The performance Impact of Communication Failure in BlocksWorld for Teams

J.Z. van den Oever

TU Delft

The performance Impact of Communication Failure in BlocksWorld for Teams

by

J.Z. van den Oever

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on <date>

Student number: 4002741
Project duration: Juli 1, 2015 – Month 1, 2018
Thesis committee: Dr. K. V. Hindriks, TU Delft, supervisor
Dr. W. Brinkman, TU Delft

TU Delft

Abstract

Multi-agent systems often communicate to perform better at their tasks. However, messages sometimes get lost after sending, and the communication is disrupted. This paper looks into the effects of messages having a chance of not arriving with an agent using the Blocks World for Teams simulation environment. The environment uses agent teams written in the GOAL programming language. First, this thesis details a pilot using existing agent teams with different communication failure models. Each agent team is run multiple times per failure model to get preliminary data. Based on this data, the thesis goes into a followup experiment. The experiment starts with the design of four agent teams, which only differ in the communication strategy they use to share information. What follows is the configuration of Blocks World for Teams, agent team size selection, the communication failure model, how the simulations are are run in a consistent manner, and finally, how the data generated by the simulations is processed. After modifying the agent teams, so they no longer waste limited resources, the experiment is repeated. The different communication strategies are affected to different degrees by communication failure. While the teams perform better than a single agent without failure, slowdown with communication failure can make the teams perform worse than the single agent. How often agents obstruct each other does not appear to be affected by how often communication fails as long as it does fail.

Contents

1 Introduction

The prevalence and mobility of robotic systems have given rise to many new applications and uses. Instead of bolted down machines in controlled environments like factories, new categories of robots are in development. They are intended for more unpredictable environments and have to adapt to new situations as they are not bound to a single location. In some cases, these robots are intended for use as a single unit, like the many robots designed as toys, or those used as assistants like robot receptionists[15]. However, others work in groups like autonomous vehicles. Sometimes with other robots and sometimes with humans.

One project trying to take advantage of such robots is TRADR, Long-Term Human-Robot Teaming for Robot-Assisted Disaster Response[11]. This project aims to create teams of robots and humans working together to provide urban disaster response services. These teams will search or explore disaster areas requiring mobile robots that work effectively in a team. Search and rescue scenarios require that people get found as fast as possible and involve investigating locations dangerous for humans. Communication is crucial to make sure that the robots[1] and humans work efficiently.

TRADR was the initial motivator for looking at the way teams of robots communicate. In what manner do robots communicate, and how does a person creating robot teams prevent misunderstandings between the robots? These questions are not just limited to search and rescue scenarios, but also other applications of mobile robots exist where communicating offers benefits. One example of such applications is warehousing, where robots are already in development. Although, in the case of Amazon[2], already being used. These robots need to navigate a changing environment and coordinate their work to fulfill orders quickly. Communication allows for better cooperation, so it is easier to achieve those goals. The work can be divided, and information on changes shared, to increase efficiency by fulfilling orders more quickly.

A similar application is self-driving cars. Like the warehouse robots, they need to navigate through an environment but on a different scale. Here the distances are longer, and more vehicles participate in the traffic. Unfortunately, self-driving cars do make mistakes, such as a fatal one involving an experimental Uber car that failed to detect someone crossing the street at night[6]. Currently, self-driving cars rely only on built-in sensors, but there is work being done to change this. Through the use of wireless technologies, cars can communicate to plan more efficient routes, resulting in people reaching their destination sooner and preventing accidents[4, 14]. However, the coverage of these wireless signals is limited in some circumstances[7, 13]. Interference from other wireless systems or physical objects can block transmissions. As a result, messages could get lost, and the self-driving cars fail to communicate. Despite that limitation, wireless communication provides greater mobility to robots than when using wires, even though physical connections experience far less interference. Wires are not only limited by their length but can also get tangled or stuck in the environment when moving around. Restricting the robot's reach even more, or getting them stuck. The cables can also be an obstacle for other robots, requiring the robots to avoid crossing paths. So despite not being as reliable as cables, wireless communication is preferred for robots that move around.

Communication protocols have been developed that mitigate the impact of communication failure by minimizing the loss of messages. However, due to physical limitations, communication failure is unavoidable in some circumstances. Disaster areas for example, like the ones the TRADR project

focuses on, can have plenty of interference like layers of rubble between the sender and receiver. The interference negatively impacts the reach of wireless signals and almost guarantees that some messages will not arrive. Previous research investigated how to recover the connections between agents in case of failure [16, 20]. In this thesis, however, I have investigated the question of how the loss of messages affects cooperative robot teams and looked into methods for handling the inefficiencies in which that results. For robot teams, inefficiencies mean having multiple robots do the same thing where only one would suffice, or repeat something that has already been finished. For example, for warehouse robots, that means more than one robot is trying to grab the same product for an order, or adding a product to the wrong order. Ideally, to handle the inefficiencies, a robot team would divide the work in such a manner that no robot does work that another robot is already doing. The team's efficiency is determined by the degree to which the robots can make sure that happens and by only idling when there is no more work to be done.

1.1. Problem Statement

The robots discussed earlier, like the ones for warehouses or self-driving cars, use software to decide for themselves how to achieve a task. Such programs are called agents. When multiple agents, like the warehouse robots, cooperate to achieve a goal as a team, it is called a multi-agent system.

This research uses multi-agent systems to investigate the problems with communication. Specifically, agents organized in teams, cooperating to perform multi-stage tasks. Such tasks require multiple actions to be completed. Warehouse robots, for example, need to collect the items someone ordered in a single package. Every item in the order would be a stage in the task. When performing multi-stage tasks, it is possible to divide work between the agents through communication. However, when problems arise with communication, decisions are made based on incomplete data. The agent teams are then less likely to make the optimal choice. The time required to complete a task increases as agents make inefficient choices. If the agents assume the information is complete, these choices can even result in actions that make it impossible for the teams to finish.

One cause for such problems is when agents assume the communicated information needs no verification. If the agent does not confirm it received all messages, it might assume some locations are not relevant to the task at hand. As a result, sometimes, the agents will not know where to go to perform the task at hand and do nothing. For example, an agent does not receive a message containing environmental data, and because there is no verification, the agent is unaware that the information is incomplete. Take a warehouse robot where a message, which mentions that another robot moved some items, is lost. Hence, the agent assumes the items are still at the previous location. However, once it gets there to pick them up, the items are not there, and it does not know what to do. The agent made an incorrect assumption, which led to less optimal decisions because the robot was unaware that the situation had changed. Alternatively, multiple robots try to deliver the same item, unaware of the others, causing them to double up on work.

Communication failure does not just lead to slower agent teams. Teams can fail their task because the agents have conflicting beliefs. A belief is a piece of information an agent assumes to be true. For example, the current state of a task is a belief. When agents have differing beliefs about the state of a task, this results in differing conclusions about what to do next. The mismatch can cause agents to get in the way of one another, or make them consume limited resources without contributing to the overall goal. For example, a self-driving car might waste fuel when taking wrong turns, and it runs out. The task then becomes impossible to finish because there is no fuel left. The impact on of such problems on how often the teams succeed, and how long they take to do so, indicates how well that agent team handles communication failure. The degree to which the team performance is affected is called the team's resilience. The more agent team efficiency goes down, the less resilient the team is. If there is very little difference in the performance after introducing communication failure, the team is highly resilient.

To determine which effects inefficient choices and resource waste have, data on what happens in such situations is required. In addition, there might be non-obvious problems while developing or working with agent teams when introducing communication failure. To that end, we look at agent team strategies applying different degrees of communication. The strategies do not just differ in how much information gets shared between agents, but also in how much communication is involved in deciding what the agent will do next.

Because the focus is on the communication failure and agent team resilience, this thesis will limit itself to multi-stage tasks shared by the whole team. The reason for focusing on multi-stage tasks is because these allow for strategies where agent teams distribute stages between them to speed up the work by working on stages in parallel. Having all agents share the same task simplifies the experiment and makes it easier to introduce the type of cooperation needed to divide the work. At the same time, such an experimental setup is more in line with the search and rescue scenarios that initially motivated this research. In the warehouse robots case presented earlier, an example of the type of multi-stage task discussed would be gathering up the items for a single delivery. Gathering every item that needs to be delivered would be a stage of the task. Only once all items have been collected would the task of preparing the delivery be complete.

After determining the effects of communication failure on multi-stage tasks, it becomes possible to identify the different types of failure and the impact thereof on the strength of the observed effects. Similarly, the resilience to communication failure between the different agent team strategies shall be compared. Based on the data, we will categorize different types of failure with the intent of making it easier to find potential communication problems and help direct research into methods for mitigating the effects.

1.2. Research Questions

This section expands on the goals discussed in the previous section and aims to turn them into the research questions that this thesis aims to answer. The main goal of this thesis is to provide insight into what happens when agents fail to communicate and how that affects agent team strategies. Before explaining how to provide that insight, we define what communication failure is within the context of this thesis: the intended recipient of a message did not receive the message that an agent sent. Based on that definition, the following questions were raised:

What are the effects of communication failure on multi-agent team effectiveness for performing multi-stage tasks?

For this research, a multi-agent team is considered effective if it performs the tasks assigned to it quickly. A team's effectiveness is how close it got to the fastest solution. Communication failure will affect this, but the effects might differ between strategies. Some strategies might only be susceptible to minor changes in how quickly the task gets resolved, while others fail in the task most of the time. By learning more about how this influences the effects, it becomes possible to make more robust agent teams in situations where communications failure is likely or even inevitable. To determine what these effects are, several agent teams using different strategies will be tested. These agent teams should be more effective than a single agent performing the same task. The tests will consist of agent teams performing the same task with and without communication failure to determine the difference in effectiveness.

How can we make multi-agent teams resilient to the negative effects of communication failure on performing tasks?

Once we learned what the effects are, we can look into improving the effectiveness of the agent teams when communication failure does occur. It is not always possible to prevent all failures, or if they can be, it often requires so much effort that it is not practical. Hence agent teams will need to accommodate these situations. By identifying which adverse effects have the most impact, the agent teams can be adjusted to mitigate that effect or maybe even eliminate it. Then the test for determining the effects can be run once more with the adjusted teams to verify if the changes improve the resilience.

1.3. Research Approach

To gather data on the effects, we will simulate agent teams enduring communication failure in a digital environment rather than a physical one. While real-world scenarios with robots inspired the research, physical agents teams have downsides that simulated teams do not have. Robots cost significantly more money than computing power does, so simulated setups, which require less hardware, make running experiments in parallel more affordable. The ability to run multiple experiments at the same time allows for reducing the total time spent on them and evaluating more variables. Similarly, measuring agent team results and automating everything does not require additional hardware because the data

is provided by the simulator.

The remainder of this section discusses the setup for the agent team simulations. First, it introduces how the agent teams will be run, and then explains why the environment is a suitable choice for running the teams — followed by an overview of the experiments that use the simulation environment to acquire useful data for analysis. The experiments were done in two sets, first a Pilot and then a custom configuration full-scale experiment.

1.3.1. Simulation Environment

The Blocks World for Teams (BW4T) environment was chosen to run the agent team simulations. BW4T is a well-defined simulator for researching multi-agent systems[8] where an agent, or multiple agents, have to deliver colored blocks. The blocks are taken from rooms and need to be dropped off in a special room called the dropzone in a given color sequence. This scenario provides a simple multi-stage search and retrieval task for agent teams to complete. Chapter 2 has more detail on how BW4T works, and chapter 5 presents the environment configuration used for the experiments.

Communication is optional for agent teams running in BW4T but does offer performance benefits. The agents are only aware of their immediate surroundings. With communication, they can work on the different stages in parallel. The agents can grab blocks at any time and bring them closer to the dropzone so the blocks can then be dropped off as soon as the previous block in the sequence is delivered. Additionally, making mistakes imposes a cost since incorrectly delivered blocks are not counted towards task completion and removed from the environment. These conditions are similar to real-world scenarios like search and rescue, so the experimental results are applicable beyond this specific experiment.

1.3.2. Pilot

The pilot, which chapter 3 covers, investigates if BW4T fulfills the requirements as a tool for investigating communication failure. The investigation is required because BW4T does not provide a way to let communication fail. Additionally, the pilot has some secondary objectives: checking whether or not the negative effects do show up, identifying the measurements needed, and verifying that the results provide enough information to determine what went wrong. The last is required for creating mitigation strategies once the full experiments are executed.

Implementing failure will be attempted in two ways:

- The code that executes the agent teams is adjusted to have that feature built-in. This method works regardless of the agent team used. Additionally, this method defines how a message fails in one place, so it is easy to switch between different forms of failure when needed.

- The agent team itself emulates the communication failure. This method, however, requires manual adjustment of all teams when switching between different forms of failure.

The agent teams chosen for the pilot were created before the work on this thesis started. These teams had to communicate and should always complete the task when no failure occurs. These teams assume that all messages will arrive and have no strategies to limit the effects of communication failure. The agent team simulations used existing BW4T maps.

The method for providing communication failure for the full-scale experiment is chosen based on the difficulty of adjusting the agent team runtime compared to emulating the failure. The solution that is easier to implement is used for the custom teams later. The outcome of the simulations then determines if the results are suitable for analysis and if the data available suffices.

1.3.3. Full-scale Experiment

Following the pilot, work started on creating a set of agents specific to this experiment to control for the difference in agent strategies. The pre-existing agent teams did provide data but varied not just in the way in which they communicated. Differences in how the agents navigate and how actions get prioritized also contributed to the results differing between the agent teams. For this thesis, only the effects of communication are relevant. As such, the agent teams created for the experiment only differed in how and what they communicate. The code is the same in all other aspects. Chapter 4 explains this design in more detail.

However, it should be noted that the teams assume that communication always succeeds. There are no methods to reduce the number of messages lost, such as repeating messages in the hope that one of them does arrive. Similarly, the agents will not use protocols to identify if any information is missing, so agents do not know when the information might be incomplete. The experiment aims to find the effects of communication failure on agent team performance, and such methods attempt to prevent failure. As a result, these methods are undesirable as they could mask some of the effects the experiment tries to measure.

The results of the custom agent teams were then analyzed to determine how the communication fails, and if different communication strategies are more resilient to failure. The resilience of the agent teams was then determined by evaluating the output of the simulations as follows:

- Comparing the effectiveness of the agent team to a baseline agent team consisting of a single agent that does not communicate.
- Comparing how effective the agent team is when communication failure occurs to running without failure.

Then the results of these comparisons helped to identify the strongest effects and which agent team strategies perform better than others in the failure scenarios. Based on those results, methods for mitigating the effects of failure were then determined. Of the mitigation strategies, one was then applied to the agent teams and run through the same simulations. The results of which were then compared to the previous results. The comparison lets us determine how effective it was and if this resulted in a useful mitigation strategy.

1.4. Outline

Excluding the introduction the thesis is laid out as follows:

- Chapter 2: This chapter describes background information regarding communication with the main focus on multi-agent systems, the simulation environment, and the GOAL programming language.
- Chapter 3: This chapter details the pilot, and goes into how it was set up, what the results were, and the lessons learned for the more extensive experiments.
- Chapter 4: This chapter provides a technical overview of the agent teams created for the research by explaining the basic behaviour and the different communication strategies.
- Chapter 5: This chapter describes the main experiments and the procedures for running them.
- Chapter 6: This chapter explores the results of the experiments, evaluates the effectiveness of the agent teams, and identifies the problems introduced by communication failure.
- Chapter 7: This chapter examines the problems identified in the previous chapter and offers mitigation strategies. It also re-evaluates the agent teams to determine the effects of one of the mitigation strategies.
- Chapter 8: This chapter has recommendations for future avenues of research.

2

Related Work

This chapter discusses communication as it relates to software, how it can fail, and what sort of work is already being done to prevent it. Then it goes into what sort of environment is used to test the multi-agent systems in and how that environment works. Finally, it discusses the programming language used to create the agents and gives a brief overview of the structure of agent programs in that language.

2.1. Multi Agent Team Communication

Communication happens all the time in daily life. Be it to share information, discuss problems, or just for fun. People do it all the time, but increasingly the machines around us do as well. Sometimes it is between a person and software, as is the case for the voice-controlled personal assistants like Siri or Google Assistant. They get asked questions or given tasks to complete, such as setting reminders. However, just like humans sometimes misunderstand, so does the software. It hears the wrong words or misunderstands what the user intended. Sometimes it does not respond at all. When dealing with communication between software programs, some of those problems are solved. Misunderstandings are limited as the communication is structured through protocols, which are agreements about how to send and receive data for specific purposes. However, protocols are limited in scope since everything has to be predefined. So, most of the time, the software is structured around these limits.

Similarly, researchers have looked at when and how agent coordination is required to cooperate in environments like that of BW4T[17]. Often agents send messages to provide such coordination, how often, and how much agents communicate influences the amount of work required to make the agent teams work. The trade-off between team performance and communication complexity has been previously investigated[18].

Communication is not just about what is said, but also about how the information is transmitted. Multi-agent systems can have physical constraints that influence how and when they send messages. As such, there is research being done on implicit communication[12] or how to effectively utilize limited or shared communication channels[19]. Such methods are becoming more relevant as multi-agent systems grow in size and complexity, such as the drone delivery systems currently in development for various companies[26]. These systems are expected to deal with unreliable communication at least some of the time. Research is already being done to see how the communication systems can be made more reliable[16, 25]. However, this research generally focuses on detecting that a fault has happened or making sure that the communication systems work or recover communication[3], rather than investigating what happens when communication fails.

2.2. Blocks World for Teams (BW4T)

BW4T is a well-defined simulator for researching multi-agent systems based on a simple planning problem[8]. To perform simulations with BW4T, agent teams connect to a BW4T server, which runs separately from the agents. Previous research has already used BW4T to look into various aspects of coordination, planning, and interaction between agents[5, 9, 23, 24]. In addition, BW4T is very suitable for investigating communication failure while doing multi-stage tasks, as this section will explain.

BW4T provides a simple multi-stage search and retrieval task for agent teams to complete. A single agent can perform all stages alone, but adding more agents that communicate speeds up the process. The BW4T environment provides different maps for the simulation to run in and a map builder for custom topologies. An agent, or multiple agents, have to pick up colored blocks and drop them off in a special room in the map called the dropzone. Blocks have to be delivered in the order of the sequence. For example, if the sequence is red, white, blue, the first block to deliver is red, then a white one, and finally, a blue block. The only difference between blocks is the color, and there can be multiple blocks of any single color. When delivering blocks, any block of the right color will suffice. Delivering the sequence with these requirements is a multi-stage task where every required color is one stage of the task. As such, only maps that have at least one reachable block per block in the sequence can be completed. These blocks are found in rooms with only one doorway through which to enter, and no more than one agent can be in a room at any given moment. When creating maps, the rooms should be accessible through corridors for the blocks to be reachable. Based on these concepts, BW4T provides a customizable simulated space. More details on how to configure BW4T are available in Chapter 5.

This task is a more abstract form of real-world applications like search and rescue scenarios. The agents first have to search for the 'humans,' which in this case are the blocks. Then the agents 'rescue' them by delivering the blocks to the dropzone. Additionally, the agents have to deal with similar environmental restrictions because the maps mimic buildings with corridors and rooms. Agents have only limited awareness of their surroundings. Only the immediate surroundings are observable by an agent, which means agents will have to explore the map. The observation limitations let us investigate what happens when agents unexpectedly have incomplete knowledge about parts of the environment. In real-world situations, this would happen in locations where wireless signals cannot reach.

When blocks get delivered that differ from the current color in the sequence, the block is lost as BW4T deletes it from the simulation environment. Losing blocks imposes a cost on mistakes as limited resources get consumed. In real-world applications like search and rescue, delays or missing materials can cost lives. As such, it is essential to know how to limit wasting those resources. For warehousing applications, there is only a limited amount of goods stored. Losing items might cause orders to be impossible to fulfill, which in turn causes problems for the rest of the business.

Finally, the multi-stage task needs agents to complete the sequence in order. The sequence requires agent teams to make sure they know which stage of the task needs doing. The stage order corresponds to prioritizing different rescue targets or having to take specific steps to make rescue possible. For the warehousing scenarios, packages might need delivering in a specific order for efficient packing in delivery trucks or for a timely delivery. For these situations, knowing what happens if communication failure makes agents unaware of what the current stage of the task is, it is essential to prevent costly mistakes.

