENT STORE

3.1.1. SIZE DIALOG

At start-up, a small GUI is shown that prompts the user to enter the map dimensions of the map has that they want to create. Figure 3.1 displays the size dialog.

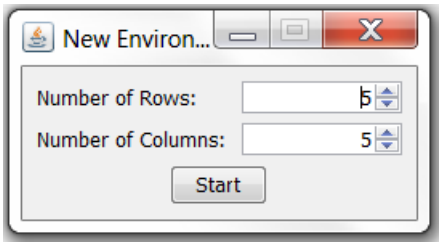


Figure 3.1: The size dialog

3.1.2. MAP EDITOR

After specifying the amount of rows and columns the map should have and clicking on the Start button, the map editor is launched. 3.2 shows the environment store after launch, with a 5x5 map. The map editor has

Figure 3.2: The map editor with a 5x5 freshly created map

many functionalities. The functionalities are described bit by bit in this documentation.

3.1.3. EDITING A ROOM

Right click on any corridor or zone, and the drop down menu in 3.3 is shown. This allows zones to be con-

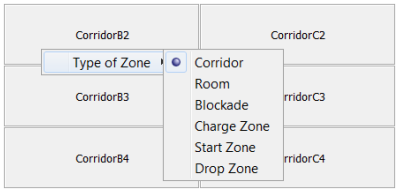


Figure 3.3: The drop down menu generated when a zone is right clicked

verted to any of the zones available in the system (charge zones, blockades, rooms, etc.). Figure 3.4 shows the different zone appearances in the map editor. As can be seen in 3.4, the room and drop zone contain a green

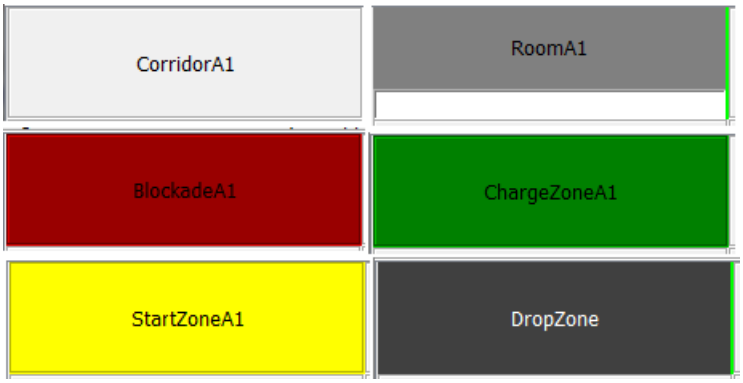


Figure 3.4: The color palette at the bottom of the map editor

border. This is to indicate which way the door is faced. This is also editable. When right clicking on a certain room, the drop down menu in 3.5 pops up. That drop down menu lists the directions the door can face while

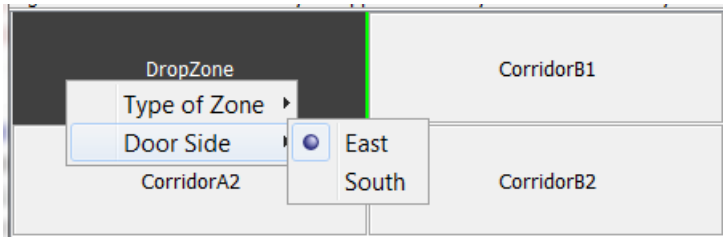


Figure 3.5: The drop down menu generated when a room is right clicked

still being accessible. In the figure, as it is the top left zone, only doors facing east or south are possible.

3.1.4. EDITING A SEQUENCE

Located below the map editor, there is a text field which contains the color sequence of the blocks to be added to the map. The same text field is also present in the rooms that aren't drop zones. The text fields are given in 3.6. The text field in the rooms can be used to enter colors, which are then used for the blocks that are placed in that room. The text field under the zones can be used to enter the colors that make up the color sequence of blocks to be delivered. To add colors, the color palette is useful.

3.1.5. COLOR PALETTE

At the bottom of the map editor is the color palette. A larger picture is given in 3.7. The color palette is used for adding blocks to rooms and the sequence to be completed. When clicking on the box containing either the sequence of the map or the blocks in the room, the color palette is shown. It allows the user to add colors by clicking on the respective colors on the palette, but colors can also be added by entering the respective

Figure 3.6: The rectangle at the bottom of both the map editor and the rooms

Figure 3.7: The color palette at the bottom of the map editor

numbers in the colors (e.g. pressing 1 adds a red block to the room or sequence) and also by entering the first letter of the color (e.g. pressing R adds a red block).

3.1.6. FILE MENU

The file menu does not only contain the standard operations for exporting a created map so that the map can be used later, but also a few other options. The drop down menu is as pictured in 3.8. The New, Open..., Save

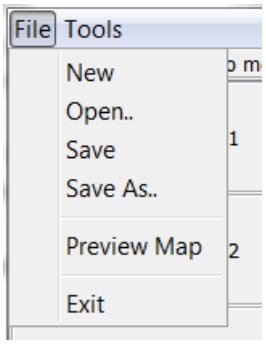


Figure 3.8: The drop down menu created when the 'File' menu item is clicked

and Save as... options are the standard file operations and need no further explanation. However, as added functionality to the options, it is impossible to save a map not containing either a start- or drop zone, as well as a map not containing a block in its sequence, and saving an unsolvable map will give a warning describing what is wrong. The Exit option also needs no further explanation. The Preview option gives a preview of the created map, as how it would be rendered when used as an actual map for bots. For a non-edited 5x5 map, the preview looks as pictured in 3.9.

3.1.7. TOOLS MENU

The other drop down menu at the top of the map editor is the tools menu. This menu contains options to use several tools. The drop down menu is given in 3.10. Clicking on the 'Randomize Zones' option creates a random map, based off of a vanilla map with reclassification of rooms to blockades, charge zones or corridors, and with reclassifications of existing corridors to charge zones. The map is always solvable, given that all of the blocks in the sequence are present in the map and no further changes are made to the map. The 'Randomize Blocks' option randomizes the blocks contained in every room according to parameters that the user has to specify. The interface for specifying the parameters is given in 3.11. This interface allows the user to specify which block colors they want to be spawned, and also the maximum amount of blocks in a room. The interface for random sequence generation is slightly different, and is as pictured in 3.12. The differences between this interface and the previous interface are the Randomize button and Result label, that create and display respectively the random sequence, and the apply and cancel button which enter the sequence in the map or cancel the creation of the sequence.

3.2. SCENARIO EDITOR

When the editor, which can be found in figure 3.13, is opened, two main parts can be seen:

1. The Configuration panel, which is used to create, open and save configuration files.

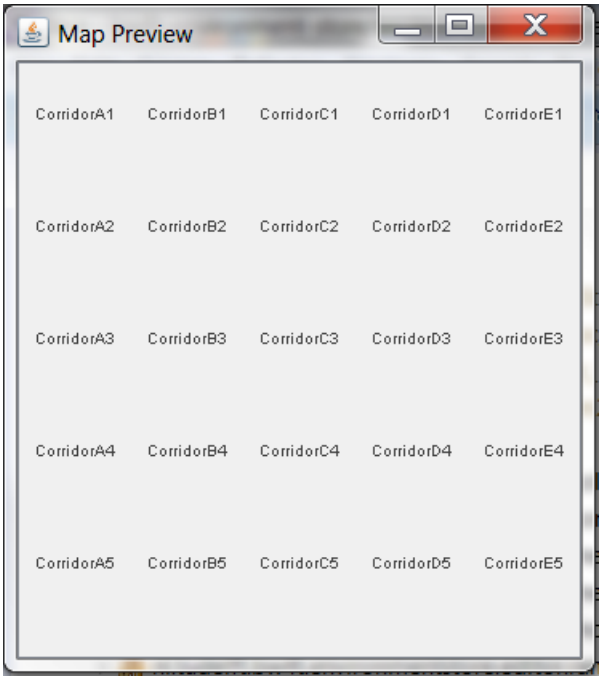


Figure 3.9: The preview of a 5x5 non-edited map, with no blocks in the sequence

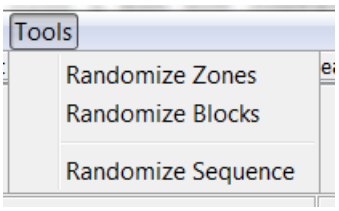


Figure 3.10: The drop down menu created when the 'Tools' menu item is clicked

2. The Entity panel, where a list of bots and a list of e-partners are being displayed. Here, bots and e-partners can be created, modified, renamed and deleted.

3.2.1. GENERAL USE

At the top of the editor a *File* menu can be seen. If that is clicked, the options to create a new configuration, open a configuration, save the current configuration, export the current configuration and to exit the editor will be visible.

