Distributed exploration of virtual environments
A game based approach

Authors
Mathias Börjesson (mathias.borjesson@gmail.com)
Andreas Mattisson (andreas.mattisson@gmail.com)
This thesis is submitted to the Department of Interaction and System Design, School of Engineering at Blekinge Institute of Technology in partial fulfilment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Contact Information

Authors:
Mathias Börjesson
E-mail: mathias.borjesson@gmail.com
Andreas Mattisson
E-mail: andreas.mattisson@gmail.com

Advisor:
Stefan Johansson
Department of Systems and Software Engineering
E-mail: stefan.johansson@bth.se
Abstract. In most games today the movements of computer players is controlled by a web of waypoints that have been predefined by the creators of the map, or automatically generated by an application on beforehand. This data is precompiled to save precious CPU cycles for the player of the game, but what if we want these computer players to be able to explore how to get around the map by themselves? How could this be done and how would the result change if we had several of these players cooperating to explore the map? We explore some of the possibilities for exploration and discovery of maps using approaches that is often found in robotics. We also look at what happens when there is a penalty for communicating between these computer players, and what effect different amounts of map coverage will have on the performance of them. By setting up a test environment inside an existing commercial computer game we developed a client side bot system that made it possible for us to test different parameters and settings. Based on the analysis of the test results we propose ways to describe and predict the effects of the number of bots and the communication rate, and we describe several ways of how to further advance on these ideas and experiments.

Keywords: AI, Agents, Exploration, Mapping, Quake 2, Games.
Table of Contents

1 Introduction ........................................ 1
   1.1 Background ................................ 1
   1.2 Related work ................................ 2
      1.2.1 Platform solutions .................... 2
      1.2.2 Algorithms and ideas .................. 2
   1.3 Research problem ............................ 3
   1.4 Methodology ................................. 4
   1.5 Outline of thesis ........................... 4
   1.6 Main Findings ............................... 5

2 The Quake 2 environment ........................... 5
   2.1 Quake 2 game server ....................... 5
   2.2 Q2BotCore Library ........................... 5
   2.3 Bot .......................................... 6
      2.3.1 Basic Navigation ....................... 6
      2.3.2 A layered approach to behaviours ....... 7
      2.3.3 Knowledge and communication ......... 9
   2.4 Strategies characteristic .................. 10
   2.5 BotCentral .................................. 11

3 Experiment setup .................................. 11
   3.1 Different amounts of bots .................. 13
   3.2 Communication rate ......................... 14
   3.3 Performance ................................ 15

4 Experiment Results ................................ 15
   4.1 Different amounts of bots .................. 16
   4.2 Communication rate ........................ 17
   4.3 Performance ................................ 18
   4.4 Knowledge over time ....................... 20
   4.5 Strategies .................................. 20

5 Discussion ....................................... 21
   5.1 Results ..................................... 21
      5.1.1 Number of bots .......................... 23
      5.1.2 Communication rate ...................... 25
      5.1.3 Performance ............................. 26
      5.1.4 Strategies ............................... 26
   5.2 Methodology ................................ 26
      5.2.1 Mapping technique ....................... 27
1 Introduction

Computer games are a good domain for experimenting and simulating artificial intelligence based algorithms. By using games as an experimentation environment instead of the real world it becomes easy to repeat experiments and as opposed to the real world you may also take control over the flow of time. Another positive aspect of using games is that sensor noise can be minimized or removed completely. There are many different ways to analyze and interpret an unknown environment and the focus of this thesis will be on implementing a set of algorithms for exploration and analyzing how differences in the setup of agents will affect the outcome and if any patterns can be found in these variations.

First person shooters, a phrase coined in the early 1990’s, has become very popular for gamers all over the world. These types of games take place in three dimensional environments with the camera mounted where the players head should have been and it usually shows the players weapon or hands at the bottom of the screen to give the illusion that you are looking through the eyes of the characters.

The computer controlled opponents in these games are getting smarter and in newer games such as Half-life 2 they have squads of enemies cooperating to surround or flank you from the sides and suppress fire to cover team members. More details about the history of first-person shooters can be found at following reference\(^1\).

Popular titles such as Half-Life and Quake 2 use webs of preset waypoints to allow the bots to roam the maps and find their way from one place to another without running into walls and getting stuck or lost on the way. This makes the navigation of the bots fast and reliable, however this is a static and not a very flexible solution and some paths might even be missed because they are not very obvious.

As some parts of this thesis go in to technical details we would recommend our readers to have some basic knowledge or experience in software development as well as first person shooters in general.

1.1 Background

We want to focus this study on how we can allow the bots\(^2\) to explore a map and generate their own knowledge model of how to navigate the map in such a flexible way that it can be reapplied to other systems and environments. We will use technologies from the field of robotics as this is a field of research where it is easy to find articles about exploration methods, and apply them to our simulated environment. To further advance this system we will also allow the bots to communicate with each other and share their knowledge of the map, as well as adding limitation to their sight and communication range. By running simulated experiments with the ability to repeat them with the same preconditions


\(^2\) Bot is commonly used for computer controlled players, and is short for robot
several times we want to visualize through statistics and diagrams the effects of some parameters in the environment and this model can be of interest for other researchers of either robot navigation or game AI developers.