One more thing to note is that BW4T supports two optional settings, agent collisions, and human agents. If the collision option is on, the agent teams would have to move around each other. BW4T provides percepts that agents can use to determine where they and other agents are. Human controlled agents follow the directions provided through GUI. However, moving around other agents can be done without communication. So the option is not used for this thesis. Human controlled agents are similarly excluded. This research only considers the communication between software agents in our research questions. Additionally, adding humans would make running the experiments significantly more time consuming and complicated.

2.3. GOAL

There are several different tools for programming agents like the Java Agent Development Framework (JADE), or SARL. Both provide tools for writing multi-agent programs. However, for this research, the GOAL programming language was used because of familiarity and access to existing agents.

GOAL is an agent programming language used to make agents that make decisions based on goals and beliefs. By taking actions, the agent will change the environment until it believes it has achieved the goal. For example, an agent might have the goal to open a door, and then try to open the door. When that succeeds, the agent's beliefs are updated to reflect that so it knows when the goal is achieved. The goals and beliefs are defined using predicate logic; the goal to open a door would be represented as "door(door_id, open)." Agents achieve a goal when they believe the goal is true. So when the beliefs are updated to include "door(door_id, open)", the agent achieves its goals.

GOAL provides the building blocks for working with goals and beliefs in this manner. Agents written in GOAL consist of six components: the MAS file, knowledge, the initial state, the event module, the main module, and action specifications. These components will be explained using the same door opening example from before:

The MAS file specifies how the agent team should be run. It usually starts with defining the connection to the simulation environment by specifying the client and, if necessary, the connection parameters. Then it defines what types of agents there are in the the team. The definitions specify which files to use for the main module, the initial state, and the event module. Finally, it specifies a launch policy that determines when to start the agents.

A knowledge file provides additional logic to derive information for the other components as predicate logic rules implemented in prolog. For example, the agent might know how to determine if a door is locked. The knowledge file would represent checking if the lock on the door is open as follows: ”unlocked(door_id) :- attached(lock_id, door_id), lock(lock_id, open).” First, we check which lock is attached to the door, then we check if the lock is in an open state. Knowledge files need to imported as such before they can be used in another module. Modules can import multiple knowledge files.

The initial state module is used to provide starting beliefs and goals for the agent. These can be static or based on some initial information by the environment. For example, the agent could always start with a goal of ”openedDoors(3)” if we want the agent to open three doors. Sometimes the environment provides information when starting up the agent; these messages can be processed in the same way as the events module.

While an agent is running, the event module processes messages sent to the agent by the environment. The messages are called percepts, and the module uses rules to turn them into beliefs and sometimes goals. For example, the environment sends the agent a list of locks and doors it can see. These then get processed into beliefs like ”door(door_id, open)” or ”door(door_id, closed)”, depending on the state of the door.

The main module has lists of rules made out of two parts: The condition, which is a check to see if a combination of goals and beliefs are true, and the action, which can be specified by an environment, like sending messages or an action defined in the action specification. So a rule might have the condition ”goal(door(door_id, open)), bel(door(door_id, closed))” to check if there is a door the agent wants open but currently believes to be closed. Then the action would be to open the door, which is defined in the action specification because the action is specific to the environment.

The action specification consists of three things: The name and parameters of the action, the preconditions, and the postconditions. The name and parameters get sent to the environment, so it knows what the agent wants to do. Opening a door would be ”open(door_id)” to show which specific door we want to open. The preconditions get checked when we try to execute an action. Unless these conditions are true, the action is canceled. For example, we only want to open closed doors, so the precondition ”door(door_id, closed)” is added to ensure that. The postcondition modifies the beliefs of an agent. After opening the door, we no longer believe the door is closed, so now the agent believes ”door(door_id, open).”

3

Pilot

The pilot was the initial research in agent communication failure using already existing components. This chapter will not provide any detailed analysis but focuses on two things. (1) The primary goal was to determine if introducing communication failure to BW4T agents written in GOAL was possible and how to implement the failure. (2) The secondary goal was to validate if the effects of communication failure were measurable with the data from the BW4T environment. This chapter first explains how the pilot was set up, then discusses the results, and finally provides conclusions based on the results.

3.1. Setup

The pilot used pre-existing BW4T GOAL agents on one of the maps that comes with the BW4T server. The different agent teams were chosen to look at communication in different ways while still finishing the sequence under ordinary circumstances. The following agent teams, made for previous communication research, were used:

- Blockbuster mincom
- Blockbuster dropzone only
- Blockbuster rooms only
- Maximum Coordination

The Blockbuster agents' intended use was for unpublished research on how increased communication relates to increased performance and if the added complexity was worth it. Personal communication provided the agent team code. The behaviors of the three agent teams are similar but differ as follows: 'mincom' only communicates when delivering a block. Whereas 'dropzone only' and 'rooms only' both communicate which rooms the agent visited and what blocks they have found. The only difference between the two agent teams is that 'dropzone only' team, has the agents go back to the dropzone after every action. For example, if a room was visited before and no block to deliver was there, it would go back to the dropzone. Using that location as a marker to determine a new action is required.

'Maximum Coordination' was first created by me for a university project to study how agents could work together with a human-controlled agent and communicate their intent. For this experiment, 'Maximum Coordination' was modified to no longer include the human-controlled agent and removed the communication intended just for humans. This change leaves the agent team with the following behavior: The agents first try to visit rooms at random and then deliver the blocks found as soon as they find the right block. When these agents decide to visit a room or deliver a block, the other agents are informed. If multiple agents try to perform the same action, the team will decide which agent can continue performing that action.

All four agent teams are then run through the BW4T environment using a python based script that started the BW4T server and an agent team. Every agent team got tested without any communication failure, which served as the baseline for comparing the results with failure.

For every failure model described below, each agent team was run 200 times except for the sending failure, which was repeated 50 times. The repeating simulations are to compensate for the randomness in the failure models and agent behavior. The agents acted in a map called 'SuperRandom,' which comes packaged with the BW4T server. The map generates a random sequence every time the simulation starts and allows teams of up to ten agents rather than the limit of three for the other maps. The random sequence made it so we could test the failure models for different types of sequences. The agent teams all had three agents, which is the minimum size required for agents to be sending messages to multiple agents. Having multiple receivers is important because some failure models can cause an agent to send a message that is only received by some of the other agents. For every simulation, the team had five minutes to complete the sequence before being stopped. If the task did not get completed in that time, the agent would most likely never finish, and therefore was considered a failure.

The pilot tested three different failure models to determine if they could be implemented. The model chosen for the full experiment would come from the viable models. The 'receiving failure' model tested different percentage chances of not receiving a message. The goal was to create a model mimicking interference in wireless communications. The 'communication cutoff' model cut off all communication after 30 seconds as would happen with networking hardware failures. The cutoff happens at 30 seconds to put it just after the halfway point of most simulations. For the reference data, the agent teams generally took a little under one minute to complete the sequence. Due to time constraints on the time scheduled for the pilot, the final model was only run for two agents with just one failure chance. The 'send failure' model is like receive failure but with a chance of not sending the message instead of not receiving. It has only been run with a 50% chance of not sending a message. This model intended to simulate partial networking hardware failure as a variant of the cutoff model.

3.2. Results

The BW4T environment logged the results of the experiments. Every simulation with an agent team resulted in a log file. These simulations will be referred to as 'simulation runs.' Through the use of a python script, the logs were processed to determine how often the agent teams were successful. The difference in the success rate for each failure mode was compared to the baseline using a chi-square analysis. All agent and failure mode combinations resulted in a chi-square $p < 0.01$ except for Coordination with $p > 0.05$ ($p \approx 0.07$). As a result, the success rate of the Blockbuster agents with communication failure is significantly lower than the reference data. The Maximum Coordination agent still has a notably lower success rate, but not as low as the Blockbuster agents. Further analysis was done by visualizing the success to failure ratio in figure 3.1.

The first point that stands out in the agent results is that the agent teams sometimes fail when performing the baseline simulation runs. Further examination showed that these were due to issues external to the agents. Some of them were fixable in the experiment methodology. For example, by making improvements to the test runner, but others resulted in fixes in GOAL to prevent them from happening during the main experiment. One of these errors was a problem while loading libraries for multiple agents at the same time. As they both tried to access the library at the same time, the agent team as a whole would crash. However, the BW4T log files did not include the agent team error messages. As such, separating these failures from communication-related task completion failures was not an option.

Additionally, the 'Maximum Coordination' agent team seems significantly more successful than the Blockbuster agents at completing the task when communication failures occurred. The difference in p values between the Blockbuster teams and Maximum Coordination corroborates this. The difference can be explained by how this agent team diverges from the others when it comes to communication. This agent team uses the $SequenceIndex$ percept, to update how far into the sequence it is. Where the other three agent teams only communicated about delivering blocks and ignored the percept. When looking at the communication failure results, there is a significant difference in the amount of success simulation runs between agents using that percept and those that do not. Giving agents the ability to correct their beliefs on the sequence state could be a basis for a method to reduce the impact of communication failure.

However, what the graphs do show is that the Blockbuster agents differ from 'Maximum Coordination' in how the increasing communication failure chances affect the success to failure ratio. 'Maximum

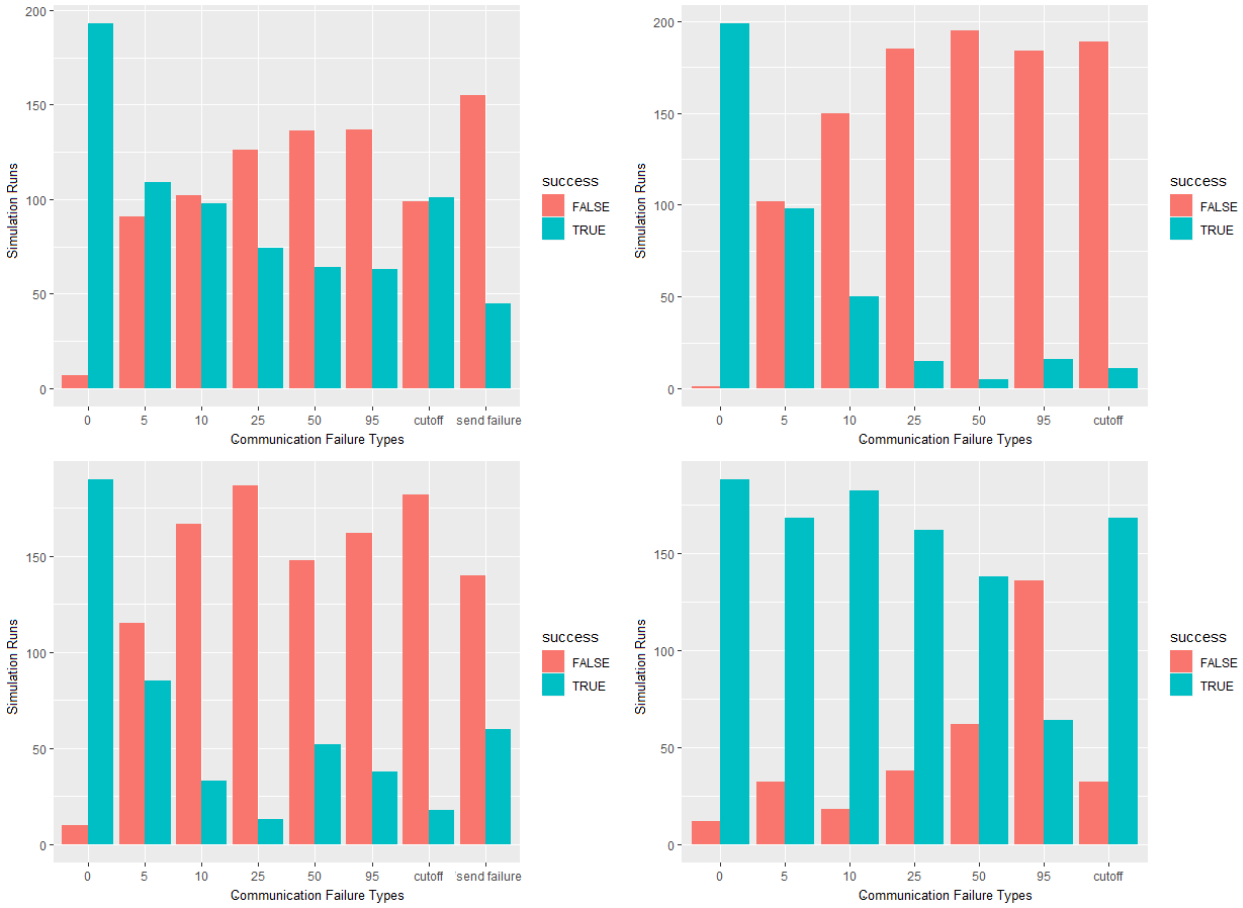

Figure 3.1: These are the results for top left: Blockbuster Mincom. Top right: Blockbuster Rooms Only. Bottom left: Blockbuster Dropzone Only. Bottom right: Maximum Coordination.

Coordination' has a noticeable decrease in task completion, as the communication failure chance increases. The decrease might be related to the team's coordination strategy. Agents send each other messages about their intentions. If there is a conflict, the agent furthest away will do something else. However, if the agents are not fully updated on what the other agents are doing, they may make incorrect decisions. However, the Blockbuster agents seem to be less predictable. The unpredictability might be in part because of differences between the agents other than communication, such as navigation strategy. 'Maximum Coordination' selects the closest relevant blocks and rooms, whereas the Blockbuster teams select these randomly. As a result, there is not much to say about what the effects of communication failure could be.

Finally, the results are not very useful for comparing the different failure models. The limited test results for 'send failure' seem to be similar to 'receive failure.' However, there are only a few simulation runs done for this model. Of note is the 'Dropzone Only' Blockbuster agent seems to succeed slightly more often, while the 'Mincom' team succeeds less often. The difference is likely because 'Mincom' shares information on the relevant blocks. The agents will spend less time searching for blocks and will thus deliver the wrong block more often. The 'receive failure' and 'communication cutoff' models do have enough data. The results appear to have a significant variance between the agent teams. Though 'Maximum Coordination' does perform noticeably better for the communication cutoff. The performance difference is caused by the use of the $SequenceIndex$ percept. The Blockbuster agents will all try to deliver the remainder of the sequence individually while 'Maximum Coordination' does not. This variance makes it difficult to determine the extent to which communication failure explains the differences.

3.3. Conclusions

BW4T provided enough information to analyze the agent team success rate in the case of communication failure. However, the more detailed statistics BW4T provided were only available in case the team successfully finished its task. Due to the small scale of the pilot, there were few successful runs, which did not result in statistically significant data. However, if more data is available, it is possible to analyze the run time, time spent idle, the incorrectly delivered blocks, and amount of rooms visited.

Multiple methods of modeling communication failure were successfully evaluated using BW4T and GOAL. There already was a GOAL runtime used for previous experiments that could specify a percentage chance of not receiving a message. This runtime was the easiest way to introduce communication failure to pre-existing agent teams. However, the runtime was for an outdated version of GOAL. The 'communication cutoff' model blocked all communication after a given amount of time passed. This model was used to validate if it was possible to block communication within the GOAL code itself after modifying the agents. This type of modification does require additional time to implement per agent team. However, those changes only have to be made once even if the runtime gets updated. Similarly, modifying the agent teams to have a chance of failure when sending messages was likewise successfully implemented.

In the end, the preferred failure model was 'receiving failure.' Unlike 'communication cutoff' and 'send failure,' that model more closely resembles the type of failure seen in physical environments, such as wireless communication interference. Some agents could still receive the message while others did not. The failure model was implemented by adjusting the agent code. There are only a few agent teams in the full-scale experiment, and the modified runtime was challenging to keep up to date.

Finally, based on the chi-square analysis results of the simulation runs, it can be seen that communication failure does influence the success rate of agent teams. The degree to which the success rate is lowered appears to differ depending on the agent team's communication strategy. Though based on the difference between the Blockbuster agent teams and the Maximum coordination team, mistakes about the state of the task reduce the task completion rate significantly. So correcting agents on what the current stage of the task is, could be an excellent method for mitigating the effects of communication failure.

3.4. Hypotheses based on the pilot

This section introduces several hypotheses which were formulated based on the pilot results. The hypotheses supplement the research questions by describing the expected effects of communication failure. If correct, potential mitigation methods can be derived based on the effects described.

1. When messages are lost, agents can be mistaken what the other agents are doing, resulting in agent teams doing duplicate work.

   The agent teams use communication to differing degrees for coordination. So when messages related to coordination are lost, the agents will not be able to coordinate the tasks as effectively. As a result, the agents will start work, or continue doing work, that a different agent has already been doing. So the amount of duplicate work should increase with lost messages.

2. When messages are lost, agents can improve their success rate by validating which stages of the task have been completed. Not verifying the current task state before consuming limited resources can result in the task becoming impossible to complete if the resource is required for later stages.

   The pilot results suggest that agent teams ('Maximum Coordination') that do not rely on communication for tracking the stage completion are more likely to succeed. Being able to verify independently if a task is still necessary allows agents to consume resources only when required. For BW4T, that means delivering blocks to the dropzone when it is not of the current color required as all blocks delivered there are removed from the environment. This ability would stop agents from failing the task due to a lack of resources. However, the agent teams designed for the later experiments will not immediately implement such measures. Such checks are not required when the design does not account for communication failure. Follow-up experiments can then account for this problem to see what the improvements are.

3. When messages are lost, agents can improve their success rate by making sure they do not block access to any location. Otherwise, agents can unintentionally block access to required resources and make the current task impossible to complete.

   Sometimes agents will know what stage of the task they are in but do not know where to go to complete the task. For example, because another agent moved a block, but the message with the location update got lost. If there are no other goals, the agent will not do anything. These agents will, if they are in a room, block access to it as only one agent can enter a room at the same time. This problem can also occur without communication failure. For example, if not all block information is shared, then an agent can check one room, but a block is dropped off in that room after the agent visited. For this reason, teams in the later experiments do account for this to some degree by making sure the dropzone is always accessible.

4. When messages are lost, agents can miss messages on the environment as observed by other agents, resulting in some agents being unaware of the resources required to complete the current stage of the task. These agents will idle until the task reaches a stage where the resources are known or the task is complete.

   Visual inspection of simulation runs has shown that agents do sometimes idle and then later continue when another agent completes the current stage. The agent teams for later experiments will not have to compensate for this as there will always be at least one agent that is not idle. The agent that sent the message that got lost does know about the resource and will continue doing work. The result is that the agent team will behave as if it temporarily has fewer members, and the task will take longer to complete.

5. When messages are lost, agents can miss messages when negotiating their next action, resulting in agents waiting on a response that will never arrive.

   These systems are not required for the type of task in BW4T but can improve the speed at which the team finishes because they can work ahead. The task assignment does not always have to be explicit but still requires information on the other agents and what they know to infer what they will do correctly. For the type of coordination mechanism described in the hypothesis, agents can become inactive with lost messages. Even the whole team can end up deadlocked if this happens for every agent. Agent teams will not have this problem if all messages arrive. As a result, communication failure is not always taken into account when building task assignment systems. Checking two types of coordination mechanisms, one where agents wait, and one where the agents start and listen for disapproval, can show the impact of this problem.

4

Agent Design

The agent programs are written iteratively using an agent team consisting of one agent as the starting point. The task completion data of this single agent team is the baseline for determining how effective the communication of the other teams is in improving the performance. The following agent teams build upon that by adding different forms of communication. To facilitate the communication added, all agent teams except for the baseline team will have multiple agents. The teams with multiple agents were the result of three iterations. Every iteration adds one new component to the communication. The last iteration implements two variants resulting in a total of 5 different agent teams, including the baseline team. The iterations of additions are shown in figure 4.1.

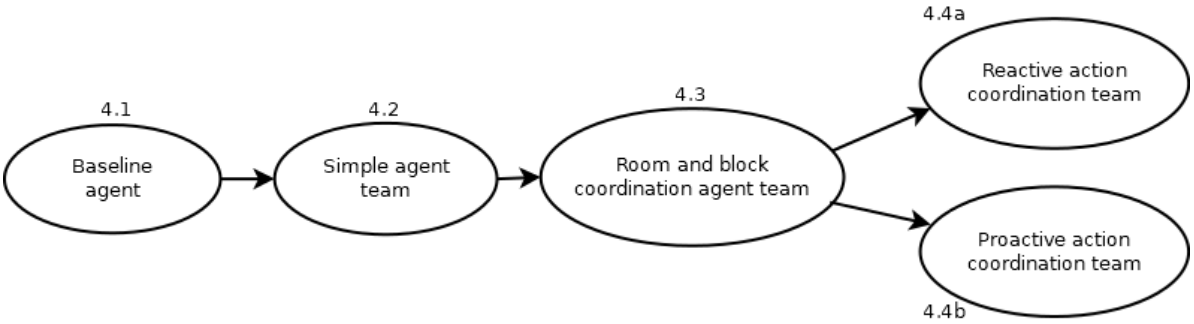

Figure 4.1: Agent types, each agent team has the capabilities of the previous one in the graph. The numbers correspond to the sections discussing the agent team.