On the left side of the editor, a scenario can be configured. The Client IP, the Client Port, the Server IP and the Server Port should already contain the default values. These can be changed if needed. It is also possible to indicate whether or not a GUI needs to be opened, if the paths the bots take need to be visualised and if collisions need to be enabled. Map files can also be imported by selecting the *Open file* button at the right section.

On the right side of the editor bots and e-partners can be created, modified, renamed and deleted. There also are a list of bots and a list of e-partners that are currently created, and below each list there is an indication of how many bots or e-partners are created in total.

Linked to the Scenario Editor are the Bot Store and the E-partner Store, which will also be discussed in this document.

3.2.2. CONFIGURATION PANEL

A picture of the configuration panel can be found in figure 3.14.

New configuration file To create a new configuration file, select *File* → *New* in the menu bar. A new con-

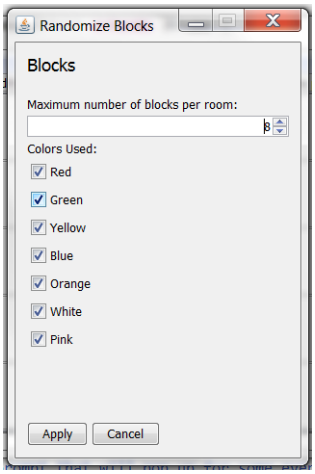


Figure 3.11: The menu appearing when the randomize blocks option is clicked in the menu

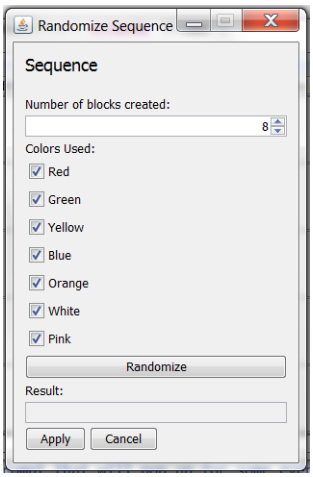


Figure 3.12: The menu appearing when the randomize blocks option is clicked in the menu

figuration with the default values will be created. Creating a new configuration will reset all previous changes made to the configuration, so make sure the current configuration is saved.

Open configuration file To open an existing configuration file, select $File \rightarrow Open$ in the menu bar. A window will pop up where the folder can be selected in which the configuration file is saved. Once the right folder is selected, select the file and click the *Open* button.

Save configuration file To save the current configuration file, select $File \rightarrow Save$ in the menu bar. If the current configuration hasn't been saved before, a window will pop up where the folder can be selected to save the configuration file to. Enter a name for the file and click the *Save* button.

Save configuration file as To save the current configuration in a new file, select $File \rightarrow Save\ As$ in the menu bar. A window will pop up where the folder can be selected to save the configuration file to. Enter a name for the file and click the *Save* button.

Export configuration to mas2g To export the current configuration to mas2g, the configuration has be first. Once that is done, select $File \rightarrow Export\ as\ MAS\ project$ in the menu bar. A window will pop up where the folder to which the configuration will be exported can be selected. Enter the file name and click the $Export\ MAS\ project$ button.

Exit the Scenario Editor If the Scenario Editor needs to be closed, select $File \rightarrow Exit$ in the menu bar. Clos-

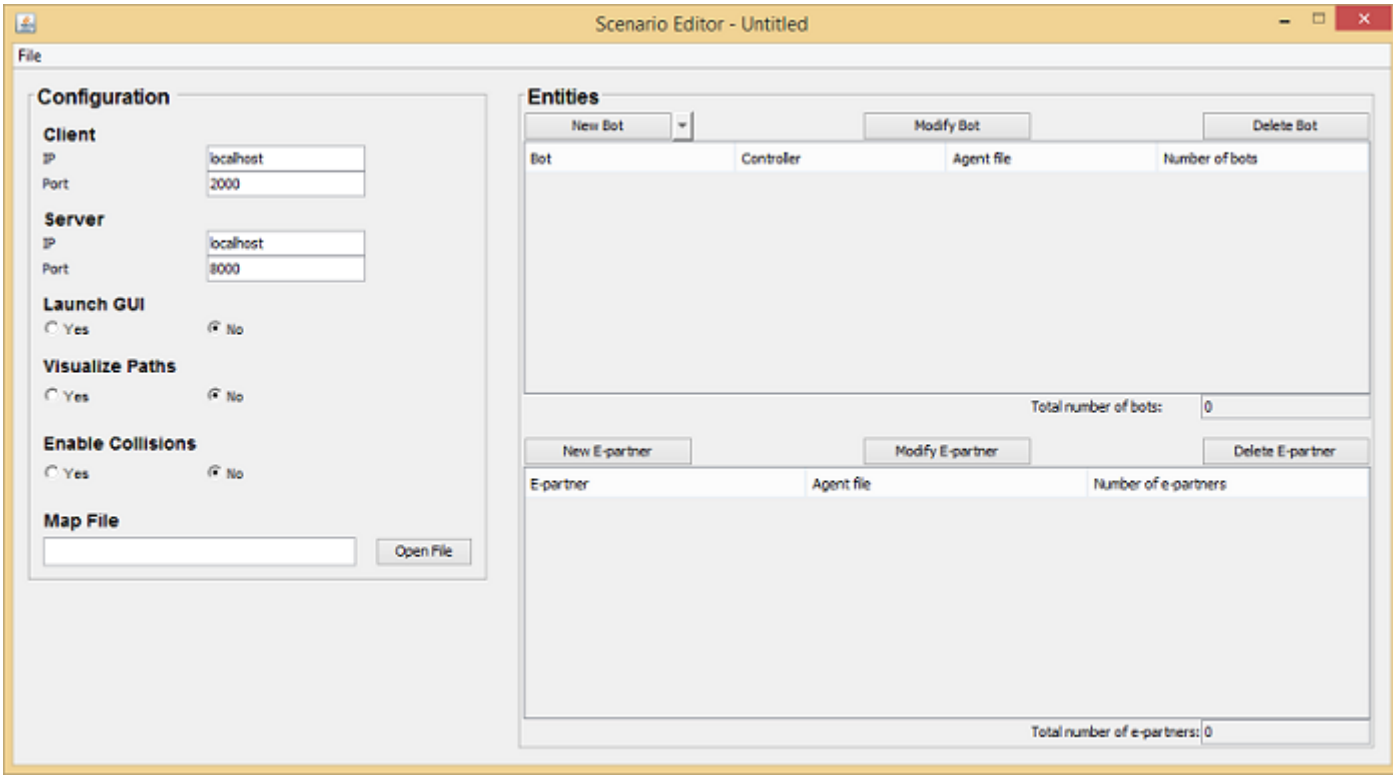


Figure 3.13: Scenario Editor

ing the Scenario Editor will not save any changes that have been made, so make sure the current configuration is saved first.

3.2.3. ENTITY PANEL

The entity panel exists of 2 parts; the top part shows the bot options and the bottom part shows the e-partner options. A picture of the entity panel can be found in figure 3.15.

Create new bot To create a new bot, click the *New Bot* button. A new bot will appear in the bot list.
If a standard bot needs to be created (which already has default values), click on the arrow next to the *New Bot* button. A menu will drop down where one of the standard bots can be selected. If one of the standard bots is clicked, the bot will appear in the bot list.

Modify bot To modify a bot, select the bot that needs to be modified in the bot list and click the *Modify Bot* button. The Bot Store will open in a new window where the properties of the bot can be modified.

Rename bot To rename a bot, either the *Modify Bot* button can be used to rename in the Bot Store, or the bot can be renamed in the Scenario Editor. To rename in the Scenario Editor, select the bot that needs to be renamed in the list with bots. Double click its name and enter a new name.

Change controller type To change how a bot is controlled, either the *Modify Bot* button can be used to change it in the Bot Store, or it can be changed in the Scenario Editor. To change the controller type in the Scenario Editor, select the bot that needs to be changed and click on its controller type. The controller type can now be selected.

Change the amount of entities of a bot To change how many entities of a type of bot are created, either use the *Modify Bot* button can be used to change it in the Bot Store, or it can be changed in the Scenario Editor. To change the entity amount in the Scenario Editor, select the bot that needs its amount changed.