After investigating the available technologies and environments we decided to use the first-person shooter Quake 2, developed by id Software, as an environment for our experiments.

1.2 Related work

Because there are a lot of small parts to consider when developing an exploration system, we have taken ideas and methods from several areas of research.

1.2.1 Platform solutions

Quake 2 Bot Core. To connect to the Quake 2 server we use a library called Q2BotCore [Swa98] which is written in C and contains a clean framework for accessing the information sent from and to the server. This library played a big role in our development and made our job a whole lot easier.

We wrote a .NET wrapper in C++ for this library so that we could easily access it from C#.

The QASE API. QASE [GFH05] is an API written in Java for Quake 2 that serves as a development platform for implementing machine and imitation learning, but also seem like it is well suited for evaluating general artificial intelligence systems. This API seem to have all the functionality we would have needed in order to implement our ideas smoothly and effectively, however it was found too late in our project for us to be able to use it.

1.2.2 Algorithms and ideas

Finding the path. Path-finding is a wide area and therefor a large quantity of research has already been done to find working solutions. After studying and comparing different possibilities we finally decided on using the A* algorithm [RP03, WW04] for searching through our graphs for a path. The main reason we chose A* was that it is very flexible and guaranteed to find the shortest path [Sto96] if one exists, and because it uses heuristics it is also quite fast. Other algorithms worth mentioning are Dijkstra’s algorithm [THCS01], which also is guaranteed to find the shortest path but does not use heuristics, and the Best-first algorithm [RP03].

We used an already finished A* library developed by Eric Marchesin called “A-star is born” and can be found at The Code Project website. 

---

3 http://www.idsoftware.com/games/quake/quake2/
4 http://www.microsoft.com/net/basics.mspx
5 http://www.codeproject.com/csharp/graphs_astar.asp
Field-based coordination. [MZ04] have written about coordinating a team of bots by using distributed computational fields (Co-Fields), almost like dynamic magnetic fields. This was implemented in Quake 3 Arena\footnote{http://www.idsoftware.com/games/quake/quake3-arena/}, the sequel to Quake 2, using a server-side approach\footnote{Changes to games, be it server-side or client-side, is generally referred to as a “mod”} to reuse some of the already implemented intelligence in the game.

Ant behaviours. Although we decided not to use ant-like behaviour [KL01] for our system, we still want to mention this, as it is a very interesting view on how you can simulate ants dropping pheromone when looking for food, and using this trail of pheromone to find its way back to the ant hill once it has found what it was looking for. If you then allow more ants to travel this trail they will drop more pheromone and maybe not walk the exact wiggly path the first ant took, but at least close enough to follow it. If the new trail of pheromone is a shorter path the ant can follow this new trail more often and therefore drops more pheromone on the new path. Since this more pheromone generates a stronger attraction for the ants it might eventually evolve in to a shorter and quite optimal path between the food and the ant hill.

One possibility of using this technique for exploration would be to make the dropped pheromones repel the bots instead of attracting them, that way they would avoid going to places already visited and find new unexplored terrain. Some of the difficulties with this is to prevent the pheromones to not leak through walls and thus affects other areas on the other side of a wall.

We did not implement this method of exploration mainly because of personal preferences and better understanding of the waypoint system we did implement.

1.3 Research problem

Hypothesis 1 The number of communicating bots during a training run is related to the resulting waypoint coverage of the map will follow a curve described as: \( y = 1 - ae^{-bx} \) where \( 1 \leq x \).

Where \( y \) is the fraction of how much of the map that has been covered, \( x \) is the number of bots, and \( a \) and \( b \) are parameters to adjust the curve and we will try to calculate these to fit our experiment results and see if we can get the curve to match. We expect to find that the coverage of waypoints after a training run will increase the most for the first number of added bots. The coverage should then increase less for each added bot, giving a curve similar to the one described above.

Hypothesis 2 An increase in the communication frequency will have a positive effect on the training data, even with a communication penalty.

Even though we have a penalty that makes the bot stand still for a few seconds after sending a message, we believe to find that by allowing the bots to com-
municate with each other as often as possible will have a positive effect on the training data.

Hypothesis 3  The effectiveness of how fast a bot can navigate towards a goal is related to the amount of information gathered during the training phase.

We believe that a high coverage of a map will decrease the time it will take for the bot to move to a random point on the map. This would be because it has a higher chance of finding a shorter path if there are more nodes and coverage.

1.4 Methodology

We started out with a literature survey covering autonomous exploration and similar topics to find out more about how it has been done before. A lot of the research about exploration has been done in the field of robotics, for example different mapping techniques and knowledge models.

When we decided on what game to use for our experiments we found different games like Quake 2, Quake 3 and Half-Life\(^8\) that had been released with their source code. By evaluating and comparing the available opportunities for each of the games we finally settled on using Quake 2 as our platform. The source code for the Quake 2 engine has been available to the public for a long time now, and this game has been referred to and recommended by numerous forums and articles [Cha01] surrounding artificial intelligence. We also found the Q2BotCore platform for Quake 2 to build our client side bots on, and the simple interface provided by this library was also one of the main reasons for our choice.