The remainder of the chapter discusses each of the five agent programs. The baseline agent defines the default behavior of the agent. The simple agent team section explains how the solitary baseline agent was modified to work in a team by communicating when dropping off a block. Room and block coordination agent team expands on that by introducing communication about what rooms have been visited and which blocks are in there. Finally, there are the Proactive/reactive action coordination agent teams who try to work ahead. These teams divide the delivery work, so only one agent tries to deliver a block in the sequence. These two teams differ in how the agents coordinate their work. The full code for the agents is available on the project's GitHub page under releases[21].

4.1. Baseline Agent

The baseline agent team is a single agent performing a random exploration algorithm. It remembers which rooms the agent explored and the blocks that are present inside the rooms. The exploration is interrupted when the agent finds a room with a block of the current color in the sequence. At that point, the agent switches to a delivery module. The agent picks up the block and moves to the dropzone. Once inside, the agent drops the block to deliver it. When the block is delivered, the agent will first check if it knows the location of a block of the next color and delivers that too. If there are more blocks

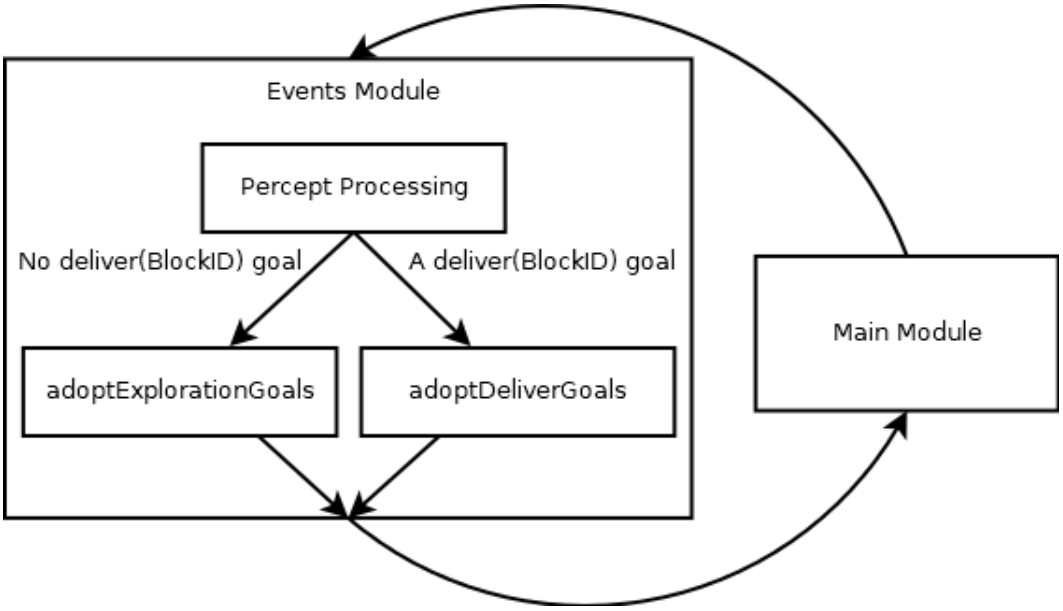

Figure 4.2: A high level flow diagram of how the baseline agent reasons.

of the same color, it chooses one at random. If the agent has not found any blocks of the next color in the sequence yet, it switches to the exploration algorithm.

Blocks are only delivered if they are of the current color in the sequence. Additionally, there is only one agent on the team, so nothing changes without it being aware of it. As a result, there is no chance of failure in completable BW4T maps.

The agent achieves this by performing the following steps:

- First the events module processes all percepts it receives from the environment to update its beliefs. For example, updating which blocks are currently present in a room.
- If the agent has a goal to deliver a block it goes into the adoptDeliverGoals module which provides the intermediate goals to get the block delivered to the dropzone.
- If there is no block to be delivered at this moment in time it goes into the adoptExplorationGoals module. This provides the goals to implement the greedy exploration algorithm described earlier in the section and the goal to deliver a block if the right colour is found.
- Then the agent goes into the main module which performs the actions corresponding to the various goals.
- Finally, if the agent has delivered the entire sequence it stops.

The goal modules and the main module are discussed in more detail below. All modules use the commonKnowledge module to define shared predicates. The sections do not explain percepts as they are part of the BW4T specification, and further clarification is available in the documentation of the environment[10]. The baseline agent shares code between it and all subsequent agents. The shared code results in some communication specific code being available to the baseline agent. However, the rule conditions do not hold for the baseline agent. As such, the impact is minimal, and this section skips such until relevant for the agent team under discussion. An example of such code is the evaluation of the goals to see if they are still relevant. There are no other agents whose actions can make goals irrelevant.

4.1.1. adoptExplorationGoals

First, we look at the adoptExplorationGoals module. The way an agent indicates that it wants to go to a room is with the $in(Place)$ goal. The main module reacts to this goal by performing the goTo action to a room with the name $Place$. Once the agent is in that room, we get a percept, $in(Place)$, and add

it to the beliefs. Goals are completed when the agent has a belief matching the goal, so just entering the right room is enough. The way we adopt the $in(Place)$ goal in this module provides the exploration behavior:

```
if bel( room(Place), not(dropZone(Place)), not(visited(Place)) ) then
    adopt( in(Place) ).
```

This rule looks at all rooms that are not the dropzone. The agent learns of these locations when the simulation starts. When entering a room for the first time, agents add a $visited(Place)$ belief in the events module. This belief lets the agent exclude visited rooms for future exploration goals.

However, without additional configuration, this rule would not implement a *random* exploration. The module has to use linear random order to check its rules. This execution order means that the various rules in the module are executed in linear order, stopping after the first rule that succeeds. The randomness of the execution order comes from the agent choosing a random valid option for the variables evaluated in a rule. As a result, the agent chooses $Place$ at random from all $room(Place)$ beliefs instead of always choosing the first in the belief base. This behavior differs from the default, linear order, which picks the first valid option in the belief base instead.

The module contains another rule to determine if it has found the next block in the sequence. As the module uses linear random order, this rule has been placed at the start of the module. This placement ensures that delivering blocks takes precedence over exploring. The following rule is the result of the agent team iterations, so it is more complicated than strictly necessary for the baseline agent. The reason the rule is in this form is that the agents share code to ensure they are the same, aside from the communication.

```
if a-goal(sequence(Seq)), bel( agentCount(N),
    nextXColoursInSeq(Colors, N, Seq),
    findall(ColorID, (delivering(_,ABlock),block(ABlock,ColorID,_)), TakenC),
    single_subtract(Colors, TakenC, UntakenC), member(Color, UntakenC),
    block(BlockID, Color, Place), not(Place = held) ) then
        adopt( delivered(BlockID) ).
```

However, for the baseline agent team, this rule can be represented in a simpler form. The rule will be discussed again in subsection 4.3 and subsection 4.4 below to explain the rest.

```
if a-goal(sequence(Seq)), bel( nextXColoursInSeq([Color], 1, Seq),
    block(BlockID, Color, _) ) then adopt( delivered(BlockID) ).
```

As long as the agent still needs to deliver colors in the sequence, the rule determines what the first color in the sequence is that was not delivered yet. The color is determined by using the $nextXColoursInSeq(ColourList, X, Seq)$ predicate from the commonKnowledge knowledge module. $ColourList$ is the list of the first X colors in the sequence, excluding the blocks delivered previously. As $X = 1$, the predicate always returns the color that the agent needs to deliver right at this moment. The rule then selects a random block the agent has seen in a visited room with the same color and adopts the goal to have it delivered.

The delivered goal does nothing on its own but is used by the adoptDeliverGoals module to provide the intermediary goals. A random block is chosen because going for the closest block would involve an additional path-finding algorithm if there are multiple blocks of the same color. The agent teams are kept simple on purpose to focus on the communication, and the random choice prevents agents from always picking the same blocks. If there are no blocks of the right color, the rule fails, and the agent will return to the random exploration described earlier.

4.1.2. adoptDeliverGoals

The delivery algorithm works similarly to the exploration algorithm. This module contains rules that are only relevant for later agent teams as well and treats those as before. For the baseline agent team, there are two relevant rules. The first is delivering a block to the dropzone if the agent holds it, and the other is getting to the block that needs to be delivered to pick it up.

```
if a-goal(sequence(Seq)), bel( holding(BlockID), nextColorInSeq(Color, Seq),
    block(BlockID, Color, _), dropZone(Place) ) then adopt( in(Place) ).
```

If the agent is holding the block, check which room is the dropzone, adopt the goal to go there. $BlockID$ is a module parameter with the ID of the block from the deliver goal. Module parameters are available to all rules in the module. As $BlockID$ is already defined, the rule only triggers if the agent is holding the specified block. The main module is responsible for dropping the block once the agent is in the dropzone.

```
if bel( not(holding(BlockID)), block(BlockID, _, Place) ) then
    adopt( in(Place) ).
```

The rule that adopts goals to get to a block makes use of the module parameter, $BlockID$, as well to determine which room the agent should enter. Then the agent adopts the in goal, and the main module takes the goTo action just like for the exploration module. The main module makes sure to pick up the block once the agent is in the correct room.

4.1.3. main

Where the goal modules are about the agent's reasoning, the main module is responsible for acting on its goal. The order of the rules in this module matters as it determines the priority of actions. As such, this module performs the rules in linear order. Random is not used here as the goals already define the specifics of the actions. The rules discussed below are only those relevant to the baseline agent team even though other rules are present in the shared main module.

The first rule is responsible for putting down a block the agent holds once it is in the dropzone and still holding the block. Of course, this is dependent on the block's color, which should be the one that needs to be delivered right now. After the agent performs the putDown action, the agent enters the updateSequence module. This module updates the $sequence$ belief to include the color of the block just delivered. Moreover, it removes the block from the beliefs to prevent the agent from trying to deliver the block again.

Then if the agent is in a room with a block it wants to deliver it should move towards the block and perform the goToBlock block action. The action will fail if the agent is already at the block, and the module will evaluate the following rules. The next rule attempts to perform the pickUp action if the agent is not holding the block specified in the delivered goal. The pickUp action specifies the agent needs to be standing near the block; otherwise, the action fails like the goToBlock action.

Finally, the agent checks if there are any $in(Place)$ goals. If there are any, the agent performs a goTo action. If the agent is already moving somewhere, the goTo action fails. Since this is the last relevant rule, the action failing results in the agent exiting the main module and starting over from the events module. If a rule and corresponding actions are successful, no other main module rules are evaluated. In that case, the agent returns to the event module as well.

4.2. Simple agent team

The first step before introducing communication to a single agent would be to put several instances of the baseline agent in an agent team. This team is called the Simple agent team to indicate it is very similar to the baseline agent without communication. However, the agent program still needs some changes to guarantee that the agent team can always complete the task. The agents need to know when another agent delivers a block. The environment does not track the sequence individually, so the agents have to make sure they are delivering the right block. The environment provides the $SequenceIndex$ percept to keep track of the index, but these agents do not use it. The focus of the experiment is on communication, so the agents will not use the information they cannot directly observe. Therefore, the agents have to inform each other when they deliver the current block in the sequence. If the agents are not aware of the blocks delivered by the other agents, they will all attempt to do the entire sequence on their own. In the best case, the team performs similarly to the single baseline agent. However, as access to rooms is limited, agents can actively interfere with each other and reduce team performance. When multiple blocks of the same color are required to complete the sequence, it could even result in the team never completing the sequence. All available blocks of the required color would have been delivered earlier in the sequence if the agents were not aware another had already delivered it. As a result, this color of block is no longer available when it is needed again in the sequence.

The agents inform each other about the blocks delivered by having the agent send a message to all other agents whenever it delivers a block to the dropzone. The communication clause is part of the

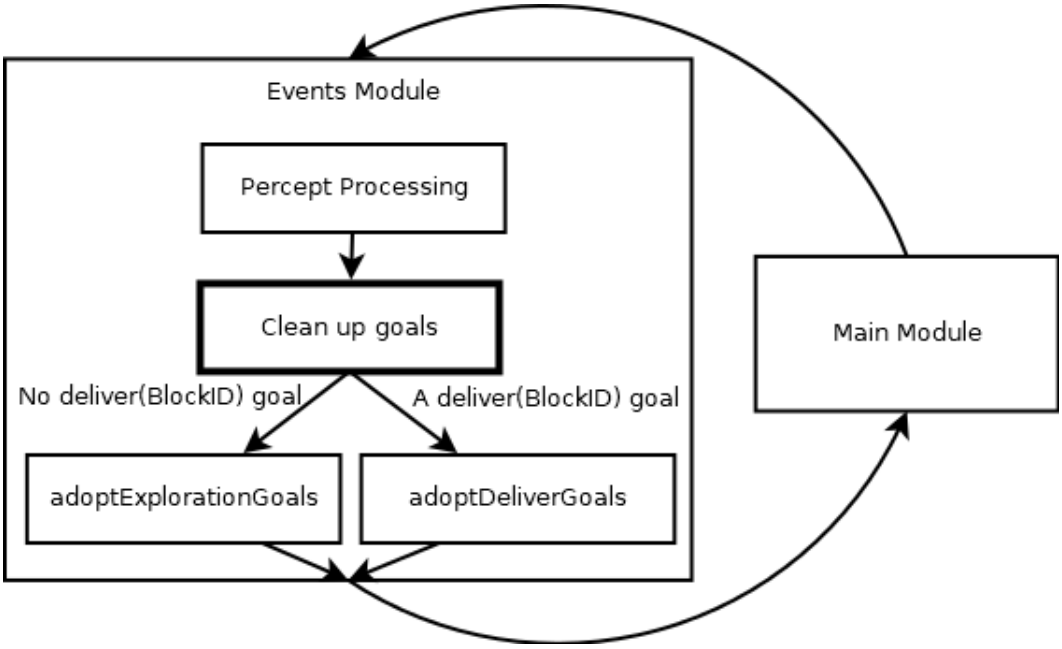

Figure 4.3: A high level flow diagram of how the agent reasoning with the dropping of goals added.

rule in the main module responsible for putting blocks down in the dropzone. The agent sends this message before entering the updateSequence module using the following command:

```
allother.send( msg(delivered(BlockID)) )
```

The agents on the receiving end process the message in a module called comBasic. When the agent receives a message, it is stored as $msg(X)$. In the case of delivered that would be $msg(delivered(BlockID))$. The messages are contained in the msg predicate to make it easier to drop random messages in the experiment. Instead of having to identify all message types individually, the code can check for anything inside a msg predicate.

The simple agent team has a team specific event module that first executes that module before the shared events from the baseline agent. The comBasic module checks if the agent received any delivered messages. If there are any, the agent executes the updateSequence module as in the main module. However, the $BlockID$ passed to it is the one received in the message.

```
forall (Agt).sent(msg(delivered(BlockID))) do updateSequence(BlockID).
```

Here is the reason for having a separate updateSequence module. The agent does the same thing after receiving a message that someone delivered a block as when it delivered the block. The agents were easier to reason about if the sending and receiving agent behaved the same. However, the agent could already have been trying to deliver that block despite the block no longer existing. When that happens, the agent would not know where to go to get the block and get stuck. The event module has an extra section added to prevent that from happening. See figure 4.3. If the agent has a deliver goal, it checks if it is still possible or even necessary to deliver. Another agent might have moved the block; the agent will find out the block is no longer in the room but not know where it is. In that case, the agent will not be able to deliver the block, and it drops the deliver goal. Similarly, if an agent has already delivered the block or the block it wants to deliver does not have the new current color in the sequence, the goal is dropped.

When an agent receives $msg(delivered(BlockID))$ and has completed the updateSequence module, there are four situations it could be in;

- The agent is holding a block, and its color is next in the sequence. It will continue trying to deliver the block.
- The agent is not holding anything and trying to pick up a block. If the next color in the sequence is the same as before, the agent will continue.

- The agent is holding a block, and its color is not the next in the sequence. In that case, the agent follows the baseline exploration algorithm, as described in section 4.1, with one difference. The first time it enters a room, it will drop the block it is holding. The block is dropped into a room to prevent the risk of losing a block that is needed later. All blocks dropped outside of a room will disappear from the environment.

- The agent is not holding anything, and the conditions above do not hold. In that case, the agent continues to follow the baseline exploration algorithm, as described in section4.1.

The agent team needs one more change as without that, the team can get deadlocked. When an agent is without options while inside the dropzone, it will not leave and block access to all other agents. This situation occurs when an agent has already explored all rooms, and other agents have picked up blocks and moved them to a different room. In this situation, the agent does not know where the next needed block is, and there is no next action as far as this agent knows. However, at least one other agent does know and will perform that task. The agent that moved the block, for example, could try to deliver it but then cannot get inside the dropzone since there is already an agent inside. The following line has been added at the end of the adoptGoals module to prevent this deadlock:

```
if bel( dropZone(Place), in(Place) ) then adopt( at('FrontDropZone') ).
```

With these changes, agent teams can always complete the scenario. The agents always clear out the dropzone to make space for the next delivery. There is always one agent who knows the location of any given block after all rooms have been visited. So there will always be at least one agent that will continue working or stop idling to pick up a block of the current color. Even the idling agents get informed when another agent delivers a block, so they are still up to date on the required color. If an agent is idling inside a room that contains a block with the currently required color, it will be aware of that block and try to pick it up so they will not block access to that block. Finally, blocks will not be delivered at the wrong moment as the agents share their progress.

4.3. Room and block coordination agent team

Like the deadlock described in section 4.2, it can happen that an agent will enter a room expecting a specific block to be there when it is not. The agent will update its knowledge on which blocks are present in the room, but the agent still does not know where to find the block. If the agent does not know of any other blocks of the desired color, it will return to the exploration algorithm because it will have no $deliver$ goal. If the agent has already visited the room the block ended up in, the agent has no way to find it. Contrary to the deadlock described before, this situation is not a deadlock as the agent that placed the block will still know the correct location. However, communication can improve performance as delivering the block would no longer depend on a single agent.

The agent team does this by introducing the comBlocks module. This module sends a message for each block it sees for which it gets new information and also processes those messages it gets from other agents. The module is executed after comBasic but before the percept processing. Not only does this module send updates when a block is seen in a different room, but also the first time an agent sees a block. Agent teams can divide the exploration work as agents are aware of the blocks seen by the rest of the team when trying to deliver them.

The code below ensures that all agents in the team listen for updates on block locations:

```
forall (Agt).sent( msg(block(BlockID, ColorID, Place)) ),
    bel( block(BlockID, ColorID, OtherPlace), Place \= OtherPlace ) do
        delete( block(BlockID, ColorID, OtherPlace) ) +
        insert( block(BlockID, ColorID, Place) ).

forall (Agt).sent( msg(block(BlockID, ColorID, Place)) ),
    bel( not(block(BlockID, ColorID, Place)) ) do
        insert( block(BlockID, ColorID, Place) ).
```

These two lines process $block$ messages for two possible cases. The first rule updates block location information in the case an agent already knows of a block with the same $BlockID$. The second rule inserts block beliefs about blocks the agent has not seen before.

Additionally, the agents share if they are holding a block to prevent other agents from trying to pick up that block as well. It updates the $Place$ for the $block(BlockID, ColorID, Place)$ predicate to 'held' once an agent picks up the block. While it would be possible to track which agent holds what block by checking who sent the message, they only need to know if the blocks are available and where they are. As such, there is no tracking of who is holding the block. When the agent drops the block in the dropzone, the block belief treated the same as before. If an agent drops a block in any other room, it sends out an update that the block is there. Below is the code showing how the agents use the $heldblock(BlockID)$ message:

```
forall (Agt).sent( msg(heldblock(BlockID)) ),
    bel( block(BlockID, ColorID, Place) ) do
       delete( block(BlockID, ColorID, Place) ) +
       insert( block(BlockID, ColorID, held) ).
```

One thing to note is that the adoptExplorationGoals module needed to be changed to prevent the agent from trying to deliver blocks already held by another agent. However, the block can still be picked up between adopting the goal and getting to the block. When that happens, the agent would try and deliver a block it cannot locate. To ensure the agent does not get stuck until the current block is delivered, the events module has a new rule which drops the $delivered$ goal in that situation.