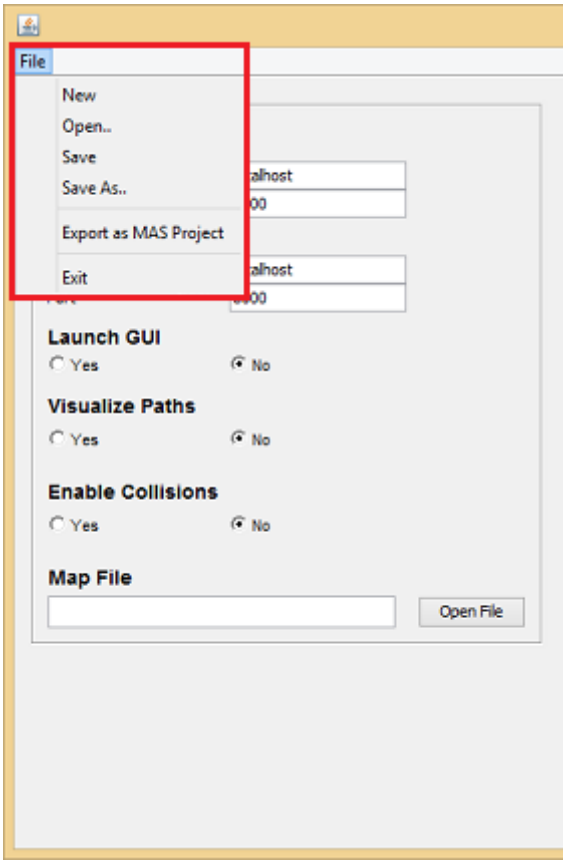


Figure 3.14: Configuration Panel

Now double click the amount given, and enter a new number.

Delete bot To delete a bot, select the bot that needs to be deleted in the list with bots and click the *Delete Bot* button.

Create new e-partner To create a new e-partner, click the *New E-partner* button. A new e-partner will appear in the e-partner list.

Modify e-partner To modify an e-partner, select the e-partner that needs to be modified in the e-partner list and click the *Modify E-partner* button. The E-partner Store will open in a new window where the properties of the e-partner can be modified.

Rename e-partner To rename an e-partner, either the *Modify E−partner* button can be used to rename in the E-partner Store, or it can be renamed in the Scenario Editor. To rename in the Scenario Editor, select the e-partner that needs to be renamed in the list with e-partners. Double click its name and enter a new name.

Change the amount of entities of an e-partner To change how many entities of a type of e-partner are created, either use the *Modify E−partner* button can be used to change it in the E-partner Store, or it can be changed in the Scenario Editor. To change the entity amount in the Scenario Editor, select the e-partner that needs its amount changed. Now double click the amount given, and enter a new number.

Delete e-partner To delete an e-partner, select the e-partner that needs to be deleted in the e-partner list and click the *Delete E-partner* button.

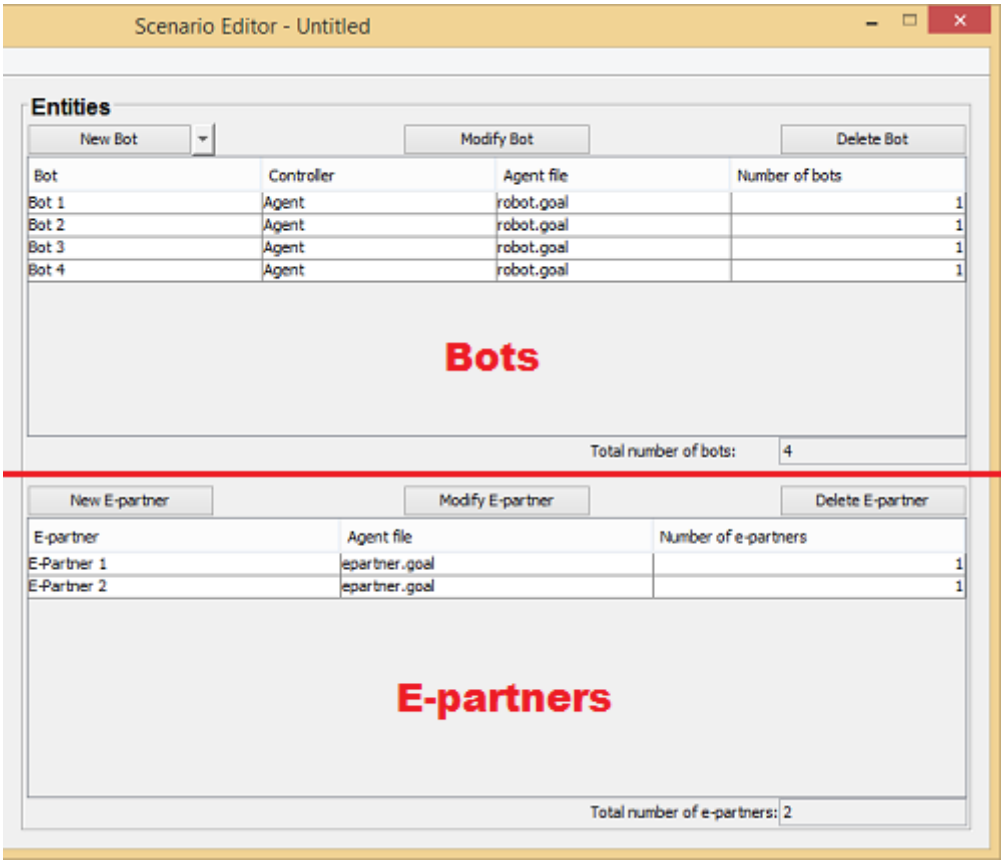


Figure 3.15: Entity Panel

3.2.4. BOT STORE

A picture of the Bot Store can be found in figure 3.16.

Rename bot To rename a bot, click on the field that contains its current name. Now the current name is able to be edited or a new name can be entered.

Change controller type To change how a bot is controlled, click its current controller type. Now the controller type can be selected.

Change the amount of entities To change how many entities of the bot need to be created, the field that contains the current number of entities should be clicked. Now the amount is able to be edited.

GOAL options The GOAL options for the bot can be added under the *GOAL options* section. Here it can be indicated what its reference name in GOAL should be, and a GOAL agent file can be selected which will control the bot. To select the GOAL agent file, click on the *Use existing GOAL file* button. A window will pop up where the folder in which the GOAL file is saved to can be selected. Once the right folder is selected, select the file and click the *Open* button.

Bot properties The available bot properties can be found under the *Properties* section. To change what properties the bot needs to have, select or deselect the checkbox next to the property description. Once a property has been enabled, use the sliders to change the value of that property.

Save modifications If the changes that have been made to the bot need to be changed, click on the *Save* button. The Bot Store window will close, and the changes will have been saved.

Figure 3.16: Bot Store

Reset modifications If the changes that have been made to the bot need to be reset, click on the *Reset* button. The changes will have been reverted to the last saved configuration.

Cancel modifications If modifying the bot needs to be canceled without saving any changes, click on the *Cancel* button. The Bot Store window will close.

3.2.5. E-PARTNER STORE

A picture of the E-partner Store can be found in figure 3.17.

Rename e-partner To rename an e-partner, click on the field that contains its current name. Now the current name is able to be edited or a new name can be entered.

Change the amount of entities To change how many entities of the e-partner will be created, the field that contains the current number of entities should be clicked. Now the amount is able to be edited.

GOAL options The GOAL options for the e-partner can be added under the *GOAL options* section. Here it can be indicated what its reference name in GOAL should be, and a GOAL agent file can be selected which will control the bot. To select the GOAL agent file, click on the *Use existing GOAL file* button. A window will pop up where the folder in which the GOAL file is saved to can be selected. Once the right folder is selected, select the file and click the *Open* button.

E-partner properties The available e-partner properties can be found under the *Properties* section. To change what properties the e-partner needs to have, select or deselect the checkbox next to the property description.

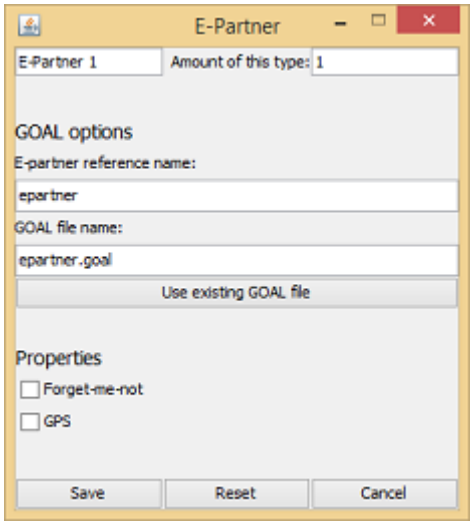


Figure 3.17: E-partner Store

Save modifications If the changes that have been made to the e-partner need to be changed, click on the *Save* button. The E-partner Store window will close, and the changes will have been saved.

Reset modifications If the changes that have been made to the e-partner need to be reset, click on the *Reset* button. The changes will have been reverted to the last saved configuration.

Cancel modifications If modifying the e-partner needs to be canceled without saving any changes, click on the *Cancel* button. The E-partner Store window will close.

3.2.6. HOW TO ADD CAPABILITIES TO ROBOTS

Looking at a scenario where a developer would like to add an extra functionality to the BW4T robots, he or she will have to undergo a certain set of steps in order to accomplish this task.
The first step is the creation of this new capability in the system itself. We have set up a decorator pattern in the "robots" package located in the Server project, which can be used to easily add new features to robots. The only thing that needs to be done is adding an extra class that will extend the AbstractRobotDecorator. It is then rather simple to add a new capability.
Let us say we want to add the ability to "go through walls", the navigation method will have to be overridden so that the robot can also go through the walls in the environment.

