Imperial College of Science,
Technology and Medicine
(University of London)
Department of Computing

Multi-Agent Based Simulation

by

Samer Arandi

Submitted in partial fulfillment
of the requirements for the MSc
Degree in Advanced Computing of the
University of London and for the
Diploma of Imperial College of
Science, Technology and Medicine.

September 2006
Abstract

Agent based simulation is a -relatively new- and promising field, it tries to overcome the limitations of the previous traditional approaches by introducing a new set of tools and techniques. One of the new approaches categorized under this field is Game-based simulation. First, we give a basic definition of the Agent-based simulation approach in the context of its relation to the multi-agents and simulation fields, a quick review of modeling approaches, their advantages and disadvantages and some of the terms and expressions used -is also given- as an introduction to illustrate where the multi-agent simulation stands, and what it can offer. Then an in-depth look into the Game-based approach is given, examining how this approach emerged and why, its relation to the agent-paradigm, the nature of games used and their architecture.

Then, we go further by building a simulation platform implementing the Game-based approach, documenting all the necessary steps and knowledge needed to master and build such a platform, explicitly articulating the advantages and disadvantages of this emerging simulation technique, then we put our platform into a practical use achieving the second goal of this project, evaluating several different coverage and exploration strategies, after providing the necessary background on this topic, reviewing a large number of coverage algorithms and strategies, and explaining the strategies we have decided to use, and -finally- analyzing the simulation results in order to evaluate these strategies, not forgetting to -also- evaluate our platform (and the whole Game-based approach in due) at the end.
Acknowledgements

It is a pleasure to acknowledge the invaluable guidance, help and support of my supervisor Professor Keith Clark, whom his advice, insightful discussions and continual suggestions kept me on the right track during this project.

I would also like to thank Hani Qaddumi Scholarship Foundation, namely Dr. Nabil Qaddumi and Ms. Maha Awad for giving me the valuable chance of studying at Imperial College.

On a more personal note, I would like to thank my parents, whom their love and care made me who I am now, and whom nothing I would say or do till the end of my life could reward them just.
# Table of Contents

Abstract ................................................................................................................................................. i  
Acknowledgments ................................................................................................................................. ii

1 Introduction ........................................................................................................................................... 1

1.1 Motivation ......................................................................................................................................... 1  
1.2 Objectives of the thesis ..................................................................................................................... 1  
1.3 Related Work ................................................................................................................................... 2  
1.4 Structure of the Report ..................................................................................................................... 3

2 Background .......................................................................................................................................... 4

2.1 Multi-Agent Based Simulation ........................................................................................................ 5  
2.1.1 Definition ...................................................................................................................................... 5  
2.1.2 Modeling Approaches ................................................................................................................... 6  
2.1.3 Modeling Complex Systems ......................................................................................................... 8  
2.1.4 Modeling problems and traps ....................................................................................................... 10  
2.1.5 Advantages of Multi-Agent Simulation ......................................................................................... 10  
2.1.6 Multi-Agents: Schools of Thoughts .............................................................................................. 12  
2.1.7 Simulation Software (Testbeds) ................................................................................................... 13  
2.1.8 Categorizing Generalized Simulation Software ......................................................................... 16  
2.1.9 Simulation Software Examples .................................................................................................... 18  
2.1.10 Games as a Simulation Software ............................................................................................... 25  
2.1.11 QASI : Quake 2 Agent Simulation Environment ...................................................................... 29

2.2 Coverage Planning ............................................................................................................................ 32  
2.2.1 Introduction .................................................................................................................................. 32  
2.2.2 A priori information ..................................................................................................................... 33  
2.2.3 Cellular Decomposition .............................................................................................................. 33  
2.2.4 Single Robot Algorithms ............................................................................................................ 34  
2.2.5 Multi-robots techniques ............................................................................................................. 38

3. The Simulation Platform ...................................................................................................................... 46  
3.1 Introduction ....................................................................................................................................... 46  
3.2 The Visualization .............................................................................................................................. 46  
3.2.1 Quake2 Level Bsp Files .............................................................................................................. 46  
3.2.2 The Level Editor – GtkRadiant .................................................................................................... 49  
3.2.3 Simulator Platform Visualization ............................................................................................... 50  
3.3 Platform Architecture and Features ............................................................................................... 51  
3.3.1 Introduction .................................................................................................................................. 51  
3.3.2 The Probe Agent ......................................................................................................................... 51  
3.3.3 Probe Limited Visibility Problem ............................................................................................... 52  
3.3.4 KQML Communication Layer ................................................................................................... 53  
3.3.5 Agent Management ...................................................................................................................... 54
4 Simulation Setup ........................................................................................................... 55
  4.1 Introduction ............................................................................................................. 55
    4.1.1 Agent’s Environment Representation ............................................................. 55
    4.1.2 Exploration Percentage Calculation ............................................................... 65
  4.2 Exploration Strategies ............................................................................................. 58
    4.2.1 Individual Strategies ......................................................................................... 58
    4.2.2 Collective Strategies ......................................................................................... 64
  4.3 Simulation Environments (Levels) .......................................................................... 69

5 Results & Discussion ..................................................................................................... 70
  5.1 Introduction ............................................................................................................. 70
  5.2 The Random Strategy ............................................................................................. 70
  5.3 The Agent-Repel Exploration Strategy .................................................................. 72
  5.4 Pheromone-Based Exploration .............................................................................. 76
  5.5 Frontier-Based Exploration Strategy ..................................................................... 79
  5.6 Overall Strategies Comparison ............................................................................. 85
  5.7 Platform & Game-based Approach Evaluation ..................................................... 87

6 Conclusion .................................................................................................................. 91

7 Future Work ................................................................................................................ 93

8 Bibliography ................................................................................................................ 94
1 Introduction

1.1 Motivation

Simulation is one of the most effective tools of modern science, it aims –using a multitude of techniques- to bring a complex and hard to understand system under control by creating an -as faithful as possible- model of this system in order to study it thoroughly, enabling, disabling, modifying certain parts or properties of the system and analyzing the results under a safe and controlled environment without affecting the original system.

Many techniques are available to support simulations, but the multi-agent paradigm with the elegance in design and abstraction it can offer, and the ease by which it can model complex systems, makes it an excellent candidate for being the method of choice for realizing simulations on computers. And among the diverse techniques available in the multi-agents field that can be used for this, a new and promising approach has emerged recently on the scene, which is the game-based approach that is offering new opportunities and facilities by taking advantage of the technology the huge gaming industry is providing. We believe that this approach has not took its fair share of usage and involvement, because most people in the research world lack the knowledge in the gaming world needed to implement or use such new approach in their research. This is where this project comes trying to make in effort in this direction.

