University of Magdeburg

School of Computer Science

Master's Thesis

Orchestration of Heterogenous Agents With Bonding Functions

Author:

Florian Uhde

2016

Advisors:

Prof. Sanaz Mostaghim

Department of Intelligent Systems (IKS)

Uhde, Florian:
Orchestration of Heterogenous Agents With Bonding Functions
Master's Thesis, University of Magdeburg, 2016.

Abstract

To produce efficient swarms in multi-agent-task based scenarios, team composition and task allocation play a critical role. This work conceptualizes and evaluates local decentralized mechanics to orchestrate a heterogeneous swarm due to a bonding mechanic in a dynamic environment. The parametric scope of this methods is analyzed and different settings are compared. The resulting systems are able to increase the efficiency in changing environments with dynamic targets, as well as as stabilize swarm performance in case of agent failure.
These mechanics are evaluated for their appliance in real world scenarios and as artificial intelligence in computer games.

Contents

List of Figures and Tables

1. Introduction

1.1 Introduction

Modern computer science tries to find algorithms to solve complex problems. One way to design an algorithm for such a task is basing it on natural or biological intelligence; So called 'intelligent systems' include swarm- and ant-simulation, evolutionary algorithms and many more [Eng07]. These systems "*Shaped by natural evolution [they] can withstand most diverse selective pressures, and the huge plasticity, exceptional flexibility, and high adaptability have resulted in an extraordinary global success*" [Mor15].

Swarm simulation tasks itself with the exploration and discovery of mechanics of naturally developed swarm systems. These systems, like ant colonies or bird flocks exhibit a kind of intelligence, by showing exceptional patterns, while being constructed by simple rules. Bonabea summarizes swarm intelligence as "*collective behavior that emerges from a group of social insects*" [BT00] .
One of the common challenges of swarm simulation is the coordination of a large group of simple agents. By cooperating, these simple agents can form emergent behaviors and work towards higher level goals, for example optimizing multi-lateral functions or finding and tracking objects.
This work conceptualizes ways to inter-mesh a swarm of agents, which are all heterogeneous in terms of capabilities. By introducing task acceptance mechanics for single agents, as well as a way of communication between agents the swarm is empowered to fulfill a predefined goal. In this case the agents are tasked with locating, tracking and chasing a number of objects, which are moving in a dynamic pre-defined environment. The agents need to gather and interpret sensual inputs, emitted by the environment and the moving objects, to form an internal representation of the current situation. Agents which excel at this kind of game are expected to do well in other scenarios with changing environments, where prioritization and multi-agent organization are required.

1.2 Motivation

The behavior and performance of agents in this scenario maps to two different real world applications. First, by looking at robustness of a swarm with only a few competent agents it helps to decide for a **minimal sensor set**[1] needed to control a swarm (like drones with visual sensors for guidance). Efficient procedures can help to reduce the total sensor count in a swarm, making space for other equipment or increasing battery life due to the lighter drones and reduced strain from sensors. This in turn can allow for extended range and the creation of multi-purpose swarms, distributing all required sensors between subsets of the swarm and giving it means to communicate findings efficiently and act accordingly.
The second scenario evaluates the ability of the agents to **navigate and orientate in a dynamic environment**. This is interesting for real world situations where you need to find or track objects in a non static or unknown environment. An example could be human rescue after catastrophes, where a swarm of drones, tasked with search and rescue, needs to be highly adaptable to different environmental influences. This case maps well onto the task the agents have to fulfill in this thesis, allowing findings to be adopted easily.
Beside these real world examples findings from this thesis can also be used in computer games. By giving the swarm the ability to adapt to different scenarios, while still maintaining swarm integrity and following a high level goal, games can **adopt to different user driven play styles and form emergence in story telling**. Instead of scripted events and patterns the game can utilize dynamic functions to react to player input and, for example, counteract the player's strategy.

With regards to these given scenarios this work intends to check different approaches and shed some light on two questions: first, can simple task processing and communication mechanics provide emergent and adaptive swarm behavior to locate and track down objects; and second, if so can these mechanics be tweaked to provide an effective solution for real world problems.

1.3 Structure of the Thesis

This thesis gives a short introduction into *swarm simulation* and commonly used formulas (Section 2.1.1) and *division of labor* (Section 2.1.2) in the following chapter. Section 2.2 provides a context of other scientific works which cope with a similar theme and positions this work in a broader scientific context. In Chapter 3 the main concepts used in this work are presented and explained. After explaining some commonly used terms it focuses on the core modules developed in this work the task acceptance and the bonding module. Section 4.1 evaluates the concepts and tests them. In the last chapter a conclusion is drawn, incorporating possible future work and open problems. It gives an outlook about what can be done with the findings and where they can be applied.

[1] A minimal sensor set being the smallest possible equipment of sensors across the swarm to still fulfill its purpose (see [BPH10])

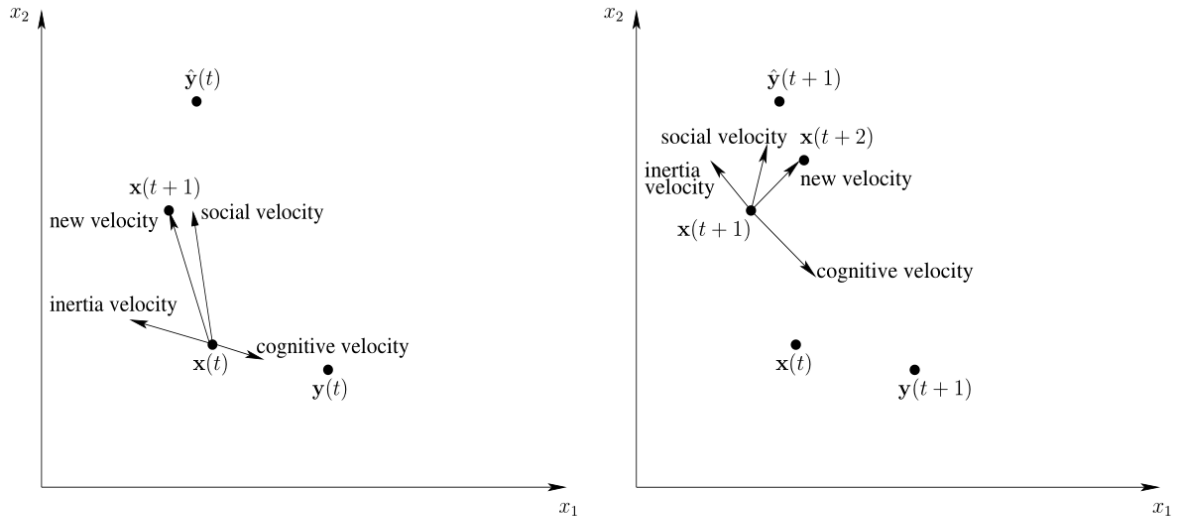
2. Background

2.1 Theory

This chapter describes some fundamental concepts behind this thesis. In Section 2.1.1 and Section 2.1.2 two core components of swarm simulation, the swarm function and the idea of division of labor are explained and put into the concept of this thesis. The related work chapters will highlight similar approaches and place this work into a broader scientific context.

2.1.1 Swarm Function

The swarm function is a mathematical representation of the inner working of a single instance. It calculates the movement direction of the agents. As stated in [CWM04], a swarm function usually consists of a repulsion and an attraction factor. The agent is drawn towards good solution candidates, but also tries to maintain its distance to other agents. Engelbrecht names three components for a swarm function: the **inertia velocity**, carrying over the value from the last calculation step, like a momentum. The **cognitive component**, like a memory, pulling the particle to its own memorized best location. And finally the **social component**, denoting the desired velocity of other agents in proximity [Eng07]. Each of this components produces a velocity, combined they result in the actual velocity for the agent at this time step. See Figure 2.1 for an exemple of these components and their influence on the agents behavior.
These components can be weighted to create different patterns. For example by masking out the cognitive part you can achieve a social-only model. Here agents will only move towards the best results in their neighborhood, regardless of previous personal best [Eng07]. The social model works well in a changing environment, (see [CD00]), which is given in this work, as the targets and thus the solution candidates tend to move around and are removed completely once fulfilled.



(a) Time Step t (b) Time Step $t+1$

Figure 2.1: Illustration of the three different components [Eng07]

2.1.2 Division of labor

Division of labor is one of the core concepts found when observing swarm intelligence in social groups. In order for this groups to thrive different activities need to be done by parts of the population. The process of distributing these tasks and allocating individuals to do them is called task allocation [KB00]. Insects have been in the focus of this research of task and role assignment[1], because of their simple, yet robust mechanics used for this. Within wasp populations of the *Ropalidia marginata* tasks are assigned based on a relative age withing the swarm, therefore distributed evenly, with the possibility to react to population changes automatically [NG99]. [Rob92] has summarized work on the plasticity within this process, pointing out the importance of flexibility in task allocation. Using the activation-threshold model where "*individuals react to stimuli that are intrinsically bound to the task to be accomplished*" [KB00] leads to a dynamic process of task adaption. Different thresholds withing a swarms population lead to a gradual 'recruitment'. This leads to a total fitness gain, when task are distributed and activated based on the agents' age.
This work adopts parts of the activation-threshold mechanics, and permutes others. The gradient thresholds are provided by different skills of the receiving agents. Depending on the type of stimuli presented agents will receive a strong urge to fulfill this task, while others have only little desire to partake of this activity. A simple implementation allows to calculate the probability to take a task with $P_{\theta_{kj}}(s_j) = \frac{s_j^\omega}{s_j^\omega + \theta_{kj}^\omega}$. θ_{kj} is called the response threshold, noting how likely an individual k will respond to a stimulus s_j,

[1] *Task* specifying: "What needs to be done", contrary to *activity*: "what is being done" and *role*: "who of this groups needs to do it" [KB00]

based on task j. ω sets the steepness of the threshold curve [Eng07]. This mechanic falls short if it comes to temporal polytheism and stability in selection over time [Eng07]. This can be addressed by allowing threshold values to vary over time (see [Eng07])

2.2 Related Work

The field of swarm based optimization is a widely researched field, often containing ant simulation and division of labor in [Rob92]. Depending on the underlying goal single agents may take up a single task [GM04] or a group of multiple agents may be required to fulfill a single duty [Mor15][SK98]. This work uses multi-agent tasks to allow agents to play a game of *hide and seek and catch*. This game has complex requirements regarding the understanding of the environment and communication [TSP+06]. With moving targets and variable effects in the scenarios this game features dynamic environments, an additional challenge which was already discussed in [CD00]. Adding in non-reciprocal swarms as observed in [CWM04] complex emergent behavior is possible using only simple bonding mechanics. This mechanics consist of team formation [Mor15][MDJ07], task allocation [SK98][KB00][MDJ07] and the movement itself. This work investigates on mechanics to intrinsically solve multi-agent tasks without the agents being aware of the constraining factors.

When looking at task allocation the spectrum ranges from one single agent coordinating the swarm, to a fully-distributed systems, with only local information at hand [DZKS06][LMAM04]. On the one hand global agents can produce optimal task assignments in theory, in real use cases it only works well for small teams within static environments, where planning ahead is possible [DZKS06]. On the other hand Fully-distributed systems allow for fast results, based on local knowledge, are robust to failures of single agents, but may produce globally suboptimal results [DZKS06]. [KB00] proposes a simple yet solid algorithm, the *activation-threshold mechanism*, which this work's task allocation is based upon. This algorithm is expanded in [AME04] to allow for a single threshold over the whole population, by only evaluating local demand.

The core requirement of multi-agent task systems is the team formation [GD08][MJT13], represented by the bonding behavior in this work. The overall structure within the teams "*dictates the interactions among the agents, and can play a significant role in the overall performance of the agent society*" [Gd05]. Traditional approaches use skill values of the agents to maximize the expected values of teams [Gut08], while more recent approaches focus also on the overall synergy between the agents [LV12]. This allows to monitor diversity and adjacency of skills when planning a model for team formation. Diversity within the teams plays an important role [MJT13], especially in dynamic environments, like the one featured in this work. A diverse composition can allow for much better results than just multitudes of a single well performing agent [MJT13]. In [LMAM04] a way of measuring and adapting diversity, respectively specialization is given.

Another approach is to use a market-like bidding system to allow the agents to form ad-hoc teams when dealing with different task [DZKS06]. This systems can deal well with failure and allow a dynamic allocation based on parameters [SABS05]. Ad-hoc

regrouping has been considered in [Mor15], where groups of agents negotiate exchanges via moderators.

After team formation a system needs to be in place to allow the agents to communicate with each other. [GD08] analyzes different communication structures for a network of agents. Team formation and flow of information is controlled by setting states of availability for each agent. Graphs can be used to model the underlying structures and to analyze their effect (see [Gd05] for graphs within dynamic team formation). These graph can be observed further to measure the formation of beneficial structures like hubs and portals, which enables information to traverse graph faster [New03].
This work uses a locally distributed system as a base to incorporate communication mechanics to create a flow of information between the agents. Contrary to [DZKS06] the task allocation system is rather simple, based on a permuted activation-threshold mechanism [KB00].

3. Orchestrating a swarm

In this chapter the core concept of the work is conceptualized. Behaviors, which determine the inner workings of an agent, are defined and explained.

3.1 Terms and definitions

This sections will cover most of the frequently terms used terms in this work, as well as semantic meaning and limitations of these.

3.1.1 Agent

All individuals in this simulation are called agents. One can distinguish between hiding agents and seeking agents. The hiding agents use a simple behavior. They can move along pre-defined paths or can be controlled by the user via mouse input. Hiding agents are located by the seekers through their emitted signals. A hiding agent is *visible*, it emits a *sound* when moving and leaves a trail of *odor*.

Seeking agents are controlled by the modular artificial intelligence conceptualized in this work. They are all arranged in a swarm, with different bonds representing smaller subgroups. Seeking agents receive signals emitted by hiding agents and decide how to pursue them based on this information. They communicate inside their bonds and implement different methods to evaluate and accept tasks.

3.1.2 Swarm

A swarm entitles a conglomerate of agents. In this work only the seeking agents are considered as a swarm, as the hiding agents all move autonomously without any way of communicating or cooperating. Besides the logical grouping of agents a swarm will also describe the syntactic structure in regard of distribution of abilities. For example the *single-competent swarm* is made of senseless agents, except one agent capable of receiving inputs. Section 4.1.3 goes over the different swarm configuration used in this work and their attributes in more detail.

3.1.3 Senses

Senses provide an ability set to each agent. This work deals with three different senses derived from human sensual inputs. Each agent has an individual sense profile describing how prone it is to receive (seeking agent) or emit (hiding agent) information of the particular sense (see Section 3.1.4). The profile denotes a basic capability for each sense, as well as a way to modify these for short term adjustments. Senses are considered static besides this, without any training or learning effect.

3.1.4 Signal

Signals are the medium to provide **information** from the outside world. Common sources of signals are the hiding agents or different scenario elements. Each signal anchors in a specific location, has a strength and a type. The type is based on the three available senses, seeing, hearing and smelling. Signals also have a certain range, a finite lifetime and a decay type. These influence the strength of the signal, depending on when and where agents read it. A *linear decay* type for instance will decrease the strength of a signal slowly, while *instant decay* will lead to a full strength signal, disappearing immediately after its lifetime is reached.

3.1.5 Scenario

A scenario is a predefined map, including a certain topology and scenario elements. Each scenario is constructed in a way to increase or suppress effects exhibited by the agents. The diverse topologies also favor some senses over others, forcing the swarms to adopt to different situations. This effect also stems from scenario elements producing noise or canceling different kinds of signals. The idea is to artificially simulate distinct situations like the aftermath of catastrophic events or certain topology to evaluate the swarm orchestration under these restrictions. Section 4.1.4 will present the used scenarios, their characteristics and the aspects focused by them.

3.1.6 Task

After [GM04] a task can be seen as an agent-internal representation of a sub goal, required to reach the overall goal. Each task is the direct result of a received signal. A task receives a priority by multiplying the signals received strength with the agent's skill to interpret this signal. For example an agent with 0.5 seeing strength can only generate tasks from optical signals in the priority range 0-0.5. A deaf agent will never be able to generate tasks based on sounds. Besides the priority task also have a target position and carry information about their creator. The bonding module used this information to evaluate trust between seekers.

3.1.7 Bond

A *bond* denotes a connection between seeking agents. Within a bond information can be shared freely, as in a radio network. Bonds always are uni-directional and contain one

receiving agent and many sending agents. This setup simulates the formation of groups, with possibilities of overlaps and separation. One agent can receive information from a defined group of sources and pass this information to other agents. Especially with swarms without equally distributed skills this may benefit the formation of groups, as closed, bi-directional bonds will either lead to one single big cluster of agents or bonds without any capable agent. The characteristics and arising features of this mechanics are discussed in Section 3.5.

3.2 Concept

To fit the scenarios described in Section 1.2 the simulation had to fulfill certain criteria:

- It should allow for *dynamic and changing environments*.
- It should enable *focusing and stressing of single components* in the agent AI.
- It should be *simple* and *comparable*.

In the end a concept like **hide and seek** was chosen, with adjustments to fit the above criteria. This game focuses understanding and control over certain non-trivial concepts. Hide and seek requires awareness of the state of the game. Usually the game is played in a large environment, where not all features and information are available to the seekers at every moment of the game [TSP+06]. This is interesting in matters of the cases brought up in the motivation; computer games as well as catastrophic scenarios feature a large dynamic environment. Agents/Drones need the ability to work inside of physical limitation like weight or memory and still be able to fulfill their goal. In this variation of hide and seek two kinds of agents exist: *seeking agents* controlled by AI and *hiding agents* controlled by players or following simple routes. To focus on the bonding mechanics the agents need to locate targets based on their senses and then track them down. To catch and remove a hiding agent from the game, seekers need to accumulate in a certain number around the target.
One important consideration is the diversity between the seeking agent. While swarm simulation often deals with similar and homogenous swarms [LMAM04] we face a very distinct mixture of agents. While this makes some 'classical' concepts harder to apply and to balance, it can result in great emergence and complex behavior [CWM04].

Agents are given a certain sense profile, describing their capability in the three implemented senses; seeing, hearing and smelling. Each sense is able to receive differently structured impulses:

- *Seeing* allows to establish an ad-hoc connection, which remains active as long as the target is in the field of view.
- *Hearing* is able to catch wide-range, short lived impulses emitted by players when moving around.

- *Smelling* locates long lived, locally restricted impulses, left by players to linger a long time.

This simulation uses four different configurations of swarms, each characterized by a certain combination of agents (see Section 4.1.3. All simulations take place in different, predefined scenarios. Each scenario presents a different challenge profile, some restrict certain senses, others challenge the task acceptance by providing a lot of noise and false information, see Section 4.1.4 for a detailed description of the scenarios used.

The main concept, based on the motivation is a scenario, with medium background noise and more hiding than seeking agents. This emblematizes a possible situation after a catastrophic event. A few assumptions were made:
Agents only know indications of targets: This means an agent will not be able to track a target for its own sake. Agents need to rely on the information generated by the target and emitted into the environment to track it.
Agents are able to communicate within each other: The concept assumes that some way of communication can be established between agents. They may use short range communication as well as long range communication within their bonds. This situation may not always be given, especially in real world scenarios. Section 5.3 will shed some more light about implications and ways to implement this.
Agents have a static feature set: This work does not implement any learning mechanic, agents acquire their skills at the start of a run and keep them over the course of it. See Section 5.3 for an outlook about learning and adopting agents.
Agents only have local knowledge: Contrary to [KNZ10] agents do not have access to a global storage of information. They can only rely on their own inputs (cognitive component) and information shared through bond (social component).