Now we have to think of how to create new robots with the new capability that was just added. In the Scenario Editor project, we can find the Scenario Editor graphical user interface which allows us to create new robots, e-Partners, and modify their functionalities.
The Scenario Editor leads us to the BotStore, once we choose to modify a robot: this is were the changes should happen. Suppose we take our example from above: we want a robot which can go through walls. The simplest solution would be to add a JCheckbox in the BotEditor, among the other JCheckboxes which define the different robot's abilities.
The Scenario Editor needs to receive the new command to "create" a new robot which can go through walls. That is why the information from the BotStore is stored into a model class called BotConfig, located in the Core. In this class, we would have to add a method that can check whether we have chosen to make a robot with this new capability. This class can be directly accessed (and updated) from the controller that belongs to the BotEditor.

3.2.7. HOW TO ADD CAPABILITIES TO E-PARTNERS

Adding a new capability to an e-Partner also involves changing various components in the Server and Core. We do not have a decorator pattern for e-Partners because we only had two functionalities so far.

Suppose we want our e-Partner to tell us the time it took a robot to go to a certain location. What we want to do is modify the e-Partner's percept, for when it has this capability, it should be able to calculate the amount of ticks its holder took to go to a certain zone. These changes should be made in the EPartnerEntity class, located in the EIS part of the Server.

In order for the user to be able to create an e-Partner with this functionality, changes need to be made in the Scenario Editor. Once we press on the "Modify" button when we want to modify an e-Partner, we are directed to a rather small frame which allows us to check which functionalities we want to be added to our e-Partner. What we need to do in order to make it possible for us to choose the "stopwatch" ability is to add an extra checkbox.
As we explained for adding capabilities to robots, the Scenario Editor needs to know what kind of e-Partner to instantiate. That is possible by using the EPartnerConfig class, in the Core project. The possibility to have an e-Partner with a stopwatch ability needs to be added to this class.

3.3. HUMAN PLAYER GUI

When the interface, which you can find in 3.18, is opened, two main parts can be seen:

1. The map panel, where the map and the block sequence are being displayed.
2. The message panel, where the remaining bot battery and messages between bots and e-partner are being displayed. Messages can be be sent to other bots and your e-partner.

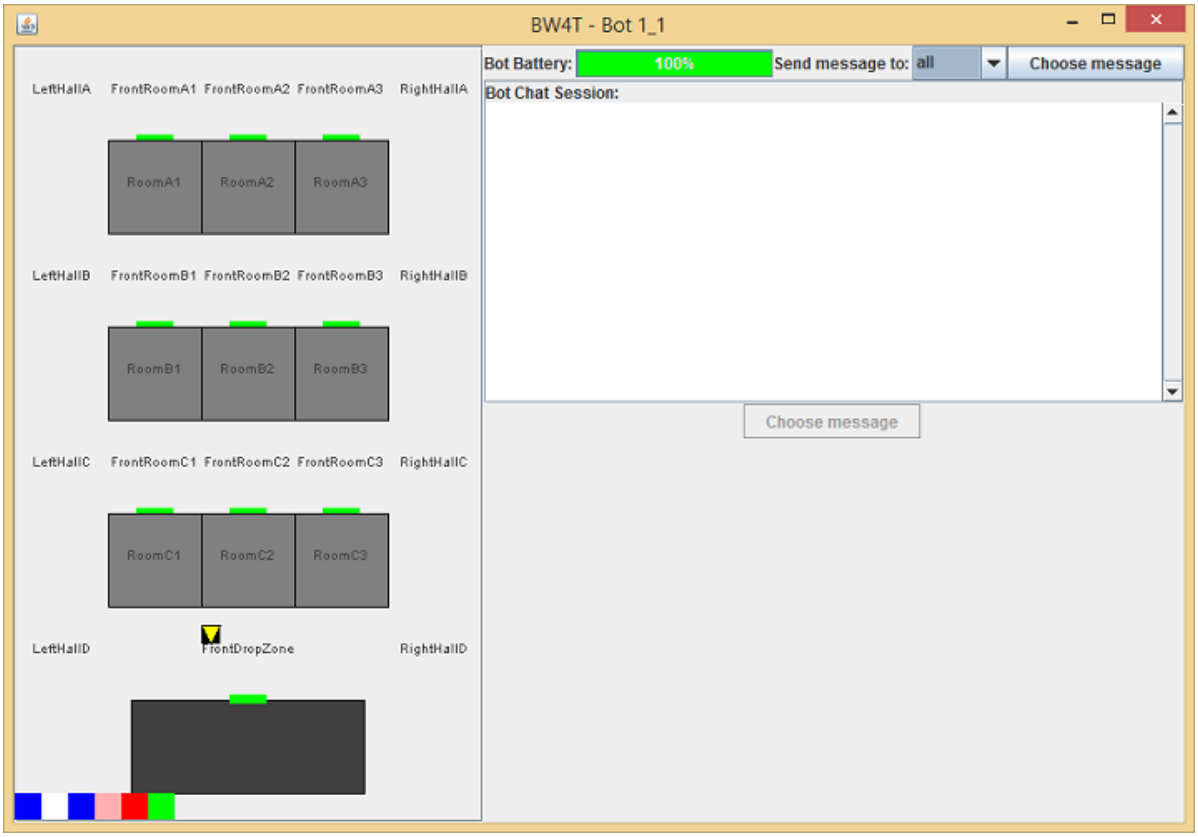


Figure 3.18: Human Player GUI

3.3.1. GENERAL USE

On the left side of the interface, the bot can be directed by clicking on the map. Different pop-up menus will appear upon clicking on different entities on the map.

On the right side of the interface, the battery capacity of the bot that can be controlled can be found. Bots can be selected to which messages need to be sent and messages can be sent here. The bot chat session and the e-partner chat session are also displayed.

3.3.2. MAP PANEL

A picture of the Map Panel can be found in figure 3.19.

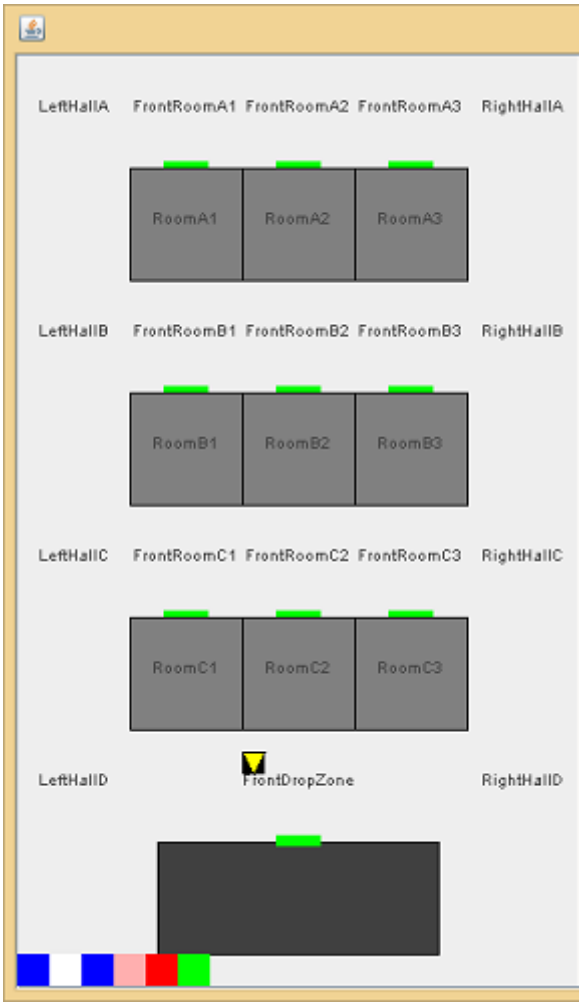


Figure 3.19: Map Panel

Go to: "*place*" To command the controlled bot to go to a room or a certain place in the corridors, click on the room or place where the bot needs to go to. The corridor menu (figure 3.20) will appear next to your mouse pointer with the option *go to here* or *Go to* "*room*".
Send message: I am waiting outside "*room*" To send a message to a bot or to all bots, click in the corridor. The corridor menu (figure 3.20) will appear next to your mouse pointer with the options *I am waiting outside* "*room*". A message will be send to all bots by default. To see how to change the receiver of the messages, go to the Message Panel section.
Send message: I am in "*room*" To send a message to a bot or to all bots, click in the room. The room menu (figure 3.21) will appear next to your mouse pointer with the option: I am in "*room*". A message will be send to all bots by default. To see how to change the receiver of the messages, go to the Message Panel section.
Send message: tell other bot(s) "*information*" To send a message to a bot or to all bots with certain information about a certain room, click on the room. The room menu (figure 3.21) will appear next to your