1.2 Objectives of the thesis

Building on what we have said in the previous section, this project tries to accomplish the following objectives:

- Providing all the necessary background and inner knowledge needed to take advantage of the game-based approach in simulation.
- Building a simulation platform using this approach and documenting every step and illustrating clearly all the underlying concepts and the required skills for developing or using such a platform.
- Using the simulation platform we have built, in a practical real world problem, which is, evaluating a number of coverage and exploration strategies, part of them are random-based and the rest are complete, and studying the effect of a number of parameters -relating these strategies- on the coverage process.
- Evaluating our simulator and the whole game-based approach, based on the experience we have gained.
1.3 Related Work

One of the pioneers in the game-based approach in AI research in general is John Laird, he has been working for quite sometime in this field, promoting the idea of using games for AI related research, he has also written a QuakeBot[15] tool based on the Quake2 game engine, he used a different technique than the one we are going to use, he interfaced the Quake2 game dll (dynamic link library) with a proxy program which in turn connects using a socket layer to a Soar engine running on another machine, this Soar engine is another AI related project he is involved in and forms the backbone to his system, because this system is more AI related, all the knowledge of the Bot was encoded as rules in the Soar engine. This system was used first for simulating military pilots in large-scale distributed flight simulations, then it was used for implementation of more intelligent or “anticipatory” bots for the Quake2 game (the meaning of the word bot will be discussed in detail later, but for now one can think of it as a game character).

Another simulation platform that implements the game-based approach is GameBots[18], this platform is based on the Unreal Tournament game, which is a game very similar to the Quake2 game, this platform take a slightly different approach for interfacing with the Unreal Tournament game, it uses the Unreal script language (C++ like script) for implementation of the bots. This platform provides very useful features for supporting multi-agent based research and simulation like visualization and data collection tools, they have built a world of Wizards as a replacement of the actual violent environment of the Unreal Tournament (guns, killing and gore!).

The nearest work to ours was the one in [32], in this work, a simulation platform was built based on the Quake2 game engine, but it was not based on the QASI SDK like ours, rather, it was built using another lower level library called Q2BotCore and was implemented in C#. This platform didn’t include a comprehensive visualization tools nor a communication layer like ours, because the main purpose of their project was to evaluate some exploration techniques, so they built the platform merely as a tool to use for their simulations and it was not meant as a generic platform, nor did their project (and more specifically their report) explain any details about the implementation, because this was not the main concern of their project. Furthermore, although the simulation runs using their platform were involving a similar problem like ours, namely exploration (which is the most suitable problem to study using a platform based on this type of game we both adopted), a number of differences exist, both on the scope and objectives of the project (since ours was to build a comprehensive simulation platform as generic as possible providing all the needed knowledge and implementation details) and on the range of strategies we have studied and evaluated in the simulations, their project mostly dealt with random-based strategies of reactive nature, including moving randomly, following the walls and moving in a spiral manner and so on, the strategies we have studied in our project is far more sophisticated than these strategies. Finally, they have used a completely different approach for environment representation and exploration measuring than the ones we have used.
1.4 Structure of the Report

The remainder of this report is organized as follows:

- The second (Background) chapter highlights the key concepts and principles needed to understand the two main areas this project is involved in, Multi-agent Based Simulation and Area Coverage and Exploration, this is reflected in the structure of this chapter where the first part is concerned with Multi-agents and Simulation, introducing this concept and explaining the related terms and expressions and illustrating the current approaches and tools in this field. The second part is related to the coverage and exploration, a definition is given followed by an extensive review of the existing algorithms and strategies used for coverage, explaining with a degree of depth these algorithms and how they work.

- In the third chapter we present our simulation platform, illustrating its features and details of the implementation. A brief introduction to some of the technical issues needed for understanding how our platform -as a game-based simulation platform- works, is also given.

- The forth chapter discusses the simulation setup illustrating -in detail- the exploration strategies we are going to evaluate, and explaining the methods we will use to measure the exploration and evaluate these strategies.

- In the fifth chapter the results of the simulation runs are presented and analyzed in relation to the task of evaluating the different coverage and exploration strategies and the effect of different parameters on the coverage procedure. After that a detailed evaluation of the Game-based simulation approach and our platform is presented.

- The sixth chapter has the conclusion and the seventh has the future work.
2 Background

This chapter provides the reader with the key concepts and principles needed to understand and become familiar with the topics & issues discussed in this project.

This chapter is divided into two main sections, the first one serves as a comprehensive introduction to the field of Multi-Agent-Based Simulation, presenting definitions and basic concepts, illustrating and evaluating the available modeling approaches and simulation software tools and finally and -most importantly- discussing the Game-based simulation approach which our platform simulation will be based upon.

The second section defines the concept of area coverage and the terms and expressions involving this field, after that a review of a number of coverage algorithms using a multitude of techniques to achieve coverage are presented, classified within a certain framework that is also explained.
2.1 Multi-Agent Based Simulation

This section is comprised of three parts, the first one (sub-sections 1 to 6) serves as an introduction to the area of Multi-agents Based Simulation, definition is given, followed by a quick review of modeling approaches, illustrating the pros/cons of each approach, trying to present the reader with the basic terms and concepts involved in this field in order to effectively understand any later discussion. The second part of this section (sub-sections 7 to 9) deals with agent-based simulation software, we try to define such software and capture the trends and techniques currently employed, a brief examination of many simulation software tools (representing each trend) is also presented.

The third -and final- part (sub-sections 10 and 11) is an in-depth look into a relatively new approach in simulation: Game-based simulation. We examine how this approach emerged and why, its relation to the agent-paradigm, the nature of games used and their architecture.

Finally, we examine QASE as an example of this type of software and the different facilities and tools it offers.

2.1.1 Definition

One way of understanding and characterizing Multi-Agent Based Simulation (MABS) is as the intersection between two main paradigms in computer science: Multi-Agents and Computer Simulation [1].

Multi-Agents is the area in computer science dealing with issues of design, implementation and coordination of agents which are computer systems (programs) that have special characteristics that vary from one implementation to another but include being: autonomous, reactive, proactive, has an internal state and social (interact with other agents in its environment) in addition to other properties.

Computer Simulation is the area concerned with the study of different techniques for simulating phenomena on a computer, using a multitude of techniques like: discrete events, object-oriented, and equation-based simulation. The phenomenon simulated is an event or a sequence of events in a natural or an artificial system (or a combination of both) which could be either existing or non-existing at the time of the simulation [1]. A natural question that comes to one’s mind is why? Why doing Simulation? To answer this we must notice that the process of simulation is all about building a model that is as valid as possible, and by validity we mean that it matches – at least – those properties of the target, we are interested of, and by constructing such models we hope to satisfy a multitude of goals, as illustrated in [2] : to get more understanding and insight into the target, -put another way – the model have an exploratory goal. Another goal for the model could be to predict the behavior of the target under certain circumstances that might or might not exist (hypothetical) due to difficulty or cost of inducing these circumstances in reality, for example we could build a model of a hospital and see how it will behave in the presence of an epidemic.