The development environment we used was Microsoft Visual Studio 2003 .NET\(^9\) and the most parts of our system was implemented in C# with the exception for a wrapper we wrote in C++ that translated the .NET calls to be used in the library we utilized for connecting our bots using the client-side approach.

To test our hypothesis we designed several scenarios to expose the bots to such as a changed probability of communication or different amounts of bots. These experiments will be explained more in detail in Section 5.1.

By gathering metrics from test runs we will compile diagrams that we analyze with statistical methods to test if the hypothesis hold true or not.

1.5 Outline of thesis

In the following section we will take you through the different parts of the system we developed as well as how the training and testing of our bots was performed. Section 4 shows the raw results of our runs with diagrams and tables filled with extracted values. These values and trends seen in the diagrams are discussed in Section 5 where we will confirm or disprove our hypothesis described in Section 1.3. After our results have been discussed we will conclude our findings, and

\(^8\) http://half-life2.com/  
\(^9\) .Net is a shorter name for Microsoft Visual Studio .NET
finally we will finish up with discussing the possibilities of how to continue from here, what are some of the options available for continuing the work we have done.

1.6 Main Findings

The work presented in this thesis is built upon our literature survey and some of the interesting findings we made during this. Through the hypotheses described in the previous section we will present a generalized model for prognosing and visualizing the resulting effects from changes made to the exploration setup. By developing an experimental environment that implements algorithms for exploration we will try to confirm or disprove each one of our hypotheses previously defined, and through discussion in the concluding section share our experiences and thoughts about the hypotheses and results which may serve as a stepping-stone for future research.

2 The Quake 2 environment

Our system is primarily made up of four parts: The Quake 2 game server, the Q2BotCore library which is used to connect and communicate with the game server, the Bot where most of the magic happens, and the BotCentral which is the communication agent responsible for handling the communication between bots and visualize the current game state. See Fig. 1 for a visual representation of this. We will go into more depth on these in the following subsections.

2.1 Quake 2 game server

We ran the game server on a Linux machine as a dedicated server. It is also possible to run the server in a graphical mode, but this was not something that we had use for and would only slow the server down. If we wanted to enter the game and see the bots running we could connect a normal Quake 2 client and enter in spectator mode. Once the server is up and running it is possible for game clients to connect and play on it, or as in our case, connect our bots and have them roam the running map.

2.2 Q2BotCore Library

This library is what we use to connect and communicate with the server. We wrapped the Q2BotCore library [Swa98] to C++ .NET so that we could access it from C#, and we also convert some of the C-structs to more managable .NET classes.

The library takes care of the communication protocol for us, and we merely only had to port it to C# so we could code our first bot.
2.3 Bot

The bot is the computer controlled played connected to the server. This module is the largest part of the system and handles the exploration and interprets of the game state.

We decided to implement our bots using the client-side approach, because this means that we do not have to worry about how the game engine works, and it also prevents us from affecting the environment except for moving around as far as the physics and collision detection allows us to. Also, by running our bots on the client-side we can run the bots on a different CPU than the server. We may even distribute the bots over several machines if it would be necessary. We consider this to make our solution more flexible because the only commands they send to the server are either a move or turn command, and thus the interface to the server could easily be replaced by something like an interface to a robot to be placed in the real world.

2.3.1 Basic Navigation.

For navigation and basic collision avoidance we are using what we call feelers which is a trace from the bots origin on the map, to another point in front of the bot. The implementation of this works in such a way that the feeler function tests a line from the bot to, for example, a point a few units in front of the bot. The function returns a number between zero and one that shows how long the
A description of how we utilize this method for collision avoidance can be found under Section 2.3.2

2.3.2 A layered approach to behaviours

The behaviours of the bots are implemented in a layered manner and applied to the movement of the bots in each tick (every tenth of a second) of the game, so that each of the layers can transition into the next. If a layer such as the Collision Avoidance concider that it is necessary to avoid something it will apply the avoidance algorithm to the movement of the bot, and break the behaviour ladder so that the next behaviour layer is never reached. Because the behaviour of the bot is layered as it is, it will never try to apply a strategy like Random Walk when it is about to collide with a wall or another obstacle. This because the Collision Avoidance layer will react and stop the bot from continuing to the next levels of behaviours until it no longer needs to avoid the obstacle. See Fig. 2 for a list of behaviours.

![Diagram of layered behaviours](image)

**Fig. 2:** The bottom-up layered behaviours of a bot.
Check Health. When the amount of health is reduced to zero (or sometimes below zero) from for example falling, drowning, getting shot or telefragged\(^{10}\) the bot will die and no longer be able to move. This layer would take care of resurrecting the bot if it would be killed which is done by triggering the shoot action once.

Prevent getting stuck. There is a small risk of getting stuck in cramped or wedge shaped areas if the collision avoidance would for some reason not do its job correctly. If this layer senses that the bot has not moved from its current spot for ten ticks it will react and force the bot to rotate to a random angle. Even if this new angle does not directly show an open path it will still allow for the collision avoidance layer to try and navigate out of it until the coast is clear. Although this might not be the most optimal way to get unstuck we found this to be a simple and well working solution. If the bot would get stuck again after rotating this whole procedure will be run again.