As stated in Section 1.2 this works aims for simple mechanics to provoke emergent behavior and the usability in real world scenarios. Looking at the definition of a swarm function according to [CWM04] there is a repulsion and an attraction part. The repulsion is generated by using Unity 3D's[1] own navigation systems. Agents are configured in a way they avoid each other to a certain extend. While this cannot be put on the same level as repulsion mentioned in [Eng07] it allows to spread the agents while moving through the environment. A much larger repulsion, forcing the agents to cover a larger area, can't be implemented in this simulation, as agents need to converge to the target to mark it as done. Estimating the optimal moment to converge to catch a target would require the agent to have full understanding of different simulation parameters, which was excluded in the assumptions.
The attraction is generated from bonding. Agents with the same target will converge towards each other and align in their movement, if the environment allows for this. This leads to one of the core challenge the agents have to face: *Prioritization*. The agents should try to catch as many targets as possible. These hiding agents have a certain spread over the scenario and a distinct position, both unknown to the seeker.

[1]Unity 3D is the game engine used for the implementation of this concept. See Section 4.1.1 for more details.

The mechanics implemented need to account for the fact that agents can only rely on spacial mapped information. Each input carries a position but no target, so agents can not be sure that they are chasing different targets, or are focusing the same hider. This leads to the need in *swarm topologies* to allow the accumulation of enough agents to finalize a target, while still maintaining a broad spread over the scenario to locate other targets and pull the different groups towards them.

3.3 Overview

A single run in the simulation starts by placing seeking and hiding agents into the scenario. All agents are aware of the topology of the scene in the sense that they can decide on an optimal route between two given points. The hiding agents can be controlled by user input, sending them to clicked positions in the scene. Without user control the agents will walk around a way-point path, looping over it until killed. Also as soon as the simulation starts the hiding agents will start emitting different signals, which are received by the seeking agents.

The internal working of a seeking agent consists of three major steps. First it will gather and receive all signals from the scene. This is done by logging every intersection with an olfactory or auditive signal and casting a visibility check for the hiding agents. This yields a set of signals, all different types with different strengths. The agent now transforms this set into a set of tasks. Each tasks has a priority based on its signal strength and the agents capability of interpreting this signal type. This list of tasks is passed into the task acceptance module which picks the best task in the current situation. At the beginning of the simulation agents are grouped in subsets, so called *bonds*. After choosing their personal best task agents communicate this task to their bond. Like a shared discussion each agent proposes its best option. The bonded group then decides on the total best task and pursues the target as a group.

Figure 3.1 Simplified definition of a SAgent

```
1: SAgent {
2: CurrentTask : Task                    ▷ Task currently pursued by the agent
3: _myImpulses : Impulse[]               ▷ Array of all received impulses
4: _myTasks : Task[]                     ▷ Array of all generated tasks
5: _myBonds : Bond[]                     ▷ Array of all receiving bonds
6: _holdingBonds : Bond[]                ▷ Array of all sending bonds
7: navigation                            ▷ Unity 3D navigation component
8: base                                  ▷ Base class, handling movement
9: }
```

For the purpose of simulation different settings can be activated in the simulation master. The game can be set into *headless* mode, running simulations with minimal graphical output to reduce GPU overhead and increase the frame rate. In certain simulations the 'kill by targeting' mode was enabled. In this mode agents to not physically need to

Figure 3.2 SAgent: Unity method loop

```
1: procedure UPDATE( )
2: Evaluate bonds:
3:     if _myBonds ≠ null then
4:         _myBonds.OrderByDescending(Priority * Trust)
5:         BondTask ← first valid task from _myBonds
6:         if CurrentTask.Priority < BondTask.Priority * BondTask.Trust then
7:             CurrentTask ← BondTask
8: Assign current task:
9:     if CurrentTask ≠ null then
10:        navigation.Target ← CurrentTask.Location
11: Gather and process impulses:
12:    _myTasks.Clear()
13:    for all Impulse imp in _myImpulses do
14:        _myTasks.Add(new Task(imp))
15:    _myImpulses.Clear()
16:    base.UPDATE( )
17: procedure LATEUPDATE( )
18: Gather and accept tasks:
19:    if GatherTaskFromNearby then
20:        _gatheredTasks ← Tasks from agents in proximity
21:        _myTasks.AddRange(_gatheredTasks)
22:    TASKACCEPTANCEMODULE(_myTasks)
23: Bonding:
24:    BONDINGMODULE( )
```

Figure 3.3 SAgent: Task acceptance module

```
1:
2: procedure TASKACCEPTANCEMODULE(tasks)
3:     best ← tasks.BestTask()
4:     if TaskAcceptance mode == SIMPLE then
5:         if CurrentTask.Priority < best.Priority then
6:             CurrentTask ← best
7:     else if TaskAcceptance mode == SINGLE then
8:         if best.Priority ≥ FirstThreshold then
9:             if CurrentTask.Priority < best.Priority then
10:                CurrentTask ← best
11:    else if TaskAcceptance mode == DOUBLE then
12:        if best.Priority ≥ FirstThreshold then
13:            if best.Priority ≥ SecondThreshold then
14:                if CurrentTask.Priority < best.Priority then
15:                    CurrentTask ← best
```

Figure 3.4 SAgent: Bonding module

```
1: procedure InitBondingModule( )
2:     _myBonds ← newList < Bond > ()
3:     if Bonding mode == RANDOM then
4:         b : Bond ← newBond()
5:         b.Add(RandomAgents(AgentPerBond))
6:         _myBonds.Add(b)
7:     else if Bonding mode == SKILL then
8:         b : Bond ← newBond()
9:         b.Add(RandomBetterAgents(AgentPerBond))
10:        _myBonds.Add(b)
11:    else if Bonding mode == REINFORCED then
12:        for i:int = 0; i<NumberOfBonds do
13:            b : Bond ← newBond()
14:            b.Add(RandomAgents(AgentPerBond))
15:            _myBonds.Add(b)
16: procedure BondingModule( )
17:    for all b:Bonds agent is part of do
18:        b.Add(CurrentTask)
```

reach a hiding agent. Instead it can be marked as removed by targeting it. As soon as the number of targeting agents surmounts the required number to kill, the hiding agent is removed and all agents targeting it will be put into sleep mode for a certain time. Sleep mode simulates the effect of moving towards the agent and punishes clumping of agents. The time an agent is put to sleep is based on the distance between agent and target: $sleeptime(agent, target) = \frac{distance(agent,target)}{maximumDistance} * 10$, $maximumDistance$ being the maximum possible range between agent and target, thus mapping the fraction within 0 and 1. 10 is the time in seconds it would take an agent to walk the maximum distance. This mode is especially useful when exploring parameters in a simple environment, where the focus is more on the interaction and internal mechanics of agents, and less on their interaction with the environment.

3.4 Task acceptance

The task acceptance module deals with the question which objective to follow. In this experiment each agent can only have one task at a time and will pursue the location emitting the base signal. A task acceptance module selects the most promising task from the set of tasks generated by signal inputs. This work evaluates three different kind of task acceptance. By changing the task acceptance module the simulation is influencing the process of labor division as described in Section 2.1.2. In [KB00] an activation-threshold mechanism is used, which distributes task through agent classes based on an extrinsic impulse. [DZKS06] points out the importance of decentralized task allocation when dealing with dynamic environments, like in this simulation.

The *greedy and simple* approach just picks the task with the highest priority, regardless of the currently selected objective and absolute strength of the picked one. This will lead to quick decision making, but may result in a lot of backtracking, where agents are moving back and forth between targets, or even a lock situation.

A more sophisticated approach using a *threshold* allows to set a minimum priority for a task to be considered. This is expected to perform well in scenarios with a certain background noise, by keeping the agent from switching tasks because of noise too often. This approach has the downside that an agent may ignore potential good tasks if the threshold is set too high, or will be influenced heavily by noise because of a low threshold. Section 4.1.5.2 will consider this challenges and discuss different setting for the threshold.

The third module implementation extends on the threshold idea by adding a second, upper threshold. An agent will consider the first threshold while not having any task, but to overwrite an already accepted task the new one must surpass the higher threshold. This forces the agent to stick to the already started task unless a much better alternative presents itself. By holding on to tasks loss of progress is prevented. *Double threshold* is expected to work very well in surroundings where a lot of short lived impulses tend to occur, like the *Forest* as described in Section 4.1.4. This emulates the idea behind the concept int Section 2.1.2. Instead of a probability function two different states are simulated per agent. A low initial threshold will circumvent the problem of missed weak tasks. A higher secondary threshold will allow the agents to stick to their tasks and ignore other tasks until they are done, as long as these task are not significantly higher prioritized.

3.5 Bonding

The bonding module manages communication and cooperation between agents. It simulates collective decision making behavior by creating communication channels between agents[Ken99]. Agents will share information and current targets with their bonded counterpart. At start the bonding modules are initialized and form subgroups according to their configuration. This work examines three different bonding strategies with increasing complexity.
Each agent A receives input from a bond α, which in turn receives its information from other agents B_i. A itself can be part of other bonds β_{ii} with agents O_{iii}, with $i, ii, iii \in \{0, AgentCount - 1\}$ and potentially $i \neq ii \neq iii$.
To simulate overlapping groups bond connections are not bi-directional. This implies that $\{B_i\} \neq \{O_{iii}\}$, which means that A may give its own information to other agents than it is receiving information from. This simulates the flow of information between different groups. A may propagate a high prioritized task it received in the current frame from his bond α into all bonds β_{ii}.

The simplest approach of this implementation just meshes agents together randomly at start up. The *Agent Per Bond* parameter allows the customization of the degree of separation within the swarm. This systems create one bond for each agent, with varying

number of members. A high *Agent Per Bond* parameter will lead to a well connected swarm, which behaves in a uniform way. A low value will create a highly separated network and produce groups too small to actually kill hiding agents.
The second approach works like the above one. It will also create one bond per agent, based on the bonding division parameter. This time the partnering agents won't be picked completely at random. Instead, every agent will try to only bond with agents which are better in at least one skill that the agent itself. This will raise the average quality of the information provided through the bond by assembling better teams [Mor15]. This is expected to work especially well in swarms with only a few competent agents, as it guarantees a connection to at least one good agent. Emergence of leaders is reinforced by this approach. They are more likely to appear in many bonds as information providers, and thus influence more agents in total.
The last approach can be seen as an variation of the other two. While maintaining each of the two bond creation patterns it establishes more than one bond per agent. Each single bond is assigned a *trust* value, indicating the likelihood to provide good information. During the simulation the agent will reevaluate the quality of each bond by comparing suggested objectives with actual events. If the agent participates in a kill it will increase the trust of all bonds which suggested to go after the just killed target. The trust value of other bonds will decrease slightly.

4. Evaluation

4.1 Experiment

The following chapter illustrates how the concept was implemented and what kind of experiments were done with the implementation. To test the variety of settings and functions of bonding mechanics a simulation framework was created as described in Section 4.1.1. Within this framework different metrics were defined and evaluated.

4.1.1 Test framework

To run this tests a framework was constructed in the Unity 3D[1] game engine. This engine provides tooling to calculate and visualize the swarm simulation in an efficient way, while also providing a framework to calibrate different aspects of the tests easily. Using Unity's internal component system (see [Uni16] for an in depth explanation of *GameObjects* and *Components*) different behaviors were developed to execute the concepts developed in Chapter 3. Beside being a popular game engine, Unity 3D is also suited for simulation and serious games[2]. [Pen15] points out some benefits of Unity and compares the engine to its direct competitors. Unity itself implements different patterns commonly used for developing large complex systems. Each object in Unity is describable by the sum of its component. Different implementations can be found via the service locator pattern[3], implemented by the *GetComponent()* function.
This system allows for dependency injection[4], by switching out different implementations

[1] https://unity3d.com/ Unity Technologies, accessed 27.05.2016

[2] https://unity3d.com/unity/industries/sim, accessed 20.07.2016

[3] A pattern to hide implementation details and locate the right components with a universal locator class. See: https://msdn.microsoft.com/en-us/library/ff648968.aspx Microsoft Cooperation, accessed 20.07.2016

[4] A pattern allowing the replacement of loosely coupled class mechanics, without recompiling the code base. See: https://msdn.microsoft.com/en-us/library/hh323705(v=vs.100).aspx Microsoft Cooperation, accessed 20.07.2016

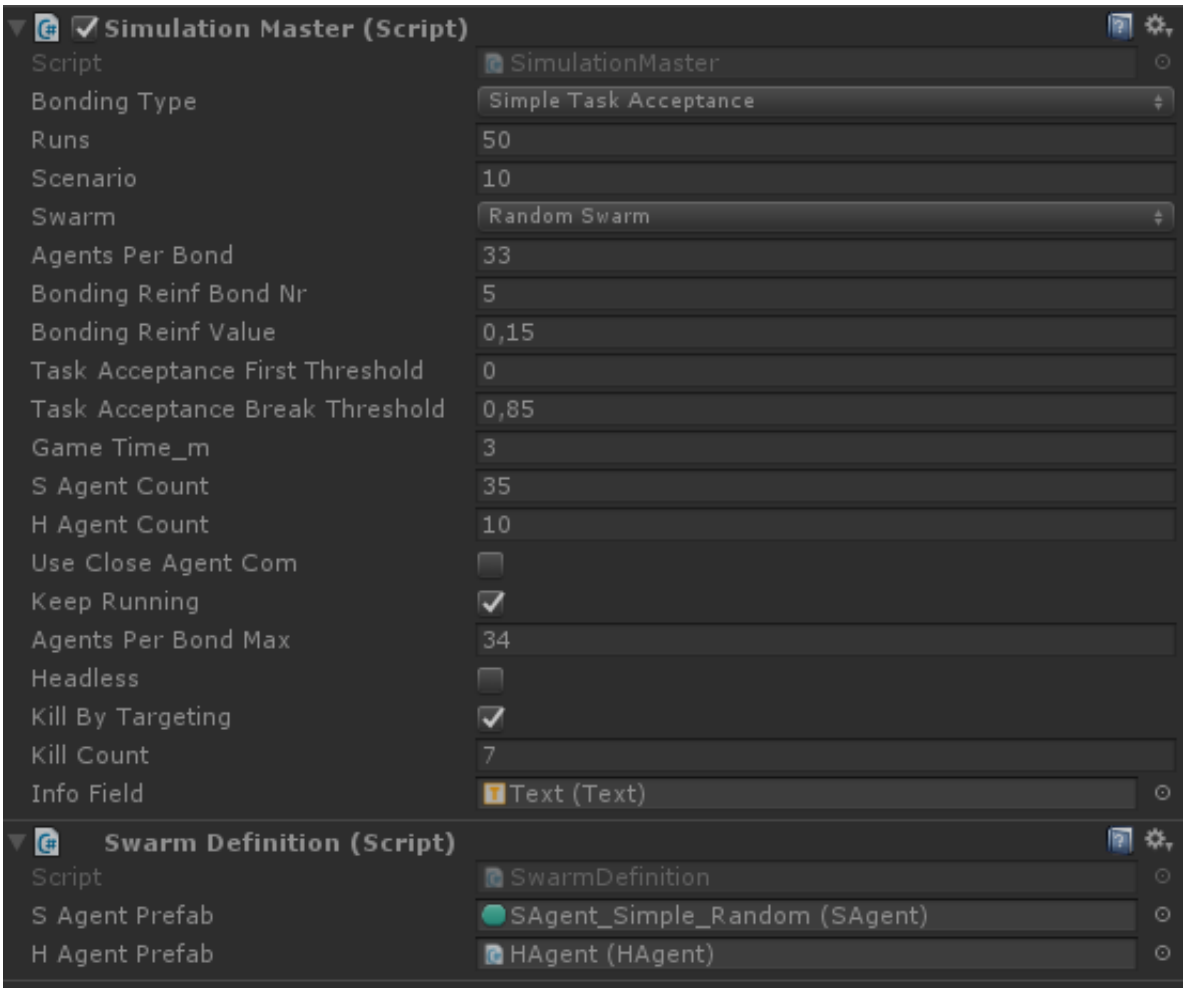


Figure 4.1: Example configuration of simulation master

of core components. This simplifies the process of creating different characteristics per mechanic and testing them. The agents were implemented derived from a base class and agnostic to their specific behavior (e.g. what kind of bonding is actually used). In combination with Unity 3D's component system this allowed for easy switching of the test population and batching of test runs. Due to C#[5] being Unity 3D's programming language it was very easy to connect custom analysis tools and link them to the pipeline.

Another useful feature of Unity 3D is the extensibility of the editor itself. Custom dialogs and controls were engineered to allow running the simulation directly from the Unity 3D tools, therefore conserving all setting options provided by them. Trough the *SimulationMaster* class a central point of control was established, providing all means to configure and save a simulation run. A rundown of the available parameters can be seen in Figure 4.3. Figure 4.1 show an example configuration for a simulation. The simulation

[5]https://msdn.microsoft.com/en-us/library/kx37x362.aspx, accessed 24.07.2016

master stays persistent over all runs, which are organized by the scene system in Unity 3D[6] and loaded in additively. When starting the game inside the editor the simulation master instantiates all background workers, like logging and instance factories. After configuring and starting the simulation the game finishes run after run, writing log data into comma separated value files.
During a simulation run the editor can be used to inspect all elements of the simulation and adjust parameters on the go if needed. In Figure 4.2 a snapshot from a running simulation is shown. From left to right the *scene view* shows the current situation in the level, the *scene tree* denotes all active game objects and allows to select them to load their *inspector*, visible on the very right. The inspector reflects all the current values which define the selected game object. The values are grouped by their modules and may be changed during run time, to evaluate different configurations.

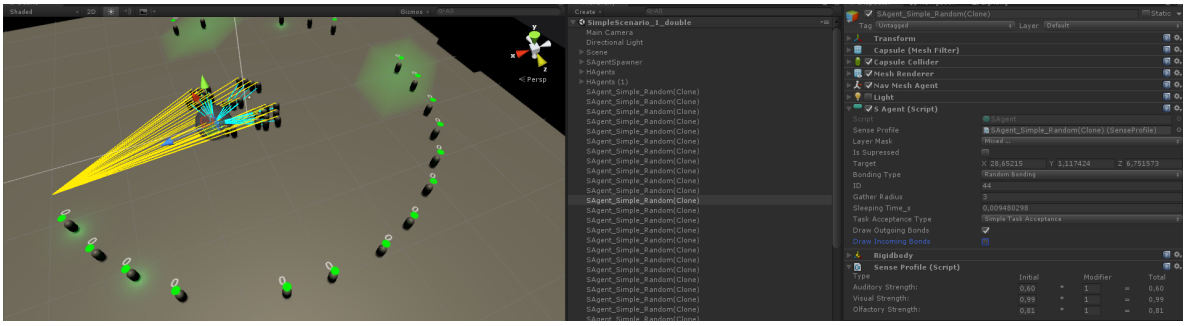


Figure 4.2: Scene view, scene tree and inspector of SAgent during a run

4.1.2 Metrics

Distances

During the runs a multitude of data is collected and exported for further processing. For this work some key points have been chosen as main points of interest to evaluate quality and performance of features. While the agents are not imagined to know their distance to the individual targets the **total distance** is a good measurement to evaluate swarm performance. This can be seen as an analogy of the often used *Pareto front*. While a *Pareto front* contains optimal solutions to a *Multi-objective Optimization* problem the location of hiding agents are one partial solution to the problem assigned in this work [MT04]. The task of the swarm is to reduce the number of hiding agents to zero. Following this through we can reason a function based on total swarm distance $td(S,H) = \sum_{s=0}^{S} \sum_{h=0}^{H} distance(s,h)$ with S being all the seeking agents and H being all the hiding agents. Minimizing this function can be interpreted as the seekers closing in on the hiders and therefore also working towards the higher goal. This correlates with commonly used pattern in evaluating swarms by their closeness to known *Pareto solutions* [LE08][dSC09].