A third goal for building the model could be as a part of the design process which is what usually called a “prototype” for a certain unit or stage of the system built for purpose of testing.
Before going into details about Multi-agent based simulation it will be useful to see first what other paradigms and approaches are available as tools of simulation and to contrast them with the Multi-Agent Based approach.

### 2.1.2 Modeling Approaches

Simulation and modeling usually has many tools and approaches but most of them can be categorized into two main areas:

- Numerical/Analytical approaches
- Symbolic approaches.

- Numerical/Analytical approaches are mathematical by nature, they include tools like differential equations, stochastic statistical and probabilistic distributions. This approach tries to capture the properties of the target system and formulate it into explicit mathematical equations which maps all the elements that constitutes the target system into parameters and all the processes into operators that tries to describe the system's dynamics. Examples of such approaches are the use of Chaos Theory or the formulation of a set of differential equations to describe the system.

- Symbolic approaches: symbolic approaches usually rely on some form of Artificial Intelligence combined with different levels of knowledge and cognition, Multi-Agent systems is a major example of these approaches, another example is formal logic specification of models, which usually expresses a model by a first-order logic language where temporal transitions and implications are expressed as inference rules [2].

The multi-agent simulation approach depends on the notion that we can map the different interacting modules and entities that constitutes the target systems into agents (components) that can be programmed or instructed to mimic each entity they represent inside the artificial environment the agents “lives” in. The combination of the agents and their interaction with other agents and the environment, simulates the target system.

The Multi-Agent approach is considered relatively a new one compared to the other more established approaches and tries to overcome some of the shortcomings of these analytical approaches which were identified by many. Some of these shortcoming are (as noted by [3]) related to the following aspects:

- **Micro to macro relationship.** One must define input and output parameters at the same level. It is then not possible to relate a global parameter such as the population size to a local parameter like the decision process of an individual in a population.

- **Complexity and realism of parameters.** Complexity in models leads to the definition of new parameters whose relation to reality is not obvious. Detailed models usually require complex differential equations including awkward parameters. An example demonstrating this in [4] is that in a prey animal-predator mathematical model a certain coefficient a, which indicates the efficiency a predator can convert a food into offspring, is oversimplification considering the large and complex factor that controls such a relation like ( hierarchies, dominance, territories , etc ..) .

- 6 -
- **Taking actions into account.** Numerical methods do not represent actions, i.e., activities which result in a modification of the world. They only see them by their measurable achievement or in terms of their probability to happen.

- **Multitask behaviors and conditional task switching.** Actions cannot be considered as proceeding from decisions whose outcome depends on some conditions of the world. One can describe a hunting process by an equation that relates the number of preys to the probability for a predator to find one but this equation will not show the numerous strategies used by the predator, though these strategies have a strong influence on its efficiency.

- **Qualitative information.** Numerical simulations cannot cope with qualitative data such as the relation between a stimulus and the behavior of an individual. These relations, though central to ethological models, are far beyond their scope.

All this does not mean that the two approaches are contradictory, but as [4] denotes, they are intended to be used at different levels. Multi-agent models are used at a local level as analogical mappings of a real system. From this description, one can derive global parameters that can be studied and incorporated into a mathematical model (more on this to come next).

Further more, [2] notices that the multi-agent approach (as an example of a symbolic approach) can embed numeric values and –even- update them as the whole simulation structure is updated, which we count -for sure- as an advantage for multi-agents approach since it can augment certain aspect of the other approach in a flexible way.
2.1.3 Modeling Complex Systems

Modeling complex systems that usually have large and complex interactions between entities that compose the system usually is classified to two levels to make it easy to understand and analyze such systems:

- **Micro-level**
- **Macro-level**

The Micro-level deals with the low level interactions between individual entities (agents) -inside the systems – and their behaviors, knowledge, actions, etc...

The Macro-level deals with the overall characteristics of all these entities (agents) and the resultant behavior of the interaction between all the individuals and entities in the systems.

Others like [4] went further by introducing an intermediate level between the two levels, calling it the group-level which deals with small groups of entities (agents) and differentiation of agents roles and activities within these groups and how these groups in-turn relate to the overall system.

The expressive power of Multi-Agents and its ability to support individuality makes it best suited for working at the micro-level, while the analytical numerical method is best suited for working at the macro-level by virtue of its summarizing and overall averaging mathematical power.

A good phrase to summarize this issue is found in [6] "... agent-based modeling is most appropriate for domains characterized by a high degree of localization and distribution and dominated by discrete decision. Equation-based modeling is most naturally applied to systems that can be modeled centrally, and in which the dynamics are dominated by physical laws rather than information processing."

**Emergence**

One of the most common phenomena one will encounter when dealing with complex systems, whether when designing, implementing or modeling is emergence. Emergence as [7] defines it:

The existence of a coherent pattern that arises out of interactions among agents.

Since entities (agents) in the system don’t usually work alone and they interact with each other, this interaction produces a new and different result, something that is more than the sum of its participants, for example in an ant colony, each ant follows very simple and basic rule, it wanders freely until it finds food, it carries this food back to the colony leaving behind it a pheromone trail, other ants can follows this trail as an indication for a path that leads to food, this simple interaction between the ant and its environment and the ant and other ants in the colony (system) results in the emergent phenomenon of feeding the whole colony, if we examine each ant alone (locally) we will note that its actions are very simple and are not obviously contributing to the feeding and well-being of the colony but if we look at what the ants are doing collectively (globally) we will start to see a coherent pattern or an "emergent” one, its as if the interaction between entities is itself plays a major role in the system.
Bird flocks, traffic jam, stock market and ant colonies are all examples of emergent phenomena.

This emergence idea can be extended further into a hierarchical system where each emergent structure is considered as an entity whose collaboration and interaction with similar entities results in a higher level emergent structure and so on. Mapping all this to our previous discussion of micro/macro levels it becomes obvious that interactions at the micro-level results into the emergent of behaviors and structures at the macro-level, but this change is not in one direction only, since things happening at the macro-level can also influence the micro-level, an excellent example for such a phenomena is found in [7]: "Applause occurs when spectators join in what appears to be spontaneous synchronized clapping. There is no conductor that coordinates this. When everyone starts, the clapping is totally unorganized. Each person’s tempo is wildly out of phase with the next person. Eventually groups of people begin clapping at the same tempo. People in the audience sense the emerging rhythms and adjust their clapping to join it. The emerging applause rhythm grows even stronger and more people conform to it. Eventually, the entire audience is clapping in a synchronized pattern. This entire process can take place in a matter of seconds with even thousands of individuals" .... "In the example of spectator applause, the interaction of humans in the audience produced the dynamics of applause. However, there was another phenomenon occurring in this example: individually, the spectators adjusted their applause rhythm based on the applause that they heard."