Collision avoidance. In this layer the bot utilizes the feelers to avoid running into walls. The bot will project a single line straight forward 120 units to see if the current direction of the bot is good, or if it is getting too close to a wall. The threshold for the front feeler is set to 0.6 so that when a wall gets within 72 units this layer will react and start to look at the surroundings as well. We divide the view of the bot in to four sections, as seen in Fig 3, and project three traces with 10 degree intervals within each of these sections. The bot is then turned towards the center of the section which has the highest sum of trace values, meaning that the section that reaches the longest before colliding with a wall will be the new direction for the bot.

Movement along a vector. This layer will move the bot towards a specific target stored in a queue until it is has arrived. This is done by simply turning the bot to face the target point every time this layer is visited, which is done every tick unless one of the previous layers has to intervene. When this target is reached the bot removes the target from the front of the queue. If no orders are available it will proceed to the next layer.

Movement along a path. When the bot is about to move to a point on the map by using its knowledge this layer will calculate a path from the waypoint closest to the bot, to the point that lie closest to the target. Each step in this path is then placed in the queue handled by the previous layer, which will move the bot towards the target, step by step. This layer uses the A* algorithm to calculate the shortest path to the target.

\(^{10}\) Telefragging is a term used for when a player is teleported or resurrected at a location where a living player is standing. This would instantly kill the player standing at the spot.
Apply strategies to movement. A single strategy is assigned to the bot at startup, and in this layer the assigned strategy will be applied to the bots behaviour and movement. The details about the strategies are explained in Section 2.4. If the strategy does not have anything at the moment it will go up another step in the behavior layers to the next and final layer.

Idle. This is the final layer of the bot which will just stops the bot. As the name implies nothing else interesting is happening here but waiting for a new order to perform.

2.3.3 Knowledge and communication.

The knowledge of the bot is stored as a graph of nodes and links describing where the bot has been, or where another bot has been if they have communicated. When two bots are close to each other they can send their graphs to the other bot, which will then be merged together so that both bots will contain similar knowledge about the map.

We have also implemented a cost for sending messages between the bots. The cost for sending a message is that the bot will stop for a few seconds while it is exchanging information (this delay is simulated). The effect of using penalty when exchanging information between bots will hopefully show us how the exploration will be affected. Will the knowledge that the bots have together differ much when using different rates to allow the bots to share waypoints between them? The effect will be that due to more communication the more penalty time will pass, which may affect the amount of exploration they will be able to do.
Are there any special rates of communication which perform better than the others and are they better or worse than using full communication to maximize the knowledge for all the bots.

### 2.4 Strategies characteristic

Detailed information about how the strategies works will be presented here. It will show how different strategies will act on the present scenario.

**Random Walk.** The Random Walk strategy makes the bot run around the map turning a random amount of degrees between ±6 each tick. This will make the bot run either S shaped curves or sometimes running in circles of different sizes. How this strategy was implemented is shown in Fig 5.

Although this strategy applies a random movement, this strategy as well as the others are controlled by underlying layers of behaviours. This means that although it will run randomly it will still avoid running in to walls and prevent getting stuck.

**Follow Left/Right Wall.** These two strategies are quite self-explanatory. They make the bot stay close to a wall on either the left or right side, and then follow this wall along the map.

Both strategies have feelers pointed toward the wall they follow. If the wall they are following get further away from the bot it turns toward the wall. In Fig. 4 the follow left strategy will turn to the left and follow right strategy turn...
RandomWalk ()
{
    if ( ticksSinceLastChange >= 30 )
    {
        currentRotation = Random number between −6 and 6 degree
        ticksSinceLastChange = 0
    }
    currentDirection = currentDirection + currentRotation
    ticksSinceLastChange++;
}

Fig. 5: Pseudocode for random walk strategy

right at the corner. Neither of these strategies take in to account if the area they
turn to has already been explored or not. The performance of this two strategies
are not good since they could get stuck in the same cycle of the map.

Gap Finder. This strategy looks for unexplored areas in its known graph and
will pull the bot towards holes in the web of waypoints.

Gap finder will look at both directions and look if any direction has any
knowledge. That direction which has minimum knowledge will be the chosen
direction which the bot takes. If there has not been explored at any direction
the bot choose one direction and save the other way in the memory.

When an area does not have any gap around the bot, and it haven’t found
any gap in a while, it start to check the memory if there are any place it has not
been too. If so, it start walking toward that point, to explore further.

Spiral. Spiral will have the bot follow the outer regions of a room, and once it
gets back to where it has already explored it will move one step towards the
middle and continue following the outer regions. Working its way towards the
middle in a spiral motion to fully explore the room before moving towards the
next.

This means that this strategy is a complement to the follow-wall strategies,
since they only took care of the outer regions.

2.5 BotCentral

The BotCentral is the part of the system that handles messaging between the
bots as well as coordinating the training and testing sessions. It can also visualize
the bots waypoints and the messages being sent.