[6] https://docs.unity3d.com/Manual/MultiSceneEditing.html, accessed 27.07.2016

Parameter	Description
Bonding Type	Used bonding module
Runs	Number of runs
Scenario	Scene template to simulate in
Swarm	Factory for producing the swarm
Agents Per Bond	Number of agents to feed information into one bond
Bonding Reinf Bond Nr	Number of bonds created initially per agent *[Reinforced Bonding]*
Bonding Reinf Value	Value to increase bonds trust when scoring a kill *[Reinforced Bonding]*
Task Acceptance First Threshold	Priority threshold a signal need to surpass to be considered *[Single Threshold TA]*
Task Acceptance Break Threshold	Priority threshold a signal need to surpass to be considered, if there is already a accepted task *[Single- & Double-Threshold TA]*
Game Time_m	Maximum run time of a single run
S Agent Count	Number of seeking agents to spawn if spawn point is present
H Agent Count	Number of hiding agents to spawn if spawn point is present
Use Close Agent Com	Allow seeking agents to share tasks when in close proximity
Keep Running	Execute multiple run sets. Works with *AgentsPerBondMax*. After *Runs* runs have been simulated the wired parameter (in this case *AgentsPerBond*) will be increased and a new run set will start.
Agents Per Bond Max	Upper limit of incremented value. If *KeepRunning* is active the simulation will create run sets until this value is reached
Headless	
Kill By Targeting	Set *KillByTargeting* mode active
Kill Count	Number of agents required to remove a target

Table 4.3: SimulationMaster parameters

While this function td may never reach 0, even on a successful run, its slope can still give an idea about swarm performance. The steeper the slope of a trend line is, the faster the swarm is working towards the goal.
Besides the total distance the simulation also measures the sum of the **minimum distance** as $md(S,H) = \sum_{s=0}^{S} distance(s, closest(H))$. This value is expected to jump more than the total distance and can be seen more as a quality indicator looking at single agents, instead of the whole swarm. Different insights can be obtained by this measurement. In a very well working scenario the jumps, which will occur every time an agent is removed, will be quite small. This means that the agents are already positioned close to the next target. Big jumps however show misplanning and long ways which will lower the efficiency in the end. The size of the jumps also allows to indicate the size of the killing group. More agents in the group will lead to bigger jumps, as the closest target for more agents is suddenly farther away. In general it is desirable to have a steep declining curve with as little jumps back up as possible.

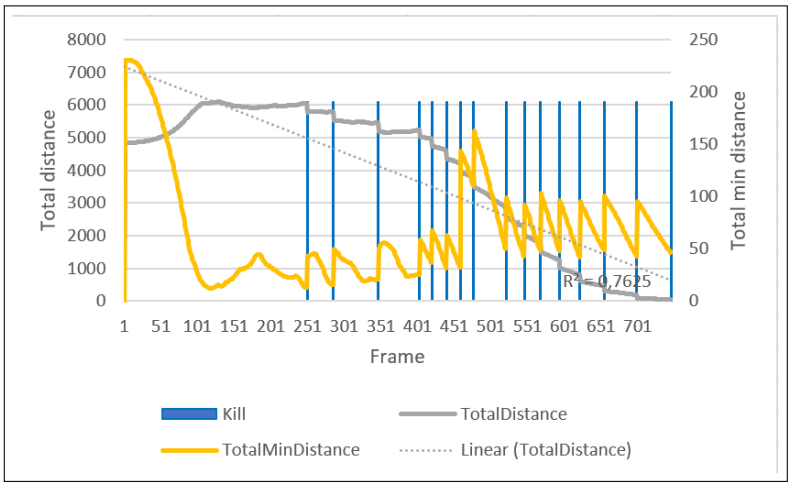


Figure 4.4: Sample run H:16, SA:10

Figure 4.4 shows a sample analysis of a single run. The settings for the run were the same as for bonding decision testing and can be found in Figure 4.8. The blue bars indicate moments when a hiding agent was captured and removed from the playing field, the yellow line is the minimum distance, the gray line represents the total distance and the dotted gray line a trend line for the total distance. All values are sampled once per frame[7], the frame count being the x-axis. The scenario used was an empty arena with a ring of hiding agents, in whose center the seeker are created. In the first phase a rapid decline of minimum distance can be observed, as the agents all move away from the center towards different agents. Total distance is on a rise because of the same reason. Around frame 120 the minimum distance reaches a local minimum, as most agents are close to their target, yet unable to kill it. It rises again as agents regroup and are finally able to make their first kill at frame 251. Because the closest target for some agents is removed the minimum distance value takes a jump up, while total distance is decreased. In this run two main group of agents have formed, one starting on the west edge of the circle, the other on the east end. Both groups move downwards towards the south terminal point of the circle. When the last agent is cleared

[7] a *frame* being one simulation step.

away both groups need to move up through half the circle to catch the next agent and merge in this process. This can be seen in the large jump at around frame 450. Here the last close agent was killed and the next target is way further up. Also the size of the jumps increases, indicating that more agents are part of the killing group.

Both of the distance values have to be taken with a grain of salt. Depending on the number of agents needed to kill a hiding agent and the total number of all agents it may be possible to minimize both functions and still have no success at all. A simple scenario for this is a run with an equal amount of hiding and seeking agents. Each seeking agent picks a different hiding agent as a target. The swarm will minimize both functions *td* and *md* and yet will not be able to finish a single task. It allows to get a base idea of the inner working of a swarm and can provide insights about performance when combined with other metrics.

Timing

Another obvious metric is the time stamp of the moment a hiding agent is removed. Or, more precisely the relation of the different time stamps. Looking at Figure 4.4 the vertical blue lines denote a moment a hiding agent was removed. In the beginning the swarm needs a moment to consolidate itself until it is able to perform the first kill. Here the single groups form themselves and start moving towards their first targets. It can be reasoned that at around frame 400 the number of groups with enough agents to kill increased, as the spacing between the kills decreases. The values around 450 solidify this assumption. When the minimum distance takes a jump at frame 450 there is no delay in the kill time, which means that another group was already positioned to make another kill. Right after the first big increase of distance this group kills of one the upper agents. Afterward there is a small delay, where all groups need to reposition themselves. Around frame 600 the groups appear to merge together, as the overall timing between two kills increases, and then stays consistent.

4.1.2.1 Efficiency

While evaluating the swarms with their distinct configurations different metrics where collected and evaluated. One metric used was the effectiveness of a swarm, describing its overall fitness and task solving capability. For this the time the swarm needs to kill a number of hiding agents is measured. It is calculated by the formula $e = \frac{deactivatedagents}{time} * 1000$. The scaling by 1000 was introduced solely to improve readability in the console application generating the results.

Figure 4.5 shows some baseline graphs for effectiveness. For the test data a scenario with six hiding agents was designed. Data was created for timing from 10 seconds up to 60 seconds, in steps of 10 seconds. Each line represents a simulated number of kills. For example, the yellow line shows the efficiency of a swarm which was able to kill two hiding agents in 10, 20, 30, 40, 50 and 60 seconds respective. While this metric allows to estimate a good general fitness value for each swarm it is also prone to errors of measurement. Evaluations using this metric need to address the fact that some frame

rate variations within Unity 3D may result in a skewed result, therefore some kind of trimming should be done with gathered data, to reduce the impact of stray values. To increase the explanatory power of this function simulation time was adopted to allow agents in most of the cases to fulfill the imposed goal. This fixes one axis of the function, allowing for more leveled results, while imposing a penalty for configurations that completely fail the task.

4.1.2.2 Swarm topology

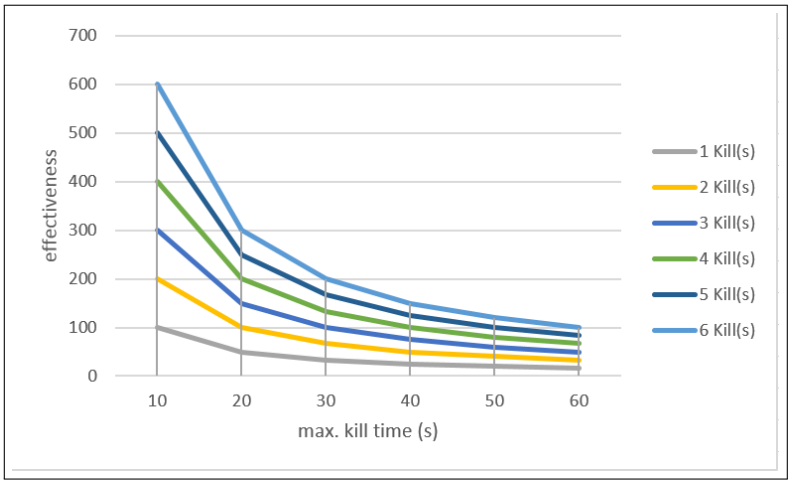


Figure 4.5: Effectivity of kill count with different timings

Another important factor to consider is the topology of the observed swarm. Bonding influences the flow of information in a swarm heavily and thus the capability to locate and track valuable targets. Kennedy presents different pattern from social communication in [Ken99], which are partly adapted in this chapter. Contrary to [Ken99] these features are not fit onto the swarm, they rather emerge from the bonding mechanics. To elaborate concepts and their characteristics measured during testing a sample network is provided in Figure 4.6. For the first two bonding modules these metrics can be evaluated once per run, as they are static and remain the same. For the third module, it is important to note that more connections exist due to the multitude of bonding connections present in each agent, but not all of them will be used.

Bidirectional connections

A bidirectional connection means that two agents A and B are inter-meshed in a way that both are contained in the other receiving bond. This means a full two way communication is possible between these agents and tasks can be exchanges freely. During experiments the number of bidirectional connections is measured, as well as the largest bidirectional network. This is interesting, as agents within this network have access to all tasks produced by their partners over a few frames. If the small delay of information transmission, which is one frame per hop, is neglected agents within a bidirectional network can be grouped logically to one meta agent. They can provide each other with the best tasks quickly and spread high priority tasks from outside the network to all partners. The size of the network will heavily influence the cohesion in the

swarm, as a very small number of high-priority tasks will be shared with all members of the network. In Figure 4.6 two bidirectional connections can be seen, between agent 1 and 2 and between agent 2 and 4. This means the largest bidirectional network covers 50 % of the swarm (3 of 6 agents).

Masters & slaves

Due to the asymmetric connection pattern it is possible that some agents are part of a one-sided communications. These agents are called **master agent** if they are only part of other bonds, that means they never receive tasks from other agents. If an agent only receives tasks, without pushing its own tasks in any bonds it is called a **slave agent**. Formalized the following relations for agent A stand: $master(A) \rightarrow \alpha_i \neq \emptyset \wedge count(\beta) = 0$ and $slave(A) \rightarrow count(\alpha_i) = \emptyset \wedge count(\beta) \neq 0$. α being the receiving bond, containing agents B_{ii} which provide information to A, and β being the set of bond receiving information from A (see: Section 3.5).

A master relationship can be created when using the skill based bonding with an agent

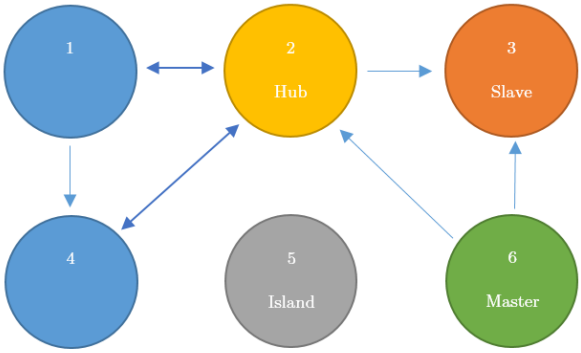


Figure 4.6: Sample topology

which has the best swarm-wide value in every skill. It will not be able to form a receiving bond, as it can not find any agent which is better in any skill. Slave relationships can be formed by chance, if an agent is just not picked to participate in any bond, or, when using the skill based bonding, for agent that do not dominate another agent in any skill. An example for master and slave agents can be seen in Figure 4.6. The green tinted master agent 6 provides his tasks to agent 3 and 2 while not receiving any tasks itself. This agent will therefore always pursuit his own highest prioritized goal. This can be beneficial, for example in the following situation: Agent A_2 has a very high skill value for smell, agent A_6 is mediocre at spotting targets. While A_2 will pull most of the swarm to

its current target (based on olfactory inputs) A_6 may pursue a different spotted target and provide tracking information as soon as the rest of the swarm finished A_2's target. Agent A_3 tinted orange has the slave role in this setup. It will only receive information, without the possibility to pass its own findings on. A_3 can therefore only tag along with other agents, if their task prioritization are high enough, or can move to its own targets without the ability to call for aid. This is generally an undesirable state, a high number of slaves may lead to bad performance of the overall swarm.

Hubs & islands

Similar to Section 4.1.2.2 hubs and islands allow to analyze the general topology of the swarm in more depth. **Islands** are agents with no connection to other agents. They can only rely on their own skills and may not provide insights to other agents. A high number of islands negates any behavior achieved by bonding mechanics, and no flow of information can be established. **Hubs**, being the counterpart to islands, denote agents with a high number of incoming and outgoing connections. In this work an agent is classified as hub when it is connected to at least 33% of the swarm population with incoming *and* outgoing connections. These connections do not have to be bidirectional.

An example of one hub and one island is given in Figure 4.6. Agent A_2 in yellow receives input from A_1,A_4 and A_6 while sharing its information with A_1, A_4 and A_3. Three incoming and outgoing connections each are over the threshold of 33% (1.98 agents) and A_2 can be considered a hub. Hubs allow information to spread quickly within a swarm, a package p which reaches a hub at time t_i will be distributed to at least $ceil(\frac{n}{3}) - 1$ agents[8] of the swarm at t_{i+1}. Hubs can be seen a broadcaster of information and filters, roles that Kennedy points out to be very important in effective communication [Ken99]. At the same time can hubs also suppress valuable information. From all the tasks they receive only the best is forwarded. This can lead to clumping and the loss of diversity within the swarms solution space [Ken99]. As soon as the threshold, defined by the number of agents to kill a target, is surpassed, the efficiency of the swarm suffers.

Contrary to A_2 there is the gray A_5 without any connection to other agents. A_5 is similar to the slave A_3, devoid of A_3's ability to receive tasks and follow them. A_2 can only follow private goals and has, just as A_3, no way of communicating them. Just as in Section 4.1.2.2 islands are not desirable and tend to decrease the performance of a swarm when stacked in number.

4.1.3 Swarms

Different types of swarms are expected to behave different in the test scenarios and reach varying successful results. These swarms are conceptualized to provide unique strengths and weaknesses to be able to analyze the used mechanics under these aspects. The evaluated swarms are:

[8] n being the number of agents, -1 because the providing agent could be part of the outgoing connections, therefore already knowing the information

- **Random**: In this swarm sense capabilities are randomly assigned a value between 0 and 1 on all agents.
- **Single Competence**: In the first version this swarm has only one agent which is able to execute all senses perfectly (strength of 1), all other agents have no senses at all, and are forced to follow the leader. The second variation has three agents, each able to perform one sense perfectly, while all the others have no competence in any skill.
- **Byzantine**: Derived from the byzantine generals problem [LSP82] this swarm features agents with a negative skill value. This agents will from time to time provide false information, trying to sidetrack the swarm. This allows to simulate sensor failures in real world drones, as well as provide a source of 'human error' in computer game AI. The modification can be combined with the first two characteristics.

4.1.4 Scenarios

As stated already in Section 3.1 a scenario is a *predefined scene* within the simulation environment. A scenario features some way of instantiating agents. These can be done with the use of spawn points, using Unity 3D's prefab system [Pen15] or by manually placing agents in the scene. Furthermore special elements, so called modifiers, may be added to create certain requirements withing the scenario. Modifiers are able to boost certain traits, like emitting loud noises whenever an agent passes it, or suppressing them, like making agents blind / invisible when inside its range.
Scenarios can be roughly divided into simple and complex variants. The **simple variants** were used to evaluate single parameters, constructed to allow the swarm to focus on singular mechanics, trying to minimize outside influences. These simple scenarios mostly feature an evenly distributed number of hiding agents, with spawn points for seeking agents in the middle. Their aim is to focus on task selection, bonding topology and to gather insights about the effect of different bonding parameters.
Complex scenarios employ findings from the simple variants to construct complex situations. Examples are changing the way signals propagate during the run, forcing previously powerful agents to rely on others, or demand complex patterns to locate and surround the targets. They usually do not utilize pre-placed hiding agents and rely rather on spawn points. This allows to tweak the number of hiding agents over different runs and evaluate the effect on the overall performance. A complete list of scenarios can be found in Section A.1.

4.1.5 Parameters

For the evaluation a certain set of parameters was used. This section explains the creation and fine-tuning of the parameter set. Within the chapter selected reports and graphs will be highlighted. A full rundown of raw data and report sheets can be found in the Appendix.

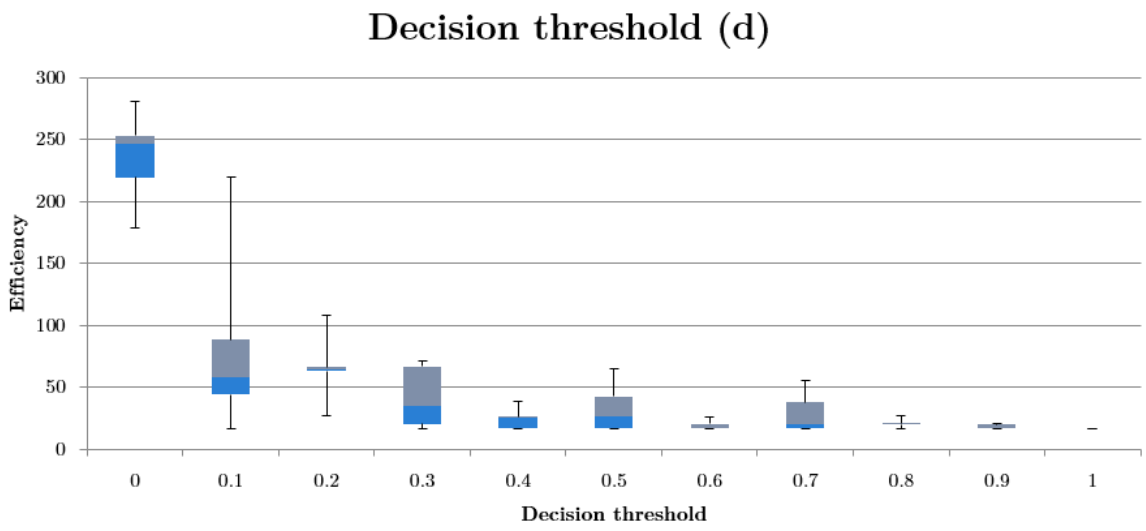


Figure 4.7: Decision threshold results

4.1.5.1 Simulation Runtime

One of the core decisions was the maximum time of three minutes per run. Due to the nature of this game, requiring multiple agents to reach a target to kill it, it's quite possible to end up in a deadlock state. Therefore a maximum timing was introduced, after which the run will be terminated. After initial experiments the time frame was set to *three minutes*. This time has proven to allow most of the swarms to show their potential, without extending the simulation too much with bad swarms. When introducing new simulation configurations or scenarios spot tests were performed, using a range of different parameters to evaluate the optimal time frame. These tests were conducted to evaluate if the swarms extend behavior which was cut short by the artificial time limit and to adjust the time in this case. For the more complex scenarios at the end of the evaluation the time limit was increased to ten minutes. This has proven helpful for configurations with a windup-time, like the reinforced bonding.

4.1.5.2 Decision Threshold

The decision threshold parameters controls which priority a task must possess to be considered a valuable aim by task acceptance module. A higher decision threshold means that agents will only take over high prioritized tasks, thus having a high probability of improving the success of the swarm. But a high threshold will also mask out possibly rewarding tasks with a lower threshold due to external influences. Noises generated by surrounding, which may disturb agents, can be filtered out by this threshold.