![Figure 1: Bi-directional effect of emergence phenomenon](image)

**Top/Down approach**

In the context of our understanding of complex systems in terms of micro/macro and emergence a complex system structure can be understood and described (for design or modeling purposes) as either following:

- **top-down approach**: Where it corresponds to a system designed by an engineer to respond to a certain need or purpose he had in mind while designing the system, and such case we can consider it following a centralized organization.
- **bottom-up approach**: Where it corresponds to many decentralized entities interacting with each other without a coordinator to produce an emergent behavior, and its obvious that this kind of approach is of exploratory nature. It must be - also -
obvious by now that multi-agent systems follows this approach as a way to model and describe systems [4]

2.1.4 Modeling problems and traps

[2] Identifies two traps the modeler could fall in, the first is called the trap of "verisimilitude" which plagues the Multi-Agent (symbolic) approach and happens when the modeler is tempted - by the richness of representations available - to put plausible details not because they are required but just because they are plausible. The dual of this trap in numerical analysis is the trap of "Tractability" which happens when modeler subordinates everything to be able to model the system analytically.

Another problem noted by [4] that will face the modeler and which applies -most obviously- to Agent-Based approach and other symbolic approaches is that because every feature of the system under modeling has to be made explicit (as a preliminary step to turn it into a specific program or agent) it is no longer acceptable to allow certain aspects to remain unclear or even to define a concept in a vague or obscure way, all elements, all components all mechanisms must be thought out in minute detail.

Another Trap noted by [5] for the Multi-Agent approach is that Agent-based models allow a great deal of freedom in terms of model design, analysis and result presentation to a degree that could be abused by modelers into not following a strict an rigorous criteria in the rules that governs the modeling process where each model follows a different procedure, maps the input data and the resulting output data without proper scientific justification and not shows explicitly the implementation details of the simulation so that it could be repeated later, which undermines the scientific validity of the experiment. Where on the other hand, the traditional analytical modeling, relies on a very well established, and rigorous methodologies that adheres to the basic rules of scientific experimentation.

Finally, even if a model is experimentally validated and its assumptions are justified just because a model stays a model, one can always dispute any parallel behavior to be as reality [2].

2.1.5 Advantages of Multi-Agent Simulation

Before delving further into the details of Multi-agent based simulation it is useful to illustrate some of their advantages that justifies embracing such an approach. [4] mentions a few points on this matter:

- In contrast to the classical approaches that only help to define a model and then compare the similarity of its results to reality, Multi-agent approach allows us to participate in the process of modeling by giving us the ability to change the model and test if this change improved the similarity or not and so on, until the model converges to the real system, its as if the model itself evolves dynamically until it forms a good match.

- The multi-agent approach also allows the integration of all other approaches like differential equation and quantitative variable in the same model.
- Easy to introduce changes by only introducing a new type of agent carrying out a new model of behavior and interaction without the need to change the whole system.

- It can cross the Macro/Micro barrier by controlling the Macro level behavior with action occurring at the Micro-level and vice versa as it shows in the clapping example illustrated before.
2.1.6 Multi-Agents: Schools of Thoughts

In the field of Agents there is a distinguishing between two types or paradigms of agents, which are Cognitive and Reactive Agents. Cognitive agents are agents that has an explicit representation of their environment and maybe of other agents in that environment and uses this to reason about its situation and what its action in the future, further more these agents -usually- have intentions or explicit goals defined inside the agent that drive him on. Because of all previous properties these agents can have plans to achieve their goals and can decide what course of action to take based on its environments, internal state, goals and plans. It obvious that these agents must have a good deal of intelligence and sophistication, and this is why such agents –usually– have a knowledge base that stores all the know how they require to achieve their tasks.

As a consequence of all this, these agents enjoys a good deal of dependency and can achieve their tasks in an autonomous way, nevertheless these agents can interact with each other in a numerous sophisticated models of cooperation, competition and negotiation see [4] and [3].

Reactive agents on the other hand, are agents that do not (necessarily) have a representation of themselves nor their environment (including other agents as well) nor do they possess the ability to plan or reason about their actions, i.e.: they cannot anticipate the future. The actions of these agents are merely reactions to the state of their environment, and this leads us to note that for this agent to achieve a certain goal, it has to be infused inside it as a set of a state (precept) and a corresponding action, this of course relieves the agent from having an internal state or memory component and any other type of sophisticated knowledge bases.

To overcome their simple structure and consequently simple abilities these agents exist in large numbers and they are equipped with the ability to form groups, their interaction and evolution within these groups and the resulting emergent behavior is what gives them their appeal and power that enables them to deal with complex problems in an efficient way competing with the (usually less numerous) cognitive counterparts [4].

The most famous example of such agents at work (in nature) are ants colonies where each ant via very simple reactive rules to certain events (precepts) in the environment (like pheromone trails) will tackle successfully a complex task as maintaining the well being of an entire colony.

One must notice that in such reactive models the focus is not on the individual level of an agent (because agents here do not even have distinctive personality) the focus is rather on the overall group level where one starts to see the collective emergent behavior.

Of course one can always have hybrid agents, for example a reactive agent that possess a certain level of cognition and planning capabilities.

Now after this introduction to Agents and Modeling we can start examining Agent-based Simulation in more detail, starting with Simulation Testbeds.
2.1.7 Simulation Software (Testbeds)

Definition

A Testbed is the Infrastructure inside which the Multi-Agent simulation is conducted, it should basically, contains all the tools needed for implementing the agents, the environment in which the agents will "live" and –possibly- interact and the means for recording the results of the simulation or experiment. As the complexity and the level of simulations gets higher, more advanced requirements will be needed, the Testbed should support other functionalities and properties like the ability to control the simulation in real time – if needed -, the ability to change certain parameters easily and then repeating the simulation, the ability to present a visual view of the simulated world and agents inside it and the ability to present the results in a meaningful – usually graphical - way and maybe applying some statistical analysis to the results, not to mention ease of use and a fast learning curve (especially if the Testbed is to be of general nature).

The importance of a Testbed as an infrastructure in the Multi-Agent Simulation area is of central importance, as [8] argues the adoption of the Multi-agent approach (in a whole) as a sound approach in the scientific research field is dependent (among other factors as well) on the existence of a stable, reliable and easily accessible infrastructures for agents, an analogy is drawn from the case where technologies like Cellular phones and Web browsers came to popular widespread only after the development of such stable and reliable infrastructures to support such emerging technologies, we can extend this argument to denote the importance of Testbeds (as a special case of Multi-Agent Infrastructure) to the Multi-Agent Simulation approach (as a sub-category of Multi-Agent Systems).