3 Experiment setup

We developed an application for running bots as well as the BotCentral, and
scripted our runs in XML. After a script has been written we can load it in to
the test application which then would step through it. The only time human
interaction is needed is when we wish to change scripts or change the map on
the server, however some of our scripts took as long as eight hours to run this
was not a big problem. The three maps we created for testing are 2376x1696
units big and a single bot covers 32x32 units. These maps can be seen in Fig 7,
8 and 9.

We use two different ways to measure the exploration data for the results, the
first way is the node count where we sum up how many nodes the bot has placed
on the map. Each new node is placed 75 units away from the closest known node
and therefore a high number of nodes would mean that a large space has been
covered by the bot. The second way we measure the data is by calculating how
many percent of the map has been covered. The coverage is calculated using a
grid and a waypoint representation of the map. The representation was manually
generated offline by testing points with even distances from each other over the
map to see if they were too close to a wall or free to place, which resulted in a
very nicely covered map as seen in Fig 6 as a blue graph. The grid was manually
adjusted to find a good size that would fit up to about three nodes at most,
and this grid size was used for all map coverage calculations so that the results
would be comparable.
Fig. 7: Map 1 has many open areas to navigate through and a room with what could be viewed as an open door.

The coverage percentage is calculated by counting the number of squares that contain at least one node from the representation, if the square contains at least one third as many nodes from the bots graph, then the square will be counted as explored. For example, if a square was filled by three nodes from the representation it would be enough if the bot had one node inside the square to consider it explored, if the square has four representation nodes it would not have been counted as explored because $\frac{1}{4} \not\geq \frac{1}{3}$. In Fig 10 we have written some pseudo-code to further explain how this was done.

3.1 Different amounts of bots

The setup for this test was done by scripting four different training runs with one to four bots. Each run would be timed at three minutes for exploration and no penalty for communication, because this is not what we want to test in Hypothesis 1. Each one of these runs was repeated nine times on each map to get a larger population, for comparison and the average number of nodes for the bots on each setting was calculated together with the standard deviation, which are both shown in Section 4 where we present our results.
3.2 Communication rate

To test what effect the communication rate would have on the bots, as mentioned in Hypothesis 2, we put together a script with four bots in each run and stepping from 10% communication probability up to 100% with 10% intervals. If two bots came within 100 units range of each other they would at 100% communication rate send their knowledge to the other bot. The penalty for sending this information was to stop the sending bot for 10 seconds, as well as a cooldown preventing it from sending another message for 15 seconds which gives the two bots five seconds to explore before checking if they should send a new message. If the communication rate is below 100% the bot will generate a random number between 0 and 100, and if this number is lower or equal to the set communication rate it will send the message. If the random number is above the rate no message will be sent but the bot will still activate the cooldown that prevents it from sending messages for 15 seconds.

We ran 15 tests per 10% communication rate on each of the three maps and as in the previous test we also calculated the average and standard deviation values for the nodes in each bot.

Fig. 8: Map 2, this map is characterized by long corridors.
Fig. 9: Map 3 is similar to Map 1 with much open space, but around the opening in the middle of the map there are a lot of walls to avoid if the bot is to get to the other side.

3.3 Performance

The performance tests were set up so that one bot entered the map, loaded up a set of previously explored waypoints, and is then given a random point from the BotCentral to try and navigate to. The bot will receive 10 random points to try and find its way to, and a 32 second time limit for each point. When the bot has a low map coverage there is a possibility that the target point is not inside the known areas of the bot, in that case the bot will move to the known waypoint closest to the target, and once reached it will turn towards the target and try to walk towards it from there. If the point is not reached it will be removed, and a penalty of 20 seconds is added to the bots total time.

4 Experiment Results

In this section we show some data and diagrams over the results we extracted from the experiments that were described in the previous sections.
MapCoverage() {
    for each square {
        representationNodes = representation nodes inside the square
        if ( representationNodes > 0 ) {
            botNodes = bot nodes inside the square
            if ( botNodes / representationNodes <= 1/3 )
                filledSquares = filledSquares + 1
            squares = squares + 1
        }
    }
    coveragePercent = (filledSquares / squares) * 100
}

Fig. 10: Pseudocode for calculating map coverage.

4.1 Different amounts of bots

We executed nine training runs for every setting on the three maps, as described in Section 3. In Fig. 11, 12 and 13 we see the average number of placed waypoints for the bots in each setting, each bar is also marked with the standard deviation to show the spread of the results.

Fig. 11: The result from training different amounts of bots on Map 1.
4.2 Communication rate

A higher chance of communication will make the bots share their knowledge more often and this would give the best performance at a rate of 100% communication. However if we add a penalty by stopping the bot for a few seconds for sharing knowledge, the question is whether full communication is still the best and how the amount of discovered space is affected by this penalty. The results from the communication rate tests are compiled into the diagrams shown in Fig 14, 15 and 16.
4.3 Performance test

Performance test is a test to see what impact the map coverage has on the time it takes for the bot to move to a random point on the map. One bot is started and
Fig. 16: The result from training on Map 3 with different amounts of communication rate settings.

connected to the server and MapCentral, and the MapCentral will then generate ten target points within the map. Each point is tested so that it is not placed inside a wall, and it is then given to the bot as a target. For each point the bot is given a time limit, if the point is not reached within this limit the point will be flagged as missed and a penalty is added, then the next point is placed. At the end of the test a summary is written to a text file with the total time it took for the bot to visit all points, and also the number of missed points is displayed.