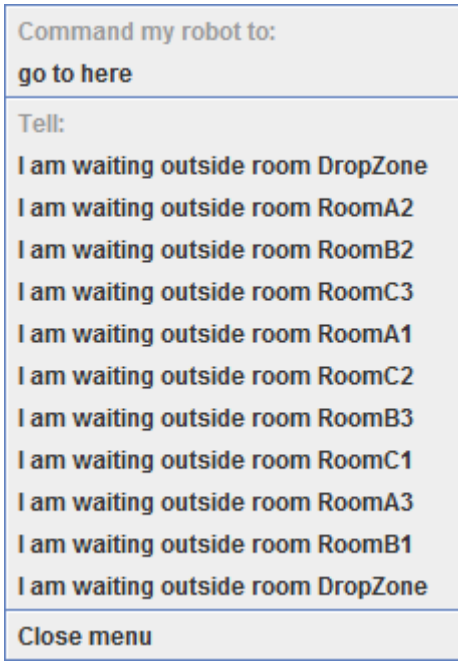


Figure 3.20: Corridor menu

mouse pointer with the messages you can send. A message will be send to all bots by default. To see how to change the receiver of the messages, go to the Message Panel section.
Send message: ask other bot(s) "*question*" **about certain room** To ask a bot or all bots a question about a certain room, click on the room. The room menu (figure 3.21) will appear next to your mouse pointer with the possible questions that can we asked. A message will be send to all bots by default. To see how to change the receiver of the messages, go to the Message Panel section.
Pick up block To pick up a "*block*", click on the block that needs to be picked up. The block menu (figure **??**) will appear next to your mouse pointer. Click on *Go to* "*color*" *block*. When the controlled bot is standing on the block, click on the bot. The block menu when standing on block (figure 3.23) will appear. Click on *Pick up* "*color*" *block*.
Drop block To drop the block that is currently being held, click on the room in which it needs to be dropped. The room menu when holding block (figure 3.24) will appear next to your mouse pointer. Click on *Put down block*. Do note that the controlled bot should be in the same room as where you want to drop the block.
Pick up e-partner To pick up an e-partner, click on the e-partner that needs to be picked up. The e-partner menu (figure 3.25) will appear next to your mouse pointer. Click on *Go to e−partner*. When the controlled bot is standing on the e-partner, click on the e-partner again. The e-partner menu when standing on e-partner (figure 3.26) will appear. Click on *Pick up e−partner*.
Send message to e-partner To send a message to the e-partner that is currently being held, click on the e-partner. The e-partner menu when holding e-partner (figure 3.27) will appear next to your mouse pointer. Click on *I am going to* "*room*". This option will only appear if the e-partner has GPS enabled.
Drop e-partner To drop the e-partner which is currently being held, click on the e-partner. The e-partner menu when holding e-partner (figure 3.27) will appear next to your mouse pointer. Click on *Put down e−partner*.

3.3.3. THE MESSAGE PANEL

A picture of the Message Panel can be found in figure 3.28. The second choose message button will only be enabled when the bot is holding an e-partner. Below this button, the e-partner chat session will appear when the bot holds an e-partner for the first time. Figure 3.29 shows the Message Panel when the e-partner has been held and dropped.
Select message receiver To select who a message needs to be sent to, click on the dropdown box (Figure

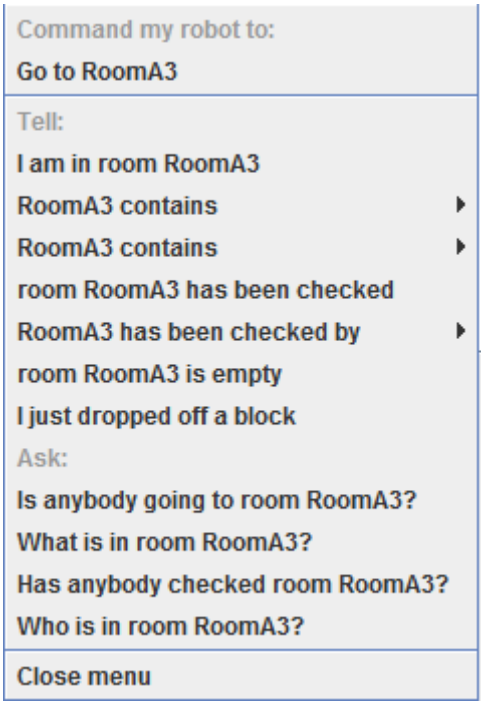


Figure 3.21: Room menu

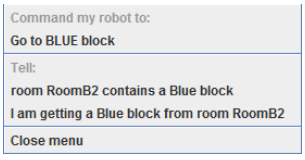


Figure 3.22: Block menu

Figure 3.23: Block menu when standing on block

3.30) at *Send message to*:. The receiver is by default *all*. When set to *all*, all other bots receive the message. When set to a specific bot, only that bot receives the message.
Send message to bot(s) To send a message, click on the *Choose message* button. A menu (figure 3.31) will appear next to your mouse pointer with the possible messages to be sent.
Answer a question To answer questions asked by other bots, click in the bot chat session box. A menu (figure 3.32) will appear next to your mouse pointer with the possible answers to be sent.
Send message to e-partner To send a message to the e-partner that is currently being held, click on the choose message button. A menu (figure 3.27) will appear next to your mouse pointer. Click on *I am going to* "*room*". This option will only appear if the e-partner has GPS enabled.
Drop e-partner To drop the e-partner that is currently being held, click on the choose message button. A menu (figure 3.27) will appear next to your mouse pointer. Click on *Drop e – partner*. The choose message button will be disabled after dropping the e-partner.