To test the decision threshold used in task acceptance (see: Section 3.4) a simple Arena setup was used: A single seeker faces five hide-agents. The hiding agents are placed in a forest-like arena, each with a randomly generated waypoint net. The hide agents have a different strength, starting at sending power of 1 with the first one and decreasing by 0.2

$(1, 0.8, 0.6, 0.4, 0.2)$. To kill a hiding agent off the seeker needs to target it for 3 consecutive seconds. If this time span passes the hider is removed. One run ends as soon as either all five hiding agents are removed or the total time passes 60 seconds. This time-frame was sampled beforehand and proved to be long enough for this simple scenario. Agents which do not finish a run after 60 seconds won't be able to finish it at all. Every time this was the case the agent was not able to generate a task to one or more signal sources due to its quality being too low. In the first iteration the time is measured, until all HAgents are deactivated, or the time limit has passed. Then the formula $e = \frac{deactivatedagents}{time} * 1000$ is used to calculate the effectiveness of different threshold levels[9]. In the second iteration a noise layer was added to the scenario, providing extra, random impulses for the agent. The noise will generate impulses with a strength between 0 and 1 at random positions inside the arena.

The test runs have shown that decision threshold provides no immanent effect or even benefit for the swarm. Due to the agents being unable to match a signal with its cause decision thresholds were not able to mask out noises efficiently. Higher thresholds (>0.4) lead to very bad results, as the agents often were not able to accept any task at all[10]. Even lower thresholds, while allowing to mask out weak noises, did not provide any benefit. It can be observed in Figure 4.7 that using no threshold at all was the result in every setting. With a setting of 0.1 for the threshold data already strays a lot, because the agent was often not able to generate tasks, while performing decently in the other cases. It is up to debate if a fine tuned threshold will convey a benefit in specific scenarios. For the course of this work the threshold has been defined to be 0 for all following test cases, to allow the focus on bonding mechanics.
In [Ken99] agents can function as filters in social communication networks, reducing the noise inside of the network. The decision threshold may be useful in such an scenario, when more complex mechanics are put into place. Interesting results could be achieved by evolving the decision threshold over time making the agent adaptive respond to environmental effects [LMAM04][Mor15]. Adapting thresholds can be used to regulate the amount of tasks circulating the swarm. Master agents, being the main communication nodes (see Section 4.1.2.2 and [Ken99]) can increase or decrease the threshold based on how *busy* the swarm is right now. This can allow the swarm to adapt towards an efficient configuration by altering its diversity [LMAM04].

4.1.5.3 Bonding Parameters

Agents per bond

The *AgentsPerBond parameter* is applied when initializing the bonding module and defines the separation within the swarm. When creating bonds the module adds agents to the bond until it contains a predefined number of agents.

[9] The scaling by 1000 was introduced solely to improve readability in the console application generating the results

[10] An agent with skills <0.4 will not be able to generate a task with priority >0.4

Seeking agents: *20*
Hiding agents: *32*
Runs per config: *40*
Kill count: *5*
Timeout: *5 minutes*
Swarm-type: *Random*
Bonding-type:
$\texttt{AB}_1$: *Random*
$\texttt{AB}_2$: *Skill-based*
Scenario: *S2 Arena Double*

Figure 4.8: Config $\texttt{AB}_1$ and $\texttt{AB}_2$

This value influences the amount of connections in general but especially the amount of bidirectional communication lines. These are expected to have influence on overall swarm performance. See Section 4.1.2.2 for an in-depth explanation of bidirectional connections and other swarm topology features in the simulation.
The scenario used for this test, S2 Arena Double is an empty arena with 32 hiding agents in a circle-like structure. A swarm of 20 seeking agents was spawned in the center of the area. The static parameters for the simulation ($\texttt{AB}_1$ & $\texttt{AB}_2$) can be seen in Figure 4.8. One full simulation circle of 40 runs was performed for each possible *AgentsPerBond* value n; with $n \in \{1...SeekingAgentCount\}$ and $SeekingAgentCount = 20$. From the sampled metrics the overall efficiency of each instance of n was calculated and then compared with each other. The first test $\texttt{AB}_1$ used *Random* swarm generation and bonding mechanics (See Section 4.1.3 for a description of the swarm modes and Section 3.5 for overview over different bonding mechanics).

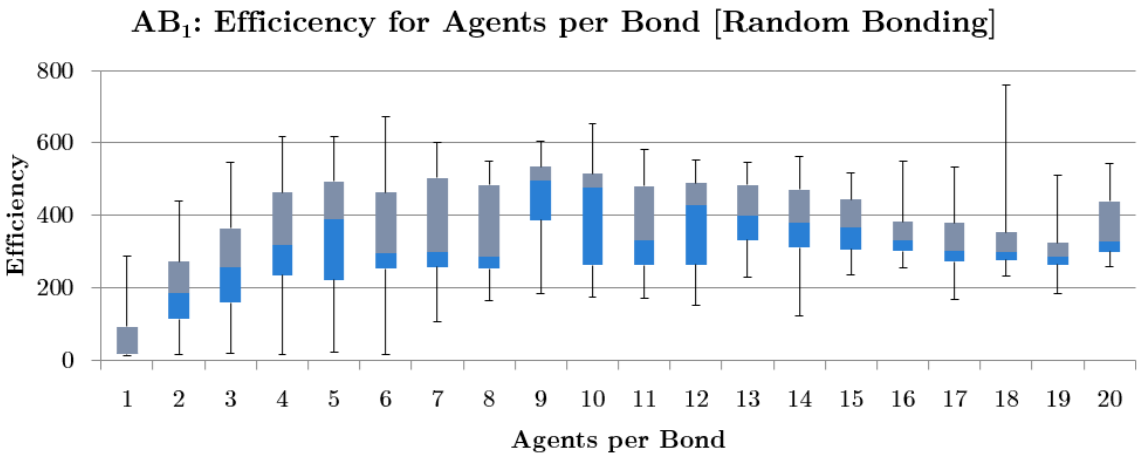


Figure 4.9: Efficiency of different Agent Per Bond values with random bonding

Figure 4.9 shows data and insights gathered from $\texttt{AB}_1$: For each set of runs the efficiency was calculated in a post process. The graphic shows a box plot of the generated values. The blue and gray areas show the lower and upper quartile, together they contain 50 % of all values. The two whiskers indicate how far off the median value the extreme values stray. It can be observed, that for only a single agent per bond values tend to be very low, with a few runaway values going up to around 300 efficiency. Agents tend to follow their own inputs and can only gather enough instances at a target by chance. As soon as agents are added to the bond the overall efficiency starts to rise until it stabilize at

the highest point at 9 and 10. This is because in the beginning bonds are not able to get enough agents invested and focused on a single target. By the time 5 agents are added into one bond the value reaches a local maximum, forming groups which are able tot instantly kill a target (See Figure 4.8: Killcount of 5). After plummeting a bit at 6 to 8 agents per bond, because bonds starts to snap agents away from other bonds, overloading targets with too many agents and therefore slowing the overall progress, efficiency reaches a global maximum at 9 and 10 agents. At this point bonds overlap in a way that allows to quickly switch between targets, by always presenting a high quality target, while simultaneously forming multiple groups with close to 5 agents going after different targets. The last property gets lost slowly when creating bigger bonds, as seen from 12 onwards, connecting too many agents with strong leaders, therefore forcing them to blob and form one single big group. The graph shows that the best efficiency can be achieved with values at 9 or 10, with a falloff of around 100 efficiency to the next values. Using the formula $e = \frac{deactivatedagents}{time} * 1000$ from Section 4.1.2.1 a difference for 100 would mean to catch $0.1 * time$ more agents, or 1 agent per 10 agents caught. Transferred to $\texttt{AB}_1$: A swarm of the efficient group performs 10 % better.

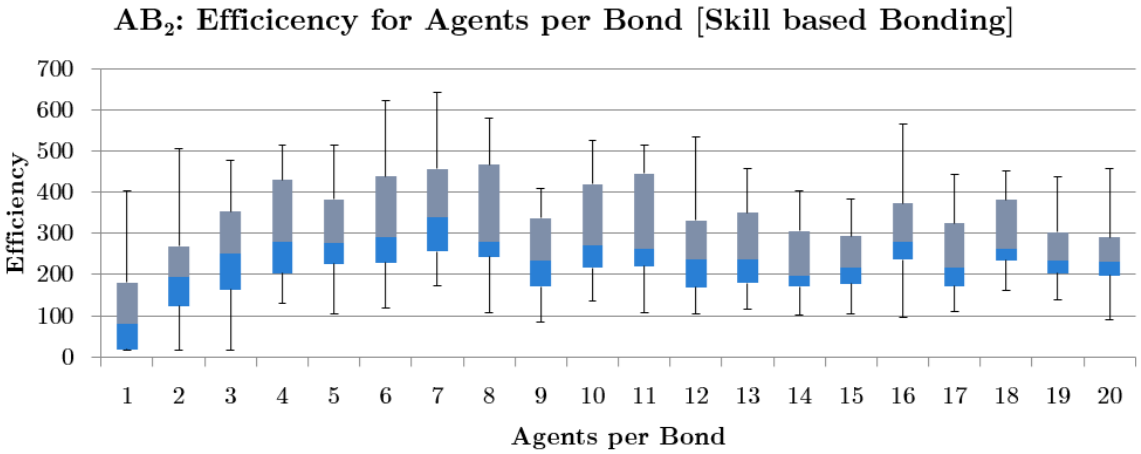


Figure 4.10: Efficiency of different Agent Per Bond values with skill based bonding

For the second experiment $\texttt{AB}_2$ the runs were repeated with the skill based bonding module. This time the agent tries to fill its bond with *AgentsPerBond* other agents, which are better in at least one skill. The resulting graph, shown in Figure 4.10 is similar to $\texttt{AB}_1$. It reaches it peak point at 6 and 7 agents per bond and starts decaying after this value is surpassed. This shift forward can be explained by the reduced availability of bonding partners. The mechanics which provoked the loss of efficiency in $\texttt{AB}_1$ takes effect earlier here. While bonds overall have a higher maximum performance, they are less diverse and promote a stronger leader selection. This aligns with Marcolinos findings in [MJT13], where diverse groups outperformed strictly optimized groups. Figure 4.11 shows a direct comparison of the median values, highlighting named effect.

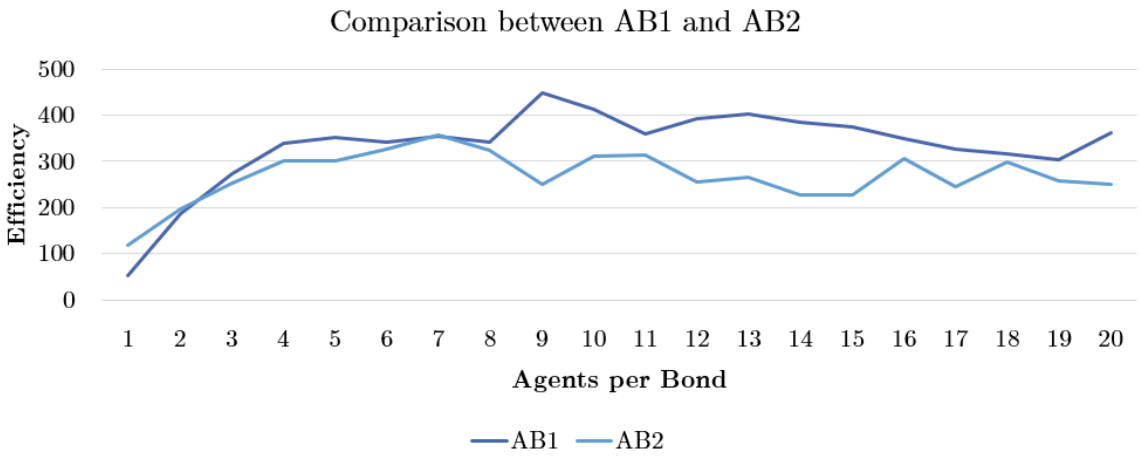


Figure 4.11: Direct comparison of the median values of $\texttt{AB}_1$ and $\texttt{AB}_2$

Reinforced Bonding

When using the reinforced bonding module two parameters are responsible for the configuration of this module: **Bond Count** (BC) and **Bond Reinforcement Value** (BR). After generating the swarm each agent is assigned BC different bonds from which it can receive information. Each bond is assigned an internal *trust* value, set to 1 at the beginning. Trust acts as an indicator about the quality of the provided information. Whenever a target is removed agents which are involved evaluate all of their bonds. When a bond had suggested the recently removed target its trust value is increased with a function, scaled by the reinforcement value (BR). This module is tested against a byzantine swarm, founded on the fact that in a normal environment the existence of multiple bonds will just decrease the spread in values. Results would be very similar to results from Section 4.1.5.3 and data will show an overall higher and more even curve. One can think of this as a normal bonding strategy, where only the best of the BC bonds is considered. In the byzantine case however, where agents need to filter out a high amount of 'foul' agents, access to multiple bonds can provide a lasting benefit.
As stated before, the value of the **Bond Reinforcement Value** determines how much reinforcements are required to clearly differentiate one bond from another. A very high value will quickly promote bonds which are able to score kills. A low value provides a smooth increase of values, allowing for a more controlled filtering. Fast convergence (high value) is beneficial, when the chance for pure bonds is quite high e.g. with low corruption rate, or a very high bond count with a medium corruption rate.
The formulas for in- and decreasing trust values are:

$$increasedTrust(trust, reinfValue) = trust + \frac{\sqrt{trust+2^{reinfValue}}}{25} \text{ and}$$

$$decreasedTrust(trust, reinfValue) = trust + \frac{\sqrt{trust+reinfValue}}{25}.$$

A deeper analysis of the formulas for different reinforcement values and several steps

can be seen in Figure A.6 for the evolution of bonds for different BR and in Figure A.7 for the demoting counterpart. The *increasing function* provides stronger changes than the *decreasing function*. This is to offset a false-positive penalty, when a healthy bond focused on a different target. Also the decreasing value is capped at zero, so bonds can technically be *muted* but cannot get negative trust values[11]. The effect of this parameter is more prominent in the reward function, leading to a quicker promotion of successful bonds and a leveled degeneration of failing tasks.
Tests were run in scenario S2-Arena Double with the settings from Figure 4.12. In the center of the arena and 35 seeking agents were spawned. Then 35% of the agents were marked to go rogue. In this case 13 of the 35 agents were turned into byzantine agents. The agents were generating falsified tasks every three seconds, the same time agents need to focus an individual target to remove it from the simulation. A bond which distributes a lot of this falsified tasks will be called an *unhealthy* or *bad* bond, while a bond free of byzantine agents will be a healthy bond. Bond health quickly degenerates when adding byzantine agents. See Figure A.5 for the chances to create a complete healthy bond, depending on agents per bond (1-14) and number of rogue agents (1-18). For this simulation a value of 35% provided a reasonable chance to have at least some healthy bonds. This consideration is important for this evaluation, as the effects of reinforced bonding will be most explicit when there is a difference in bond quality.

Seeking agents: *35*
Hiding agents: *32*
Runs per config: *30*
Timeout: *15 minutes*
Swarm-type: *Random*
Bonding-type:
Random Reinforced
Agents per Bond: *5*
Bond Count: *1-31 (steps of 5)*
Reinf. Value:
0.5-2.5 (steps of 0.5)

Figure 4.12: Config: Reinforced Bonding

In the simulation the number of accessible bonds per agent started at 1 bond each and then increased their bond count in steps of 5 until it reached its maximum value of 31 bonds per agent. For each configuration of *bond count* the *reinforcement values* from 0.5 to 2.5, with a step size of 0.5 was tested.
Results of the experiment are presented in Figure 4.13. For the evaluation the runs were post processed and the average efficiency was calculated, while 5% runaway values were removed. The figure shows two different views of the average efficiency and a box-plot of all values. The graph A, showing the bond count data plotted against different reinforcement values, highlights the difference between using normal bonding (BC of 1) and the reinforced bonding strategies. Another notable characteristic is the fall-of at a BR of 2. This is because in the simulation unhealthy bonds often were able to secure a superior position by getting an early kill, basically locking agents with their false information. A smaller value and more even curve can counter this problem by only changing trust values in very small steps.

[11] This clamping is represented in the code and not reflected in Figure A.7

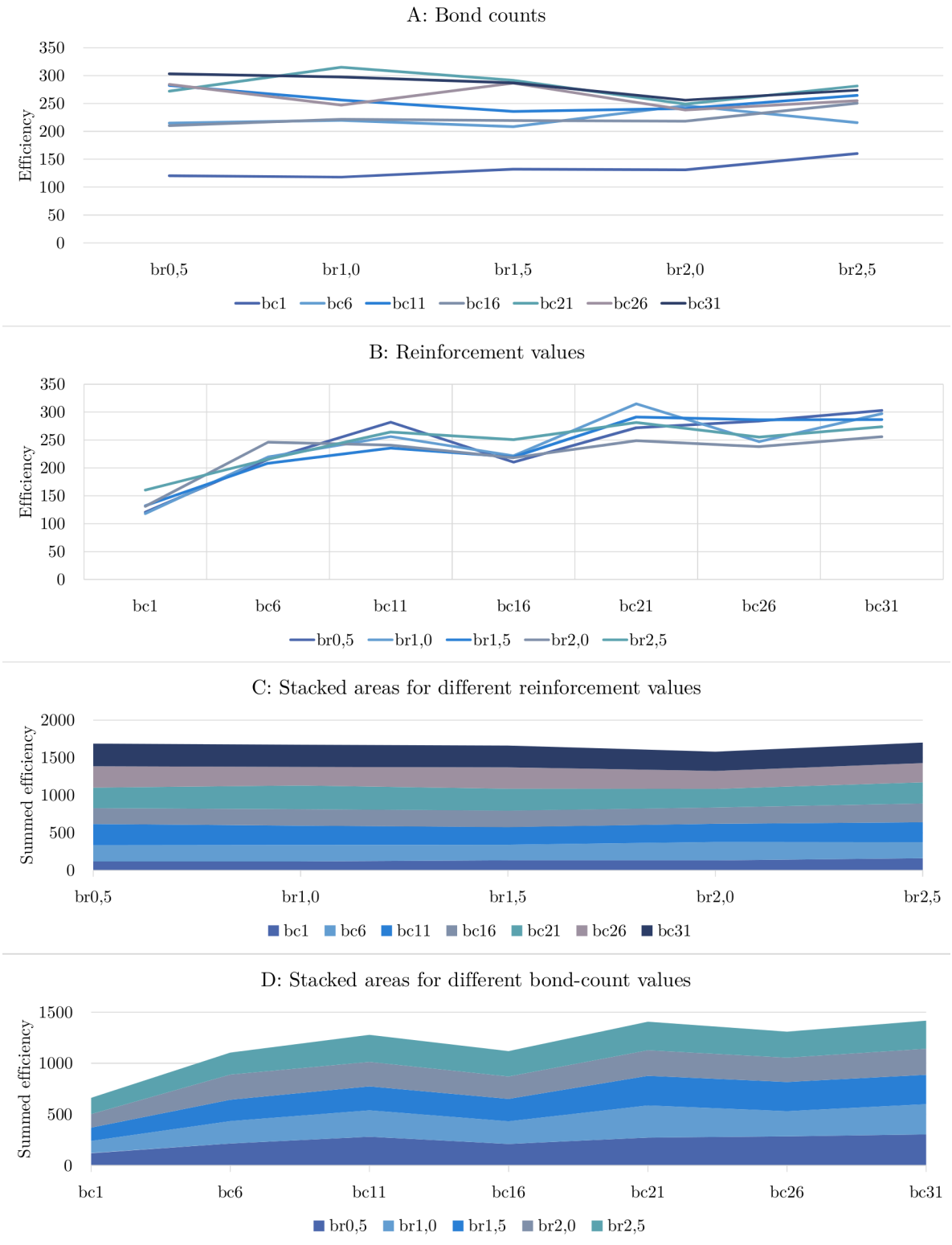


Figure 4.13: Evaluation reinforced bonding 1.BC: Bond count, BR: Bonding reinforce value