The test is performed nine times on every bot, which have different amount of information about the environment. The information will be at 10% intervals from 10% to 100% knowledge to get a reasonably overview. The total time on the same amount of knowledge, that has been tested nine times, will be calculated to a average \( \frac{\text{Total time}}{\text{Total amount points}} \). This will be the results of how good the current knowledge the bot has.

On the performance test this means that there will be a total of 81 tests.

The diagram in Fig. 17 shows that overall time it took to perform the test given the amount of map coverage. The percentage in the diagram shows the map coverage stored in the bot at the time of the test. As we see at the bot with 100% coverage, the time represent the average time it takes to travel from point A to B and with no searching delay. A bot with less knowledge will have to search nearby points to find possible routing path. The penalty time is 20 seconds for a missing point and at the diagram the average penalty time is shown.
Fig. 17: Performance of the bot relate to its knowledge about the space. Higher knowledge admits to lower testing time (lower time is better). The penalty is given when a point is not reached.

4.4 Knowledge over time

By using the random walk strategy we want to see how much information the bot is gathering over time. A look at the diagram in Fig. 18 we see that the exploration rate (the yellow area at the bottom that shows the difference made at each timestamp) is high at the start, but is slowly reducing over time, which is a quite expected result. At the beginning when we have a space totally unknown and the bot will keep running over unexplored terrain, but after a while it will be starting to cross previously used paths more and more often and therefore add less new nodes to the knowledge.

4.5 Strategies

We also compared the result of different strategies to see how they affected the exploration of the bot. In Fig. 19 we see the different results for these strategies.

Spiral finder was the best overall strategy, which could be since the strategy runs around the walls and when it comes back to where it has already been it spirals towards the center, giving a good coverage of that room. The strategies who follow the walls did not investigate much at the middle of the map and therefore lack information about the middle of the rooms. They however got great results on Map2 which is generally made up of long corridors. The big
Fig. 18: The diagram shows how a single bot explore a map over time. The rate at which new nodes are added is reduced over time when it gets harder to find undiscovered areas in the environment.

surprise was the performance of the random walk strategy, performing fairly good on all maps.

We also tested some combinations of strategies to see if there could be some setups that would give better results than using the same strategy for all bots. Fig. 20. The results from our tests show that two bots with gap finder strategy and one with the spiral got the best combined result. The spiral strategy focus much more on the outer parts of the rooms and the two gap finder took care of a lot of the central details.

5 Discussion

The following sections will discuss the experiment results extracted in the previous sections as well as the methodology we used.

5.1 Results

To answer Hypothesis 1 and 2 we ran the test described in section 4.1 and section 4.2. We ran bot training with different settings on the number of bots or the communication frequency respectively, and by running each training several
Fig. 19: Strategy diagram that cover the different strategies the bot can have. Are run with a single bot and show the performance of each strategy in each map.

times, saving the training data and resetting the memory after each training, we get an average result together with a standard deviation to analyze.

A good coverage of the map is defined by the spread of the waypoints of the bot. This is calculated by placing a grid of arbitrary resolution (this will be discussed more in depth later in this thesis) over the map, every square in the grid that is filled by a waypoint will be considered covered. By calculating \( \frac{\text{covered}}{\text{total}} \) we can see how many percent of the map that have been explored. A visualization of the graphs from two bots can be seen in Fig. 6 where the blue waypoints was a map reference.

To measure performance of the bots for Hypothesis 3 we designed a test where the bot will load a set of training data that is to be tested, and then the bot is given a random point on the map to get to in a limited set of time. If the bot does not make it to the point in time the point will disappear and be recorded as missed. This is repeated a number of times for the bot, and the result will be the number of completed goals compared to the number of goals given, as well as the time it took for the bot to work its way through the test.

Analysis of the knowledge a bot or a group of bots have, is done by our own analysis tool. A representation of the map is loaded into the application. Then one or more bots could be added and an overview of the knowledge is displayed.
Fig. 20: Strategies mixed to see different effects on strategies performed together. Here we used three bots, with two of the same strategy.

The knowledge is presented in either percent of the map that is covered, this method has been described earlier, or the number of nodes placed on the map.

5.1.1 Number of bots

More bots gave better coverage as previously proposed. There are several reasons for this, one is that due to more bots they have a greater probability of exploring new areas. They also communicate more often if there are a lot of bots since they come across each other more regularly, and that should increase the spread of the knowledge which in turn will increase the average coverage.