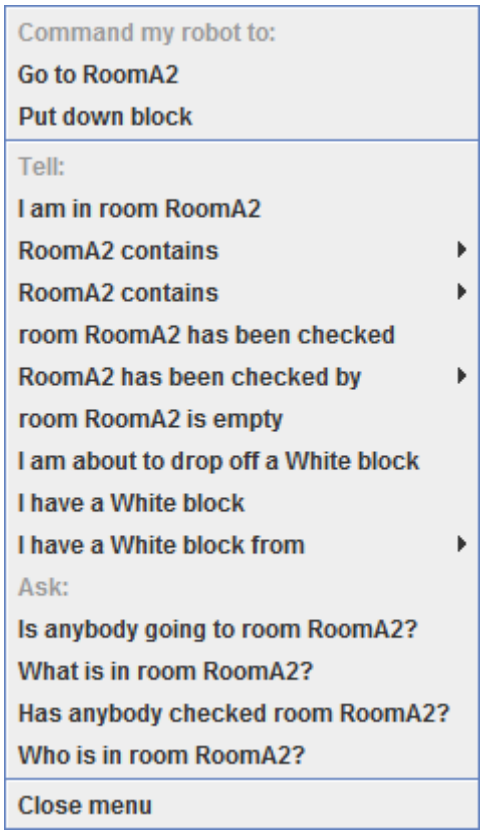


Figure 3.24: Room menu when holding block

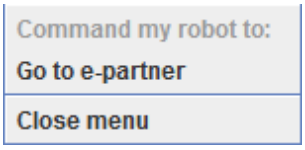


Figure 3.25: The e-partner menu

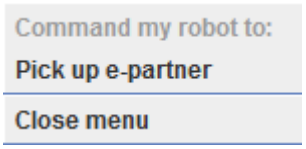


Figure 3.26: E-partner menu when standing on e-partner

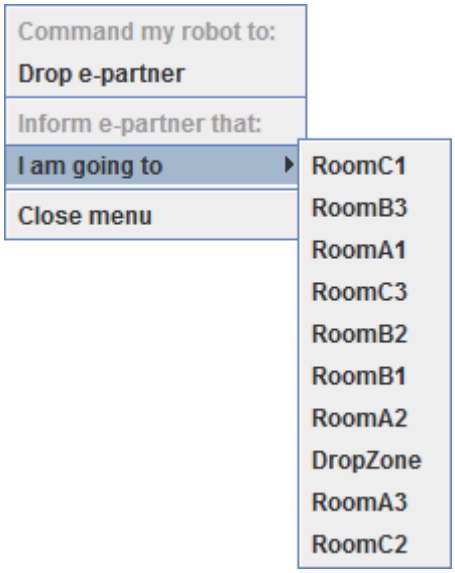


Figure 3.27: E-partner menu when holding e-partner

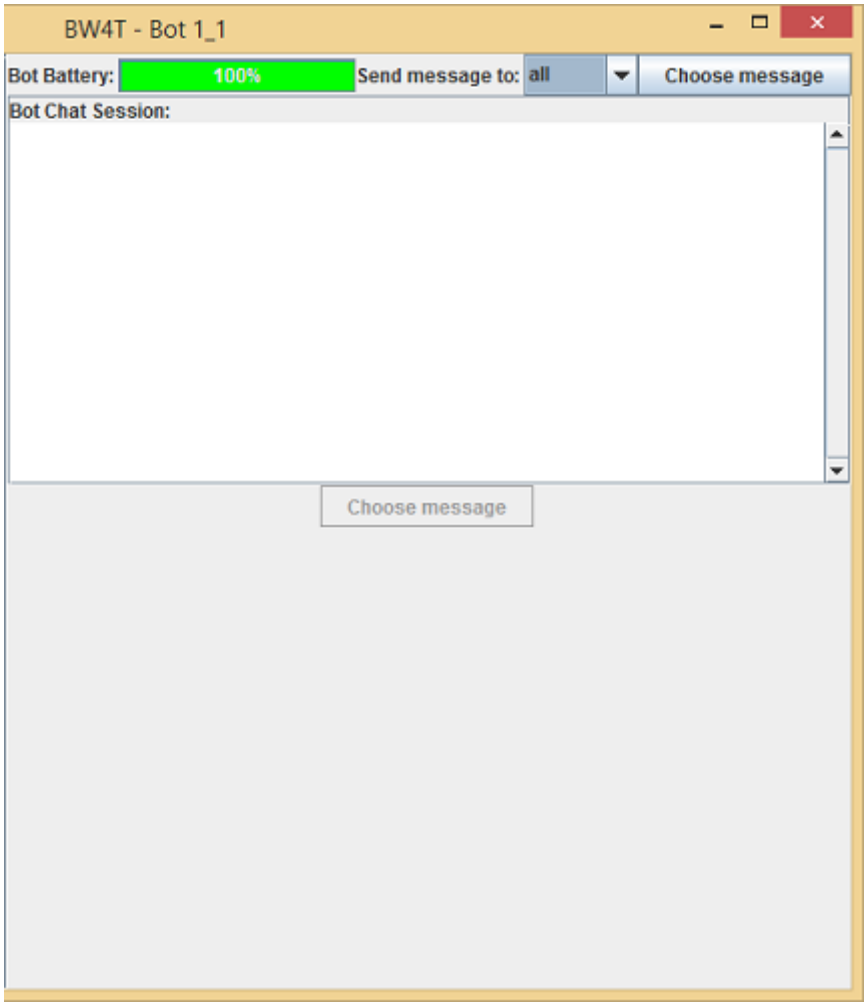


Figure 3.28: Message Panel

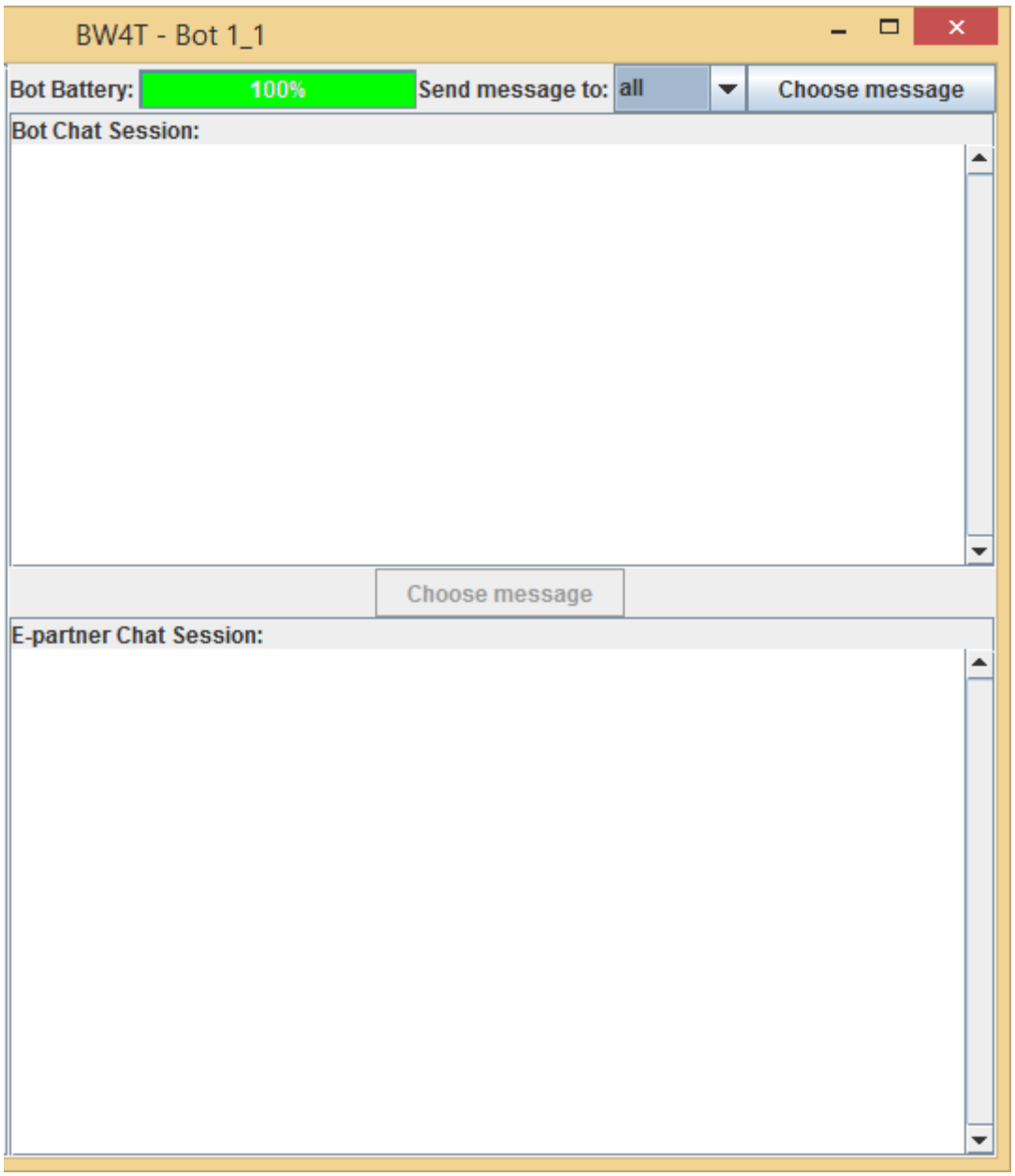


Figure 3.29: Message Panel with e-partner section

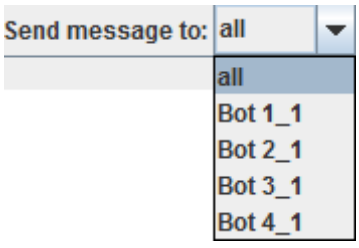


Figure 3.30: Select receiver dropdown

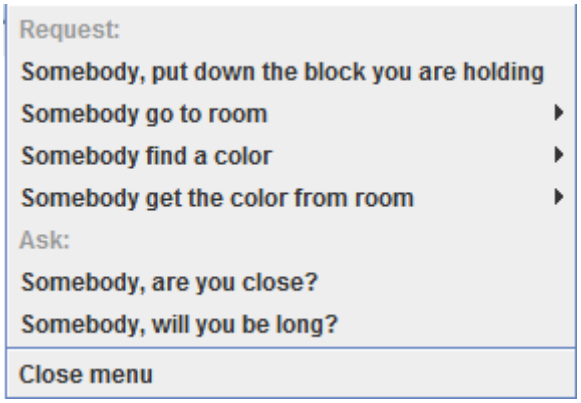


Figure 3.31: Choose message button menu

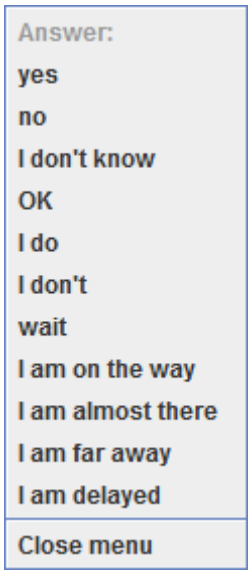


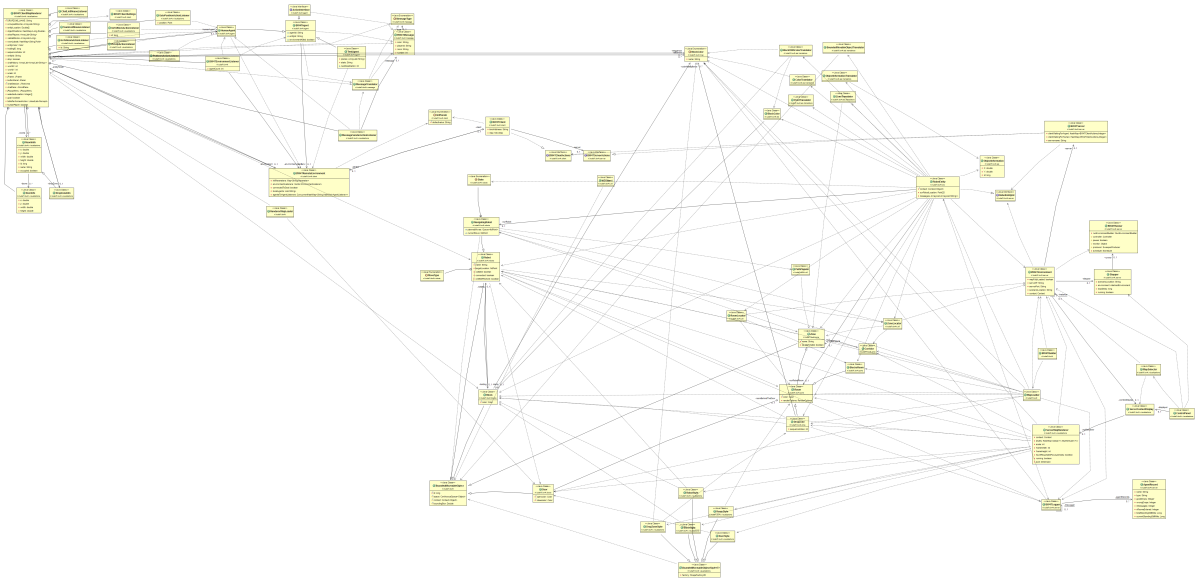
Figure 3.32: Bot answer menu

4

TRANSFERABILITY

4.1. REFACTOR

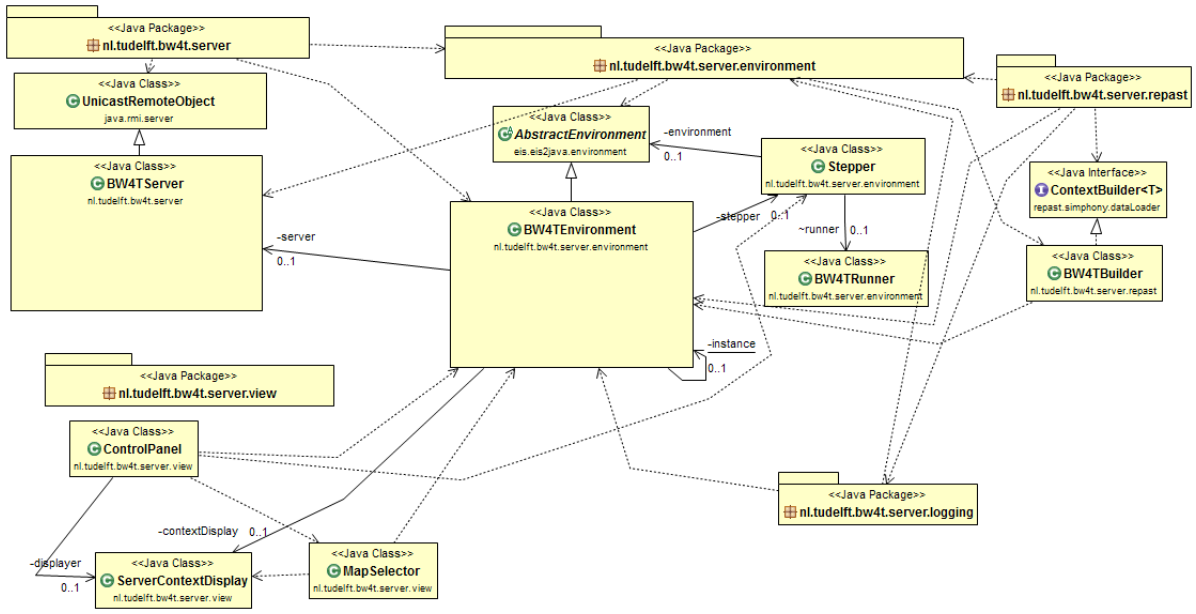
4.1.1. HOW IT WAS



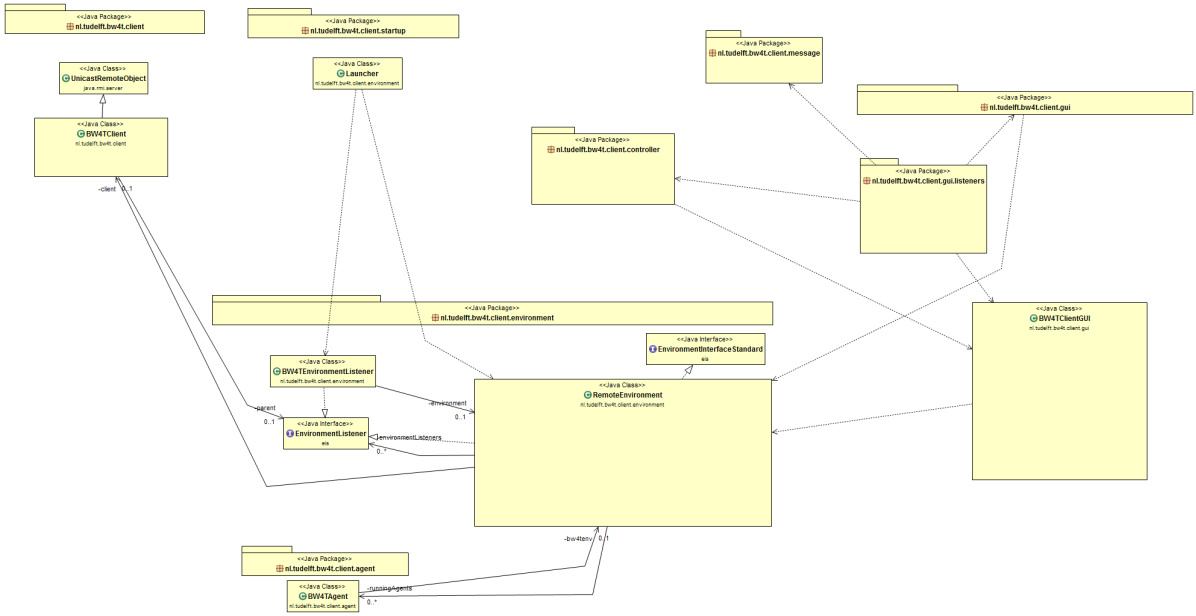
As can be seen in the UML diagram above, many classes were connected, Documentation was inadequate and no tests were written. These factors made it very hard for us to understand and debug the code base. It took quite some time to find out what a piece of code exactly did, what function a method had, what the difference between methods and classes was and what the connection between pieces of code was. After some weeks of exploring the code base and the UML diagrams belonging to the code base, we found out that there was no clear structure at all. There was a client and a server, but all communication went through the BW4TEnvironment class. So the first thing we did, was deciding how to split up the code, such that we got a clear Client-Server Architecture.
Take a look at the attachments as well for some examples of the old code. In the old system, Repast was used. Problem was that many functions in Repast were custom written as well and the entire Repast package needed to be installed. This caused the system to grow very large (500 MB for Repast only).

4.1.2. WHAT WE ACHIEVED

Server



Client



The customer wanted a clear structure to be added to the code base. Besides that, the code had to be rewritten so that it would be easier to adjust and maintain the code. Adding new features should than automatically become easier as well.