Finally, because the agents already share block data, the agents do not need to visit rooms that other agents have visited. The agents already know which blocks are in those rooms. To make sure agents only explore rooms that no other agent entered, the agent team added the comVisited module between the comBlocks module and the events module. This module sends a message when an agent visits a room for the first time. While agents could derive this information from block related communication, being explicit provides more clarity about the intent of the communication.

```
forall bel( msg(_, visited(Place)), not(visited(Place)) ) do
    insert( visited(Place) ).

forall percept(in(Place)), bel( not(visited(Place)), room(Place) ) do
    allother.send(msg(visited(Place)) ).
```

The message sent holds the identical belief used by the baseline exploration algorithm and is added to the beliefs by the receiving agent as if it had visited the room itself.

4.4. Proactive/reactive action coordination agent teams

The communication added in section 4.3 leaves one situation where an agent would do nothing. The team has already visited all rooms and picked up all blocks of the current color in the sequence. The agents not holding blocks do not have anything to do until a new color needs delivering. The agents only try to deliver the current block in the sequence instead of working ahead.

The last two agent teams implement nearly identical planning algorithms to divide the delivery work. The reason for making two teams is because there were at least two distinct communication methods with which the agent teams can coordinate their actions. These two methods should have similar results when the communication has no failures but could differ in performance once communication starts to fail. The proactive agent team is expected to handle communication failure better. However, the goal is to determine if the expectation is correct and to determine how different the performance is. The full details on how the teams communicate are discussed below for both teams. However, this section will first explain the parts of the algorithm that are the same between the teams.

The teams both communicate which block they intend to deliver to the other agents. The communication they share is part of the lookahead module. Here, the agents communicate which block they intend to deliver, $msg(delivering(BlockID))$, and when the agent no longer intends to deliver that block, $msg(notDelivering(BlockID))$. Without additional measures, the agents would broadcast this intent repeatedly. The rule adds a belief when it sends the message that stops it from repeating itself to match how the agents send the other messages. The rule which broadcasts when the agent is no longer going to deliver the block also delivers that belief.

Earlier, section 4.1.1 explained the simplified version of the rule to adopt the $deliver(BlockID)$ goal. The simplified version only looked at the current block in the sequence. However, the full version skips

to the next color if another agent is already delivering it. The rule skips ahead for each color that an agent is delivering. The rule is presented again below to show how to achieve this behavior:

```
if a-goal(sequence(Seq)), bel( agentCount(N),
    nextXColoursInSeq(Colors, N, Seq),
    findall(ColorID,(delivering(_,ABlock),block(ABlock,ColorID,_)),TakenC),
    single_subtract(Colors, TakenC, UntakenC), member(Color, UntakenC),
    block(BlockID, Color, Place), not(Place = held) ) then
        adopt( delivered(BlockID) ).
```

Near the start of the rule is the $agentCount(N)$ belief. This predicate returns the number of agents if the agent team starts with the $lookahead$ belief. If the agent does not have that belief, $agentCount(N)$ returns 1 to match the simplified behavior. Agents can specify an init module, so only the advanced agents have the $lookahead$ belief.

Then $NextXColoursInSeq(Colors, N, Seq)$ returns a list of the colors as $Colors$ that still need delivering up to a maximum of N colors. The number is up to in case there are less than N colors left to deliver. Then the rule creates a list of colors called taken $TakenC$ by finding $delivering(Agent, BlockID)$ beliefs and their colors. The list $Colors$ then has the list $TakenC$ subtracted from it and stores the result as $UntakenC$. This list only contains the colors no other agent is delivering. The agent then picks a random color from $UntakenC$ and tries to deliver a block matching it. The remainder of the rule follows the same principles as with the block com agent.

The adoptDeliverGoals module has only one change: agents that are working ahead will wait in front of the dropzone. Only once the block should be delivered will the agent enter the dropzone to drop it off.

4.4.1. Reactive action coordination agent team

This team is called reactive because the agent sends a message broadcasting which block it wants to deliver but does not act on it. Then the agent reacts to the other agents in the team who give an okay, $msg(doDeliver(BlockID))$, or a denial, $msg(doNotDeliver(OtherBlockID))$. Only once all agents give it the okay will the agent start on the shared algorithm described above. The rules for this behavior are placed after the modules responsible for adopting goals in the events module. This placement allows the rules to use the most up to date goals.

If an agent receives a $msg(delivering(BlockID))$ for a block of a different color, the agent will respond with an okay message. Once an agent receives a message of intent from another agent for the same color, it will check who gets to deliver that block. First, the agent checks if this color is needed again as part of working ahead. If more agents are going to deliver a block of this color than required, the agent will compare its name with the name of the other agent as assigned by the BW4T environment through string comparison. If the other agent's name is greater than its own according to that comparison, that agent gets to deliver the block. The agent doing the check informs the original agent by sending an okay message. The original agent will, in turn, deny the plan of the agent with the smaller name as its name is the greater.

The conflict resolution is a simple comparison in this case to reduce complexity. Another option, for example, could be to compare the distance to the block and dropzone. Alternatively, the agent could use the same string comparison without the okay or denial messages since they already know the agent names. This experiment does include the okay and denial messages to evaluate what happens if the conflict resolution does require communication. Some methods might require knowledge local to a single agent.

4.4.2. Proactive action coordination agent team

The proactive team is proactive because the agents act first and verify if they should after. The agents proactively adopt their goals first and then check if anyone else is doing the same. The rules implementing this behavior are placed in the same location as for the reactive agent team for the same reasons.

The agents use almost the same conflict resolution method as the reactive team. However, instead of sending a message on if it approves or disapproves, the agent drops its own $delivered$ goal when it has the smaller name in string comparison. Then it communicates that it has dropped its goal by sending $msg(notDelivering(BlockID))$. After all, the agent could infer that someone else is doing the

Slow down	Could fail
-	$msg(delivered(BlockId))$ is not received: wrong blocks delivered and lost
$msg(visited(Place))$ is not received: slows down	-
$msg(block(BlockID, ColorID, Place))$ or $msg(heldblock(BlockID))$ is not received: slows down	-
-	$msg(doDeliver(BlockID))$ or $msg(doNotDeliver(BlockID))$ is not received: Might wait for permission forever
-	$msg(delivering(BlockID))$ is not received: Two agents perform the same task and the current color in the sequence is potentially never delivered

Table 4.1: Table of the effects communication failure might have. On the left success is guaranteed if given time. On the right situations may arise where success is no longer possible.

same and will continue to do so. The other agent will do the same check but does nothing as it has the greater name.

4.5. Possible Failure Types

This section discusses table 4.1, which provides an overview of which communication failure scenarios the agent teams could encounter. The table splits the scenarios into communication failures that can slow the agent teams down, and failures where the teams might never finish. Important to note is that even simulation runs with only failures that slow the agent team down might not finish before the simulation run has a timeout. These runs then end up as failed simulation runs in the data. Additionally, the agent team designs do not account for communication failure. The reason for not taking communication failure into account is that the goal of the experiment is first to find the problems. Later experiments can take the problems into account to mitigate some of the effects of communication failure.

The agent team slowdowns occur when the agents miss some of the messages informing them about the blocks that are present in a room. Only agents that are aware of a block can try and deliver it. This lack of information reduces the effective team size as some agents do not know enough to contribute work.

Agent teams fail to complete the sequence for several different reasons. $msg(delivered(BlockID))$ not being received affects all agent teams. When the message does not arrive, the agent loses track of which block in the sequence it should deliver. The agent then wastes a limited resource by delivering colors that were already delivered. As a result, the task might become impossible to finish. The team has used up all blocks of a single color when they could be required later in the sequence. However, without a way to keep track of the sequence outside of communication, the agent will not be aware of this problem. All the information the agent has indicates that it is delivering the right block. Compounding this problem is that when the agent delivers the wrong color, it also sends out a $msg(delivered(BlockID))$. If other agents receive that and they were still aware of the correct color, they will now skip ahead and deliver the wrong colors too.

Additionally, this type of failure can cause a different problem. After delivering an incorrect block, the agent will then broadcast another sequenceUpdate. That message causes other agents who receive the message to skip past a block that would still need to be delivered. This problem only gets worse over time as the agents get more out of sync. Due to this behavior, all agents sometimes believe the team completed the sequence when it has not.

The agent plan negotiation failures cause an agent in the team to wait for an event that never happens. Both 'Advanced' teams try to pick up blocks for delivery in advance, one for each agent in the team. However, two or more agents can still try and deliver the same block if the message with an agent's $delivererering$ intent is lost. The same happens for the advanced proactive team if the message denying that intent is lost. The team tries to work ahead as much as possible with the team size, so

two agents doubling up results in another block in the sequence not getting delivered. When that block is the first block in the sequence, the team ends up waiting forever.

Finally, the Advanced Reactive team has a problem specific to it. Agents ask for permission before trying to deliver a block. However, if one of the responses is lost, the agent that should receive them will never take another action. The team will not finish the task if this is the case for all agents in the team.

5

Experiment Design

This chapter discusses the configuration of the BW4T environment, such as the map topology and block sequences. The selection of agent team sizes and communication failure chances follows after. Then there is a section on the initial experiment design and a section on the changes required to run the experiment in a cloud setup. The next session details the changes made to accommodate for a cloud environment. Finally, this chapter discusses the processing of the log files produced by the experiment to make them suitable for analysis.

5.1. Environment Setup

The experiment will consist of multiple simulation runs, as discussed in section 1.3. The simulations are run using BW4T, the details of which are available in section 2.2. For this experiment, BW4T provides us with the following measurements:

- If the agent team completed the delivery of the sequence before the timeout.
- How long the agent team took to complete the sequence.
- How many blocks have been correctly delivered.
- How many blocks have been incorrectly delivered.
- How many rooms each agent has entered.
- How much idle time each agent had.

The remainder of this section describes the configuration of the BW4T environment. The environment does provide some standard setups by default; however, for this experiment, those were not sufficient. First, this section discusses the environment topology and the design of the maps. Then the section goes into the sequence used by the simulations and how it was chosen. Finally, we go into the optional settings.

BW4T allows for customizing the maps for the agent team simulations. There needs to be at least one room with blocks, a single dropzone, a corridor to connect them, and a spawn point so agent teams can complete their task. Other than these constraints, rooms, and corridors can be freely placed. The only places with blocks inside are rooms.

The configuration of these elements matters because it influences the agent team performance. The minimum amount of work required to complete the task is affected by the size of the map. When an agent tries to visit a room further away, the agent takes more time to reach it. The amount of agent spawn points in a map limits the number of agents we can use. The effectiveness of communication is influenced by how concentrated the blocks are. If the blocks are spread out, a single agent will need to look around more for information and have to travel longer if tasks are not divided.

For this specific experiment, the focus was on map variations that affected communication. The position of the dropzone and the rooms are constant, as is the number of rooms. There only needs to

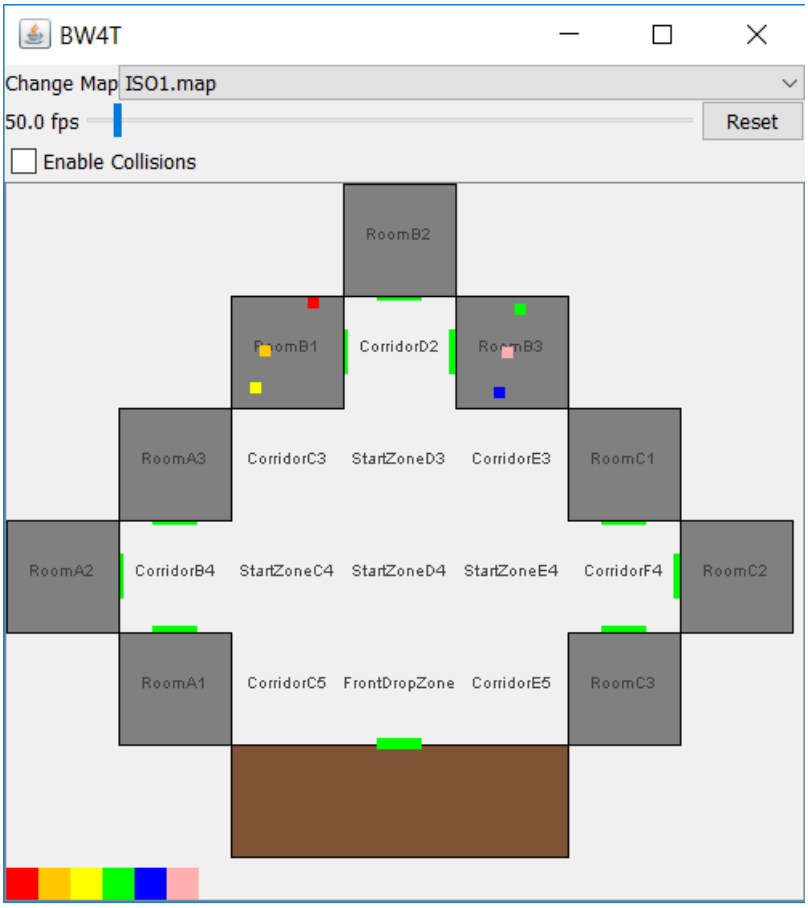

Figure 5.1: Screenshot of the BW4T map topology used.

be enough rooms that communication helps. However, the distance between them and the dropzone should be the same to limit the effects of distance on performance. For this reason, the maps have nine rooms that are equidistant to the dropzone. The number of rooms is equal to the number of rooms in the standard BW4T maps.

When looking at block placement on a BW4T map, there are the following variables to consider:

- The degree of block dispersion
- The number of redundant blocks

The first refers to the method for distributing blocks around the map. There are limits on how many blocks can be in a room at the start of the simulation run. However, there can be multiple blocks of the same color in a single room. Blocks can have a high degree of dispersal, the blocks are spread out over all rooms, or a low degree, the blocks are concentrated in one room. The latter case depends on the total number of blocks not exceeding the room limit. If the blocks are all in just a few rooms, the agents will have more variance in how quickly the agents find the blocks. Additionally, agents are more likely to have to wait for another agent to exit a room as only one agent can be inside at the same time. In the case where the blocks are all spread out over the different rooms, individual agents are more likely to find relevant blocks when visiting a random room. However, if something goes wrong with communicating the contents of a room or the agent team does not communicate that the knowledge between agents differs more. The experiment includes both options as follows: the blocks are divided over two rooms or spread over as many rooms as possible. In the first case, all blocks of the same color are in the same room. Two rooms are the minimum, as that means no single agent automatically knows of all blocks. Blocks of the same color share a room for the same reason. In the latter case, no two blocks of the same color are inside a single room if there are more blocks than rooms. If blocks of the same color share a room, then it mitigates the purpose of dispersing the blocks. Any random room

is less likely to contain a relevant block then when blocks of the same color are together. Additionally, any given room would contain fewer different blocks, and the agent would have to explore more rooms to see all relevant colors.

The number of blocks refers to how many blocks there are of each color. If there are more blocks of a single color than there are needed to finish the sequence, the extra blocks are called redundant. When there are redundant blocks, the agent team can make mistakes while still being able to complete the sequence. This experiment considered the following types of redundancy: no redundancy, one redundant block per color, and two redundant blocks per color. Higher redundancies were considered, however, due to limitations on the number of blocks in a room that was not possible.

For the sequence options, there are two categories: all blocks are the same, or the colors are mixed. When all blocks are the same color, it does not matter in what order they are delivered, as the sequence only cares about the color. Such a sequence means that the current block in the sequence does not matter; only if the agent team finished the task or not. As a result, the agents do not have to communicate about the intermediate steps. Whereas the other option, a mixed color sequence, does require agents to communicate about intermediate task completion. As communication is the focus of this experiment, all maps have a sequence where every block is a different color. The length of these sequences determines how much work the agents have to do. However, as long as there is more than one block, the length does not affect the types of communication used. For the experiment, the maps stick to the default length, six blocks, used for the maps that come with the environment.

With one sequence and three redundancy levels and two block placement methods, the experiment uses a total of 6 different maps for every combination. These maps look like the map in figure 5.1 and differ only in block placement. These maps are called ISO1 to ISO6. The first three have the blocks spread out over two rooms. ISO4 to ISO6 have the blocks spread out over as many rooms as possible. ISO1 and ISO4 have no redundancy, but redundancy increases as discussed before for the higher numbered maps. So ISO2 and ISO5 have two blocks of every color while ISO3 and ISO6 have three.

5.2. Agent Team size

GOAL agents are organized in agent teams, the size of which can be adjusted. The agent team's size specifies the number of agents present in the simulation. GOAL itself does not pose limits on the number of agents. However, the available computing power and BW4T maps do limit how large teams can get. Additionally, implicit map limitations might limit the effectiveness of large teams. Rooms only allow one agent to enter them at the same time. As a result, teams with more agents than the number of rooms available or than there are blocks in the sequence benefit less from increases in team size.

Testing every team size is not possible due to time constraints. Therefore, his experiment uses a sampling of the team sizes to determine if team size interacts with communication failure effects. It could be that larger teams experience different effects or effects at different strengths than smaller teams. A team of only one agent is the smallest team size possible, but that does mean there are no agents to communicate with. However, it will still be useful to run the simulations. However, a team of one agent would provide a performance baseline for the agents with communication. Each agent team should use less time to complete the sequence than the baseline.

The next team size used for the experiment are teams of three agents. Three is the minimum amount of agents needed to have a team where it is possible to communicate with more than one agent at the same time. There need to be at least two agents receiving a message to have a chance of individual agents to receive different subsets of the communicated information. So agent A might not receive a message while agent B does.

The third team size is five agents. This number is just over half the amount of rooms available in the default map setup and one less than the number of blocks required for the sequence. As a result, the method for dividing the tasks within the team matters. Not every agent can continue exploring after just one set of rooms, and the same holds for the sequence. Even with the highest number of redundant blocks, there are not enough to have every agent try to deliver a given color in the sequence.

The final team size is eight agents, one agent less than the number of rooms in the chosen maps. Experience with the environment has shown that 8-10 agents start to tax the hardware available. When the hardware is overloaded, the simulation results might be distorted. However, maps have nine rooms, so it is possible to make teams with a size close to the number of rooms available. As mentioned above, it might be interesting to see how that influences the agent team behavior.

5.3. Communication

The failure mode used in the experiment is a fixed percentage chance of receiving failure. However, that percentage could be any number between 0 and a 100% of the messages failing. Just like with team sizes, a sampling of failure percentages will suffice to identify trends. The sampling will use 25% increments starting at 0%. However, no messages failing and all messages failing could have different results from only a few or nearly all messages failing. Therefore 5% and 95% will be tested in addition to the 25% increments. Making the complete range of tested percentages $0, 5, 25, 50, 75, 95$, and 100%.

5.4. Experiment Setup

As specified by earlier in this chapter, each simulation uses a combination of the following experiment variables: Teams of 3, 5, or 8 agents. The environment uses one of six maps. These six maps have the blocks concentrated in two rooms or dispersed over all rooms, and both have three levels of block redundancy. Finally, agents might not receive a message sent with a chance of 0%, 5%, 25%, 50%, 75%, 95%, or 100%. The experiment repeats every permutation of these three variables several times. Additionally, the baseline team of one agent has simulation runs for all six maps. The baseline agent only runs with 0% communication failure as there are no other agents with which to communicate. Table 5.1 provides an overview of all the parameters excluding the baseline agent as it was not run with all communication failure chances.

Table 5.1: An overview of which variables were used for the experiments. For every permutation of these variables the experiment ran multiple simulations.

Agent Team	Simple agent team		Room and block coordination team		Proactive action coordination agent team		Reactive action coordination agent team
Map							
	No redundancy		One set of redundant blocks		Two sets of redundant blocks		
Concentrated blocks	ISO1		ISO 2		ISO3		
Dispersed blocks	ISO4		ISO5		ISO6		
Failure chance	0%	5%	25%	50%	75%	95%	100%

Simulation runs need repetition because of the random elements of the experiment, like the random chance of communication failure. Additionally, all agent teams have some random aspects as well. When an agent has multiple valid options, it chooses one at random. As there is no prior knowledge on the effects of the random components, a desired experimental run time was used to determine the number of repetitions performed. Based on the baseline agent doing 40 repeats of every map with no timeout, a successful run will not take more than 110 seconds. The highest number in the results we got was just under 99 seconds, and we added a margin in case communication failure increases the run time above that of a single agent. When each simulation runs with the 110-second timeout, it takes 231 minutes, 3(teams)$*6$(maps)$*7$(failure chances)$*110$(timeout), to do every combination of experiment variables once if every simulation times out. If all three agent teams run in parallel for one week, the full set of simulations can be repeated about 40 times, $24*60*7$(one week)$/231 = 43.6$. The number was rounded down to allow for system setup and data retrieval. Forty simulation runs of the single baseline agent add 440 minutes, 6(maps)$*110$(timeout)$*40/60$, which does not notably affect the total runtime.