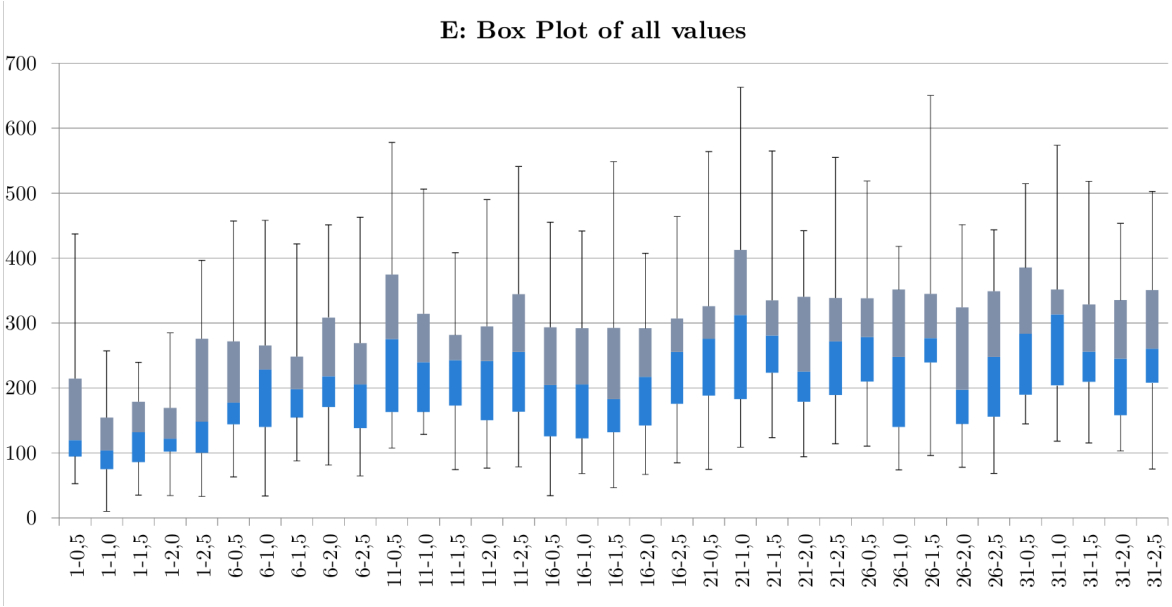


Figure 4.14: Evaluation reinforced bonding 1. BC: Bond count, BR: Bonding reinforce value

The second graph (B) shows the different reinforcement values plotted against the bond count configurations. The overall observation is that efficiency correlates with the number of bonds. At 16 bonds a small falloff occurs, which can be explained by the number of byzantine agents. In total efficiency is rising with bond count, as agents do have a higher chance to have access to a (more) healthy bond. On the other hand higher bond count increases also the number of connected agents and therefore the chance to be connected to a byzantine agent. In the beginning this chances balance out, up to around 16 bonds the negative effect of getting connected to another rogue agents is slightly more severe. This is based on the fact, that byzantine agent generate tasks at the same frequency as agents can kill targets. At 16 bonds the chances are very high that an agent is somehow connected to all of the 13 rogue agents, being exposed to every falsified task. From this point on additional bonds can only increase the quality of available information, by either incorporating a strong healthy agent for example. Graph C and D show the stacked total areas of to graphs above. Reinforcement values (with the exception of 1) tend to balance out, and which value is used best depends on many other parameters.
This experiment again highlights the problem already encountered in Section 4.1.5.3: It requires a lot of manual testing and adjustment to find the 'right' difficulty for a given experiment. If the scenario is to easy to solve results will not differ enough in efficiency to cover runaway values and still be distinguishable. Another working application of the bonding reinforcement will be shown in the complex scenario ByzantineSwarm.

4.1.6 Further experiments

With all the parameters evaluated and classified further experiments were conducted. In this runs more complex scenarios were tested to see how well the swarm performs in different complex situations. The scenarios itself are multi-layered in terms of mechanics and require the swarm to combine different core skills to solve the tasks successfully. A list of the scenarios used here can be found in Section A.1. The following scenarios, except the *Byzantine Swarm*, were performed by a seeking swarm of 20. If not specified otherwise, 32 hiding agents were placed in the scene. Parameters were based upon findings from the previous chapters, setting agent per bond to 9 and using the reinforced bonding module. Reinforcement parameters were 1 for the reinforcement value and 21 for the number of bonds per individual. The first three scenarios where run in a unique scene each, the topology and features can be seen in Figure A.4.

Complex 1: FourSeasons In this scenario the arena is divided into four quarters (See C2). Three quarters block out one type of information each. This scenario calls for the swarm switching leader often, as previous *best candidates* can no longer provide insightful tasks. This will strain the bonding system and evaluate the benefit of having a decentralized, diverse swarm [MJT13]. While traversing the arena seeking agents will be cut from their signals and need to rely on other senses or bonded agents to guide them. The result of this run can be seen in Figure 4.15.

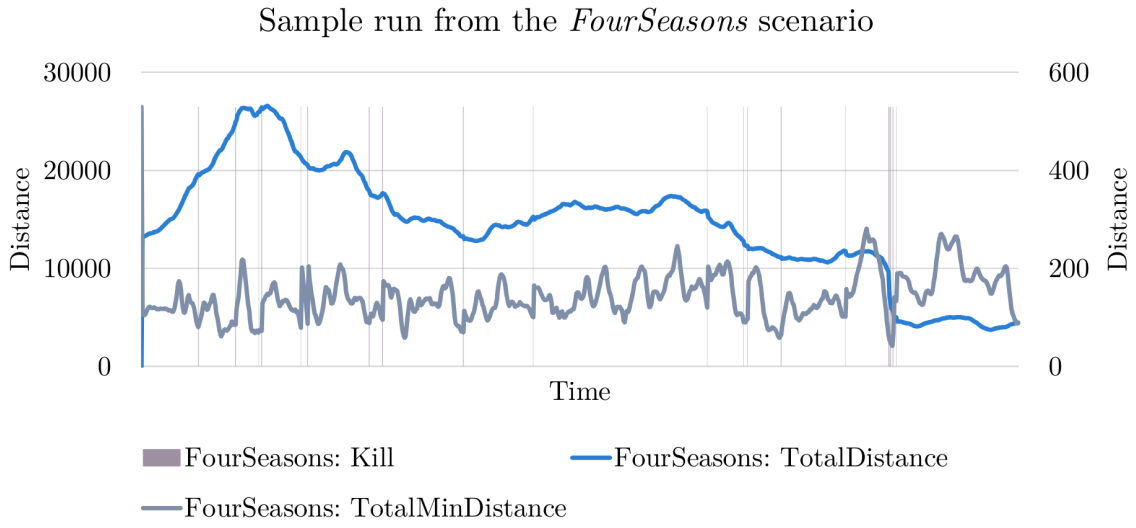


Figure 4.15: Sample run of the *FourSeasons* scenario.

Looking at the evaluation and the overall results the swarm performs well in this environment. Although targets are split over all quarters and move around, basically changing their information signature, there is no major decline in overall swarm performance. However after targets become more sparse the efficiency starts to drop. This is because of the locality of some information. To follow a trail of smell agents need to locate it first.

To orientate oneself by sound agents need to be in the range of its emitter. This explains the rise of the total distance in the center of the graph. While single agents were able to track targets, the topology of this swarm did not allow them to communicate their findings efficiently. Shortly before the end of the curve both distance functions rapidly decline. This is an indication that either the targets went into a region where they are easier to track, or that a well connected agent (see *Hub* in Section 4.1.2.2) reestablished tracking. Based on the fast decline it is probable that the latter is the case, that an agent found a target and was able to direct the swarm towards it. By getting into proximity of the target other agents were enabled to take over tracking by odor or sound, resulting in a series of quick kills.

Complex 2: Forest The *Forest* scenario has a multitude of obstacles in the arena, blocking view and hindering navigation. A topology map can be seen in C 1. The difficulty in this scenario is of a similar kind as seen in the FourSeason scenario. This time, instead of relying on different senses agents also need to cope with a hindered navigation ability. Beside this general hindrance the scenario devaluates the seeing ability. This ability has proven to often be superior over the other senses, but does not allow a constant tracking in this scenario. Figure 4.16 presents the progress of a sample

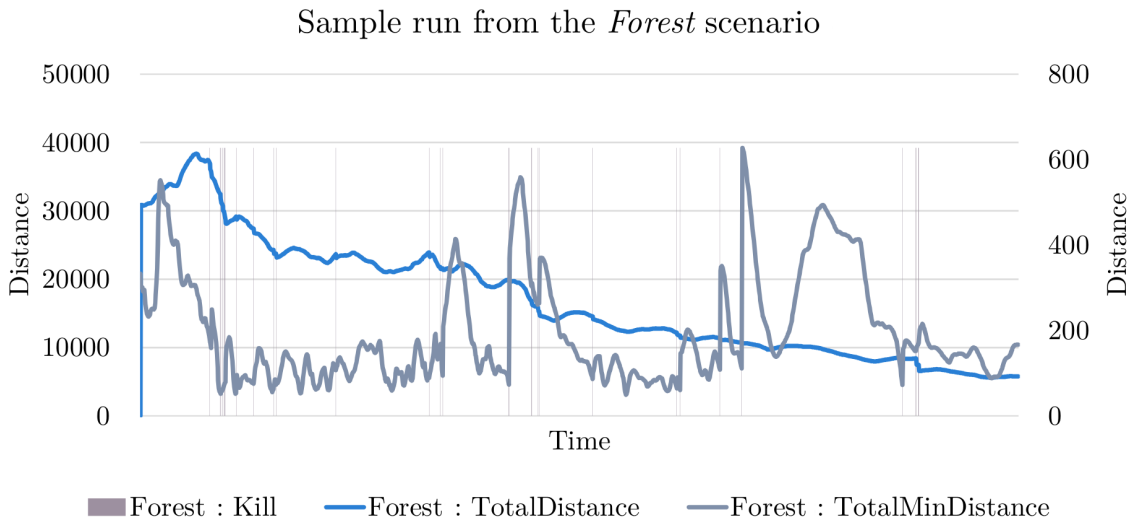


Figure 4.16: Sample run of the *Forest* scenario.

run. Similar to the result of Section 4.1.6 the distance tend to jump up at some points. Unlike before this jumps occur mostly after a kill has been executed. This leads to the interpretation that the jumps are not necessarily related to bad swarm performance, but rather to bad placement of the target. The swarm is moved away from the main group of targets to score a single kill. This is reinforced by the total distance function, which increases slightly before a kill, showing the movement of the swarm into a generally less favorable position. Only in the last 25 % of the evaluation the swarm seems to loose tracking, as the distance increases in a similar fashion to Section 4.1.6.

Complex 3: Legion While being simple element-wise, *Legion* is the complex scenario with the worst overall performance (As seen in Figure 4.20). The scenario features the huge amount of 140 targets, distributed evenly over the arena. Again a topological map can be found in Figure A.4. The swarm spawn in the middle of the targets and while kills happen frequently the overall fitness, based on distance to targets, degenerates.

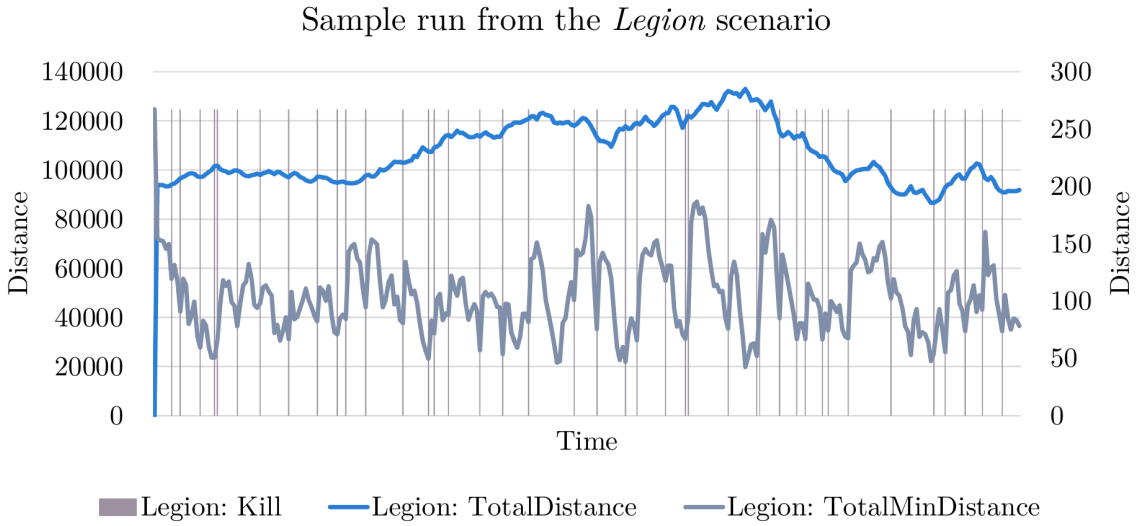


Figure 4.17: Sample run of the *Legion* scenario.

The extend of the fluctuation withing the minimal distance function gives away that a lot of backtracking is happening. The problem uncovered by this scenario is the missing spatial information of tasks. Agents do not evaluate tasks by their proximity, but only by the strength of the received signal. In this particular case it leads to the swarm working through the field in a ping-pong like matter, wasting time and resources by moving back and forth.

Complex 4: Byzantine Swarm As mentioned in Section 4.1.3 byzantine agents will create tasks based on false / non-existent information. This Scenario is used for an in-depth analysis of the reinforced bonding module. It extends on the evaluations of the parameters in Section 4.1.5.3. It also uses to the Arena Double scenario, 35 seeking agents and the parameters from the *'Reinforced Bond Count'*-experiment. During this simulation, the percentage of byzantine agents was increased to 51 %. Which leads to 18 of the 35 agents marked as byzantine agents. This means that appropriately every 30 ms a misleading information is generated and spread as a task among the swarm. It generates an environment, harsh enough for the swarm to be reliant on this mechanic to filter out the remaining healthy bonds. The number of bonds per agent also started at 1 and increased up to 10 bonds. While the results in Figure 4.18 are still volatile a general increase of fitness can be observed. It is to be noted that this experiment simulates a harsh failure in the swarm, more than 50 % of the agents provide false information,

corrupting bonds with a high probability (0,999999707 % chance of at least 1 rogue agent per bond)[12].

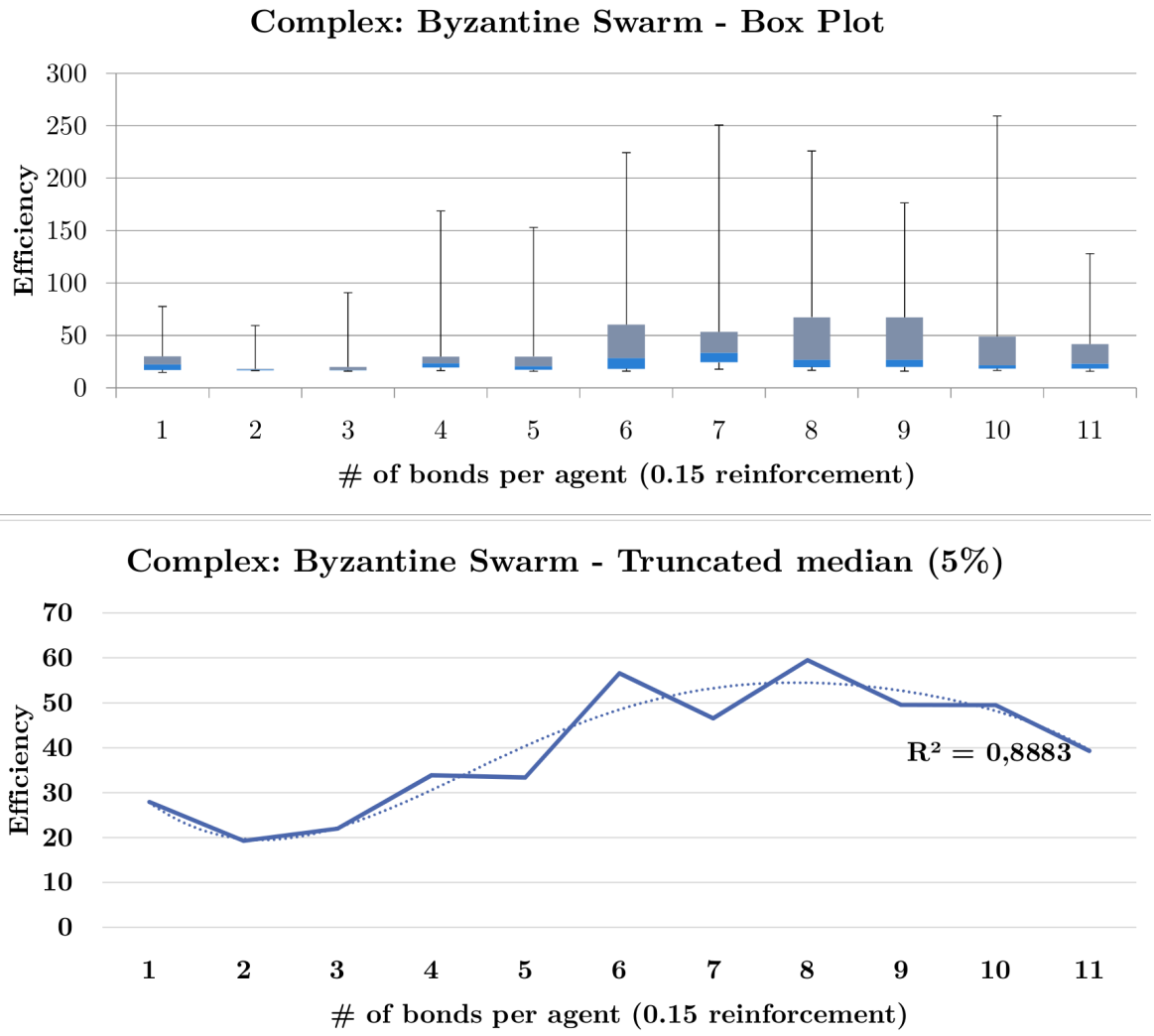


Figure 4.18: Evaluation of reinforced bonding with a high failure rate (51 %)

The high runaway values occur when, by chance, bonds without any byzantine agent are created, or byzantine agents happen to generate weak tasks. The latter being more likely, working with a random swarm. After nine bonds per agent the values start to decrease again. After this point the chance that an agent is connected to a byzantine agent, which happens to create a strong task is too high. Based on this, healthy bonds won't be able to score kills. Without kills agents are not able to rank bonds accordingly, therefore they stay responsive to byzantine agents.
It can be stated that reinforced bonding has a positive effect on the swarm and can help to account for failures in the swarm infrastructure, as long as the scenario and swarm

[12]See Figure A.5 for an evaluation of chances for a healthy bond.

configuration is within a certain corridor. The reinforced mechanics have the biggest impact in a very harsh environment and using the current formulas quickly move the swarm's focus towards healthier bonds. With adjustment to the formulas, like harsher degeneration of health for false information, the system can also be adopted to quickly mute inefficient bonds. With the (re)evaluation of bonds being executed at the kill of a target these can lead to a complete lock-down of agents, therefore a adoption to the mechanics is advised too. Bonding can ease the effect of the three kind of failures named in [DZZS04] '*Partial Robot Malfunctions*' and even '*Robot Death*' [DZZS04], as long as the communication network between the agents is still intact.