We decided to put classes which the client needs and the server does not need into the client package. This does not cause any errors in the server, and anything needed by the client is present. Afterward, we did the same for the server. All classes needed by the server and not by the client are put in the server package. The classes that are needed by both, we put in the core package. The core can be seen as a library, it can be used by both the server and the client when needed. For the client and the server, simplified diagrams are shown above. We chose to use Maven as an outline for the project, because Maven has a perfect way of handling those parts and the dependencies between them. Other dependencies such as EIS and Repast can be put in the resources folder and Maven takes care of the rest.

After everything was set up, we took a deeper look at the code. Trying to figure out what could be replaced by Repast was a big task that, for now, unfortunately led to nothing. Repast is a big bunch of interfaces which can guide your simulation of agents. Unfortunately, there were some problems with the connection to GOAL. GOAL can be paused and resumed whenever the user wants to, but Repast will continue to run, which will lead to false simulations and results. However because we only had 5 weeks to refactor the entire

code base, we decided to leave Repast as it is for now. The chance of breaking the project was too big, as we had to deliver a fully working product in only 5 weeks.
But we did manage to get the Repast functions more efficient. In combination with Maven, we pulled Repast apart and only added the dependencies that are really needed. Now the Repast functionality used is only 8 MB. Besides that, we made sure the newest version of Repast is used. So we started looking at what could be refactored. We created our own UML class diagram, and got really shocked about all connections and tried to figure out what the god classes are and how we could make them smaller. We immediately saw that BW4TRemoteEnvironment was the hugest class in the project, so that had to be refactored. BW4TClientMapRenderer was the second god class we handled. We tried to pick pieces of code that belong together and diffused it to a different class.
Then we found the custom BW4TLogger, a logger implemented to print anything the bot does. We thought that it would be very practical to export the logging to Log4j (an apache library). Now two kinds of logs are made, one kind that prints all the bot's actions to the log file and the other prints the debug issues in the code to the console. Now it can easily be adapted, if anything new from a bot is wanting to be logged, you just log it to BOTLOG level and it appears in the log file.