For Hypothesis 1 we proposed that the relation between map coverage and the number of bots could be generalized as a mathematic formula. To see if the parameters for the formula we proposed can be adjusted to fit our results we start by calculating the percental coverage of the map to set the scale between 0 and 1 which is the range of the function. In Fig. 21 we show the calculated coverage from our experiments and a trendline with the parameters adjusted to fit the bars. The parameters were different for each map and their values can be
Fig. 21: The coverage of the different maps processed from the data provided in the experiment results. The rectangles are the calculated values using the formula with the respective parameters shown in Table 1.

viewed in Table 1. Because the parameters are different for each map we would argue that they are a product of the different characteristics of the maps we used, such as the amount of tight corridors or large open spaces. The training time for the bots would also have an effect on the parameters, and if we were to speculate we would say that an increase in training time would raise the overall height of the bars in the diagram but still keep the shape of the curve. It is very difficult to find a way to analyze how and by what these parameters are affected by individually so initial tests on the first bot numbers are required to get the initial curve and when we have that we can adjust the curve with the parameters to allow us to predict future results. Given these parameters we can test what would happen if we would, for example, use 10 bots on map 1, and this equation is displayed in Fig. 22. Which means that if we had added 10 bots to the map they would after three minutes have covered somewhere around 88% of the map and this sounds as a very reasonable result. From what we have seen in our diagrams and tests we consider this hypothesis to hold true.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map 1</td>
<td>0.70</td>
<td>0.18</td>
</tr>
<tr>
<td>Map 2</td>
<td>0.75</td>
<td>0.10</td>
</tr>
<tr>
<td>Map 3</td>
<td>0.70</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Table 1: The parameters used to predict the trend for what will happen if more bots are added.
\[ a = 0.7 \]
\[ b = 0.18 \]
\[ x = 10 \]
\[ \text{coverage} = 1 - 0.7e^{-0.18 \times 10} \]
\[ \text{coverage} = 1 - 0.7e^{-1.8} \]
\[ \text{coverage} \approx 0.88 \]

\textbf{Fig. 22:} Calculating a prognosis-value using the formula from Hypothesis 1 and the tweaked parameters discussed in the discussion part of this hypothesis.

\section*{5.1.2 Communication rate}

In Hypothesis 2 we proposed that even if we used a penalty for communication we expected to see the average map coverage to increase with the rate of communication. Looking at the diagrams in Section 4.2 this is not necessarily true, the amount of nodes even out near the higher rates of communication in all of the three diagrams. There even seems to be a slight decrease around 90\% in all the three maps however it is hard to see a pattern here. If there would be no penalty the coverage should logically increase with a higher communication rate but with the 10 second penalty we use here the higher rates seem to be slightly worse. This could be a natural variation but after looking at the diagrams and comparing and discussing results we believe that there is another variable that plays a big role here.

If we were to, say, double the penalty for communication we believe that there would be a much more noticeable trend in the diagrams. We would expect that because the penalty would take away even more precious exploration time from the bots the highest communication rates would make the map coverage curve drop more noticeably and reveal a peak, maybe somewhere around 60\% - 80\% looking at the diagrams we currently have however this is just a speculation.

With our current results we can neither confirm or disprove Hypothesis 2, as more training runs are required to get a clearer result. The time it takes to run enough training sessions for a more complete set of data makes it impossible for us to test this now. The current data used for this hypothesis took slightly over eight hours per map to run (counting both the 3 minute training and the additional 16 seconds for startup and shutdown of each run) as well as a good amount of time to compile and calculate the results into diagrams so that they could be analyzed. Because of the time it would take to test several variations to the penalty time we will therefore leave this question open for future work.
5.1.3 Performance

A higher coverage of the map did indeed increase the performance of the bot when it was to find its way to random points on the map. The A* algorithm will find the shortest known path to the target and a good coverage will have a higher probability of having a path leading directly to the target which in turn will increase the chance of the bot finding the target.

5.1.4 Strategies

Random strategy was the strategy we used when testing Hypothesis 1 and 2. We wanted a strategy that would spread out the bots randomly over the map to try and make the meetings between the bots random. If we had for example used a strategy that made the bots follow a wall they would probably line up by the wall and either stick close together all the time and thus sending messages as often as the cooldown would let them, or they would be spread apart and because they would follow the same pattern they would never run into each other. Because we wanted to avoid odd effects like this, and perhaps other unknown ones, we decided on using our random walk strategy.

Although they were not part of the hypotheses we still extracted some data from their performance which were used when deciding on what strategy to use for the other tests. The random walk strategy worked surprisingly well in most cases and this was also another reason we kept this as the main strategy. Best strategy over all when we summarized the values turned out to be the spiral strategy, which performed good on all three maps, however we found the random properties of the random walk strategy more appealing for the reasons stated earlier. Looking further into the good performance of the random walk we found that other people have also gotten good results from using a random method for navigation [MG04]. We believe that a reason for our random walk strategy to perform so well is that it is only random when it is out in the open, while when a wall or obstacle gets in the way the underlying behaviours take over to make the bot avoid running into it. The gap finder also performed well until we tested it in tight corridors where it seems like it ran into some problems, looking at the graph in Fig. 19 for Map2. On this map the strategies that follow walls had the clear upper hand because of the long corridor-like pattern that dominate that map, which would also hint at why spiral came at a third place on that map. The spiral did however not perform as well as the follow wall strategies on that map and that could be because instead of continuing to follow the walls it would start to spiral a corridor, and once the corridor is fully explored it could have trouble finding where it left the unexplored spaces.