The experiment itself will be set up under the following conditions:

- All the simulations will be run on identical virtual private servers (VPSs).
- The simulations will be run on a dedicated instance that only has the operating system running in addition to the simulations.
- Only one simulation is run at the same time.

These conditions exist to prevent the limit on available processing power on the system from influencing simulations, due to the software or capabilities differing between machines. Each agent team will use a duplicated instance of TU Delft VPSs running in parallel. The servers were Windows 10 instances that had java installed. As the instances are virtualized in the university cluster, their performance is identical, which satisfies the first condition. This setup runs slower than simulations on a non-virtualized machine. However, being able to parallelize means less time is spent waiting for all the results.

The simulations will be run through a java interface to automate the entire process. It will gather and group the server logs for each set of 40 simulation runs with the same variable set. Every set uses a new instance of the GOAL runtime per simulation run. The Java interface needs to manage the timeout and repeats as the GOAL runtime did not support the combination of the two. The BW4T server gets restarted when switching to a new set of simulation runs.

5.5. Corrections made to experiment setup

Running the simulations in the cloud came with some surprises. As mentioned in section 5.4, the simulations took longer to run in the cloud than they did on a desktop, but the parallelization makes up for that. However, this difference in performance revealed some shortcomings of the BW4T simulation environment. While running agent teams on the VPSs, problems occurred, which did not happen on physical machines.

One such issue had an agent attempting to pick up a block in a room lying in one of the edges of a room. If approaching the block took too long, the agent repeated the move to block command. This command caused the agent to reset its position in the center of the room. Then, the agent once again attempts to move to the block. Only to be interrupted by the command again, and this loop repeated ad infinitum. On faster machines, the block does get picked up quickly enough to prevent such a loop from occurring. This fix for the loop was to introduce code that ensures the agent will not repeat the command until the agent is at the block's location.

```
% In the actions module a pre- and post-condition was added: goingToBlock.
define goToBlock(BlockID) with
    pre {
        in(_), not( state(traveling) ),
        not( atBlock(BlockID) ), not( goingToBlock(_) )
    }
    post { goingToBlock(BlockID) }

% In the events module the following line was added:
forall percept( atBlock(BlockID) ) do delete( goingToBlock(BlockID) ).
```

By checking in the action preconditions, if an agent already believes it is $goingToBlock$, we can prevent the action from being repeated. The postcondition makes sure that the agent adopts the belief $goingToBlock$ with the BlockID towards which the agent is moving. The ID is used in the events module to check if the agent has arrived at that specific block so that the belief can be deleted. In that case, the agent is near the block and picks it up rather than performing the go to block command again.

The other issue was harder to identify but of a similar nature. Figure 5.2 illustrates how an agent sometimes gets stuck in the doorway to a room. In that situation, the agent has seen the room contents and tries to act on it by picking up a block. However, BW4T has not yet registered the agent as being inside the room. As a result, the agent tries to perform an illegal action: pick something up from a room without being inside. However, the agent does stop as it got a new move action. Then the agent keeps repeating the illegal move action, which results in the agent not doing anything.

An agent state check was added to the event specification for the $goToBlock$ action shown earlier to fix this issue. Checking the state to see if the agent is traveling ensures that the agent can only call the action can only once it has arrived inside the room. At that point, the belief $state(traveling)$ changes to $state(arrived)$ because the agent perceives a percept from the environment that the state has updated. The agent is then guaranteed to be inside the room, and it can pick up the block without any issue.

After fixing these two issues, there was still a small chance ($< 1\%$) of an issue occurring. The small chance of failure was found after a test run of the different agents running in parallel for a day

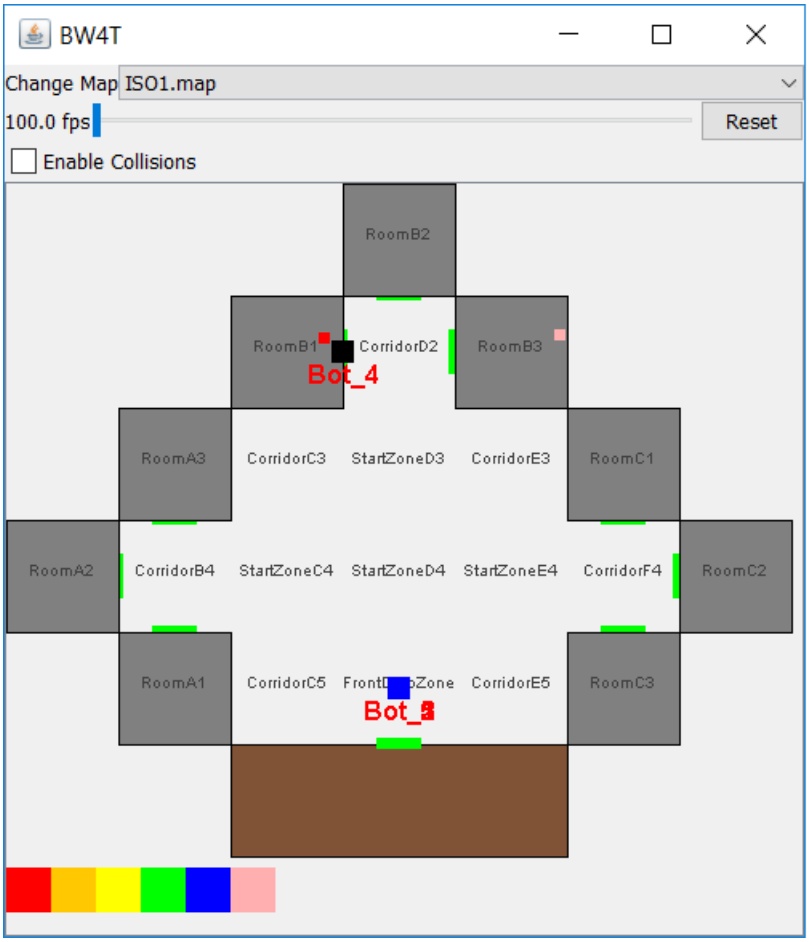

Figure 5.2: Screenshot of BW4T where an agent tries to pick up a block from a room it has not yet fully entered.

without communication failure. However, the rarity of these cases made the problem hard to diagnose. The problem appeared to be another timing issue, but it did not seem to occur when using debugging tools to find the cause. While it would have been easy to exclude the data from the test runs without communication failure, it is not so easy for those with communication failure. The difference between a simulation run that failed because of lost messages, and this issue was not clear. The rarity of the problem does mean the effect on the results of the data analysis is minimal and not fixed as a result. Simulation runs with this issue for the no communication failure baseline have not been removed either. The inclusion ensures that the effect on the data is the same for each set of simulation runs

5.6. Data processing

The data used for determining how well the agents perform under the various experiment parameters comes from the logs the BW4T server keeps. These logs were processed to extract the measurements of interest. First, this section will explain how the log files get stored, then what they contain, and finally, how they were processed for use by data analysis tools.

5.6.1. BW4T Server Log File Organisation

BW4T names logs based on the time when the server started and the amount of simulation runs since starting the server. Every time a new simulation run is started, the previous logs get renamed as follows: If there is no number appended to the filename yet, it appends ".1". Otherwise, it ups the number of all logfiles by 1. So <server start date and time>.1 would become <server start date and time>.2. After running all simulations, the log file with the highest number appended to it is the oldest.

The filenames do not contain the experiment parameters, so the log files were stored in a directory structure that encodes those. Each set of logs with the same parameters was placed in the same

directory when running the simulations. The logs first get grouped by agent team, then by map, followed by agent count, and finally, the percentage of communication failure as an integer. For example, the logs for simple_team running on map ISO3 with five agents and a 25% communication failure chance are stored in ”simple_team/ISO3/5/25”.

5.6.2. BW4T Server Log File Contents

Each log file contains several lines prefixed with the time precise to the millisecond. The content of the file is as follows in order:

- A message indicating the simulation run started
- Multiple lines that provide the name of each room and the blocks inside it. The blocks are identified by a capital letter indicating the color
- A line specifying the colors of the sequence the agent team needs to complete.
- Multiple lines specifying one of two things:
  - The action an agent has performed. The line identifies the agent and the action they performed, including the arguments.
  - A notification that a specific agent collided. In our setup, that means the agent collided with a closed door.
- Once the sequence is completed, or the simulation run timeout is reached, there is a line with the total runtime of the BW4T server so far in minutes and seconds.

Additionally, there is information that only gets added to the logs if the agent team successfully delivers the sequence. The added information is a set of statistics per agent with the following information:

- The bot handicap. Bots can be configured to have handicaps like lower movement speed or an inability to hold things. For these experiments, this should always be none.
- The number of correct block deliveries.
- The number of incorrect deliveries.
- An amount related to messages sent by human agents. Always zero in this experiment because no human agents were used.
- The number of times the agent entered a room.
- The time in seconds, with decimals for milliseconds, that the agent has been idle.

5.6.3. Data Extraction Process

The logs are processed to extract the relevant measurements and save those as a CSV file[21] to make them more accessible for analysis. The extraction was done using a python script that iterates over all log files, using the directory structure to match the log files to the experiment parameters.

The script goes over the contents of each log file to first determine the time the simulation run started by checking the timestamp of the first line in the log file. The script skips over the following lines until it reaches the simulation run end message. The timestamp of that message is used to determine the run time by taking the difference with the start time. Calculating the run time corrects for the simulation run going past midnight into the next day when calculating the difference. In case the agent team did not succeed, the runtime is specified as -1 to indicate the timeout cut off the simulation run. If the agent is successful, the script parses the following agent statistics: the number of correct and incorrect deliveries, the number of times an agent entered a room, and the idle time. These statistics are then summed up to keep the results per simulation instead of per agent.

Finally, the processed data is output in the CSV file. It has one entry per simulation run with the following columns in this order: agent_team, map, agent_count, com_failure, run_time, idletime, good_drops, wrong_drops, rooms_entered. agent_team specifies which agent team ran in the simulation. map is the identifier (ISO1-6) of the map used for the simulation. agent_count is an integer (3, 5, or

8) with the number of agents in the team. com_failure is an integer indicating the percentage chance of a message not arriving, so 50 would mean a 50% chance of failure. run_time is a decimal number indicating the number of seconds the agent team took to complete the sequence where a -1 indicates a failure. idle_time is the same as run_time but indicates how long the agents have spent idle. good_drops is an integer indicating how many blocks the team delivered correctly where a -1 indicates the run failed. wrong_drops is the same as good_drops but indicates how many blocks were incorrectly delivered. rooms_entered is also the same as good_drops but indicates how often the agents in the team entered a room.

6 Analysis

This chapter presents the results of the experiment described in chapter 5. First, it provides an overview of the data for the baseline agent team and then the other agents. Then the agent team performance without communication failure is compared to show that the communication provides performance benefits if no failure occurs. Following that is a section which examines the task success rate as communication failure is introduced, and finally, the results for the effects of communication failure on the agent team performance.

6.1. Descriptives

Table 6.1: Single-Agent Baseline descriptive statistics for run time, time spent idle, and rooms entered.

	N	Run time (s)		Time spent idle (s)		Rooms entered	
		Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)
Map							
ISO1	40	77.6 (10.0)	79.9 (54.1 - 94.4)	2.97 (0.335)	3.05 (2.17 - 3.51)	17.1 (2.10)	17 (12 - 20)
ISO2	40	79.7 (7.69)	81.0 (56.8 - 93.6)	3.04 (0.254)	3.07 (2.31 - 3.55)	17.6 (1.68)	18 (13 - 20)
ISO3	40	79.1 (12.1)	80.0 (54.7 - 97.0)	3.01 (0.396)	3.02 (2.23 - 3.62)	17.0 (2.45)	17 (12 - 20)
ISO4	40	83.9 (5.80)	84.5 (68.3 - 95.6)	3.17 (0.202)	3.20 (2.64 - 3.57)	18.5 (1.06)	19 (16 - 20)
ISO5	40	73.0 (7.64)	72.7 (56.9 - 89.8)	2.80 (0.254)	2.77 (2.25 - 3.38)	15.9 (1.28)	16 (13 - 19)
ISO6	40	68.3 (6.02)	68.3 (54.2 - 79.1)	2.65 (0.207)	2.64 (2.19 - 3.01)	14.8 (1.16)	15 (13 - 17)
Overall	240	76.9 (9.82)	77.9 (54.1 - 97.0)	2.94 (0.328)	2.97 (2.17 - 3.62)	16.8 (2.05)	17 (12 - 20)

For the baseline team, there are 240 observations divided equally between the six maps. The first three maps (ISO1-3) have the blocks clustered in two rooms. The following three maps (ISO4-6) spread the blocks out over all the rooms. Each set of three agents starts with no redundant blocks and then add more redundant blocks as the map numbers increase. Only one agent was present for the baseline, and therefore, no communication failure could occur. As such table 6.1 is only grouped by maps. The table does not list incorrectly delivered blocks because there were none. The table shows some differences between the maps, ISO5 and ISO6 have a lower mean and median for runtime, time spent idle, and rooms entered. Lastly, the single-agent baseline still has some time spent idle due to the processing limitations of the cloud platform. The time required to process the percepts when the agent arrived, to the agent taking the next action, resulted in around three seconds total of the agent being idle per simulation run.

The results for the other agent teams are presented in two ways, the task completion rate and performance indicators. The split exists because the performance indicators are only available when the agent team completes its task. The task completion results are a mapping of run times to success/failure. A run time indicating the agent never finished, encoded as -1 in the data, gets mapped to $Failure$, and every other run time maps to $Success$.

There were a total of 19262 observations, of which 7461 were successful. The first thing to note in table 6.2 in the overall column is that not all variables are the same size. When looking at the agent team group, for example, ”simple agent team” has 5000 observations while ”room and block

Table 6.2: Task completion rate

	Failure (n=12316)	Success (n=7551)	Overall (n=19867)
Agent Team			
proactive action coordination agent team	4175 (84.3%)	776 (15.7%)	4951
reactive action coordination agent team	4219 (85.5%)	713 (14.5%)	4932
room and block coordination agent team	1975 (39.6%)	3009 (60.4%)	4984
simple agent team	1947 (38.9%)	3053 (61.1%)	5000
Map			
ISO1	2252 (68.2%)	1049 (31.8%)	3301
ISO2	1911 (57.7%)	1401 (42.3%)	3312
ISO3	1708 (51.6%)	1605 (48.4%)	3313
ISO4	2594 (78.7%)	701 (21.3%)	3295
ISO5	2148 (64.9%)	1161 (35.1%)	3309
ISO6	1703 (51.0%)	1634 (49.0%)	3337
Number of agents in team			
3	3499 (52.7%)	3135 (47.3%)	6634
5	4143 (62.6%)	2478 (37.4%)	6621
8	4674 (70.7%)	1938 (29.3%)	6612
Receive failure chance (%)			
0	19 (0.7%)	2861 (99.3%)	2880
5	2042 (72.1%)	792 (27.9%)	2834
25	2060 (72.9%)	765 (27.1%)	2825
50	2063 (72.9%)	766 (27.1%)	2829
75	2029 (71.6%)	803 (28.4%)	2832
95	2063 (72.9%)	767 (27.1%)	2830
100	2040 (71.9%)	797 (28.1%)	2837

coordination agent team” only has 4984 observations. This difference is expected due to limitations in the environment. When the last simulation run fails, the environment never finishes the simulation run before it is shut down. As such, the log file is incomplete, making it impossible to determine if an error occurred. These simulation runs are not in the data as a result. Including the last run, if successful, slightly skews the data towards successes. However, as the effect is limited, no attempt was made to compensate.

When looking at the task completion rate in table 6.2 per agent team, the action coordination agent teams perform worse than the other teams. Additionally, the task completion rate in the same table does not seem to go down as the chance for communication failure goes up. The presence of communication failure is what matters as only the simulation runs without failure have a notably higher task completion rate.

For the successful runs in table 6.5 and 6.6, it can be seen that the action coordination agent teams are generally quicker to finish the task without communication failure than the other agent teams. When there is no chance of communication failure, these teams have mean run times of 29.9 and 29.3 rather than the 46.3 for ”room and block coordination agent team” and 52.0 for ”simple agent team”. When communication failure does occur, the average run time goes up for all agents and are all ≈70 regardless of communication failure chance percentage for ”simple agent team” and ”room and block coordination agent team”. However, the action coordination agent teams do not perform as well. These have run time means for the different failure percentages of ≈75 or more compared to run times around 70 for the other two agents. The limited number of successes for the action coordination agent teams might be skewing that data.

The idle time of the action coordination agent teams, on the other hand, with averages of 31.0 and 30.7, is a bit better than 34.3 for ”room and block coordination agent team”. Yet significantly worse than ”simple agent team” with 11.0. The lower idle time might be related to ”simple agent team” requiring every agent to investigate the map as that information is not shared. What should be noted is that all teams, other than ”simple agent team”, have less idle time once communication failure occurs. The

Table 6.3: ”Simple agent team” descriptive statistics for run time, time spent idle, incorrectly delivered blocks and rooms entered.

	Run time (s)		Time spent idle (s)		Incorrect Deliveries		Rooms Entered	
	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)
Map								
ISO1 (n=422)	64.8 (11.1)	64.4 (39.8, 96.1)	10.3 (4.37)	9.31 (4.85, 23.6)	0 (0)	0 (0, 0)	49.2 (16.3)	48 (28.0, 88.0)
ISO2 (n=571)	66.6 (11.4)	67.4 (38.8, 98.6)	10.9 (4.18)	9.83 (4.42, 23.7)	2.79 (1.76)	3 (0, 5)	50.7 (14.9)	49 (22.0, 83.0)
ISO3 (n=665)	67.6 (11.4)	68.5 (36.4, 94.1)	12.1 (4.90)	10.9 (5.03, 69.3)	6.12 (3.27)	7 (0, 10)	54.5 (13.6)	52 (27.0, 90.0)
ISO4 (n=232)	69.1 (13.6)	69.1 (40.3, 94.9)	10.9 (5.06)	9.55 (5.27, 31.3)	0 (0)	0 (0, 0)	51.3 (18.7)	43 (31.0, 103)
ISO5 (n=464)	67.7 (12.7)	68.2 (42.8, 105)	10.2 (4.28)	8.97 (3.18, 32.6)	2.25 (1.62)	3 (0, 5)	47.6 (13.9)	42 (16.0, 81.0)
ISO6 (n=699)	63.5 (10.6)	64.2 (35.1, 99.9)	11.6 (5.37)	10.7 (4.50, 89.4)	5.66 (2.84)	7 (0, 10)	52.5 (14.4)	51 (23.0, 89.0)
Number of agents in the team								
3 (n=1331)	69.7 (10.7)	69.1 (38.6, 105)	7.41 (1.39)	7.33 (4.42, 17.3)	3.76 (3.38)	3 (0, 10)	38.2 (5.49)	38 (22.0, 56.0)
5 (n=996)	65.3 (11.3)	64.9 (35.1, 102)	11.3 (1.89)	11.1 (7.06, 32.6)	3.49 (3.19)	3 (0, 10)	52.4 (4.82)	52 (36.0, 69.0)
8 (n=726)	61.1 (11.9)	62.8 (36.4, 97.0)	17.7 (4.35)	17.5 (3.18, 89.4)	3.01 (3.11)	3 (0, 10)	73.7 (7.32)	74 (16.0, 103)
Receive failure chance (%)								
0 (n=717)	52.0 (7.71)	50.9 (35.1, 76.5)	11.0 (4.20)	10.0 (4.42, 32.6)	0 (0)	0 (0, 0)	52.6 (16.7)	51 (22.0, 103)
5 (n=396)	70.8 (8.91)	69.5 (52.5, 97.0)	11.4 (5.40)	10.1 (5.31, 69.3)	4.46 (2.97)	5 (0, 10)	51.0 (14.4)	49 (30.0, 89.0)
25 (n=382)	71.3 (9.22)	70.3 (53.9, 105)	10.9 (4.59)	10.1 (5.31, 69.3)	4.51 (3.01)	5 (0, 10)	50.2 (14.6)	48 (30.0, 90.0)
50 (n=381)	70.0 (8.76)	69.1 (55.5, 104)	11.2 (4.64)	9.84 (5.28, 29.4)	4.68 (3.09)	5 (0, 10)	51.8 (14.9)	50 (28.0, 84.0)
75 (n=406)	70.6 (8.39)	69.6 (55.4, 95.0)	11.2 (4.49)	10.1 (3.18, 23.9)	4.47 (3.02)	5 (0, 10)	51.1 (14.5)	49 (16.0, 88.0)
95 (n=377)	70.4 (8.86)	68.7 (55.7, 95.7)	11.3 (5.93)	9.99 (4.85, 89.4)	4.69 (2.96)	5 (0, 10)	50.8 (14.2)	49 (29.0, 85.0)
100 (n=394)	70.4 (9.16)	68.9 (53.7, 100)	11.0 (4.55)	9.74 (4.90, 25.3)	4.58 (3.01)	5 (0, 10)	50.5 (14.7)	49 (28.0, 86.0)
Overall								
(n=3053)	66.2 (11.7)	66.2 (35.1, 105)	11.1 (4.79)	9.92 (3.18, 89.4)	3.49 (3.27)	3 (0, 10)	51.3 (15.1)	49 (16, 103)

Table 6.4: ”Room and block coordination agent team” descriptive statistics for run time, time spent idle, delivered blocks, and rooms entered.