4.1.3. ATTACHMENTS

This section contains some examples of the old code base that we found. It were examples of what we wanted to adjust and make sure it never happens again.
Empty methods.

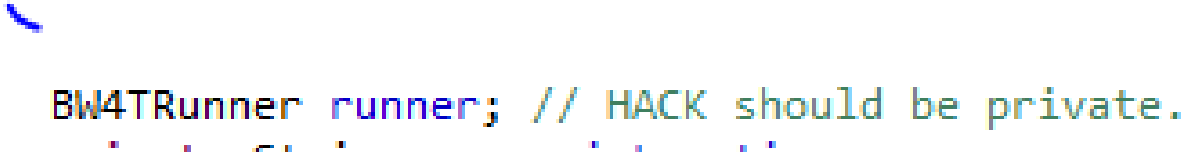
```
public void execute(RunState toExecuteOn) {
    // required AbstractRunner stub. We will control the
    // schedule directly.
}
```

Two methods that look alike, but the difference is nowhere to be found .

```
/**
 * Free an agent on the server
 *
 * @param agent
 *            , the agent to free
 * @throws RemoteException
 *             , if an exception occurs during the execution of a remote
 *             object call
 * @throws RelationException
 *             , if an attempt to manipulate the agents-entities-relation
 *             has failed.
 */
public void freeAgent(String agent) throws RemoteException,
        RelationException {
    server.freeAgent(agent);
}
```

```
/**
 * Unregister an agent on the server
 *
 * @param agent
 *            , the agent to unregister
 * @throws AgentException
 *             , if an attempt to register or unregister an agent has
 *             failed.
 * @throws RemoteException
 *             , if an exception occurs during the execution of a remote
 *             object call
 */
public void unregisterAgent(String agent) throws AgentException,
        RemoteException {
    server.unregisterAgent(agent);
}
```

Hacks



Commented code.

```
public void init(Map<String, Parameter> parameters)
        throws ManagementException {
    // System.out.println("re-initializing repast simulator");
    setState(EnvironmentState.INITIALIZING);
    takeDownSimulation();

    Parameter map = parameters.get("map");
    if (map != null) {
        mapName = ((Identifier) map).getValue();
    }
    try {
        launchRepast();
    } catch (IOException e) {
        throw new ManagementException("launch of Repast failed", e);
    } catch (ScenarioLoadException e) {
        throw new ManagementException("launch of Repast failed", e);
    } catch (JAXBException e) {
        throw new ManagementException("launch of Repast failed", e);
    }

    setState(EnvironmentState.RUNNING);
    // System.out.println("re-initialised");
```

No checks whether code satisfies pre condition.

```
    /**
     * kill the client interface. kill at this moment does not kill the server,
     * it just disconnects the client. Make sure all entities and agents have
     * been unbound before doing this.
     *
     * @throws RemoteException
     * @throws NotBoundException
     * @throws MalformedURLException
     */
    public void kill() throws RemoteException, MalformedURLException,
            NotBoundException {
        server.unregisterClient(this);
        Naming.unbind(bindAddress);
        UnicastRemoteObject.unexportObject(this, true);
    }
```

Nothing tested, everything is assumed to work.

```
/**
 * This class implements the repast {@link Runner}. This handles the calls to
 * the repast stepping - when do bots move, how often, and pausing etc. This is
 * modified copy of TestRunner_2 (see #2009 and #2236). I did not give this much
 * thought, I just plugged it in and it did what I hoped for - being able to
 * control Repast. Otherwise, scheduling/running is not done here at all, but
 * from the {@link BW4TEnvironment} directly by calling {@link #step()}.
 *
 * @author W.Pasman 11mar13
 *
 */
public class BW4TRunner extends AbstractRunner {
```

TODO's and should-be-fixed's.

```
 public void runInitialize() {
     Parameters params = new Parameters() {
         /**
          * This should be changed. Temporary fix in order to be able to run BW4T.
          */
         @Override
         public void setValue(String paramName, Object val) {
             // TODO Auto-generated method stub

         }

         @Override
         public boolean isReadOnly(String paramName) {
             // TODO Auto-generated method stub
             return false;
         }

         @Override
         public String getValueAsString(String paramName) {
             // TODO Auto-generated method stub
             return "asd";
         }
```

4.2. ENVIRONMENT STORE

In the beginning of the project the customer made clear what their desires were for the Environment Store. These are listed below:

- Random Map Generator
  - Number of Maps
  - Number of Rooms
  - Number of Bots
  - Blockades
  - Without constraints like of for example a maximum of 26 columns
- Make a map by hand
- Save environments

All these things are roughly implemented as they are described. We will walk them through, not completely in order.
The creating of maps by hand is changed completely. Before we changed it it simply asked for a number of rows and columns and then generated rows of rooms with a drop zone on the bottom. After that you could determine what blocks every room contained and in what sequence they had to be collected. All that is left of this is the grid which is generated by the number of rows and columns that are specified. This grid is empty, this means that every zone is a corridor. Then rooms can be added in every zone by hand. Also the spawn points of the bots can be determined and the drop zone can also be placed wherever the user likes. Extra additions are blockades and charging zones. The charging zones are added because the bots can now have batteries which need to be charged somehow. The reason we implemented it like this is because the grid could easily be adapted to a fully custom made map. The maximum number of rows and columns is not removed but it is raised to 100. We did not raise it to unlimited because it would simply get too big and you could not get a clear overview of the map that way.
The 'random map generator' is an option in the map editor. At this point the number of rows and columns of the grid is already specified. In this generator the rooms are still added like rows of rooms and not completely random locations because you would get too many unsolvable maps. Every time you generate the map you get a slightly different amount of rooms. This was done to improve the randomness of the generated maps. The number of bots is also not generated here. We chose not to do that because the bots can be added in the Scenario Editor. We left out the option to automatically create a certain amount of maps at once. This is because the way we do it now you can generate a map and after that you can easily make additional changes. Of course the maps the user creates can be saved and it can be easily loaded in the BW4T environment.

4.3. SCENARIO EDITOR

The customer wanted us to create a Scenario Editor that generates MAS files. The MAS files should be able to be used to run various types of simulation runs. In the Scenario Editor they wanted to be able to add a number of bots. Per bot they wanted to be able to change the bot type, change the capabilities of the bot and change how the bot will be controlled (by an agent file or by a human player). They also wanted to have a Bot Store in which the capabilities of bots can be managed and different types of bots can be created, including pre-configured bots and e-partners.
For the design we drew up some prototypes which were shown to the customer for some feedback. The feedback they gave us was that they wanted some buttons to be positioned differently, they wanted two tables instead of just one and they also wanted us to add an extra column in the tables. They also had questions about a 'use GOAL' option we had included in the editor which we discussed and decided to leave that option out, because it wouldn't make a big difference and it could still be changed manually. We then started implementing the GUI. When we finished them, we showed them to the customer again and they agreed on the design.
Even though the customer initially wanted the Bot Store to contain e-partners as well, we decided against it. We figured making a separate E-partner Store would be more clear for the users. If we would have added e-partners to the Bot Store as well, the store would have too many options and the window would be cluttered. We showed them the designs, and they agreed with it.

4.4. HUMAN PLAYER GUI

The Human Player GUI gives a human the possibility to control an agent in the environment. This was an already existing feature in the bw4t-client, but the customers had some additional wishes. Most of these

wishes have already been fulfilled. For the wishes that haven't been fulfilled, it will be specifically stated. They for instance wanted the message box for the agents and e-partners to be separated. The e-partner message box should only be shown when the human has picked up an e-partner. It also had to be clear with which bot you are currently communicating. Another aspect was about coming up with a clever solution for avoiding a long list of rooms when having an option list with rooms. We didn't come up with a good solution to this, but one possibility would be to have an option to type in the room (instead of selecting it with the mouse). They also wanted the remaining battery percentage to be shown somewhere. Furthermore they wanted options that weren't possible (such as picking up a block when having the gripper disabled) to not be displayed. During a demo we were also told that the blocks were quite small (and thus hard to click) and the same goes for the map. One thing that we did as a result of this, was to make it possible to zoom in and out and to make it possible to use the mouse to scroll over the map. We didn't change the block size though. Partly because this would mean a change in the path finding algorithm. Because the customer also wanted to make the system more realistic, they asked that the e-partner could only be seen by humans that were close to it. A last wish - of which they didn't expect us to finish it but to mention it in the report - was implementing failing actions. So failing to pick up a block, seeing the wrong color of a block or dropping a block while walking. It should be possible to let the user specify the chance in that actions have to fail.