5.2 Methodology

What we experienced when we started out was that finding articles and reports that touched first-person shooter games was very hard and a lot of the information that was available on the internet had not been published work which
meant that it had not been subject to proper reviewing so we had to look at it with a sceptical attitude. One of the hardest parts of our work was designing tests to get good and consistent sets of data that could be used for analysis, as well as how to do the analysis on this data. The implementation we did took a lot of time because it consisted of so many puzzle pieces that needed to work together, so that we could get a good test-bed to work with.

In retrospect we believe that it would have been better to use the QASE API as foundation to build the system on, because this would probably have saved us a lot of time and effort. It implements a lot of functionality that can be tedious to implement such as instead of tracing single lines as was available through Q2BotCore this API also has methods for tracing spheres and rectangles of different sizes.

5.2.1 Mapping technique

We have used a simple graph solutions which is a common practice for navigation and moving around. The bot stores the current position as a node and creates arcs between the nodes when it walks, which will give a form of a web. The arcs can be given an cost to make it possible to search in different terrains. If for example we have a hill it will be slower to pass and then the cost is higher, then running on a flat ground around the hill. This is however not used since we only use 2D maps. A graph is also easily searchable when we want to find a path from one place to another, and there are many algorithms out there that handles this very well. There are a few disadvantages when using waypoints, for example the handling of moving obstacles and changes to the environment can become difficult to handle.

There are other ways to classify knowledge about an environment, for example Voronoi diagrams and topological maps [Thr98], but we decided that waypoints was a good way to do our mapping. This was because we found it to be easy to perform pathfinding on and it also gives a nice overview of the map when visualising the knowledge for analysis.

5.2.2 Strategies

By using maps with different characteristics for the training, the strategies could give slightly different results. The GapFinder strategy turned out to be the best overall strategy, but because the layout of Map2, with mainly corridor-like properties, is quite different from the other ones. Our random walk performed surprisingly well on all tests and so we finally decided to use this strategy to generate our training data for Hypothesis 1 and 2 to isolate the results from any possible strategy specific phenomenon which could add more dimensions to the results and therefore make them harder to analyze and comprehend.
6 Conclusions

Our test results show that the effect of adding more bots really did decrease for each added bot, and that at a certain point it would be pointless to add more bots because it will only make an insignificant improvement on the output. At what place one should stop adding bots would be highly dependent of the environment and the available resources, but the formula we defined in 1 could work as a guide for this. The parameters we extracted from our test results will vary from map to map, but from what we experienced when testing and training was that they seemed to work as a reasonable prognosis for the result.

The communication rate was very difficult to analyse because the expected results did not match with what we got from our training runs. When we visualized the data in diagrams it was hard to see a consistent pattern even if we added markings for the standard deviation and compared different functions for trend lines. In Hypothesis 1 we ran tests to see what happened when we just added bots, but here we added a penalty for communication to see how this would impact the result. The final result here was not the continued improvement when increasing the communication rate, but instead something that seems to be more of a natural variation near the end but not really peaking at full communication rate. To confirm or disprove this hypothesis we need to gather a lot more data for analysis and with the timeframe available to us this was not feasible to do at this time. We do however believe that this is a good stepping stone for future work.

By running performance tests on the bots at different amounts of map coverage, with the knowledge taken from previously run training sessions we found that the higher the coverage was, the more targets were found by the bot and the time for doing a complete test-run decreased.

7 Future Work

Many of the ideas we used for exploration were taken from real world robots and thus it would be natural to try and test this system on real robots as well. This could be for a example a rescue-robot that is to find a person in a area, which brings up a several new areas of future work; replanning and object recognition to name a few. If for a example an explosion occurred, and obstacles has gotten in the way of a known path, we could no longer rely on our known information and would have to update our knowledge to replan the current task.

7.1 Co-Fields

If we combine our exploration algorithms with the Co-Fields covered in [MZ04] new interesting options may evolve, such as having the bots run in a formation when exploring. This could allow for more efficient and accurate exploration of maps and would open up possibilities such as allowing individual bots break
away from the formation to explore rooms the group pass by, only to later rejoin
the leader and share its findings with the rest.

7.2 Area recognition
To detect what kind of area the current bot is in, a strategy form for the area
could speed up the exploring. By using neural network [Bis96] it would be possibility to detection the current area and by using state machine choose the right exploring strategy.

If we for example simulate a hall, the feelers in front and back of the bot
would have a longer range then the ones pointing to the side of the bot. By the
conclusion by the neural network that it is a hall we are in. State machine point
out a strategy that is it faster to not explore for details and instead run to the
end of the hall to explore further away.

7.3 Genetic programming for strategies
By mixing strategies through the use of genetic programming [WBL98, Koz94] it
would be possible to find a good combination of strategies. Also with the added
functionality of area recognition the bot could dynamically switch strategies
dependent on how the current area looks. With genetic programming it would
also be possible to optimize what strategy would be best suited for each scenario.

7.4 Fighters
In this thesis the bots were just made as explorers, but by adding fighting in
to their behaviour they could be set up to work in teams against other teams
which could provide very interesting effects and results. To do this we would
think that switching from Q2BotCore, which is quite low-level and not initially
object oriented, to using QASE API we would get a better range of tools at our
disposal for allowing the bots to hunt for equipment and chase players.
Bibliography


All URL’s listed in this thesis have been validated on 2006-01-26