	Run time (s)		Time spent idle (s)		Incorrect Deliveries		Rooms Entered	
	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)
Map								
ISO1 (n=387)	61.7 (13.2)	63.5 (36.8, 95.0)	25.8 (30.1)	11.2 (3.34, 129)	0 (0)	0 (0, 0)	40.9 (15.7)	34 (17, 79)
ISO2 (n=576)	65.2 (13.2)	66.8 (34.7, 98.4)	15.3 (13.4)	10.6 (3.74, 78.3)	2.78 (1.78)	3 (0, 5)	46.6 (15.6)	41 (19, 81)
ISO3 (n=745)	66.1 (13.1)	67.8 (32.3, 94.5)	12.8 (5.56)	11.2 (5.04, 44.9)	6.13 (3.22)	7 (0, 10)	52.0 (14.6)	50 (21, 89)
ISO4 (n=230)	65.2 (16.8)	63.6 (34.6, 95.8)	36.3 (33.5)	22.8 (4.69, 130)	0 (0)	0 (0, 0)	36.3 (10.4)	34 (22, 77)
ISO5 (n=458)	65.5 (13.5)	66.9 (36.3, 104)	13.4 (9.25)	10.2 (5.41, 51.9)	2.26 (1.62)	3 (0, 5)	44.0 (13.4)	40 (23, 81)
ISO6 (n=684)	62.9 (10.6)	63.6 (34.1, 103)	12.4 (5.75)	11.1 (4.68, 39.3)	5.57 (2.84)	6 (0, 10)	50.2 (14.9)	47 (22, 88)
Number of agents in the team								
3 (n=1301)	68.4 (11.5)	68.6 (36.3, 104)	8.82 (5.30)	7.59 (3.34, 43.4)	3.79 (3.36)	3 (0, 10)	36.7 (6.86)	37 (17, 56)
5 (n=983)	63.4 (12.6)	64.5 (34.4, 103)	16.4 (13.3)	11.8 (7.95, 79.8)	3.48 (3.13)	3 (0, 10)	47.9 (9.53)	51 (22, 68)
8 (n=725)	58.8 (14.1)	62.0 (32.3, 99.7)	31.4 (26.1)	19.9 (11.9, 130)	3.08 (3.12)	3 (0, 10)	63.1 (17.7)	73 (29, 89.0)
Receive failure chance (%)								
0 (n=712)	46.3 (6.56)	45.8 (32.3, 68.6)	34.3 (29.1)	24.4 (4.50, 130)	0.00 (0.00)	0 (0, 0)	32.7 (6.00)	32 (21, 51)
5 (n=382)	70.7 (9.33)	69.4 (50.5, 104)	11.2 (4.57)	10.4 (3.34, 27.2)	4.51 (2.92)	5 (0, 10)	50.4 (14.3)	49 (17, 84)
25 (n=374)	70.3 (8.94)	69.0 (53.9, 100)	11.1 (4.61)	9.86 (5.02, 27.7)	4.64 (2.99)	5 (0, 10)	50.4 (14.3)	49 (29, 86)
50 (n=378)	69.4 (9.02)	67.6 (53.9, 103)	11.0 (4.53)	9.42 (5.24, 30.1)	4.71 (2.84)	5 (0, 10)	51.4 (14.8)	50 (29, 88)
75 (n=383)	69.9 (8.45)	68.8 (54.2, 95.3)	11.4 (4.86)	10.1 (5.06, 29.6)	4.63 (3.04)	5 (0, 10)	51.3 (14.8)	50 (29, 89)
95 (n=382)	70.2 (8.76)	68.8 (54.8, 95.7)	11.6 (5.51)	10.4 (5.21, 68.3)	4.57 (3.03)	5 (0, 10)	51.4 (14.8)	49.5 (29, 89)
100 (n=398)	70.0 (8.45)	69.0 (55.4, 97.7)	11.5 (4.89)	10.1 (5.03, 29.6)	4.59 (2.95)	5 (0, 10)	51.5 (15.3)	49 (30, 85)
Overall								
(n=3009)	64.5 (13.1)	65.5 (32.3, 104)	16.7 (17.7)	11.0 (3.34, 130)	3.52 (3.24)	3 (0, 10)	46.7 (15.3)	43 (17, 89)

decrease in idle time might be because the agents spend more time trying to find information that got lost.

Tables 6.3 and 6.4 show ”simple agent team” and ”room and block coordination agent team” have a median of 3 blocks incorrectly delivered. The action coordination agent teams will not be discussed here due to the limited number of successful runs. The grouping by map shows that the different maps have different amounts of incorrectly delivered blocks. The increase in incorrectly delivered blocks corresponds with a higher success rate in table 6.2. When looking at the number of blocks delivered for the different numbers of agents, the median lowers when the number of agents in a team increase. The median goes from 3 for three agents to 0 for eight agents, though the means are closer with ≈3.8 for three agents and ≈3.1 for eight agents.

6.2. Comparison of Agent Team Baselines

As can be seen in tables 6.1, 6.3, 6.4, 6.5, and 6.6 all agent teams are quicker to complete the task than the single-agent baseline when no communication failure occurs. Additionally, a difference in the effectiveness between the agent teams exists as intended. As detailed in chapter 4, the agent teams build upon each other to improve upon the agent with less communication. The two action coordination agent teams had a similar performance because they differ in how they resolve conflicts, not in their degree of communication.

Figure 6.1 shows that the random factors involved in each run do have a noticeable effect on the run time. However, adding agents to the team does reduce the variance. However, the reduction in

Table 6.5: ”Proactive action coordination agent team” descriptive statistics for run time, time spent idle, incorrectly delivered blocks, and rooms entered.

	Run time (s)		Time spent idle (s)		Incorrect Deliveries		Rooms Entered	
	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)
Map								
ISO1 (n=120)	31.3 (5.63)	31.2 (18.3, 50.1)	30.1 (15.9)	27.5 (3.92, 65.3)	0.00 (0.00)	0 (0, 0)	22.9 (3.82)	23 (15, 32)
ISO2 (n=135)	37.0 (16.7)	31.2 (19.5, 89.7)	27.3 (18.0)	22.5 (4.16, 93.6)	0.0519 (0.254)	0 (0, 2)	23.7 (3.85)	23 (13, 37)
ISO3 (n=148)	42.6 (21.5)	34.9 (19.2, 94.5)	27.4 (18.9)	22.6 (3.41, 83.5)	0.122 (0.480)	0 (0, 4)	25.5 (6.10)	24 (15, 51)
ISO4 (n=120)	30.6 (8.99)	28.5 (14.5, 54.4)	34.7 (19.7)	29.8 (4.04, 94.2)	0.00 (0.00)	0 (0, 0)	20.9 (2.59)	20 (17, 31)
ISO5 (n=120)	27.6 (8.09)	25.6 (15.2, 51.0)	31.3 (15.2)	29.3 (5.14, 93.9)	0.00 (0.00)	0 (0, 0)	19.8 (3.27)	19 (13, 28)
ISO6 (n=133)	29.7 (13.5)	24.8 (17.0, 86.6)	26.1 (17.1)	23.7 (3.33, 87.7)	0.0827 (0.327)	0 (0, 2)	19.6 (3.57)	19 (13, 27)
Number of agents in the team								
3 (n=262)	39.9 (13.4)	36.6 (22.8, 92.8)	17.5 (11.4)	14.7 (3.33, 58.7)	0.0420 (0.219	0 (0, 2)	19.3 (3.48)	19 (13, 33)
5 (n=259)	32.6 (13.9)	29.6 (17.0, 91.0)	33.8 (18.9)	31.0 (4.44, 94.2)	0.0502 (0.320)	0 (0, 4)	21.7 (3.78)	21 (15, 43)
8 (n=255)	27.8 (14.9)	24.1 (14.5, 94.5)	36.9 (15.6)	36.3 (6.73, 89.4)	0.0471 (0.277)	0 (0, 3)	25.5 (4.36)	25 (17, 51)
Receive failure chance (%)								
0 (n=720)	29.9 (7.39)	28.9 (14.5, 54.4)	31.0 (17.2)	28.2 (3.41, 94.2)	0.00 (0.00)	0 (0, 0)	21.6 (3.68)	21.5 (13, 33)
5 (n=14)	77.8 (11.0)	78.9 (63.6, 92.0)	10.6 (14.1)	6.84 (3.33, 58.7)	0.786 (1.05)	1 (0, 4)	30.8 (12.1)	27 (16, 51)
25 (n=9)	81.8 (13.4)	88.3 (55.8, 94.5)	6.14 (1.92)	5.55 (4.44, 10.7)	0.222 (0.441)	0 (0, 1)	28.9 (8.33)	29 (18, 45)
50 (n=6)	79.6 (12.6)	83.8 (59.7, 91.0)	5.65 (0.688)	5.57 (4.74, 6.56)	0.333 (0.516)	0 (0, 1)	26.7 (4.41)	26.5 (21, 33)
75 (n=14)	80.3 (9.88)	80.9 (56.3, 93.2)	6.24 (1.84)	5.66 (4.14, 9.31)	0.571 (0.646)	0 (0, 2)	30.0 (5.62)	30.5 (20, 38)
95 (n=8)	74.9 (8.91)	74.6 (61.0, 85.2)	6.61 (1.83)	6.51 (4.77, 10.2)	1.25 (1.04)	0 (0, 3)	29.5 (7.69)	28.5 (19, 41)
100 (n=5)	82.9 (7.75)	86.0 (69.8, 89.7)	6.75 (1.85)	6.73 (5.18, 9.74)	0.600 (0.548)	0 (0, 1)	30.6 (5.98)	28 (26, 40)
Overall								
(n=776)	33.5 (14.9)	29.7 (14.5, 94.5)	29.3 (17.8)	26.9 (3.33, 94.2)	0.0464 (0.274)	0 (0, 4)	22.2 (4.64)	22 (13, 51)

Table 6.6: ”Reactive action coordination agent team” descriptive statistics for run time, time spent idle, incorrectly delivered blocks and rooms entered. ’-’ as a value indicates no available data.

	Run time (s)		Time spent idle (s)		Incorrect Deliveries		Rooms Entered	
	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)
Map								
ISO1 (n=120)	30.0 (6.57)	29.1 (19.3, 50.2)	29.5 (16.3)	27.2 (3.35, 73.3)	0.00 (0.00)	0 (0, 0)	22.2 (3.22)	22 (14, 33)
ISO2 (n=119)	32.4 (7.44)	31.4 (19.2, 51.0)	32.8 (18.8)	29.6 (4.24, 82.7)	0.00 (0.00)	0 (0, 0)	22.7 (3.04)	23 (15, 30)
ISO3 (n=118)	30.5 (6.34)	29.7 (19.7, 47.0)	29.4 (16.1)	27.6 (3.84, 70.6)	0.00 (0.00)	0 (0, 0)	22.0 (3.42)	22 (14, 30)
ISO4 (n=119)	30.9 (8.37)	29.7 (16.1, 56.6)	35.5 (19.1)	32.6 (3.89, 91.7)	0.00 (0.00)	0 (0, 0)	20.3 (2.24)	20 (17, 27)
ISO5 (n=119)	27.1 (8.38)	25.2 (16.0, 76.2)	30.4 (18.0)	26.0 (3.13, 80.3)	0.00 (0.00)	0 (0, 0)	18.9 (2.83)	18 (13, 26)
ISO6 (n=118)	24.8 (5.37)	23.8 (16.3, 40.8)	26.3 (15.1)	25.5 (3.40, 68.4)	0.00 (0.00)	0 (0, 0)	18.4 (3.03)	18 (13, 26)
Number of agents in the team								
3 (n=241)	36.3 (7.07)	36.1 (21.4, 76.2)	18.1 (11.7	16.0 (3.13, 68.9)	0.00 (0.00)	0 (0, 0)	18.7 (2.94)	19 (13, 27)
5 (n=240)	27.4 (5.15)	27.0 (16.4, 42.2)	34.7 (16.3)	31.2 (7.28, 82.7)	0.00 (0.00)	0 (0, 0)	20.2 (2.56)	20 (14, 28)
8 (n=232)	24.0 (3.78)	23.8 (16.0, 35.3)	39.6 (16.2)	37.5 (7.45, 91.7)	0.00 (0.00)	0 (0, 0)	23.5 (2.78)	23 (18, 33)
Receive failure chance (%)								
0 (n=712)	29.3 (7.38)	28.1 (16.0, 56.6)	30.7 (17.5)	27.8 (3.35, 91.7)	0.00 (0.00)	0 (0, 0)	20.8 (3.41)	21 (13, 33)
5 (n=0)	-	-	-	-	-	-	-	-
25 (n=0)	-	-	-	-	-	-	-	-
50 (n=1)	76.2 (NA)	76.2 (76.2, 76.2)	3.13 (NA)	3.13 (3.13, 3.13)	0.00 (0.00)	0 (0, 0)	16.0 (-)	16 (16, 16)
75 (n=0)	-	-	-	-	-	-	-	-
95 (n=0)	-	-	-	-	-	-	-	-
100 (n=0)	-	-	-	-	-	-	-	-
Overall								
(n=713)	29.3 (7.58)	28.1 (16.0, 76.2)	30.7 (17.5)	27.7 (3.13, 91.7)	0.00 (0.00)	0 (0, 0)	20.7 (3.41)	21 (13, 33)

variance is an expected pattern. The additional agents mean the team as a whole is more likely to visit the rooms with the blocks it needs sooner. The variance does result in some overlap of run times between agents. However, the overlap is not an issue. The same tables show that an agent team without communication failure at its worst will still complete the task faster than the baseline agent at its worst. The maximum run time for the agent teams without communication failure is 76.5s, while the baseline has a maximum runtime of 97.0s. The same does not hold for the time spent idle, and the number of rooms entered. However, these indicators are cumulative over all agents in the team. As such, these indicators do not indicate a worse performance compared to the baseline but is merely a result of having multiple agents.

6.3. Effect on Task Success Rate

This section analyses the completion rate by using the odds ratios in table 6.7. The data excludes the simulation runs do not have any communication failure because the intended use for these is providing a baseline in the analysis of the successes in section 6.4. Then the following Binomial logistic regression model was used with the data to calculate the odds ratios:

$$TaskSuccessRate \sim AgentTeam + Map + AgentCount + CommunicationFailureChance.$$

This section analyses the completion rate by using the odds ratios in table 6.7. The table excludes the simulation runs without any communication failure because the intended use for these is providing a baseline in the analysis of the successes in section 6.4. Then the following Binomial logistic regression

Figure 6.1: Plot of mean agent team run times by map grouped by team size. The error bars have a length of 1 standard deviation. Left: teams of 3. Right: teams of 8.

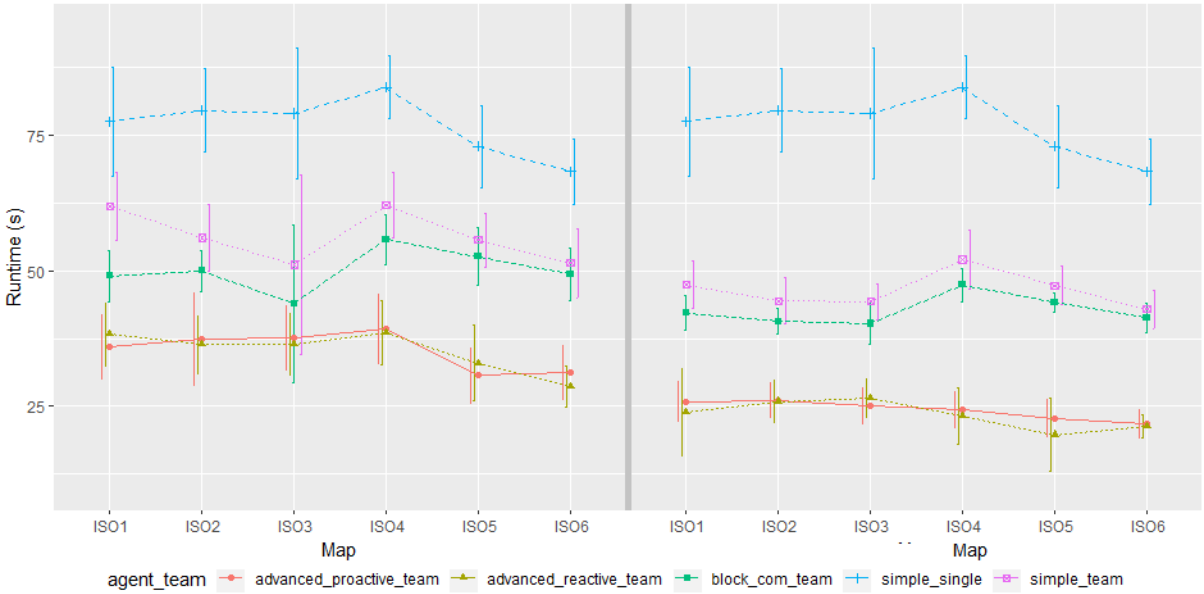

Table 6.7: Odds ratios of the agent team success rate, excludes the baseline simulations runs with no receive failure chance. The ISO1 map is used as the baseline for the maps and is thus not present in the table.

	OR	**2.5 %**	**97.5 %**	p
proactive action coordination agent team	0.0044	0.0033	0.0058	<0.0001
reactive action coordination agent team	<0.0001	<0.0001	0.0003	<0.0001
room and block coordination agent team	0.9569	0.8633	1.0606	0.4010
map ISO2	3.2552	2.7601	3.8436	<0.0001
map ISO3	7.3318	6.1446	8.7668	<0.0001
map ISO4	0.2363	0.1953	0.2851	<0.0001
map ISO5	1.4551	1.2383	1.7106	<0.0001
map ISO6	7.9006	6.6130	9.4596	<0.0001
agent count	0.6434	0.6262	0.6607	<0.0001
communication failure chance (%)	1.0003	0.9988	1.0017	0.7261

model was used with the data to calculate the odds ratios:

$$TaskSuccessRate \sim AgentTeam + Map + AgentCount + CommunicationFailureChance.$$

The McFadden R^2 for this model, 0.634, shows that the model predicts the majority of the variance. Some of the variances can be explained by the agent teams making randomized choices. However, an attempt was made to improve the model by introducing the relevant interaction effects between the independent variables. Of these interactions, only the effect between ”proactive action coordination agent team” and the number of agents had a significant odds ratio. As ”proactive action coordination agent team” and ”reactive action coordination agent team” fail to complete the task in almost all simulation runs with communication failure, the interaction effects were excluded from the final model. ”Proactive action coordination agent team” succeeds 56 times out of a total 4231 simulation runs with communication failure, while the ”reactive action coordination agent team” only succeeded once during 4212 simulation runs.

The action coordination agent teams were expected to have a lower success rate than the other two agents because of the plan negotiation failures. However, the reduction in success rate for ”proactive action coordination agent team” was higher than expected as it is only affected by one of the failures specific to action coordination agent teams. ”Proactive action coordination agent team” still had more successes than ”reactive action coordination agent team”, but only marginally. As such, there is no further analysis of these teams.