Specialized and Generalized Simulation Software

At the start scientists using Multi-Agent Simulation had to write their own software simulation tools to conduct their experiments, the results were Simulation Software built for specific task, although later a more generalized version of the software could be released, but this generalization only meant the ability to define more models of agents or add/remove some properties and rules for an agent in the Simulation software without being able to make the system applicable in other domains or fields without an extensive effort and a good deal of software rewriting. We will demonstrate an example of such Simulation systems before going to the other generalized type.

The EOS Testbed

Our example is the EOS software Testbed [9] and as this is our first example, we will be delving in a quite good detail to get a sense of the notion of multi-agent based simulation.

The problem

The Testbed was put forward to test a theory that tries to explain a certain archaeological phenomenon, namely “the transition from a relatively simple to more complex society during the course of the Upper Paleolithic Period in Southern France and Spain 15-30 thousand year ago” [9] among the theories put forward to try and explain such emergence in complexity was one that attributes this to certain
ecological condition that took place at that period and resulted in population concentration which in turn lead to social complexity in order to organize the activities of such large concentration which is necessary for the population survival.

The ecological conditions were:
- a strong pattern of concentration of diverse and abundant food resources both at particular locations and particular annual times.
- a relatively high degree of stability and predictability of spatial distribution of such resources.

**The Experiment**

The Testbed was designed to facilitate experiments involving the variation in the behavior of agents and the possible dynamics of resources in the environment, to see if a complex hierarchy of the population of agents will emerge in the environment. Needless to say that the testbed consisted of an environment, agents and resources.

The environment was a two-dimensional square grid with a typical 10,000 by 10,000 units, agents and resources were spatially located in this environment and their location was simply a pair of coordinates.

The resources representation in the testbed supported the experimentation needed to test the theory, i.e.: the resources location could be a cluster or a single instance, each type of resource provided certain level of energy for the collecting agent, there was also resources only attainable by group effort or "complex resources", also some resources were available for only for a certain interval of time in certain time periods.

Agent's goal is to stay alive by consuming resources which increase their internal energy and consequently keep them from dying, because after each time unit elapses, this internal energy level is decremented by one. Agents were designed with a production system architecture where each agent has a set of rules and a memory of dynamic facts changing overtime.

Communication between agents was achieved via the environment taking into consideration sensory range and spatial considerations.

Besides Agents realization of resources, they were equipped with a social model of concepts like leader and group which is a necessary part of the simulation to see the effect of certain parameters on the development and complexity level of this social model.
Although the experimenter could control the simulation via setting different parameters like initial location of agents, speed of movement, sensory range and list of skills but the most important factor is the set of rules that governs the agent actions and behaviors.

The software followed an Object-oriented style and used Prolog language. A graphical interface provided a dynamic view of the environment showing both agents and resources, the ability to interact with the simulation in real-time is also provided with simple measurement facilities.

When describing this Simulation Software it was inevitable to describe the simulation itself, because the simulation testbed cannot be separated from the simulation itself which is why this kind of testbed is a specialized one.

One thing to note about this simulation is the cognitive nature of the agents model, which is a common theme for simulations of societies whose members have a degree of anthropomorphism which is the case here since this is a simulation in social archaeology. Note also that they used Prolog language for development of this software where the rules and facts of the production system were all expressed in prolog. Which is a suitable choice because of the cognitive nature of the agents and consequently the need for AI reasoning and techniques. For complete details of this simulation and its results see [9].

The second type of Simulation Software is the generalized one, where the simulation framework can be separated (variably) from the simulation or experiment itself.
Such Simulation software emergence was a natural process, since almost always after the success of a certain specialized simulation software dealing with a certain problem, the writer(s) of such software (usually) soon think of generalizing it, and making it applicable in other cases to a wider scientific audience, and this generalization varies form trying to make the software applicable to other experiments within the same field to making it applicable to many/any experiment in many/any fields which requires quite an extensive effort.

Another reason for building such generalized simulation software is that usually a scientist in need to conduct a simulation in the field of biology for example are not supposed to know programming, and even if he does, it is a fair assumption that he is not as good a programmer as a scientist, and even if this was the case it will be a waste of his time to build simulation software while he could invest this time in doing what he is supposed to be doing in his field of expertise, add to that the notion that science does not usually favor re-inventing the wheel time after time !

All these factors lead to the emergence of generalized simulation software whether as a result of generalizing a special case or writing a general tool from scratch as we will see next.

### 2.1.8 Categorizing Generalized Simulation Software

It is always difficult to categorize Software in general into clear and distinct groups and the same applies to simulation software, each tool provides a set of functionalities and facilities that support simulation in one way or another that could be similar to what another tool offers but is more specialized or generalized, some of them shares the same functionalities but different in the approach, some uses the same approach but provide different implementations, but one could always see a general trend that is followed by one tool or the other, we will try to capture a few of these trends in simulation software providing examples of simulation software embracing each trend and giving a brief overview of at least one of those software tools later.

The reader should not conceive the following types as strict distinct categories since other genres might exist combining properties of one or more of these categories.

#### General Simulation Software (Frameworks)

These software usually provide a general domain-independent set of libraries supporting all the functionality needed for simulation, from defining the model of agents, environment, works flow of the simulation, monitoring the simulation, collecting data and presenting the results.

In most cases these Tools are implemented in an object oriented language which provides constructs like inheritance and encapsulation necessary to support such approach in design where the user of such libraries is supposed to use the predefined models and modify them to “fill” them with content and map them to his domain. As a result such Frameworks are very flexible and ductile but this comes at the cost of putting a big burden on the user as he has to implement most of the details of his model using the programming language of the framework.

Examples of such Software is Swarm[16] (more on swarm later) which is the most famous and widely used one, other Frameworks include Repast[37], Ascape[38] and Mason[39] all of which have been influenced by Swarm, each one of those Framework has its own unique way of providing functionalities to the users adding,
removing and improving what the others offer but all of them follow the same
generalized trend of providing domain-specific tools that are mapped by users later.

**Extensions to the General Simulation Software**

These software are extension of the more general frameworks in the previous
category, and they are extended either for providing more specialized support for
simulation in a certain domain, or for providing a higher level layer that offers an
easier interface for using the services of the more underlying complex Framework,
hiding much of the details and shortening the time needed to program and start the
simulation.

Extension following the first approach could range from simply providing a
framework of models dedicated for a particular field, for example providing a
complete set of classes implementing all aspects of modeling economics, to a
complete module that adds a certain functionality to the engine of the underlying
framework, for example the NeuroLib module that adds the basic functionality of
Neural Networks services to Swarm agents or Kenge[40] extension that adds
discrete2d surfaces with Cellular Automata to Swarm.