Complex 5: Maze The *Maze* is another very specific scenario: A maze is constructed in the arena, guarding the four targets in the center. Two groups of seeking agents are placed in the maze, one group with a walkable connection to the targets (upper left) and a single agent close to the targets, but without a path to them. While the first group could reach the targets they can not see them and are to far outside to smell or hear them. However the single agent is well withing the radius of the sound signal and therefore able to direct the other group through bonding. The Figure 4.19 shows how

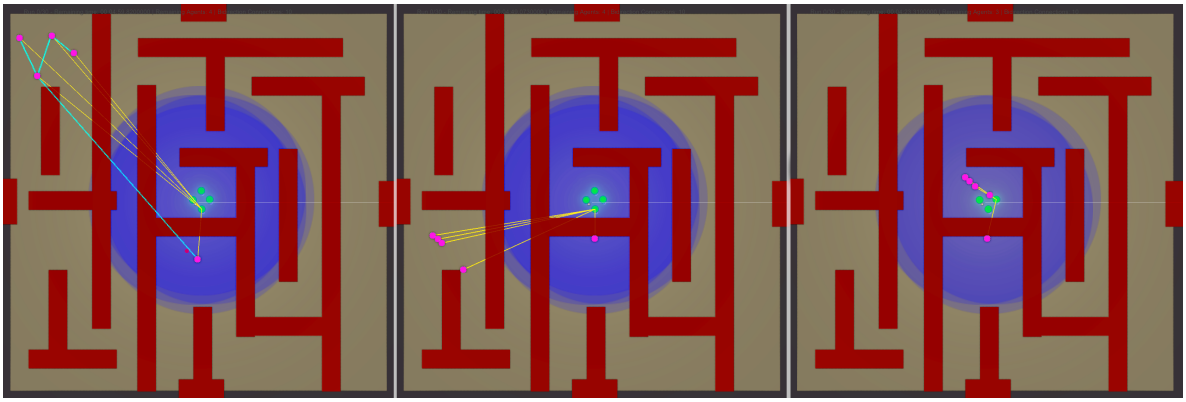


Figure 4.19: Maze scenario: Snapshots at three different timings.

a run in the maze works with agents per bond set to one. Pink dots are the seeking agents, green dots are targets. Yellow lines denote the collective target of the bond (Which means the target the agents decided to follow, not necessary the target each agent generated. Aqua blue lines show bonding connections. The first frame shows the single agent inside the sound signal, connected to a single agent withing the other group. The other agents connect themselves to this agent and withing each other. The middle frame shows how the group of four free agents is able to navigate the labyrinth, guided by the single agent, which still receives auditory information. The last frame pictures the arrival of the free group, which is able to remove the targets, resulting in a successful run for the seeking swarm.

Summary

The complex scenarios have highlighted more specialized situations and tested the swarm inside them. The swarm was able to perform in each situation, but additional limitations and open problems were identified. Especially, looking at the first three complex scenarios allows to reason some properties of the implemented system. Figure 4.20 shows the values of *Forest*, *FourSeasons* and *Legion* in direct comparison. Forrest performs best, only taking small hits in the overall performance. *FourSeasons* shows a slightly increased deficit, which can be related to the loss of tracking described in Section 4.1.6. The *Legion* scenario, which does not feature any constricting elements in its scene structure, achieves the lowest rating. This emphasizes the problem with agents not being distance aware.
The bonding scenario shows that even under extreme conditions the swarm is able to produce reasonable results. The mechanic of the bonding system allow for stabilization and increase the robustness of a swarm.
Lastly the *Maze* emits behavior which are especially interesting for game development. The implemented flow of information allows different agents to combine their information and their search space (In this case the region within the maze they can reach) and to complete the task by merging this information. This can be used in games to form emergent, yet somewhat controlled behaviors for agents, allowing them to react at player interaction.

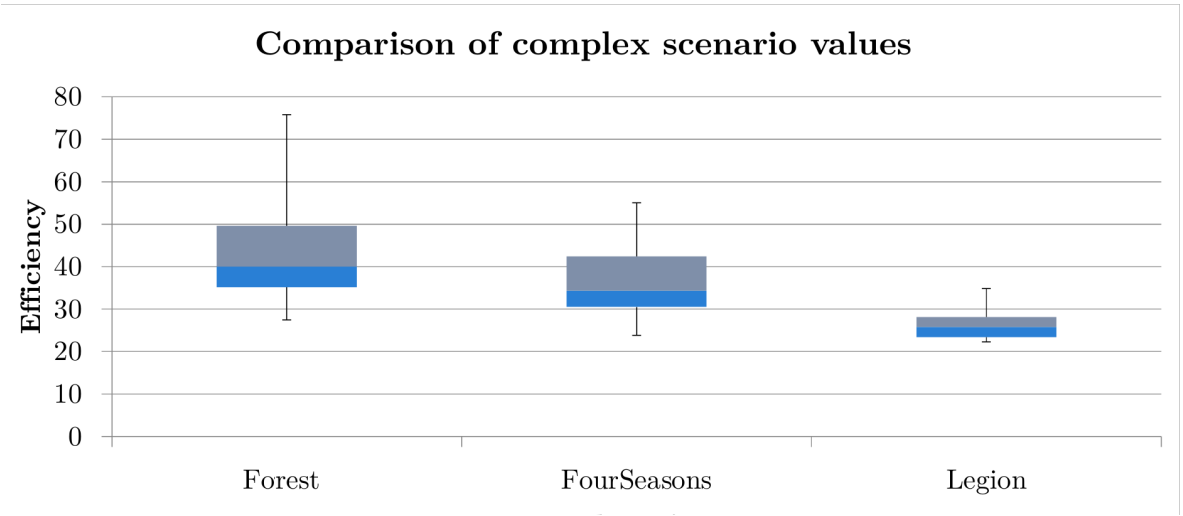


Figure 4.20: Comparison of complex scenario 1,2 and 3

4.2 Results

This chapter sums up the experiments done in Section 4.1 and clusters them.
The first system, implementing a simple task acceptance mechanics in agents, showed no benefit in performance. It rather downgrades the swarm efficiency. The results of test runs with active decision threshold can be seen in Figure 4.7. The reason for the

systems failure is missing information. An agent can not discriminate between a valid task and noise/false input. Therefore filtering based on this system can not provide better results. Giving the agent access to the source of the signal (at least with a certain probability) could solve this problem. [KNZ10] uses global storage to circumvent this kind of problem, agents can use such a storage system to counter-check other task and evaluate them. Similar to reinforced bonding, tasks can have a trust value, which is increased when other agents *confirm* the validity of the task.
The second intended use of the decision threshold was focusing tasks, to reduce the effect of sidetracking. While this property may still hold, it relies heavily on the right design of scenario and balance of parameters. With the abolition of task acceptances main feature the decision was made to not evaluate this property any further. Instead part of its characteristics where moved inside the agents logic (see Figure 3.3).
With the omission of the decision threshold parameter this section focuses on two major components: The influence of the swarm topology and and the influence of bonding values.

4.2.1 Influence of bonding parameters

The parameters of the bonding module shape the behavior of the swarm and are magisterial for its success. In Section 4.1.5.3 three major parameters were exemplified and tested. The *Agent count per bond* defines the separation of the swarm. It influences every other parameter in a unyielding way. In multi-agent task systems a successful team formation is the key for efficient swarm behavior [MJT13]. Bonding provides agents with a simple and decentralized way of interoperability [Gd05]. The agent count parameter describes the number of agents responsible of providing information to a single agent. It directly influence the interconnectedness of the swarm and the *informational-reach* of each agent. The experiment in Section 4.1.5.3 shows that efficiency correlates with connectivity within the swarm. Up to a certain point the larger zone of interest is beneficial for agents. Because of the nature of the task chosen in this work, *catching targets with a certain number of seeking agents*, a point exist from which more agents do not provide any benefit. Looking at the two extremes of this parameter on the one side single minded agents exist, each following only its own inputs, resulting in a potential large spread over the search space[13]. On the other side of the spectrum is a fully inter-meshed swarm, always focusing on a single, most-promising task. Experiments showed a good guiding value with *Agent count per bond* set to approximately two times the required number of agents to score a kill. Bonds with this number of agents provide enough adhesion to form subgroups capable of removing targets, while simultaneously providing a big enough spread of targets within a single bond to reduce downtime and backtracking.
The tests AB_1 and AB_2 in Section 4.1.5.3 showed that a diverse group within a bond, for example created by random bonding surpasses a very skill focused approach. Especially when no global instance allocates tasks for groups, but instances decide for themselves which goal to pursue [MJT13]. This may prove false for special situations. Using the Three Competent Swarm (See *Single Competence* in Section 4.1.3), where only

[13] This will become even more prevailing in more complex sense profiles. See Section 5.1

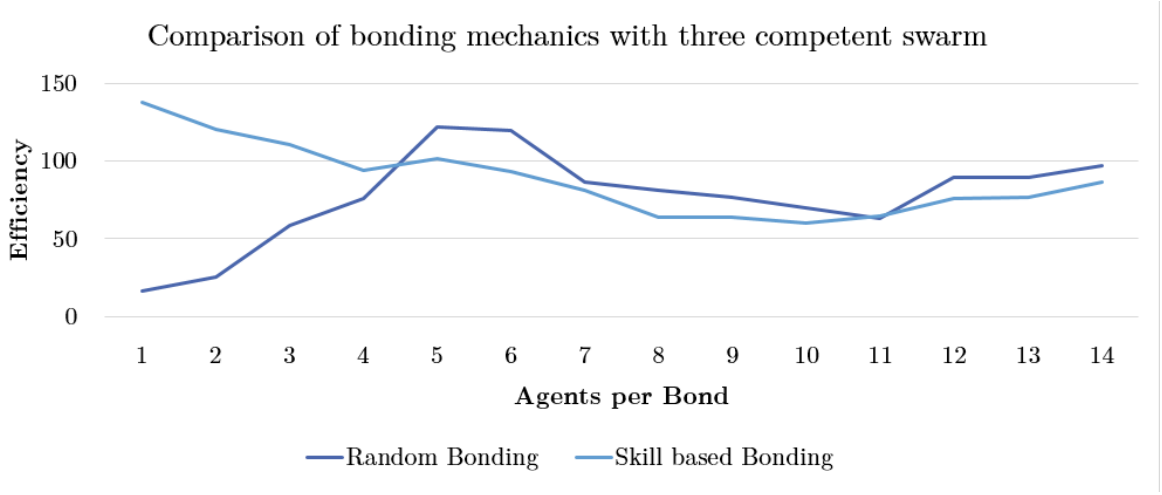


Figure 4.21: Comparison of bonding mechanics using *Three Competent Swarm*

three agents have a single available skill, the skill based bonding outperforms random bonding early on, because it help to prevent useless bonds where agents with no skill at all connect to each other (Figure 4.21).

A problem which surfaced during testing is the granularity of the *AgentsPerBond* parameter. When tasks are to easy results tend to be very close in terms of efficiency and there is no real distinction to be made. This is because the step from *'Not enough connections to coordinate agents'* to *'Too much connections and just one blob'* was within a very small range. This was circumvented by running initial small scale tests to determine an appropriate scope. So while experiments were able to narrow down useful categories for this setting, additional methods have to be deployed to reduce the required manual adjustment of this parameter. Because bonding is heavily influenced by this single value alone, results are volatile and tend to change drastically with scenario and other influence factors. Section 4.2.2 shows some of the cascading bearings of this value and Section 5.3 points out possible ways to pin down the parameter.
After evaluating the agent per bond parameter another layer was added to the bonding module. Instead of receiving information from just a single bond agents were provide a multitude of bonds, each assigned an internal trust value (on a per bond base). This value is adjusted every time the swarm is able to score a kill. Internally each agent, which participated in the kill, will evaluate all his bonds, increasing the trust for all bonds which suggested this successful target, and demote the trust of the other bonds. The formula of trust adjustment can be seen in Section 4.1.5.3, with a visual representation in Figure A.6 and Figure A.7. Demoting a bond is slower than reinforcing trust and can be offset quickly by successful targets. After a certain run-time agents are able to sort out *bad* bonds and highlight effective connections. As seen in Figure 3.1 line 4ff the trust value is factored into the bonding task acceptance. By this *healthy* bonds with a high trust value are able to compete with potential corrupted values. Section 4.1.5.3 pointed out these mechanics are highly dependent on different factors: Overall difficulty, swarm composition and the connection graph between agents. In Section 4.1.6 a scenario

containing a large scale swarm failure is constructed. Over half of the agents contain 'malfunctions' and provide wrong input signals. The experiment showed that with a slow enough reinforcement value and enough bonds to choose from agents are able to stabilize and perform.
The method used to adapt a bonds trust value fulfills its intention, but have to carefully tweaked by hand and can yield unwanted results if parametrized outside of the tested variable scope (see Section 4.1.5.3 and Figure 4.13). Right now the method exhibits some potentially unwanted characteristics. Agents can get locked up in a single bond. When a bond is able to score some kills early on, but is in total a rather unhealthy bond it can bind agents to itself. When the trust value of the unhealthy bond rises far enough, respectively the other bonds are demoted far enough, the agents won't be able to tag another kill. Without a kill there is no reevaluation of the bonds, therefore the situation becomes persistent. To prevent this from happening some measures can be taken: Just like in [KNZ10] a global index can be used to rate the bonds. Instead of only reevaluating own bonds when scoring a kill, all bonds will get evaluated by a central instance whenever a target is removed. A more complex but also powerful approach is the introduction of dynamic bonds. Right now agents stick with the bonds they receive at the beginning. There is no direct way for an agent to mute another agent. It can only demote the bond containing the agent in question. Allowing bonds to change their own bonds will enable them to be reconfigured to evolve their potential. Similar to [Mor15]'s approach of group formation with reconfigurable agents, bonds can connect ad-hoc to close agents and permute their properties. This can allow for complex behaviors, like morphing the overall topology of the bond into another model with better properties [Ken99]. Changing the *agent per bond count* should yield significant changes in swarm behavior, when evaluated dynamically. One way of achieving a dynamic adoption is shown in [Men04], where good agents are allowed to reduce the number of connected agents, while poorly performing agents could increase theirs.
When preserving the current bonding system changes can also be made on the adjustment functions. A system with dynamic functions, based on factors like passed simulation time, total stability of the trust values across the swarm and division of bonds, can adopt to different situations. Depending on the goal of the functions, a possible scenario can undertake only very small changes in the beginning and increase them while the simulation runs. Giving this kind of meta information can specifically target problems with the current functions. If a switch of the most predominant signal type happens, like smoke appearing, making visual strength worthless, the system can use meta information to speed up convergence. In this case it would look for bonds with a slightly lowered trust value, which got a lot of positive reinforcement lately. A combo like system, which increases the gain for every correct consecutive target will allow this bonds to rise quickly and take the place of the degenerated heavy-on-visual bonds. This would defy the concept of a simple local system and require an existing understanding of the coming task, but may result in a more optimized swarm.

4.2.2 Influence of topology

Earlier topology features were defined and their influence was briefly discussed. Evaluation of these feature became increasingly difficult with a raise of complexity. During test runs the configuration of the swarms was always exported as meta information, allowing to link it back later onto the result. Yet reinforced bonding proved to be a problem, as it generates way more connection as it actually uses. The post processing pipeline was modified to connect this meta information with efficiency results generated by test runs. By assorting this data by build number, scenario used and other parameters clusters were generated, containing information about efficiency and bonding from similar runs. These clusters were evaluated by calculating the *Pearson- and Spearman-correlation*[Cho10], to find relationships between internal values. In total around 10000 runs were sampled to generate the correlation data. The result of this evaluation can be found in Figure A.8. The data set shows a strong correlation (> 0.8) between the *number of hubs* and our *largest bidirectional sending network*, as well as between the *number of bidirectional connections* and the *hub count*. A solid correlation (> 0.75) can be observed between the *bidirectional connection count* and the *size of the largest bidirectional sending network*. While these correlations are expected, they emphasize the influence of hubs to the overall communication power. Between the *slave count* and the *average sending net*, as well as between the *island count* and the *largest sending net*, a strong (> 0.8) anti-correlation can be observed. All of this values are explainable by the definitions of these features in Section 4.1.2.2.
The focus of this evaluation was to gather insights about the correlation between swarm efficiency and topology features. Looking at the first column, it can be seen that some feature, while not dominating the efficiency completely, influence the outcome of a simulation. Namely the amount of *bidirectional connections*, the *hub count* and the *reach of the bidirectional network* provide a moderate (0.5) benefit for efficiency, while *island* and *slave count* have a weak (-0.2) negative influence.
Looking back at Section 4.1.5.3 the experiment showed that from a certain point additional agents per bond do not provide any benefits, instead leading to clumping and a slight degeneration in efficiency. To evaluate the influence of the *agent per bond* parameter further the post processing pipeline was extended again to incorporate this meta information. Due to the fact that the parameter was added to the meta information at a later build version some clusters were unfit to be included in this evaluation. After merging the parameter into the data set around 7500 unique runs remained. The evaluation of correlation between agent per bond count and efficiency can be found in Figure A.9. Instead of calculating the overall correlation, clusters were grouped by their *agent per bond* parameter, providing more granular results. The table shows the correlation for different ranges withing the data set. The first entry denotes, that when looking at values with an *agent per bond* of 1 and 2 yields a correlation value of 0.07. The next row then describes ranges of length 3 (e.g. first entry calculates correlation of the efficiency and agents per bond count with the values 1,2 and 3). Two findings stand out: looking at the lowest row, where correlations for a spectrum of length 14 are presented the values appear all to be negative and there is a decent hit in values,

which is migrating through the table. The runs used in the data set used different kill counts, and the extreme values (high as well as low) tend to be multiples of the common used kill values. Therefore the data was normalized with the kill count, by generating a coverage parameter as $coverage(agentsPerBond, KillCount) = \frac{agentsPerBond}{KillCount}$. This allows the value to be normalized between different runs and express how much 'surplus' over the required values was given. A coverage of 2 would mean, that there are twice as much agents in a single bond as are required to kill one target.
Figure A.10 visualizes the aforementioned streaks and correctly relate them to the kill count. The first negative correlation appears when the system overshoots the simple coverage of 1, which means having precisely enough agents in a bond to kill the target. Especially when looking at rows in the middle of the graph, with section length between 4 and 7 the changes between positive and negative correlation are well visible.

5. Conclusion

5.1 Conclusion

This thesis conceptualizes mechanics to control and coordinate a swarm of heterogeneous agents. Agents have to navigate a dynamic environment, filled with targets. They need to kill all targets by moving a certain number of agents close to them. This work has explored effects of different task acceptance pattern and the influence of team compositions and coordination by bonding mechanics. While threshold based task acceptance had proven to have no positive effect towards swarm performance, bonding resulted in interesting behavior changes. With simple, local and decentralized mechanics agents are able to distribute and evaluate information. Bonds can sustain a swarm and keep it operating even when larger parts of the swarm malfunction. How does this finding translate to the questions posed in Section 1.2?

Section 4.1.5.3 has shown that bonding mechanics can preserve a swarm's activity level, even when a larger number of agents fail. In this test case failure meant that agents did not only grow silent, they produced false information, actively disrupting the swarms operation. This should cover the three kinds of failures from [DZZS04]: Robot malfunction and communication failure and even robot death only inhibit swarm performance, as the remaining agents need to cover for more tasks. It can be assumed that this mechanics improves swarm performance with reduced sensor set, as well as swarm activity in disastrous situations, where drone failure may be immanent.

Specific tests need to be run for such real world scenarios, because optimal bonding parameters hugely depend on the environment, number of senses and target robustness. Experiments have shown that a prior information or extensive testing is required to find parameters which lead to maximal performance. Depending on the swarm size and desired application, bonding mechanics can be switched to a deterministic approach, which allows for a greater control over the underlying topology, requiring a increased planning effort beforehand.

Application in real world situations after catastrophes longs for more considerations. In

the most cases search spaces are huge and the current system produces a decent amount of backtracking for agents. While this does not significantly decrease agents performance in this work because of generally confined areas, it will prove to be a problem in large open spaces. This has been highlighted in the *Legion* scenario, which put a focus on the deficits created by backtracking. To counter this problem the bonding system needs be location aware and to consider distance between agents and targets as a cost to factor in.

To consider the application in games these work can be extended easily to provide a robust artificial simulation of adversaries. Especially the momentary popular survival games[1] can hugely benefit from mechanics proposed in this thesis. In this scenario agents are not expected to behave as efficient as possible, instead it is desirable to implement a faulty behavior, to allow for players countermeasures. A system based on this work, can implement a local swarming behavior, connecting groups of agents to increase challenges and to produce emergent behavior. This means a behavior of the enemy agent which is not expected by players. A scenario that springs to mind are groups of enemies following a trail of odor, connecting to another group and surrounding the player in between. On the same time players can abuse the system by creating falsified information as baits to lure enemies out of critical zones. A system of agents which is simulated over a decent amount of time would have a distinct tendency withing the bonds, basically following decent leaders. Players can eliminate these leaders to disrupt agents organization and forcing a reconsideration of their strategy.