On the other hand, the ”simple agent team” and ”room and block coordination agent team” results did

conform to expectations in the number of successes. However, in section 6.1, it was pointed out that the success rate did not gradually decrease as the chance of communication failure increased. The Odds Ratios in table 6.7 confirms that the chance of failure does not affect the success rate. Additionally, based on the sampled agent sizes, every additional agent in a team decreases the chance for success. The likely cause is that more agents can fail to receive a message, which means it is more likely one of the failure scenarios happens.

Finally, there is the effect of the maps on the success rate. The odds ratio table uses ISO1, a map with no redundant blocks, as the baseline. ISO4, similarly a map with no redundant blocks, lowers the chance of success. However, adding even one redundant block per color like with ISO2 and ISO5 increases the success chance by about 3.3 and 1.5 times, respectively. Increasing the block redundancy even further with ISO3 and ISO6 raises the success chance again by around 7.3 and 7.9 times. This change in task success odds indicates that preventing agents from delivering the wrong block can provide a substantial increase in the success rate.

6.4. Effect on Agent Performance

Table 6.8: "Simple agent team" linear regression models of successful runs. The ISO1 map is used as the baseline for the maps and is thus not present in the table.

	Run time (s)	**Time spent idle (s)**	**Incorrect deliveries**	**Rooms entered**
Intercept	67.2400	-0.336820	-0.28396	13.21080
Agent Count	-7.4669	2.076632	-0.14608	7.11880
Receive failure chance	0.1194	0.008991	0.02483	0.01767
Map				
ISO2	0.9963	0.608492	2.63538	1.72908
ISO3	1.6856	1.793723	5.90581	5.47432
ISO4	5.2079	1.427021	0.25457	4.83099
ISO5	1.8695	0.771792	2.11024	1.46255
ISO6	-2.2009	1.233881	5.48102	3.14106

Table 6.9: "Room and block coordination agent team" linear regression models of successful runs. The ISO1 map is used as the baseline for the maps and is thus not present in the table.

	Run time (s)	**Time spent idle (s)**	**Incorrect deliveries**	**Rooms entered**
Intercept	63.75469	8.7945	-0.3161	8.5564
Agent Count	-1.70263	4.4181	-0.1393	5.4288
Receive failure chance	0.15480	-0.1151	0.0245	0.1386
Map				
ISO2	2.53249	-9.8272	2.6373	4.8750
ISO3	3.11515	-12.1439	5.9257	9.8367
ISO4	5.08047	10.1495	0.2785	-1.3889
ISO5	3.06101	-10.9184	2.1805	4.3828
ISO6	0.00777	-12.5928	5.3891	8.3047

This section will analyze the agent team performance on successful task completion. There are 7461 successful simulation runs in total. As the action coordination agent teams have very few successes, this section only looks at "simple agent team" and "room and block coordination agent team." These teams were then fitted to a linear regression model for run time, idle time, incorrect deliveries, and rooms visited. The independent variables were significant ($p < 0.001$ using ANOVA) for every model tested.

Table 6.8 shows the fitted models for "simple agent team" based on 3053 successful simulation runs. In the case of run time, there are no surprises in the model. When it comes to idle time, it is notable that although the effect of communication failure is significant, it is so weak that it is not considered meaningful. The difference in idle time between no failure chance and 100% receive failure, is less than a second. The agent count effect is not noteworthy either as idle time is cumulative per agent. There is no clear difference in idle time for maps with the blocks clustered together and the maps where

the blocks are spread out over all rooms. However, when the block redundancy in the map increases (ISO2-3 and ISO4-5) from no redundancy (ISO1 and ISO4), the amount of incorrectly delivered blocks goes up as expected. Agent teams have the option to recover from mistakes in the case of additional resources, which corresponds to the increase in the odds ratios for maps with redundant blocks. Finally, receive failure chance was expected to have only a small effect on the number of rooms entered for ”simple agent team.” This team only communicates deliveries, which does not affect the decision to enter a room other than changing the next delivery target. Other teams are expected to be more affected in this regard.

Table 6.9 shows the fitted models for ”room and block coordination agent team” based on 3009 successful simulation runs. Run time has similar results to those of ”simple agent team” and is within expectations. For idle time it should be noted that an increase in failure chance leads to a decrease. Most likely, the decrease is because agents will act on outdated or partial information if it misses messages. The agent then acts when it otherwise would not. Incorrect Deliveries is very similar to the ”simple agent team” results as well. For Rooms Entered, it should be mentioned that the amount of block redundancy has a notable effect on the number of rooms entered. The agents will all try to pick up the current color in the sequence but are not always aware of which blocks have been picked up. As such, the redundancy allows the agents to try and deliver blocks that are already picked up by another agent, which leads to more doors entered.

The two agent teams have a similar performance. However, receive failure chance increases the run time, and rooms entered more for ”room and block coordination agent team” than ”simple agent team.” It is the other way round for time spent idle, and rooms entered. In the case of incorrect deliveries, the effects on the agent teams are similar. However, that can be explained by the incorrect deliveries increasing in line with the number of redundant blocks. If the team delivers too many blocks at the wrong time, the task can no longer be completed, and the simulation run times out.

7

Improved Agent Team Analysis

This chapter presents the results of the experiment run with improved agents based on the results in chapter 3 and table 4.1. The agent teams have been modified to mitigate a failure situation that affected all agent teams, delivering the wrong block. The pilot data indicated that agent teams are a lot more likely to fail if there is no intermediate task state correction. When an agent misses that a block in the sequence was delivered because they missed the message, even a 5% failure cut the number of successful simulation runs in half for the pilot experiment.

For this set of simulations, the agent teams use BW4T functionality to keep track of how many blocks were delivered correctly. Then once an agent enters the dropzone, if it is about to deliver the wrong block, the agent updates its belief about the number of blocks delivered to the correct value. Since the agent no longer believes the block it is holding should be delivered, it will take the block with it out of the room without any further code changes. The agent treats the block the same as if it got a $msg(delivered(BlockID)$, and the block no longer needs to be delivered. This correction represents an agent checking what has been delivered in the dropzone and correcting itself. As a result, the problem no longer causes teams to fail. Instead, attempting to deliver the wrong block only slows the team down.

The rest of the chapter first provides an overview of the data for the agents. Following that is a section that examines the task success rate as it relates to communication failure, and finally, the results for the effects of communication failure on the agent team performance.

7.1. Descriptives

The agent team results are presented in the same way as section 6.1, with two differences. The baseline agent team has not changed, so this section will continue to use the data from table 6.1. Additionally, there are no descriptives for incorrectly delivered blocks since the agent team changes ensured that there were none. The simulation runs included those without communication failure to validate that the teams still performed better than the baseline team under normal conditions. There were a total of 19873 observations, of which 8228 were successful. As such, the success rate increased to 41.4% compared to 38.7% before the modifications. Once again, the groups in table 7.1 are not all the same size due to limitations of the BW4T environment, as discussed in chapter 6.

When looking at the task completion rate in table 7.1 per agent team, the advanced teams still perform worse than the other teams. There is no notable increase in success despite the changes. However, the total task completion rate has gone up due to the increased performance of ”simple agent team” and ”room and block coordination agent team.” However, these teams still fail a little under a third of all simulation runs despite them only being slowed down by communication failure. After the fix, the remaining ”could fail” messages on table 4.1 are only used by the action coordination agent teams. These failed simulation runs are the result of the team run time being longer than the 110-second timeout. The timeout is already longer than the maximum runtime of the single-agent baseline

What stands out about the task completion rate changes is that the increase in successful runs occurred predominantly on the maps with more redundant blocks. Additionally, the maps with the blocks spread out saw a larger increase than the maps with the blocks concentrated. It should also

Table 7.1: Task completion rate

	Failure (n=11645)	**Success (n=8228)**	**Overall (n=19873)**
Agent Team			
proactive action coordination agent team	4166 (84.4%)	771 (15.6%)	4937
reactive action coordination agent team	4234 (85.6%)	712 (14.4%)	4946
room and block coordination agent team	1619 (32.4%)	3377 (67.6%)	4996
simple agent team	1626 (32.6%)	3368 (67.4%)	4994
Map			
ISO1	2267 (68.7%)	1033 (31.3%)	3300
ISO2	1766 (53.3%)	1546 (46.7%)	3312
ISO3	1613 (48.6%)	1705 (51.4%)	3318
ISO4	2602 (78.8%)	700 (21.2%)	3302
ISO5	1945 (58.7%)	1369 (41.3%)	3354
ISO6	1452 (43.6%)	1875 (56.4%)	3327
Number of agents in team			
3	3483 (52.5%)	3154 (47.5%)	6637
5	3920 (59.2%)	2705 (40.8%)	6625
8	4242 (64.2%)	2369 (35.8%)	6611
Receive failure chance (%)			
0	22 (0.8%)	2857 (99.2%)	2879
5	1929 (68.2%)	898 (31.8%)	2827
25	1939 (68.4%)	894 (31.6%)	2827
50	1923 (68.0%)	906 (32.0%)	2829
75	1940 (68.6%)	886 (31.4%)	2826
95	1937 (68.3%)	900 (31.7%)	2837
100	1955 (68.8%)	887 (31.2%)	2842

Table 7.2: ”Simple agent team” descriptive statistics for run time, time spent idle, and rooms entered.

	Run time (s)		**Time spent idle (s)**		**Rooms Entered**	
	Mean (SD)	**Median (Min-Max)**	**Mean (SD)**	**Median (Min-Max)**	**Mean (SD)**	**Median (Min-Max)**
Map						
ISO1 (n=397)	64.0 (10.5)	63.6 (39.6, 99.0)	10.3 (4.47)	9.33 (4.60, 23.1)	49.3 (15.7)	48 (28, 84))
ISO2 (n=641)	63.6 (9.35)	64.0 (38.1, 91.9)	11.1 (4.51)	10.2 (5.25, 25.1)	54.8 (16.5)	53 (26, 95))
ISO3 (n=715)	64.2 (9.67)	64.2 (9.67)	11.8 (4.48)	10.9 (5.09, 24.4)	59.1 (17.9)	57 (26, 101)
ISO4 (n=241)	68.2 (13.2)	67.9 (41.1, 96.6)	10.7 (4.94)	9.31 (5.12, 24.3)	51.2 (19.0)	42 (30, 100)
ISO5 (n=570)	65.0 (9.53)	65.9 (40.2, 91.5)	10.5 (4.28)	9.85 (5.15, 23.5)	51.9 (15.7)	50 (29, 87))
ISO6 (n=804)	60.0 (7.50)	61.0 (36.0, 83.8)	11.6 (4.67)	10.9 (4.95, 23.6)	58.3 (19.1)	58 (26, 102)
Number of agents in the team						
3 (n=1341)	66.6 (9.35)	65.6 (42.1, 96.6)	6.92 (1.18)	6.84 (4.60, 20.7)	38.2 (4.88)	38 (26, 57))
5 (n=1105)	62.6 (9.10)	62.8 (36.4, 99.0)	11.0 (1.55)	10.9 (6.68, 21.0)	55.7 (6.61)	56 (36, 75))
8 (n=922)	60.0 (9.84)	61.4 (36.0, 91.9)	17.5 (2.30)	17.6 (7.88, 25.1)	79.1 (8.54)	80 (43, 102)
Receive failure chance (%)						
0 (n=716)	51.5 (7.64)	50.6 (36.0, 79.6)	10.9 (4.04)	10.0 (4.95, 24.3)	53.0 (16.7)	52 (26, 100)
5 (n=443)	65.9 (7.44)	64.6 (53.9, 94.3)	11.0 (4.62)	10.2 (5.14, 21.3)	55.3 (18.4)	52 (29, 98))
25 (n=445)	66.8 (7.72)	65.5 (53.6, 91.9)	11.1 (4.71)	10.2 (4.60, 25.1)	55.0 (18.0)	51 (28, 101)
50 (n=443)	67.0 (7.49)	65.5 (53.6, 91.5)	11.3 (4.65)	10.7 (4.87, 24.4)	55.9 (17.9)	53 (30, 98))
75 (n=440)	66.9 (7.55)	65.7 (54.1, 89.3)	11.1 (4.73)	10.4 (4.79, 23.7)	55.2 (17.8)	53 (29, 102)
95 (n=442)	66.8 (7.76)	65.5 (54.6, 99.0)	11.3 (4.73)	10.4 (5.08, 23.6)	56.0 (18.1)	53 (30, 100)
100 (n=439)	66.8 (7.18)	65.6 (53.7, 93.6)	11.5 (4.71)	10.7 (5.03, 23.6)	57.0 (17.9)	55 (30, 101)
Overall						
(n=3368)	63.5 (9.79)	63.4 (36.0, 99.0)	11.1 (4.56)	10.3 (4.60, 25.1)	55.2 (17.8)	53 (26, 102)

Table 7.3: ”Room and block coordination agent team” descriptive statistics for run time, time spent idle, and rooms entered.

	Run time (s)		Time spent idle (s)		Rooms Entered	
	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)
Map						
ISO1 (n=400)	61.2 (12.4)	63.2 (36.7, 92.1)	25.4 (31.0)	10.8 (4.97, 163)	41.2 (15.6)	34 (22, 77)
ISO2 (n=653)	62.2 (10.6)	63.8 (35.4, 94.1)	15.3 (13.3)	11.0 (5.21, 80.0)	52.0 (17.6)	50 (23, 87)
ISO3 (n=725)	62.4 (10.7)	63.6 (33.1, 89.3)	12.7 (5.94)	11.4 (4.97, 42.3)	57.3 (19.8)	55 (22, 101)
ISO4 (n=222)	63.5 (14.5)	59.4 (41.8, 91.0)	38.9 (36.6)	24.6 (4.92, 136	36.8 (10.6)	34 (24, 77)
ISO5 (n=562)	64.0 (10.7)	65.6 (39.4, 94.4)	13.5 (9.59)	10.6 (5.24, 64.6)	48.1 (16.0)	42 (23, 92)
ISO6 (n=815)	59.6 (7.44)	60.9 (34.9, 77.3)	12.5 (5.71)	11.3 (4.22, 37.5)	57.2 (19.8)	56 (20, 100)
Number of agents in the team						
3 (n=1314)	64.9 (10.1)	65.1 (36.4, 94.4)	8.28 (5.28)	7.03 (4.22, 38.8)	36.8 (6.33)	37 (20, 56)
5 (n=1108)	61.1 (10.4)	62.3 (34.6, 89.0)	15.5 (12.9)	11.5 (7.50, 77.5)	52.1 (11.6)	55 (24, 75)
8 (n=955)	58.6 (10.5)	61.0 (33.1, 87.5)	29.0 (25.3)	19.0 (13.0, 163)	70.9 (19.9)	79 (28, 101)
Receive failure chance (%)						
0 (n=713)	46.2 (5.87)	45.3 (33.1, 66.8)	35.4 (31.1)	25.7 (4.22, 163)	32.6 (5.94)	32 (20, 52)
5 (n=447)	66.0 (6.81)	64.8 (52.7, 91.0)	11.5 (4.91)	10.5 (5.12, 23.1)	56.5 (18.2)	54 (29, 95)
25 (n=436)	66.0 (7.20)	64.7 (53.2, 94.1)	11.5 (4.81)	10.7 (4.97, 23.2)	56.6 (18.0)	53 (30, 99)
50 (n=454)	66.3 (6.78)	64.8 (53.9, 89.3)	11.5 (4.78)	10.7 (5.12, 23.8)	56.6 (18.3)	54 (30, 101)
75 (n=438)	66.2 (6.97)	65.2 (54.4, 90.8)	11.7 (4.91)	10.8 (5.23, 24.9)	56.8 (17.8)	54.5 (30, 96)
95 (n=450)	66.0 (7.44)	65.1 (53.2, 91.5)	11.1 (4.58)	10.2 (5.08, 23.9)	56.1 (18.1)	54 (29, 98)
100 (n=439)	65.9 (7.50)	64.9 (53.1, 94.4)	11.3 (4.76)	10.6 (4.92, 21.6)	56.4 (18.3)	53 (30, 97)
Overall						
(n=3377)	61.9 (10.6)	62.8 (33.1, 94.4)	16.5 (17.8)	11.2 (4.22, 163)	51.5 (19.0)	46 (20, 101)

Table 7.4: ”Proactive action coordination agent team” descriptive statistics for run time, time spent idle, and rooms entered.

	Run time (s)		Time spent idle (s)		Rooms Entered	
	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)
Map						
ISO1 (n=119)	30.4 (6.54)	29.0 (17.4, 47.7)	29.3 (16.2)	27.0 (3.87, 76.5)	22.6 (4.04)	23 (14, 37)
ISO2 (n=132)	35.4 (14.6)	30.7 (17.4, 90.0)	27.3 (17.0)	24.5 (3.83, 85.0)	23.8 (4.19)	24 (14, 39)
ISO3 (n=145)	39.1 (19.6)	32.1 (18.5, 96.0)	25.7 (16.9)	23.6 (3.56, 73.4)	25.5 (7.80)	24 (15, 55)
ISO4 (n=119)	30.8 (7.81)	28.7 (19.0, 52.9)	34.5 (17.1)	32.4 (3.43, 83.6)	21.0 (2.82)	21 (16, 28)
ISO5 (n=120)	26.3 (5.75)	25.5 (14.9, 42.6)	30.5 (19.0)	27.3 (3.04, 92.6)	19.4 (3.33)	19 (12, 27)
ISO6 (n=136)	30.6 (15.1)	24.8 (16.1, 83.1)	25.4 (15.5)	23.4 (3.16, 68.9)	19.5 (3.62)	19 (14, 29)
Number of agents in the team						
3 (n=259)	38.1 (11.2)	36.2 (21.9, 90.0)	17.3 (11.6)	15.1 (3.04, 68.7)	19.2 (3.91)	19 (12, 37)
5 (n=253)	30.7 (13.1)	27.6 (16.2, 95.2)	32.3 (16.8)	29.9 (3.93, 92.6)	21.0 (3.67)	20 (15, 39)
8 (n=259)	28.2 (14.6)	24.4 (14.9, 96.0)	36.3 (16.2)	34.5 (5.72, 85.0)	25.9 (5.44)	25 (18, 55)
Receive failure chance (%)						
0 (n=716)	29.2 (7.11)	27.6 (14.9, 54.2)	30.3 (16.6)	27.9 (3.04, 92.6)	21.3 (3.80)	21 (12, 37)
5 (n=8)	79.0 (12.4)	80.6 (59.5, 96.0)	8.15 (2.77)	7.79 (5.21, 12.7)	35.0 (11.8)	31.5 (21, 55)
25 (n=13)	71.2 (10.9)	68.8 (56.2, 90.6)	5.84 (1.28)	5.87 (3.68, 9.36)	29.0 (6.26)	29 (18, 42)
50 (n=9)	72.9 (10.5)	74.5 (58.9, 95.2)	6.42 (2.56)	6.16 (3.28, 10.9)	31.8 (13.1)	29 (17, 55)
75 (n=8)	73.8 (9.29)	75.0 (57.1, 85.9)	7.58 (2.26)	7.19 (4.91, 11.1)	35.5 (9.17)	32.5 (23, 49)
95 (n=8)	72.7 (14.0)	73.6 (55.9, 94.3)	5.61 (1.99)	5.63 (3.16, 9.48)	27.1 (8.48)	28.5 (15, 39)
100 (n=9)	73.6 (12.4)	77.1 (57.7, 91.2)	6.71 (2.77)	6.83 (3.56, 11.6)	31.1 (12.3)	28 (18, 51)
Overall						
(n=771)	32.4 (13.7)	28.3 (14.9, 96.0)	28.6 (17.2)	26.5 (3.04, 92.6)	22.0 (5.23)	22 (12, 55)

Table 7.5: ”Reactive action coordination agent team” descriptive statistics for run time, time spent idle, and rooms entered. '-' as a value indicates no available data.