An example of an extension following the second approach is MAML[41] which
provides a macro-like language that hides some of the details of setting up the
models, GUI and environment in Swarm ( more on MAML later ).

**Visually programmed Simulation Software**

This type of Simulation Software takes a visual approach for simulation ie: there is
minimal or no algorithm code writing, instead the modeler will choose from ready
made blocks that represents models of agents, environment, actions and behaviors
of agents, conditions and graphical representations, and all that he does is to modify
and set the parameters and characteristics of all these components and their
interactions, then uses them to compose the system he want to model and starts the
simulation. The purpose of such type of software is to allow non programmers to
design and implement their ideas and models quickly and easily.

Of course this is achieved at the expense of loosing some flexibility and functionality
( as we will see in Agentsheet later ). It is no wonder that such software tools are
used mainly for educational purposes and for introducing students to the notion of
Agent-based simulations, but this doesn't mean it is not good for other more –
serious -purposes, if the user understands the capabilities of such tools well, he can
build fairly complex models easily. Examples of Such Software Tools include
Agentsheets[42] and StarLogo[43].

**Modeling/Model Description Languages**

These are not full fledged simulation tools and frameworks, these are languages to
describe models that comes – usually – with a framework that conducts simulation
that executes these languages, the idea is the same as in previous category; not to
burden the modeler with lots of programming and implementation details and rather
provide a descriptive language for defining the models involved in the simulation.
Most of these modeling languages are usually domain dependent but nevertheless
some of them can be used for more than a field. Still the modeler has to learn these
modeling languages in order to use these simulation tools effectively. Examples of
such tools are SDML[44] ( more on this later ).
2.1.9 Simulation Software Examples

Now we will try to give a brief look at one simulation software from each of the above categories.

Swarm

Swarm[16] is a multi-agent simulation platform for simulating complex systems that was developed at the Santa Fe Institute. It provides a set of ready-made libraries of object oriented reusable code the user of Swarm can use or extend/modify to meet his needs, this reusable code provides facilities for modeling agents and environments, observing results of simulations, displaying results, controlling the simulation in real time.

Although the basic unit of simulation is an agent, agents are organized in an interacting group called "Swarm" (hence the name) and included in the swarm (in addition to the agents) a schedule of events over those agents. Interestingly the swarm itself can be an agent (a complex agent) whose behavior is the emergent behavior of all the agents it is composed of, and this swarm agent can also be one among many others that form a higher level swarm and so on, this hierarchical system that support simulation of multi-level scale is one of the main advantages of swarm.
Everything in swarm is an agent, even the environment itself is just an agent that other agents interacts with, even observing the simulation has to be done by an observing agent that has all the capabilities of reporting and monitoring the simulation, which is a smart approach for separating the model itself from the observation and data collection tools so that for example changing the observation method later does not interfere with the model itself. The swarm libraries provide the following functionality for users:

- creating models of agents, both simple and complex "swarm" agents.
- creating a schedule of events to control the execution of the simulation and provide the agents with a time frame.
- creating GUI interfaces, analyzing data collected, plotting the data graphically.
- general utility libraries to ease managing agents objects, generating random numbers, etc..
To use swarm properly the user has to learn to program in Objective-C or Java, but this comes with the relieving fact that swarm has a very large community and lots of code and tutorials are out there to help the user, not to mention the live Swarm Development Group that holds a yearly conference called SwarmFest where users of swarm meet and share their knowledge and expertise in the field of multi-agent based simulation. Swarm also provides a set of libraries supporting certain domain specific models like simulations in 2d space for example, lots of other such libraries are also available as user and community contributions.

**MAML: Multi-Agent Modeling Language**

MAML[41] was developed as major landmark on the road for providing a comprehensive Infrastructure for simulation and modeling in social and human sciences fields for scientist and student who -usually- have no previous background or knowledge in programming, users should be able to write models, add GUI and executing their models and get their results easily.

![MAML simulation screenshot](image)

**Figure 4:** a screen shot from a MAML simulation (notice the menu, plotting and property windows are all Swarm components).

MAML is actually (till now) just a layer working on top of Swarm, where it provides a macro-like language that hides some of the details of setting up the models, GUI and environment, but still the actual code of the agents and their behavior is still written in Objective-C which is the language used by Swarm, and actually the compiler of MAML compiles these Macro-like directives into swarm code (Objective-C) that gets compiled by gcc.

MAML also helps by simplifying creation of objects, arrays and arrays of objects and predefine some automatically maintained variables for easy access and usage. MAML is still in the making and the current release is still an experimental. In our opinion MAML (till now) could be seen as just another extension for swarm working as a layer of macros to ease the process of using swarm, but seems that according to the goals of the MAML project they are looking for more than this in the future.
Agentsheets

As the name implies, Agentsheets[42] is a simulation environment that uses a spreadsheet approach for dealing with agents, where each cell in the spreadsheet can be occupied by an agent. Agentsheets follows a very high level visual programming approach in every aspect of the software. The environment is two dimensional grid, the size of each cell can be determined by the modeler, agents will live on the grid and can be manipulated by clicking and editing even in real-time. Agents are created in a wizard like approach where an agent is given its symbolic representation (name) its graphical representation icon or graph or animation in addition to any audio effects accompanying it and finally and most importantly its Behavior. Agentsheets adopts a fully visual approach in dealing with behavior i.e: there is no algorithm code writing, its all drag-and-drop style, thus -as with most graphical approaches to programming - agents programs is event-driven where the agent defines a method to executed when an event of interest is triggered, since text coding does not exist, the methods only contains a set of rules that test certain conditions on the form of “if-then” statements, the statements are only selection from a list of possible actions (like changing a variable, color or playing a sound ..etc) any other actions can be added only via extensions written in LISP.
Agents are allowed to have internal values (internal state) and can manipulate their internal state. Agents can send messages to each other but these messages can only contain static values, i.e., an agent cannot send the value of his changing internal state to another agent, although an agent is allowed to examine another agent's state. Agents have bounded movements and sensory range; they can only see agents in adjacent cells and can move only one cell at a time into adjacent cells, but still, agents have the capability of broadcasting a message to all or sub-part of the population. This means that agents are mainly locally focused and only by this local interaction a more global effect will emerge which underlines the paradigm Agentsheets is adopting in design, which is relying on the emergence phenomenon to achieve complex global behavior from simple local interactions. Besides the rich visual user interface for presenting construction tools, agents and the environment visualization, Agentsheets offers a rather simple graphing technique.
for the resulting data of the simulation, via a Plot functionality where you choose a certain variable and you plot its history thorough out the simulation, you can also export your data to excel.

Agentsheets comes with a very helpful tool, Ristretto with which the simulation and all tools for controlling the parameters of the simulation is ported to a java applet that can be started in a browser. Ristretto can also render simulation properties into java beans that can be combined with other JavaBeans into complete applications with complex interfaces.