All this can be implemented in a high-performance way, as bonding mechanics would act as some high level coordination, letting usual logic[2] handle the 'close range interaction', unburdening the calculations. Therefore bonding and swarm topology can be updated asynchronously with a reduced frequency.

Looking at other AI solution used in game development with Unity the only comparable candidate is the RainAI system [Riv16]. While this system also allows a perception driven system it is only capable of running pre-definded state machines, with no way of producing dynamically emerged behavior.

This work has focused on three senses to create a diverse and heterogeneous swarm. During experiments a discrepancy between the influence of the senses has surfaced. Visual inputs are often dominating other senses, as provided information are usually more accurate and have a higher availability. This has been manually balanced out in certain scenarios and by the implementation of diverse modifiers in the scene, but a more diverse and balanced sense profile is expected to provide more complex behaviors. Especially in games this is of great interest, as it is possible to equip agents with non-classical senses. A quick example: Agents can be able to track players based on the number of *a*'s in the players name. Such considerations will lead to more diverse behaviors and enable new ways to 'play' the system.

The concept of bonding has proven to be complex system, looking at the implementation,

[1] 30 % of the top ten played steam games, and 20 % of the top 100 games fall into this category. When factoring in shooter games with grouped, AI-controlled enemies 42 % of the top 100 are covered [Val16] Accessed: 08.08.16

[2] A* pathfinding, ray casting and NavMesh navigation

as well as the balancing. This is partly against the premise of simple mechanics. In Section 4.2.2 some of the underlying connections were analyzed, but degree of dependency within the system is tremendous. The separation of the swarm via the Agents Per Bond parameter function as one of the major adjusting screws, influencing nearly every other aspect of the behavior. Different applications have been conceptualized and tested, but still every case needs a throughout analysis and prior testing to generate an efficient swarm in a deterministic way. With theses parameter in place the mechanics fulfill their expectation of providing interesting and emergent behavior patterns, guiding the swarm between targets and forming ad-hoc strategies to solve the given task.

5.2 Open Problems

As stated already in the results fine tuning the test environment is still a huge and crucial task. The size of the given task and if it can be completed within the time frame, has a huge effect on the swarm's overall efficiency. Decoupling these factors can help to ease this problem, for example by introducing other metrics to measure performance, more fit for the particular field of application.
While Unity 3D provides a rich toolbox to quickly prototype and iterate implementations it also enforces a strong coupling of the components, within themselves and also with the IDE. Especially at the evaluation stage of this work some *quirks* of Unity 3D became quite prevalent. The mayor problems with Unity 3D were the tight connection between components and Unity 3D internal events. To make use of the editors rich feature set behaviors need to follow Unity 3D's event function pattern[3], loosing the benefit of C#'s loose-coupled event system. As also stated in [Pen15] Unity 3D does not make any guarantee about call hierarchy within one event hook. So there is no defined order in which components will get their *Update()* function called. This leads to problems of synchronization. In this implementation the problem was solved by inter-meshing the *Update* and *LateUpdate* function. The pseudo-code shows how the seeking agent groups all of its sequential functionality into two frames, overlapping the logic to make sure all agents have finished a step before starting the next one. By moving away from Unity 3D or by implementing own call hierarchies this problem might be solved in a more efficient and elegant way.
On problem left to evaluate is the handling of missing communication: This work assumes that agents are always able to communicate their information over the bond channels. Cutting of this communication channel temporarily, or only allowing a closer range of communication, will lead to interesting changes in the pattern of group formation. This extends to other failure (see [DZZS04]) like sensor systems and/or movement failure. Restricting the swarm in such a way will lead to more insights in regards of robustness and validate the possibilities for real world application.
The implemented system already allows agents to channel information through a near field communication. But to isolate features of bonding mechanics this system was deactivated during test runs. It's up for further test to see if a local-communication-only

[3] https://docs.unity3d.com/Manual/EventFunctions.html, accessed 04.08.2016

system can maintain the information flow required for this task and how team allocation will be affected by it.

5.3 Future Work

Based on the findings of this work there are interesting directions left to explore.
The agents simulated in this work possessed no meta information about the task they need to fulfill. Changing this can allow the swarm to form more precise strategies, especially when used for artificial intelligence in computer games. With understanding about how many agents are required to complete a task the swarm can adopt the three components from [Eng07] (Social, Cognitive and Inertia velocity) to reflect this requirements. Given the game from this work, an agent which is alone will mostly rely on his social component to move closer to bonded agents. Agents which are close to more agents than required focus on their own cognitive velocity, to move away from the blob. Groups around the right count try to stick together.
Another improvement to the agility of the swarm would be some kind of short term memory. A common way for agents to waste their time is getting stuck at targets, without enough agents around them to actually remove it. With a short memory system agents can mark such tasks as unavailable for now and move on, to come back later when the situation may have changed. This consideration is most interesting when dealing with the real world application, where efficiency is crucial. However it is hard to clearly define condition to evaluate the state of a target (*'Have at least X agents close to it'* in this works example).
[DZKS06] and [SABS05] name the concept of market based task allocation. This systems allows human to feed high level task into the system and let the agents negotiate the strategy among themselves [SABS05]. Agents use a biding system to *buy* tasks they rate as rewarding, intuitively forming teams and splitting tasks. This market based approach would fit well with systems conceptualized in this work, as it is allowing for low level task composition and dynamic environments [DZKS06]. The complex task here is the mapping between the global high level task, and the low level tasks executed by the robots [DZKS06]. A well expressed mapping is required for the robots to prioritize they biding and allocate the right tasks [SABS05] and also to efficiently decompose the mayor goal [PL05]. If such a mapping is present this system could greatly benefit from a market based allocation scheme, as bonds can quickly be com- and deposed maximizing agents utilization. When locked on a target, but short of some agents, groups can lower the fee of their bonds task, attracting bonds into it. Large groups however could increase charges, making agents pursue other tasks instead.
As another way to allocate teams and to provide a diverse variety withing the the swarm [Mor15] implements evolutionary systems (see [PdJ04]) to develop ranking schemes for multi-objective multi-agent tasks. On the one hand, this can be used to control the bonding process, allowing agents to develop a bonding strategy which is most beneficial for the current situation. On the other hand it allows for evolution of the agents itself. Using a skill successfully can enforce training of this skill, shifting features of the agent from other senses. Especially in computer games it is useable to train distinct groups of

agents, performing well in certain game situations. In other scenarios this mechanic can be used to train agents, if features of the scenario are available beforehand.

This ideas extend on the mechanics conceptualized in this work, they add another layer of emergence or push the swarm to be even more efficient. For gaming the current state of this work provide an advanced implementation, which extend swarm based AI beyond whats usual as of today.

A. Appendix

A.1 Scenarios

This chapter describes the scenarios developed during this work. The scenarios are grouped in simple and complex scenarios. The legend (Figure A.1) shows different types of starting points, which define where agents start in the scenario. Advanced elements are described directly together with the scenario.

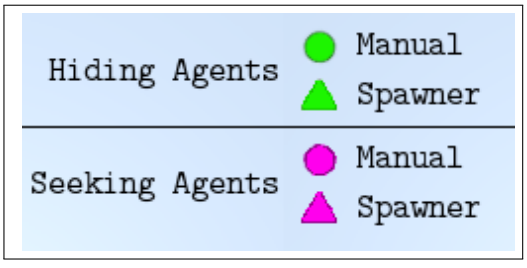
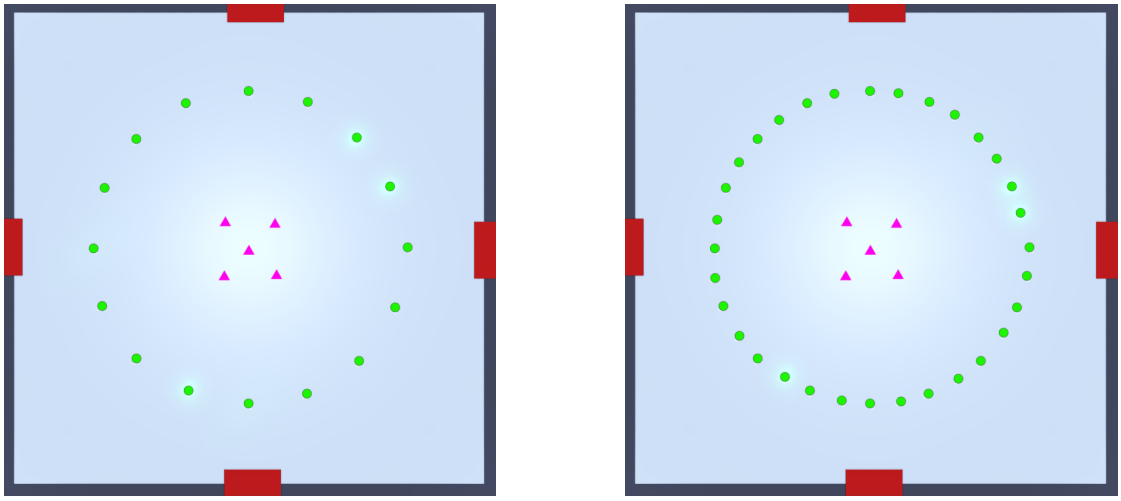


Figure A.1: Legend for scenario maps

Common elements in each scenario are objects to create the swarm on start-up. Green icons denote spawn positions of hiding agents, purple icons the spawn location of seeking agents. A circle means that the agent on this position was placed manually, therefore there will always be one agent at this spot. A triangle denotes a spawn point, so when the run starts the number of agents will be distributed among all available spawn points.

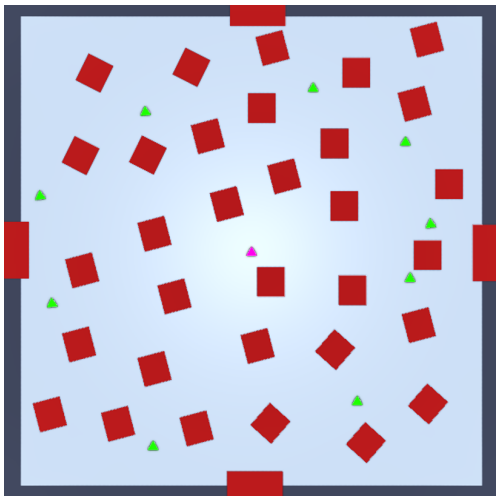
Figure A.2: Scenario: Legend



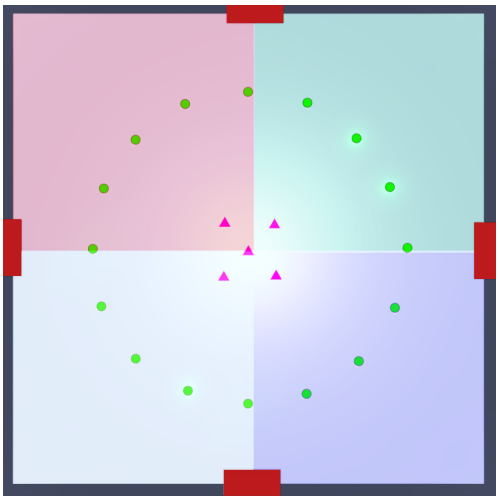
(S1) Arena: Very simple arena, which spawns seeking agents at five spawn-points in the center. Sixteen hiding agents are distributed on a circle around the seekers. This arena is used in most of the simple tests, as it allows the agents to focus on target selection and information exchange, without being influenced by environmental effects too much.

(S2) Arena Double: Similar to (S1), in this version the number of hiding agents is doubled, yet they remain distributed evenly on a circle around the center. Arena Double is used instead of Arena (S1) later on, to provide more diverse results due to the higher spread of possible targets.

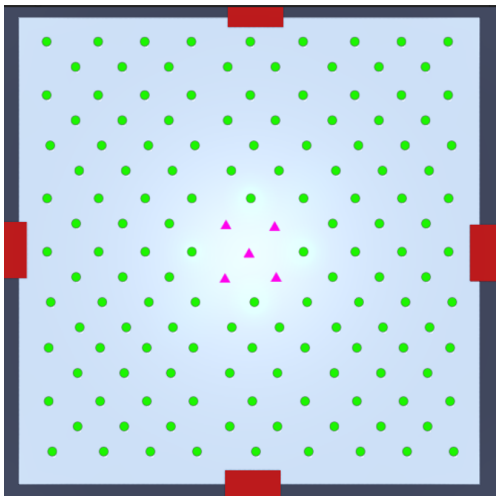
Figure A.3: Simple Scenarios



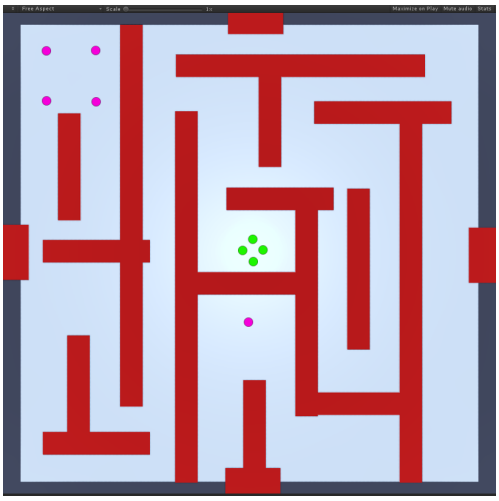
(C1) FORREST: This arena contains a loot of sight and movement blocking obstacles, colored red in the map. Agents are forced to navigate around them and often loose visual tracking of targets.



(C2) FOURSEASONS: In this scenario, based on S1, the whole arena is split into four chunks. Each colored chunk completely negates the signal ob the corresponding type, making agents invisible (red), muting their sounds (blue) and killing off their smell (green)

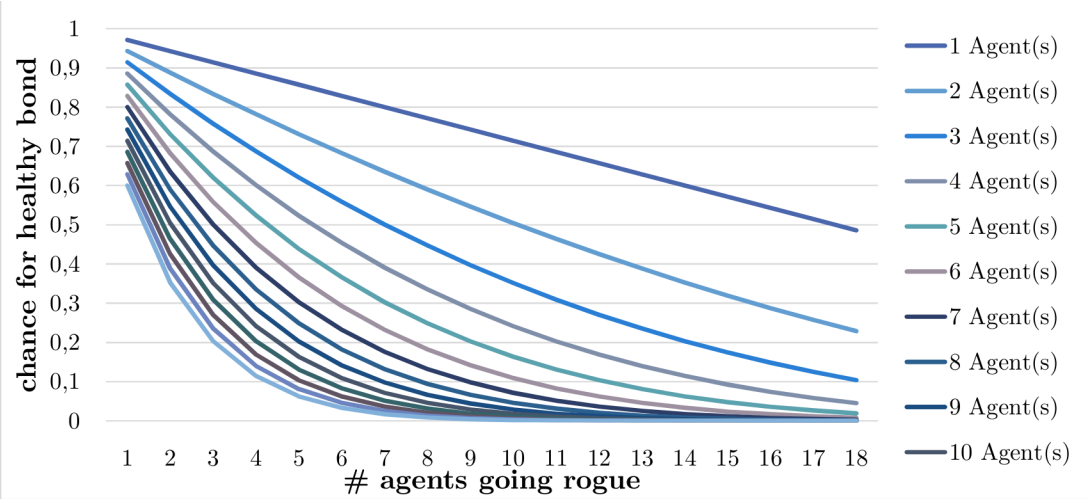


(C3) LEGION: To increase the time pressure and place a higher penalty on backtracking 140 agents were placed in this scene. Swarms need to split and work in an efficient way to reduce timely overhead by traveling back and forth.



(C4) MAZE In the maze two groups of seeking agents are created. One is able to notice the targets, but not able to reach it. The other group can pass the maze and reach the targets, but does not know about them. Bonding is required to fulfill the goal.

Figure A.4: Complex Scenarios



Agents per bond

rogue agents	1 Age	2 Age	3 Age	4 Age	5 Age	6 Age	7 Age	8 Age	9 Age	10 Ag	11 Ag	12 Ag	13 Ag	14 Ag
1 Byz	0,971	0,943	0,914	0,886	0,857	0,829	0,8	0,771	0,743	0,714	0,686	0,657	0,629	0,6
2 Byz	0,943	0,887	0,834	0,782	0,731	0,682	0,635	0,59	0,546	0,504	0,464	0,425	0,388	0,353
3 Byz	0,914	0,834	0,758	0,687	0,62	0,558	0,501	0,447	0,397	0,351	0,309	0,271	0,235	0,203
4 Byz	0,886	0,782	0,687	0,601	0,523	0,454	0,391	0,335	0,286	0,242	0,203	0,169	0,14	0,114
5 Byz	0,857	0,731	0,62	0,523	0,439	0,366	0,303	0,249	0,203	0,164	0,131	0,104	0,081	0,063
6 Byz	0,829	0,682	0,558	0,454	0,366	0,293	0,232	0,182	0,142	0,109	0,083	0,062	0,046	0,033
7 Byz	0,8	0,635	0,501	0,391	0,303	0,232	0,176	0,132	0,098	0,071	0,051	0,036	0,025	0,017
8 Byz	0,771	0,59	0,447	0,335	0,249	0,182	0,132	0,094	0,066	0,046	0,031	0,021	0,014	0,009
9 Byz	0,743	0,546	0,397	0,286	0,203	0,142	0,098	0,066	0,044	0,029	0,019	0,012	0,007	0,004
10 By	0,714	0,504	0,351	0,242	0,164	0,109	0,071	0,046	0,029	0,018	0,011	0,006	0,004	0,002
11 By	0,686	0,464	0,309	0,203	0,131	0,083	0,051	0,031	0,019	0,011	0,006	0,003	0,002	8E-04
12 By	0,657	0,425	0,271	0,169	0,104	0,062	0,036	0,021	0,012	0,006	0,003	0,002	8E-04	4E-04
13 By	0,629	0,388	0,235	0,14	0,081	0,046	0,025	0,014	0,007	0,004	0,002	8E-04	3E-04	1E-04
14 By	0,6	0,353	0,203	0,114	0,063	0,033	0,017	0,009	0,004	0,002	8E-04	4E-04	1E-04	5E-05
15 By	0,571	0,319	0,174	0,093	0,048	0,024	0,012	0,005	0,002	0,001	4E-04	2E-04	5E-05	2E-05
16 By	0,543	0,287	0,148	0,074	0,036	0,017	0,007	0,003	0,001	5E-04	2E-04	6E-05	2E-05	5E-06
17 By	0,514	0,257	0,125	0,058	0,026	0,011	0,005	0,002	7E-04	2E-04	8E-05	2E-05	6E-06	1E-06
18 By	0,486	0,229	0,104	0,045	0,019	0,008	0,003	0,001	3E-04	1E-04	3E-05	7E-06	2E-06	3E-07

Figure A.5: Chances of healthy bonds with byzantine swarms, depending of *agents per bond* and byzantine agent count

Reinf. Value	0,15	0,3	0,45	0,6	0,75	0,9
# of Reinf	Trust					
0	1	1	1	1	1	1
1	1,06550472	1,07826547	1,09597357	1,12	1,15207547	1,19443217
2	1,13180462	1,15732689	1,19274383	1,24079735	1,30494884	1,38966269
3	1,19889974	1,2371843	1,29031081	1,36239207	1,45862014	1,58569159
4	1,26679015	1,31783775	1,38867453	1,48478418	1,61308936	1,78251886
5	1,33547591	1,39928726	1,48783503	1,60797368	1,76835652	1,98014451
6	1,40495706	1,48153289	1,58779233	1,73196061	1,92442163	2,17856854
7	1,47523365	1,56457466	1,68854646	1,85674497	2,0812847	2,37779097
8	1,54630574	1,64841262	1,79009743	1,98232678	2,23894574	2,5778118
9	1,61817339	1,73304679	1,89244527	2,10870607	2,39740476	2,77863104
10	1,69083662	1,81847722	1,99559002	2,23588283	2,55666177	2,98024869
11	1,7642955	1,90470394	2,09953168	2,3638571	2,71671679	3,18266476
12	1,83855007	1,99172698	2,20427029	2,49262888	2,87756981	3,38587926
13	1,91360038	2,07954638	2,30980587	2,62219819	3,03922085	3,58989218
14	1,98944647	2,16816217	2,41613843	2,75256504	3,20166993	3,79470355
15	2,06608838	2,25757437	2,52326801	2,88372946	3,36491704	4,00031335