	Run time (s)		Time spent idle (s)		Rooms Entered	
	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)	Mean (SD)	Median (Min-Max)
Map						
ISO1 (n=117)	30.8 (7.42)	29.6 (17.9, 50.3)	29.4 (16.8)	24.9 (4.69, 74.3)	22.4 (3.50)	22 (16, 33)
ISO2 (n=120)	30.1 (6.49)	29.3 (17.6, 45.0)	29.5 (16.7)	25.2 (4.33, 90.0)	22.0 (3.64)	22 (13, 32)
ISO3 (n=120)	30.4 (6.25)	29.5 (19.2, 45.8)	30.0 (17.5)	25.6 (3.39, 76.3)	22.2 (3.42)	22 (15, 30)
ISO4 (n=118)	30.2 (7.86)	28.7 (17.6, 51.4)	34.5 (18.3)	31.5 (3.99, 86.1)	20.4 (2.36)	20 (14, 27)
ISO5 (n=117)	26.7 (7.11)	25.6 (16.3, 54.0)	30.9 (18.3)	27.3 (3.35, 82.2)	18.9 (2.85)	19 (12, 26)
ISO6 (n=120)	24.3 (4.43)	23.4 (16.2, 39.4)	27.3 (14.0)	27.1 (4.32, 67.0)	18.4 (3.38)	18 (12, 27)
Number of agents in the team						
3 (n=240)	35.2 (6.58)	35.5 (20.7, 54.0)	17.8 (11.3)	14.8 (3.35, 64.4)	18.6 (2.97)	19 (12, 29)
5 (n=239)	26.6 (4.88)	26.3 (16.2, 40.9)	33.5 (15.4)	31.6 (6.94, 86.1)	19.8 (2.56)	20 (14, 27)
8 (n=233)	24.2 (3.98)	23.4 (16.3, 36.2)	39.7 (16.0)	37.8 (7.64, 90.0)	23.8 (2.90)	23 (18, 33)
Receive failure chance (%)						
0 (n=712)	28.7 (7.07)	27.7 (16.2, 54.0)	30.2 (17.1)	27.0 (3.35, 90.0)	20.7 (3.59)	21 (12, 33)
5 (n=0)	-	-	-	-	-	-
25 (n=0)	-	-	-	-	-	-
50 (n=0)	-	-	-	-	-	-
75 (n=0)	-	-	-	-	-	-
95 (n=0)	-	-	-	-	-	-
100 (n=0)	-	-	-	-	-	-
Overall						
(n=712)	28.7 (7.07)	27.7 (16.2, 54.0)	30.2 (17.1)	27.0 (3.35, 90.0)	20.7 (3.59)	21 (12, 33)

Table 7.6: Odds ratios of the modified agent team success rate. The ISO1 map is used as the baseline for the maps and is thus not present in the table.

	OR	2.5 %	97.5 %	p
advanced_proactive_team	.0020	0.0014	0.0026	<0.0001
advanced_reactive_team	<0.0001	<0.0001	<0.0001	0.8824
room and block coordination agent team	1.0211	0.9148	1.1399	0.7094
map ISO2	5.3224	4.4995	6.3072	<0.0001
map ISO3	11.538	9.5623	13.9695	<0.0001
map ISO4	0.2631	0.2183	0.3162	<0.0001
map ISO5	2.8458	2.4268	3.3408	<0.0001
map ISO6	35.372	27.825	45.3854	<0.0001
agent count	0.7154	0.6957	0.7354	<0.0001
communication failure chance (%)	0.9996	0.9981	1.0012	0.6370

be noted that the relationship between team size and success has inversed. After modification, a larger team size results in more successful runs — likely a result of larger teams making more incorrect deliveries without this correction. When looking at the success rate by receive failure chance, not much has changed. The presence of communication failure still determines the success rate rather than the likelihood of communication failure, even if the total chance of success increased.

”Simple agent team” and ”room and block coordination agent team” have gotten faster on average, excluding the runs without communication failure, but the difference is within 1SD. The number of rooms entered has gone up for these teams as well, excluding the runs without communication failure. However, the differences are still within 1SD. The advanced teams have not been compared in detail as there are still very few successful runs.

7.2. Effect on Task Success Rate

This section analyses the task completion rate by using the odds ratios in table 7.6. Like section 6.3, the data again excludes the simulation runs that do not have any communication failure. First, the model for calculating the odds ratios is described, before analyzing the results themselves to see what has changed compared to the ratios of table 7.6.

The following Binomial logistic regression model was used with the data to calculate the odds ratios: $TaskSuccessRate \sim AgentTeam + Map + AgentCount + CommunicationFailureChance$. The

Table 7.7: "Simple agent team" linear regression models of successful runs. The ISO1 map is used as the baseline for the maps and is thus not present in the table.

	Run time (s)	Time spent idle (s)	Rooms entered
Intercept	65.1916	-0.568078	7.36989
Agent Count	-1.1383	2.116392	8.16294
Receive failure chance	0.1069	0.008846	0.03727
Map			
ISO2	-0.9339	0.396123	3.69579
ISO3	-0.3645	1.143099	8.48150
ISO4	5.4687	1.376450	5.62859
ISO5	0.2510	0.589625	3.86573
ISO6	-4.5375	0.611130	3.86573

Table 7.8: "Room and block coordination agent team" linear regression models of successful runs. The ISO1 map is used as the baseline for the maps and is thus not present in the table.

	Run time (s)	Time spent idle (s)	Rooms entered
Intercept	61.2933	9.2482	1.9318
Agent Count	-1.0736	4.2594	6.7662
Receive failure chance	0.1272	-0.1172	0.1501
Map			
ISO2	0.2485	-10.3150	7.5212
ISO3	0.4012	-12.7100	13.2216
ISO4	3.8498	12.6778	-1.2024
ISO5	2.1537	-10.9653	7.0138
ISO6	-2.3116	-13.3970	12.1694

McFadden R^2 for this model, 0.608, shows that the model predicts less of the variance than the model for the original analysis. The difference can probably be attributed to the random aspects of the agent teams having more impact after eliminating one of the communication failure types. The random elements can still cause an agent to fail if they result in the team taking longer to complete the task than the single-agent baseline team. There was no attempt to improve the model with interaction effects as this section is about comparing the results to those of chapter 6. There were no interaction effects included in that chapter either.

Even after applying the mitigation method, the 'Advanced' teams were expected to have a lower success rate than the other two agents due to unmitigated communication failure types. However, the odds ratios for the 'Proactive' team are even lower. The decrease is due to the other teams being successful more often now.

"Room and block coordination agent team" also had similar results compared to before the mitigation methods; only the p value is higher, going from 0.4010 to 0.7094. This team has no communication failure types that the "simple agent team" did not have, so the continued similarity here is as expected. The maps themselves are now stronger predictors of success, most notably ISO6, which increased from an odds ratio of 7.9006 to 35.372.

The team size now decreases the chance of success less as the odds ratio increased from 0.6434 to 0.7154. Despite the smaller increase, it is worth mentioning as the new value falls outside of the 95% confidence interval of the unmitigated odds ratio. The communication failure chance still does not provide a significant chance in task success. However, the p value has decreased from 0.7261 to 0.6370.

7.3. Effect on Agent Performance

This section will analyze the agent team performance on successful task completion. There are 8228 successful simulation runs in total. As the action coordination agent teams still have very few successes, this section only looks at "simple agent team" and "room and block coordination agent team." These teams were then fitted to a linear regression model for run time, idle time, and rooms visited. In-

correct deliveries are excluded from the model as they no longer occurred. The independent variables were significant ($p < 0.001$ using ANOVA) for every model tested.

Table 7.7 shows the fitted models for ”simple agent team” based on 3368 successful simulation runs. This section will be comparing that table with table 6.8 to determine the differences. In the case of run time, the intercept is lower after the modification. The lower average runtime for the team can explain the difference. However, it should be noted that adding extra agents to the team has a smaller effect on the run time. Where previously the agent count had a coefficient of -7.4669, now it has lowered to -1.1383. The difference could be related simulation runs that previously would have failed and been excluded, now succeeding but doing so more slowly. Additionally, both ISO2 and ISO3 now lower the runtime. Similarly, ISO5 increases the runtime less, and ISO6 reduces the runtime even more. The presence of redundant blocks might cause agents to correct their sequence more often as more agents can hold a block of any given color. When it comes to idle time, the intercept has lowered around two-tenths of a second, but agent count and receive failure have changed very little. Though the effects the different maps have lowered, likely because the agents spend more time correcting for the mistakes. Finally, the rooms entered intercept being lower is noteworthy as the expectation was that agents would be entering more rooms. The other variables reflect that by increasing compared to before the modification and the average rooms entered has gone up. The reason for the lower intercept might be that the variance increased as the higher standard deviation, 17.8 compared to 15.1 before the modification, for rooms entered indicates.

Table 7.8 shows the fitted models for ”Room and block coordination agent team” based on 3377 successful simulation runs. The run time intercept for the block com team has increased compared to before the modification. However, the average runtime has decreased. The total lower runtime is probably related to the failure chance having less of an effect, 0.1272 compared to 0.15480 before. Additionally, the team experiences the same effect from redundant blocks as ”simple agent team.” For idle time there does not appear to be much changed. The intercept is a little higher, but the maps appear to have slightly less of an effect in turn. Rooms entered has the same differences as ”simple agent team” and probably for the same reasons. However, do take note that the intercept is a lot lower, 1.9318 compared to 8.5564 before.

Both teams appear to have similar changes in performance. Comparing the agent teams, not much has changed relative to before the modification. However, the decrease in average runtime was unexpected for this change as agents require time to correct their incorrect beliefs. The lowered run time might be because previously, more agents were delivering the wrong blocks. Doing so leaves fewer agents in the team to deliver the right blocks and effectively decreasing the team size.

8

Conclusions

While working on this project, two lessons stood out. The first is how easy it is to assume that all communication will work. As well as realizing how often information is assumed to be self-evident. However, once looked at in more detail turns out to be the result of some form of communication. When working on one of the agents used for the pilot, the agent used the $sequenceIndex$ percept without questioning anything. However, when mapped to a real-world situation, something must broadcast that information. The agents would have no way of observing that a block was delivered most of the time. The other lesson was in the use of cloud systems. While they are excellent for running simulations in parallel on identical systems, there several cloud-specific considerations to keep in mind. It is more important to be clear about the project requirements as capacity needs to be reserved and, in the case of commercial clouds, paid for. Overcapacity can be more costly than when working with physical hardware. Usually, the hardware is paid for once and is reusable between projects, and then only running costs are an issue. Finally, cloud systems do not often account for GPU use. Many do not have GUIs, and even when they do, they are optimized for computationally intensive or memory-intensive operations.

The remainder of this chapter will first discuss how the different agent teams performed and why they did so. If no clear cause could be found, it will include speculation on the cause. Then it looks at possible future work.

8.1. Agent Evaluation

In chapter 7, the communication failures that could lead to impossible to finish states for the Simple Team and Block Com Team were mitigated. Now, these teams only slow down when they attempt to deliver the wrong block — however, about 32.5% of the time, these teams still fail to complete the simulation. As the agent teams are never in an unfinishable state, they need more time than the baseline agent. Table 7.6 shows that extra redundancy increases the chances of success. However, the maps with the blocks together are more likely to result in a successful simulation run. Excluding the map with the most redundant blocks that are spread out. So, while the agents are indeed obstructing each other, it is not clear why this switch in behavior occurs. It might be that with enough redundant blocks spread out, agents are more likely to locate different blocks of the color they need. As a result, the agents spread themselves out more and obstruct each other less. Communication failure chance, however, does not appear to influence how much obstruction occurs. Table 7.6 shows that the failure chance does not have a significant impact on the likelihood of success. The presence of communication failure, on the other hand, triggers the obstruction.

The two advanced agents had very few results due to the impact of a plan negotiation failure discussed in section 4.5. The agents never attempt to deliver the first block due to a failure to coordinate the deliveries. The proactive agent team is not affected by the other plan negotiation failure but sees only a minimal improvement in performance. As a result, the first failure is likely to be the leading cause of the unsuccessful simulation runs. However, further research is required to determine if agents that prefer blocks earlier in the delivery order would lead to an increase in task completion. As with the Simple Team and the Block Com Team, these teams might also be performing worse than the single-agent

baseline.

Additionally, the results for the different teams together did show what some of the effects of communication failure are. For example, agents started doing duplicate work when messages get lost. When agents in a team do duplicate work, they should spend less time being idle as the agents are working more. Similarly, the runtime should be going up because the agents are working less efficiently. Table 6.8 and 7.7 show that as communication failure increases the run time of successful simulations goes up as well. However, with the ”simple agent team,” the increased failure chance corresponds to more idle time. Do note that it increases not nearly as much as the run time does. The ”block and room coordination agent team,” on the other hand, has the idle time go down with increased communication failure chance. The agents thus spend more time active, which can only happen if the agents double up on work as the amount of work did not increase.

Similarly, the experiment in 7 showed that including a method to verify what stage a task is in can improve an agent team's success rate. The agent teams have run with and without the method. Comparing the task completion rate between them can be done with table 7.1 for the agent team results with the method and 6.2 for the agent teams without. The data in these tables shows there is indeed an increase in success. The result was an increase in the success rate from 61.1% to 67.4% for the ”simple agent team” and 60.4% increasing to 67.6% for the ”room and block coordination agent team.” The difference proves that in environments where communication can fail, there are benefits for agents to verify the task stages themselves, rather than relying solely on communication to keep up to date.

Not all effects were brought to the fore by the experiments. Something that came up during agent team design was that agents could end up blocking access to required locations and make the task impossible to complete. Due to incomplete information, it could happen that started idling inside the dropzone. Agents do this when they are not aware of any block that needs to be delivered, and they already explored all the rooms. Since agent teams need to deliver every block to the dropzone and only one agent can be inside a room at a time, the idling agent blocked the entire team. However, agents can end up with the same incomplete information if they do not receive all messages from other agents telling them about which blocks are where. As such, it is an effect to keep in mind when considering problems caused by communication failure.

8.2. Future Work

For a direct follow-up to this research, the biggest problem is that the data is limited by not being able to determine if a simulation run failed or took longer than the simulation timeout. With these additional successful runs, it becomes possible to determine if the cutoff hid some effects on the run time, time spent idle, and the rest. It would also be possible to add measurements to count how often an agent attempts to enter an occupied room as well as when and where agents spend their idle time. The new measurements would provide a better picture of how the agents obstruct each other.

Additionally, it would be useful to run another simulation set for the advanced agents where the plan negotiation failure affecting both teams is mitigated. Currently, the results do not offer much insight into the more complex agent teams and if these are affected in the same way as the less complicated agent teams. Making sure agents prefer blocks that are required earlier in the sequence is one improvement that has already been named. Other potential improvements could be to introduce timeouts on coordination negotiations. If the negotiation does not conclude within a given timeframe, the agent should assume it failed. At that point, it will try to renegotiate the next goal.

A different direction would be to look into alternatives for creating agent teams that use implicit coordination where possible. The goal would be to create a team that performs on the same level as the action coordination agent teams but also communicates as little as possible. Earlier research[22] has already looked into the feasibility of such agents. However, it would be useful to compare with agents that use more communication to see if minimizing communication makes the teams less or more resilient to communication failure.

While not directly related to communication failure, it could be useful to see how the map topology affects communication failure. From the experiments, it has become clear that block placement is a significant factor in performance with communication failure. A non-uniform topology might affect the results again. For example, redundant blocks could be further away from the dropzone. The cost of making mistakes would then increase. Are there ways to limit how often agents try to access rooms that are further away? Would those agent teams that try to explore close by rooms be more or less

resilient to communication failure.

Finally, it could be interesting to check if these results extend to different multi-agent system scenarios. These results are about search and retrieval tasks. However, delivery tasks like those an amazon drone swarm would do might have different results.

Bibliography

[1] Tucker Balch and Ronald C Arkin. Communication in reactive multiagent robotic systems. *Autonomous robots*, 1(1):27–52, 1994.

[2] Kim Bhasin and Patrick Clark. How amazon triggered a robot arms race. *Bloomberg Techology*, 2016.

[3] Yuan Fan, Gang Feng, Yong Wang, and Jianbin Qiu. A novel approach to coordination of multiple robots with communication failures via proximity graph. *Automatica*, 47(8):1800–1805, 2011.

[4] Mario Gerla, Eun-Kyu Lee, Giovanni Pau, and Uichin Lee. Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 241–246. IEEE, 2014.

[5] Maaike Harbers, Jeffrey M Bradshaw, Matthew Johnson, Paul Feltovich, Karel van den Bosch, and John-Jules Meyer. Explanation and coordination in human-agent teams: A study in the bw4t testbed. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, volume 3, pages 17–20. IEEE, 2011.

[6] Andrew J Hawkins. Uber's self-driving car showed no signs of slowing before fatal crash, police say, 2018. https://www.theverge.com/2018/3/19/17140936/uber-self-driving-crash-death-homeless-arizona.

[7] Cisco Systems Inc. 20 Myths of Wi-Fi Interference: Dispel Myths to Gain High-Performing and Reliable Wireless , 2007. http://www.cisco.com/c/en/us/products/collateral/wireless/spectrum-expert-wi-fi/prod_white_paper0900aecd807395a9.pdf [Online; accessed 10-Februari-2016].

[8] Matthew Johnson, Catholijn Jonker, Birna Van Riemsdijk, Paul J Feltovich, and Jeffrey M Bradshaw. Joint activity testbed: Blocks world for teams (bw4t). In *International Workshop on Engineering Societies in the Agents World*, pages 254–256. Springer, 2009.

[9] Matthew Johnson, Jeffrey M Bradshaw, Paul Feltovich, Catholijn Jonker, Birna van Riemsdijk, and Maarten Sierhuis. Autonomy and interdependence in human-agent-robot teams. *IEEE Intelligent Systems*, 27(2):43–51, 2012.

[10] Vincent Koeman. BW4T 3.9.1, 2018. https://github.com/eishub/BW4T/releases/tag/v3.9.1.

[11] Ivana Kruijff-Korbayová, Francis Colas, Mario Gianni, Fiora Pirri, Joachim de Greeff, Koen Hindriks, Mark Neerincx, Petter Ögren, Tomáš Svoboda, and Rainer Worst. Tradr project: Long-term human-robot teaming for robot assisted disaster response. *KI-Künstliche Intelligenz*, 29(2):193–201, 2015.

[12] C Ronald Kube and Hong Zhang. Collective robotics: From social insects to robots. *Adaptive behavior*, 2(2):189–218, 1993.

[13] D Micheli, A Delfini, F Santoni, F Volpini, and M Marchetti. Measurement of electromagnetic field attenuation by building walls in the mobile phone and satellite navigation frequency bands. *Antennas and Wireless Propagation Letters, IEEE*, 14:698–702, 2015.

[14] Tamer Nadeem, Sasan Dashtinezhad, Chunyuan Liao, and Liviu Iftode. Trafficview: traffic data dissemination using car-to-car communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 8(3):6–19, 2004.

[15] Parmy Olson. Softbank's Robotics Business Prepares To Scale Up, 2018. https://www.forbes.com/sites/parmyolson/2018/05/30/softbank-robotics-business-pepper-boston-dynamics/.

[16] Lynne E Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE transactions on robotics and automation*, 14(2):220–240, 1998.

[17] Chris Rozemuller, Koen V Hindriks, and Mark A Neerincx. On the need for a coordination mechanism to guarantee task completion in a cooperative team. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2817–2822. IEEE, 2015.

[18] Chris Rozemuller, Mark A Neerincx, and Koen V Hindriks. On the effects of team size and communication load on the performance in exploration games. In *ICAART (2)*, pages 221–230, 2018.

[19] Paul E Rybski, Sascha A Stoeter, Maria Gini, Dean F Hougen, and Nikolaos P Papanikolopoulos. Performance of a distributed robotic system using shared communications channels. *IEEE transactions on Robotics and Automation*, 18(5):713–727, 2002.

[20] Patrick Ulam and Ronald C Arkin. When good communication go bad: communications recovery for multi-robot teams. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3727–3734. IEEE, 2004.

[21] Joris Z. van den Oever. Experiment code and Results, 2020. https://github.com/jzvandenoever/agent-communication-thesis/releases/tag/experiment-results.

[22] C Wei. Cognitive coordination for cooperative multi-robot teamwork. 2015.

[23] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. Multi-robot cooperative pathfinding: A decentralized approach. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 21–31. Springer, 2014.

[24] Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. The role of communication in coordination protocols for cooperative robot teams. In *ICAART (2)*, pages 28–39, 2014.

[25] Alan FT Winfield. Distributed sensing and data collection via broken ad hoc wireless connected networks of mobile robots. In *Distributed Autonomous Robotic Systems 4*, pages 273–282. Springer, 2000.

[26] Tiger TG Zhou, Dylan TX Zhou, and Andrew HB Zhou. Unmanned drone, robot system for delivering mail, goods, humanoid security, crisis negotiation, mobile payments, smart humanoid mailbox and wearable personal exoskeleton heavy load flying machine, May 23 2014. US Patent App. 14/285,659.