A new feature of Agentsheets simulations enables it to interact via (Http and Tcp/Ip) and Read information live from webpages [12].

Although the first use that comes to ones mind when dealing with this software for the first time is educational purposes, which is a field Agentsheets has succeeded in because of the intuitive interface and easy visual approach that enables non-programmer into developing their own models effectively and quickly, an approach that is also very suitable for students who are drawn to the visually and audibly rich software, and without much imagination one can see the wide applications for such tool in education and schools, and already a number of the educational institutions are using this tool (as obvious from the Agentsheets website education section), Agentsheets is still a strong candidate in any more – research oriented – field just like any other simulation package that looks less intuitive! Being a simple, easy to use tool does not mean necessarily its weak, its rather other factors like the limited actions and events allowed and the restricted messaging and movement pattern and limited result presentation facilities that makes this tool not applicable in all fields, but nevertheless a lot of systems can be simulated using this tool and one can see from the number of simulation examples at this software website that this tool can be used in quite a large range of areas and fields.

**SDML: Strictly Declarative Modeling Language**

SDML[44] is not a complete testbed or infrastructure (according to our definition), its rather a modeling language where knowledge is represented as rules and databases in a declarative logic style.

The idea behind SDML is ”To have an expressive language in which to model the computational, learning and decision-making processes of agents. Credible internal states, which represent the structure of knowledge in relation to its environment, can be specified and modified as the agent develops” [10], designer of this language argue that logical formalism is more appropriate for modeling qualitative decisions and makes the unfolding of processes more apparent.

SDML is implemented in Smalltalk and follows an object oriented paradigm where object oriented facilities like inheritance is supported and timing facilities is also provided.

It is obvious that agents in SDML are capable of acquiring cognitive abilities, are able to reason about themselves (their internal state) and their environment, agents behavior is governed by the rules assigned to them.

SDML supports a hierarchical view of agents where complex agents can be composed of simpler ones. The simulation procedure usually starts by a wizard like series of windows, where the experimenter writes using the SDML language the rules governing the agents and the environment, and the simulation start by firing those
rules in a forward or backward chaining style where the results of executing these rules affect the agents internal states.

**Figure 6:** A screen shot of a selected number of wizard windows defining and testing a model in SDML.

This language is oriented more towards business and economical modeling and has the down side of requiring a considerable time to learn. For more information see [10].
2.1.10 Games as a Simulation Software

Another new category of Simulation Software that we rather to examine in much more detail, is Game Based Simulation Software.

The relation between computer games world and Artificial Intelligence (and consequently Agents) is an old and established one, if one examines any recent (or old) game, he will find AI techniques used everywhere within the game and in all genres, take for example the action game genre where the player is playing within a hostile environment, where he faces opponents trying to stop him achieving his goal and maybe other game characters trying to help him, both the opponents and the allies exhibit forms of human intelligence, each of them have a plan that requires interacting with other peers, coordinating their efforts, navigating the environment, finding their enemies and maybe anticipating their actions, all these activities need highly advance AI techniques that need to be implemented into the game characters inside the game engine.

With time and as games became more complex these techniques became so much sophisticated that it has to be separated and put into special module inside the game engine like any other specialized modules responsible for graphical rendering, sound effects, input processing, network management, etc. to allow individual development of each of those modules separately. And as an unwritten rule in the gaming world, as soon as a game becomes successful the game development team usually releases tools for modifying the current game and building new levels, new monsters and adding new weapons, further more some of the tools will allow the users to re-define certain parameters of the game characters i.e. their intelligence, and if that is not enough usually after the technology behind a game becomes somewhat old (for example with the release of the next version) the whole game source code could become available for free download, and anyone will have the complete source code of a full working game. People started making there own flavor of the game, adding new graphics, new levels, new sound effects and most importantly (the brave ones) started writing their own AI routine for the game characters.

This draw the attention of researchers in AI of the possibilities available here which was facilitated by the fact that AI related features were nicely modularized in separate component away from all other AI irrelevant pieces of code that has to do with other aspects of a game, and games started to be seen as an ideal platform for AI research, and as Liard [14] summarizes the benefits of such platform: it will relieve the researchers from the overhead of understanding the Environment since every thing is ready for them, rendering the complete environment, tools to build levels, design character, all code necessary for steering agents, collision detection, network protocol, it will also cut less on money (a game is like 50 dollar). And since gaming industry is a big industry (bigger than movie industry!) you will always be sure of available support from both developers and community. And to the joy of Multi-Agent based paradigm it is trivial to note that the nature of a game character cannot find a better match than that of an agent, it is autonomous, interacting with other characters (agents) and the environment, it could have cognitive nature (complex character like a monster) or reactive one (simple pickup item) not to mention that the nature of the problems faced in such game platforms like coordination between character, goal and path planning, enemy searching, team management, resemble problems under research in Multi-Agent based field, which makes these kind of platforms a candidate for becoming a multi-agent based platforms and consequently simulation ones.
Before getting too excited we must notice that such simulation platforms are not suitable for all kinds of multi-agent based simulations, because such platforms are dependent on the nature of the game itself, so they will be suitable for simulating environment that can be mapped into the environment of the game.

We will be discussing platforms based on action games and more specifically first person shooter games (games taking place in a 3d world where the player sees the world through the eyes of the character he plays with) for many reasons, first because currently these are the games that are most popular and have the tools necessary for modification and has open source available and community support, also the nature of these games with their intelligent autonomous characters roaming around in 3d environment can be mapped very easily to many real research problems in the field of multi-agents like coordination, resource allocation, path finding and planning.

**FPS Game Architecture**

One of the modes supported by a First Person Shooter game (or FPS for short) is called multiplayer mode, in this mode of playing, game players connect to the game server (run on a dedicated machine or just on the machine of one of the players) and each one of them controls a character inside the game and they start playing as either teams against each other or every man for himself! These players are nothing but clients connecting to a server over the network, when the game starts the server sends to each client its initial position in the environment (the level map) in addition to other information like their initial energy level, position of any nearby resources or other team players, using this information each client (because they know the level they are playing in) can draw a view of the level according to his position and any other team member or resources in sight, then as the game goes on the server keeps feeding the clients sensory information about the environment periodically - usually 10 times per second - and according to these sensory information the client can either move, jump or shoot, these player actions are basically sent through the network to the server to apply it to the environment and make it effective so that all other clients can see the results of these action. Of course the sensory information that is sent from the server to the client is received by the client part of the game and converted into visual information so that the players can see it and interact with it.