Reinf. Value	1,00	1,33	1,67	2,00	2,33	2,67
# of Reinf	Trust	Trust	Trust	Trust	Trust	Trust
0	1	1	1	1	1	1
1	1,22978251	1,40515413	1,71948847	2,28062485	3,28105146	5,06394355
2	1,46036362	1,81110747	2,4397765	3,56204945	5,56290278	9,12868702
3	1,69174336	2,21786002	3,16086409	4,84427379	7,84555396	13,1942304
4	1,92392172	2,62541179	3,88275123	6,12729789	10,129005	17,2605737
5	2,1568987	3,03376277	4,60543792	7,41112174	12,4132559	21,327717
6	2,39067431	3,44291298	5,32892418	8,69574534	14,6983067	25,3956601
7	2,62524856	3,8528624	6,05320999	9,98116869	16,9841573	29,4644032
8	2,86062145	4,26361104	6,77829537	11,2673918	19,2708078	33,5339462
9	3,09679299	4,67515891	7,5041803	12,5544147	21,5582581	37,6042891
10	3,33376318	5,087506	8,23086479	13,8422373	23,8465083	41,675432
11	3,57153202	5,50065231	8,95834884	15,1308596	26,1355584	45,7473747
12	3,81009951	5,91459785	9,68663246	16,4202817	28,4254083	49,8201174
13	4,04946568	6,32934262	10,4157156	17,7105036	30,7160581	53,89366
14	4,2896305	6,74488662	11,1455984	19,0015252	33,0075077	57,9680026
15	4,53059401	7,16122985	11,8762807	20,2933466	35,2997572	62,043145

Figure A.6: Evaluation of bond trust when reinforcing

Reinf. Value	0,15	0,3	0,45	0,6	0,75	0,9
# of Demote	Trust					
0	1	1	1	1	1	1
1	0,95710478	0,95439298	0,95183362	0,94940356	0,94708497	0,9448638
2	0,91501716	0,90959311	0,904474	0,89961354	0,89497609	0,8905335
3	0,87373729	0,86560051	0,85792125	0,85063006	0,84367344	0,83700917
4	0,83326533	0,82241531	0,8121755	0,80245322	0,79317713	0,78429091
5	0,79360144	0,78003767	0,76723686	0,75508313	0,74348726	0,73237882
6	0,75474578	0,73846773	0,72310548	0,70851991	0,69460393	0,68127299
7	0,71669853	0,69770563	0,67978148	0,66276369	0,64652726	0,63097352
8	0,67945987	0,65775155	0,63726501	0,61781459	0,59925736	0,58148051
9	0,64302999	0,61860564	0,59555621	0,57367273	0,55279434	0,53279408
10	0,6074091	0,58026807	0,55465523	0,53033826	0,50713833	0,48491433
11	0,5725974	0,54273903	0,51456224	0,48781132	0,46228946	0,43784137
12	0,53859511	0,5060187	0,47527739	0,44609204	0,41824786	0,39157534
13	0,50540246	0,47010728	0,43680085	0,40518058	0,37501365	0,34611634
14	0,47301968	0,43500498	0,39913281	0,3650771	0,33258699	0,30146451
15	0,44144705	0,40071201	0,36227345	0,32578177	0,29096801	0,25761997

Reinf. Value	1,00	1,33	1,67	2,00	2,33	2,67
# of Demote	Trust					
0	1	1	1	1	1	1
1	0,94343146	0,93889899	0,93468027	0,93071797	0,92697033	0,92340583
2	0,88766865	0,87860329	0,87016551	0,86224061	0,85474508	0,84761588
3	0,83271167	0,81911296	0,80645576	0,79456798	0,78332432	0,77263021
4	0,7785606	0,76042809	0,74355111	0,72770013	0,71270809	0,69844884
5	0,72521552	0,70254875	0,6814516	0,66163713	0,64289644	0,62507184
6	0,67267654	0,64547502	0,62015732	0,59637903	0,57388943	0,55249924
7	0,62094374	0,58920697	0,55966832	0,5319259	0,50568711	0,4807311
8	0,57001723	0,53374469	0,49998469	0,46827779	0,43828953	0,40976748
9	0,51989709	0,47908826	0,44110649	0,40543478	0,37169677	0,33960841
10	0,47058345	0,42523778	0,38303381	0,34339692	0,30590886	0,27025396
11	0,42207641	0,37219333	0,32576671	0,28216429	0,24092589	0,20170417
12	0,37437607	0,319955	0,26930527	0,22173695	0,17674789	0,13395911
13	0,32748255	0,2685229	0,21364959	0,16211499	0,11337495	0,06701883
14	0,28139598	0,21789711	0,15879974	0,10329846	0,05080712	0,00088339
15	0,23611648	0,16807775	0,10475581	0,04528745	-0,0109555	-0,0644472

Figure A.7: Evaluation of bond trust when demoting

AB	BC	CD	DE	EF	FG	GH	HI
0,4170859	0,8321355	-0,238052	#DIV/0!	#DIV/0!	-0,5954	0,40909	0,42894
0,4378706	0,9465159	-0,225178	#WERT!	#WERT!	-0,6343	0,388	0,31137
AC	**BD**	**CE**	**DF**	**EG**	**FH**	**GI**	
0,4258037	-0,203494	#DIV/0!	-0,08417	#DIV/0!	0,01475	0,67588	
0,4068296	-0,299021	#WERT!	-0,101399	#WERT!	-0,077	0,71939	
AD	**BE**	**CF**	**DG**	**EH**	**FI**		
-0,220605	#DIV/0!	-0,536549	-0,370485	#DIV/0!	-0,9416		
-0,290721	#WERT!	-0,590848	-0,33816	#WERT!	-0,9919		
AE	**BF**	**CG**	**DH**	**EI**			
#DIV/0!	-0,431498	0,9135633	-0,966147	#DIV/0!			
#WERT!	-0,610465	0,8950165	-0,81271	#WERT!			
AF	**BG**	**CH**	**DI**				
-0,280069	0,7387961	0,2738398	#DIV/0!				
-0,297917	0,922415	0,2823504	#WERT!				
AG	**BH**	**CI**					
0,4297041	0,2317411	0,5646838					
0,4419786	0,351572	0,6377414					
AH	**BI**						
0,2362149	0,4514208		**# Hubs**		**LargestBiDirSet**		**0,9136**
0,3005393	0,3005393		**# Bidirectional**		**HubCount**		**0,8321**
AI			**BiDirection**		**LargestBiDirSet**		**0,7388**
0,3068288			**SlaveCount**		**AvgSendingNet**		**-0,942**
0,3441923			**IslandCount**		**LargestSending**		**-0,966**

A: Average efficiency	**B: Bidirectional connections**
C: Hub count	**D: Island count**
E: Master count	**F: Slave count**
G: Largest bidirectional network	**H: Largest sending network**
I: Average sending network	

Figure A.8: Analysis of correlation between efficiency and different topology parameters

From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00
To	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00
Correlatio	0,07	0,03	0,09	-0,02	0,04	-0,01	-0,04	0,01	0,00	-0,06	0,10	-0,01	-0,27	0,32	0,06	-0,13	0,03	-0,15	0,11
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	
To	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00	
Correlatio	0,08	0,10	0,03	0,02	0,05	-0,03	-0,02	0,02	-0,04	0,03	0,10	-0,29	0,00	0,43	-0,02	-0,05	-0,07	-0,01	
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00		
To	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00		
Correlatio	0,13	0,08	0,04	0,04	0,04	-0,02	0,00	-0,01	0,02	0,06	-0,23	-0,14	0,23	0,43	0,00	-0,11	0,00		
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00			
To	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00			
Correlatio	0,12	0,09	0,06	0,03	0,04	-0,01	-0,03	0,03	0,05	-0,24	-0,14	0,05	0,29	0,45	-0,06	-0,05			
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00				
To	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00				
Correlatio	0,12	0,09	0,05	0,04	0,05	-0,03	0,01	0,05	-0,23	-0,17	-0,01	0,15	0,35	0,43	-0,02				
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00					
To	7,00	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00					
Correlatio	0,13	0,09	0,05	0,04	0,04	0,00	0,03	-0,21	-0,18	-0,08	0,08	0,23	0,34	0,44					
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00	13,00						
To	8,00	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00						
Correlatio	0,12	0,09	0,05	0,03	0,05	0,02	-0,21	-0,17	-0,10	-0,01	0,16	0,25	0,37						
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00	12,00							
To	9,00	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00							
Correlatio	0,12	0,08	0,04	0,05	0,07	-0,21	-0,17	-0,11	-0,05	0,07	0,19	0,30							
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00	11,00								
To	10,00	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00								
Correlatio	0,11	0,07	0,06	0,06	-0,14	-0,18	-0,12	-0,06	0,02	0,11	0,24								
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00	10,00									
To	11,00	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00									
Correlatio	0,10	0,08	0,07	-0,13	-0,11	-0,13	-0,08	0,00	0,06	0,16									
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00	9,00										
To	12,00	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00										
Correlatio	0,11	0,09	-0,12	-0,11	-0,08	-0,09	-0,03	0,03	0,11										
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00	8,00											
To	13,00	14,00	15,00	16,00	17,00	18,00	19,00	20,00											
Correlatio	0,12	-0,09	-0,10	-0,08	-0,06	-0,05	0,00	0,08											
From	1,00	2,00	3,00	4,00	5,00	6,00	7,00												
To	14,00	15,00	16,00	17,00	18,00	19,00	20,00												
Correlatio	-0,07	-0,07	-0,07	-0,06	-0,03	-0,02	0,04												

Figure A.9: Analysis of correlation between *agent per bond* parameter and efficiency, for different ranges of parameters

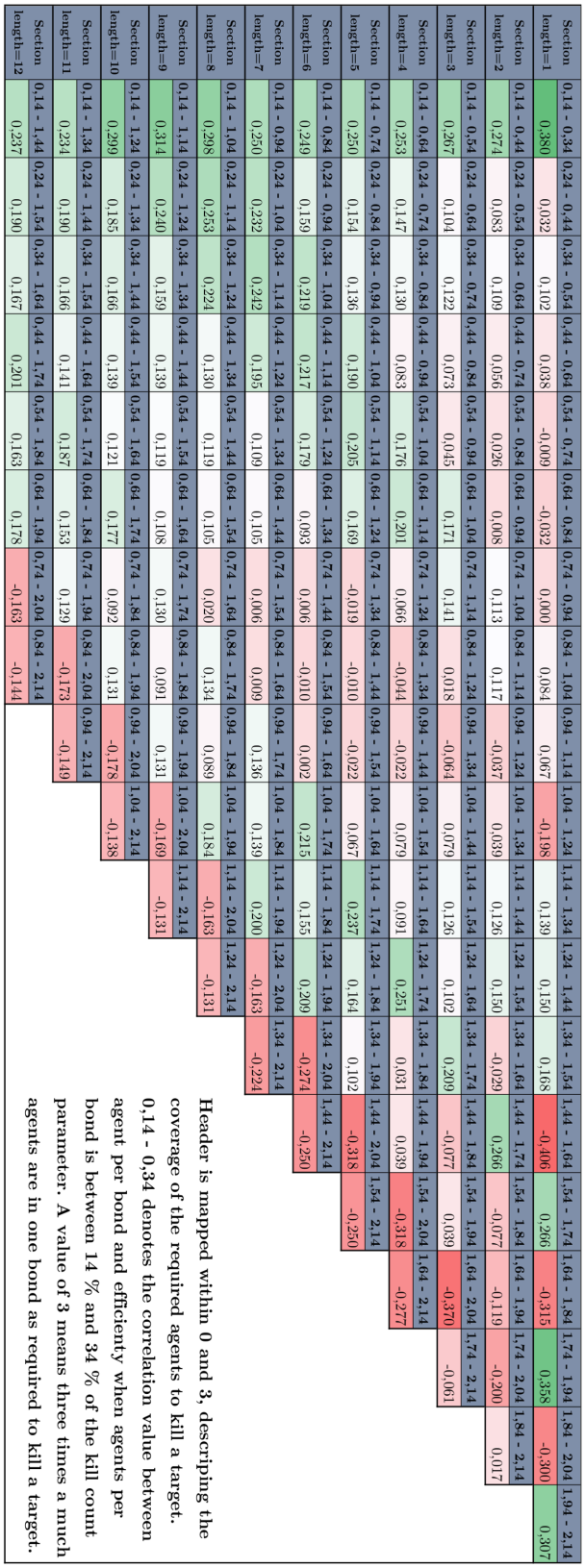


Figure A.10: Normalized correlation between bond size and efficiency

Bibliography

[AME04] William Agassounon, Alcherio Martinoli, and Kjerstin Easton. Macroscopic modeling of aggregation experiments using embodied agents in teams of constant and time-varying sizes. *Autonomous Robots*, 17(2):163–192, 2004. (cited on Page 5)

[BPH10] M. Bramer, M. Petridis, and A. Hopgood. *Research and Development in Intelligent Systems XXVII: Incorporating Applications and Innovations in Intelligent Systems XVIII Proceedings of AI-2010, The Thirtieth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*. Applications and innovations in intelligent systems. Springer London, 2010. (cited on Page 2)

[BT00] E. Bonabeau and G. Theraulaz. Swarm smarts. *Scientific American*, 282(3):72–79, March 2000. (cited on Page 1)

[CD00] Anthony Carlisle and Gerry Dozier. Adapting particle swarm optimization to dynamic environments. In *International conference on artificial intelligence.*, volume 1, 2000. (cited on Page 3 and 5)

[Cho10] Nian Shong Chok. *Pearson's versus Spearman's and Kendall's correlation coefficients for continuous data*. PhD thesis, University of Pittsburgh, 2010. (cited on Page 44)

[CWM04] Tianguang Chu, Long Wang, and Shumei Mu. Collective behavior analysis of an anisotropic swarm model. In *Proc. of the 16th International Symposium on Mathematical Theory of Networks and Systems*, 2004. (cited on Page 3, 5, 9, and 10)

[dSC09] Leandro dos Santos Coelho. *Multi-Objective Swarm Intelligent Systems: Theory & Experiences*. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2009. (cited on Page 19)

[DZKS06] M Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006. (cited on Page 5, 6, 13, and 50)

[DZZS04] M Bernardine Dias, Marc Zinck, Robert Zlot, and Anthony Stentz. Robust multirobot coordination in dynamic environments. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 4, pages 3435–3442. IEEE, 2004. (cited on Page 39, 47, and 49)

[Eng07] Andries P Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007. (cited on Page vii, 1, 3, 4, 5, 10, and 50)

[Gd05] Matthew E. Gaston and Marie desJardins. Agent-organized networks for dynamic team formation. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, pages 230–237, New York, NY, USA, 2005. ACM. (cited on Page 5, 6, and 41)

[GD08] Matthew E. Gaston and Marie DesJardins. The effect of network structure on dynamic team formation in multi-agent systems. *Computational Intelligence*, 24(2):122–157, 2008. (cited on Page 5 and 6)

[GM04] Brian P. Gerkey and Maja J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004. (cited on Page 5 and 8)

[Gut08] Christian Guttmann. *Making Allocations Collectively: Iterative Group Decision Making under Uncertainty*, pages 73–85. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. (cited on Page 5)

[KB00] Michael J.B. Krieger and Jean-Bernard Billeter. The call of duty: Self-organised task allocation in a population of up to twelve mobile robots, 2000. (cited on Page 4, 5, 6, and 13)

[Ken99] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, page 1938 Vol. 3, 1999. (cited on Page 14, 23, 25, 28, and 43)

[KNZ10] Leila Khalouzadeh, Naser Nematbakshs, and Kamran Zamanifar. A decentralized coalition formation algorithm among homogeneous agents. *Journal of Theoretical & Applied Information Technology*, 22(1), 2010. (cited on Page 10, 41, and 43)

[LE08] W.F. Leong and Oklahoma State University. Electrical Engineering. *Multi-objective Particle Swarm Optimization: Integration of Dynamic Population and Multiple-swarm Concepts and Constraint Handling*. Oklahoma State University, 2008. (cited on Page 19)

[LMAM04] Ling Li, Alcherio Martinoli, and Yaser S. Abu-Mostafa. Learning and measuring specialization in collaborative swarm systems. *Adaptive Behavior*, 12(3-4):199–212, 2004. (cited on Page 5, 9, and 28)

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982. (cited on Page 26)

[LV12] Somchaya Liemhetcharat and Manuela Veloso. Modeling and learning synergy for team formation with heterogeneous agents. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '12, pages 365–374, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems. (cited on Page 5)

[MDJ07] David Miller, Prithviraj Dasgupta, and Timothy Judkins. *Distributed Task Selection in Multi-agent Based Swarms Using Heuristic Strategies*, pages 158–172. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. (cited on Page 5)

[Men04] Rui Mendes. *Population topologies and their influence in particle swarm performance*. PhD thesis, Citeseer, 2004. (cited on Page 43)

[MJT13] Leandro Soriano Marcolino, Albert Xin Jiang, and Milind Tambe. Multi-agent team formation: Diversity beats strength? In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 279–285. AAAI Press, 2013. (cited on Page 5, 30, 35, and 41)

[Mor15] Ruby Louisa Viktoria Moritz. Cooperation in self-organized heterogeneous swarms. Master's thesis, University Leipzig, 2015. (cited on Page 1, 5, 6, 15, 28, 43, and 50)

[MT04] S. Mostaghim and J. Teich. Covering pareto-optimal fronts by subswarms in multi-objective particle swarm optimization. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1404–1411 Vol.2, June 2004. (cited on Page 19)

[New03] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003. (cited on Page 6)

[NG99] Dhruba Naug and Raghavendra Gadagkar. Flexible division of labor mediated by social interactions in an insect colony - a simulation model. *Journal of Theoretical Biology*, 197(1):123–133, 1999. (cited on Page 4)

[PdJ04] J Pieter and Edwin D de Jong. Evolutionary multi-agent systems. In *International Conference on Parallel Problem Solving from Nature*, pages 872–881. Springer, 2004. (cited on Page 50)

[Pen15] Aleksei Penzentcev. Architecture and implementation of the system for serious games in unity 3d, 2015. (cited on Page 17, 26, and 49)

[PL05] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005. (cited on Page 50)

[Riv16] RivalTheory. Rain ai features. URL: http://legacy.rivaltheory.com/rain/features/, 2016. accessed 08.08.2016. (cited on Page 48)

[Rob92] Gene E Robinson. Regulation of division of labor in insect societies. *Annual review of entomology*, 37(1):637–665, 1992. (cited on Page 4 and 5)

[SABS05] Jeff Schneider, David Apfelbaum, Drew Bagnell, and Reid Simmons. Learning opportunity costs in multi-robot market based planners. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1151–1156. IEEE, 2005. (cited on Page 5 and 50)

[SK98] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artif. Intell.*, 101(1-2):165–200, May 1998. (cited on Page 5)

[TSP+06] J. Gregory Trafton, Alan C. Schultz, Dennis Perznowski, Magdalena D. Bugajska, William Adams, Nicholas L. Cassimatis, and Derek P. Brock. Children and robots learning to play hide and seek. In *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-robot Interaction*, HRI '06, pages 242–249, New York, NY, USA, 2006. ACM. (cited on Page 5 and 9)

[Uni16] UnityTechnologies. Gameobjects in unity3d. URL: http://docs.unity3d.com/Manual/GameObjects.html, 2016. accessed 20.07.2016. (cited on Page 17)

[Val16] ValveCorporation. Top games by current player count. URL: http://store.steampowered.com/stats/, 2016. accessed 08.08.2016. (cited on Page 48)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den 15.08.2016