![figure 7: the architecture of a multi-player fps game over the network.](image)
Thus the game’s architecture boils down to the following:

- A Server running on a certain host holding the environment information including level topology players position, resources amount and positions, etc. The server also has all the facilities to simulate the physical laws governing the environment, for example it must be able to inform a client that issued a move command and who happened to stand in front a wall that he cannot move forward. The server periodically sends updates on the environment state to the client and receives their actions and (if valid according the rules of the environment) applies them.

- A Client who receives the server periodic update of the state of the environment (sensory information) and based on this information and his internal policy (player decisions) will issue actions that is sent to the server to affect the environment.

(One can notice how easy it is to map this kind of architecture to the Multi-Agent Architecture were an agent is viewed as being in an environment and has a list of percepts and actions).

As we now have a good introductory idea on the way these games are designed and built, we can examine how these games can been turned into agent simulation frameworks.

**Game-based Simulation Software Architecture**

If someone puts a probe on the network while a game is being played, he will only see a sequence of periodic messages going from the server to each client (perceptions) and a sequence of messages going from each active client to the server (actions). Now the idea of how to take advantage of such mechanism is simple, if we write a client software that implements the same network protocol the game uses and thus can receive the perceptory information and understands them and consequently replies back the actions in the same protocol to the server, then we have got our self –utilizing this very simple idea- a complete, comprehensive agent simulation framework! Notice that our client software will replace the client side of the game, and to the game server we are still like any other “human” client/player who receives percepts and responds with actions. Now we can implement the actions of the character we control the way we want, we can implement it in a cognitive way, connecting it with a database of rules, providing it with the ability to have multiple plans that is evaluated to choose the best one, or making it follow a simple reactive model.
We can also follow a competitive approach where every client competes with other clients (who could be human or other instances of our client implementation connecting to the server as a separate character) or a cooperative approach where each of our client instances cooperate and interact to achieve their goal. Notice that all this can be accomplished without the need to change even one line of code in the game, but we could also do that for many reasons, we could do that to change the rules of the environments, for example to have an environment where the server allows the clients to walk through walls or to reduce the effect of gravity, another reason could be to change the nature of the game entirely, by changing the environment from one that has to do with fighting and shooting into other one that constitutes a simple system of carts going around through a maze, and luckily there are available lots of tools and documentation around to help you do all that.

An example of such Game-based Simulation Software are SoarBot [15], QASI [17](which we will examine next) that is based on the famous Quake2 Game and GameBots[18] that is based on Unreal Tournament Game.

We will be calling the client program that controls the character (or the character itself) inside these kind of games a “Bot” as it is commonly referred to in the gaming community.
2.1.11 QASI : Quake 2 Agent Simulation Environment

This framework is based on Quake2 [46] Game by idsoftware, QASI is a java-based API which means it follows the object-oriented approach and is portable. It is oriented towards providing researchers and students with all functionality needed for experimenting with cognitive agents.

![figure 9: the typical environment inside a Quake2 game level.]

QASI works in exactly the same way described in the Game-based software simulation, it takes advantage of the network protocol and the messages exchanged between the client and server of the Quake2 game to implement the agent/environment simulation platform. It does so by providing a set of java classes that can be extended by the user of the API to implement the agent and handles all the other details and specifications of the game platform.

**Network Protocol**

QASE handles all the details and specifications of the client-side network protocol using a Proxy class, which hides all these low level details from the client or Bot, it receives the data (precepts) from the server and converts it into human readable format after parsing and analyzing these information, then it passes it as an object to the upper layers of the client code and at the same time sends the agent's subsequent actions to the server. The object that is passed to the client is called the World Object and contains all the information about the state of the environment and all the other entities.

**Bot Hierarchy**

Like most Agent-based frameworks QASE provides a set of classes a user can use or extend to implement an agent. QASI provides a hierarchy of agent classes forming different levels of functionality from the simple Bot class which all other agent classes extend and doesn't provide any functionality and so the programmer is responsible for everything, to BasicBot class that provides most of the basic functionalities like setting agents direction and speed, obtaining world information (precepts) and sending agent action to the server, the programmer only needs to override a routine called runAI which forms the brain of the bot and contains the
actual agent code implementation. Other more functionally rich classes exist like ObserverBot, PollingBot and MatLabBot classes.

**MatLab integration**

One of the important facilities provided by the QASI framework for analyzing data and processing it on the fly and consequently providing the ability to take decision based on the result of such analysis is by integrating QASI with MatLab. This can be done in two ways: either by providing a complete Matlab code template that implements the Agent in Matlab code and so is executed in Matlab or (for higher efficiency) via the class MatLabBot that provide the functionality of calling the Matlab library (from within the java client program) passing to it the data that needs processing and then receiving the result back and continue execution.

![Matlab/QASI integration](image)

**DM2 Parser and Recorder**

The Quake2 game supports a feature called demo recording, in which the game play of a complete gaming session is stored on files called dm2 files, these files is nothing but a recording of the network traffic of all messages exchanged between all clients and the server, QASI provides a class that can parse this dm2 file and treat it like a virtual server connecting to it and retrieving what the server have sent as precepts of the environment, this class also provides the ability to record such files from the client interaction with the server which could be very useful functionality for providing a comprehensive log of what happened during a simulation session and thus ( through the parser ) a reconstruction of the exact simulation could be done.

**BSP parser**

QASI also provides the client with the ability of obtaining extra information about his environment through a class called BSPParser, this class parses the map file ( which is the file describing the geometry of the level in which all the agents/bots are moving ) and thus provides the agent the capability of querying the geometry of this level, queries include: the possibility of a collision to happen in a certain direction, the distance to the nearest collision point, checking if a certain entity is visible from the current location of the bot, if the bot can fit to walk into a narrow opening or not and so on ( all these queries are supported by the ability to cast line, box and spherical traces through the geometry and getting the result back ).
Utility classes

A set of utility classes are also provided to facilitate any AI research like neural network, genetic algorithm generator and KMeans calculator.

We will be evaluating the QASI sdk in depth at the end of the fifth chapter.
2.2 Coverage Planning

2.2.1 Introduction

Motion planning (where the path of a moving entity in its environment is determined according to a certain criteria) is one of the most important aspects in a number of research areas like mobile robotics, automated manufacturing and vehicle control systems.

One of the first and main challenges for Motion Planning was the Point-Point motion planning, where a path is sought between two points, this path must be the shortest, the one with least turns, the safest (in presence of hazards) or any other property depending on the problem in hand.

Later new challenges were introduced into the motion planning field by a number of new applications like de-mining (of land mines!), grass mowing, floor cleaning, harvesting and similar applications. Such problems need Coverage Planning, where the mobile entity (or its sensor) sweeps along every point (if possible) in the environment.

An authorative study[19] by Choset, tries to define a taxonomy for Coverage Planning algorithms, which -although- doesn't cover all the algorithms (partly because it is 5 years old now), it provides a good framework for classifying different algorithms and understanding them.

According to this framework we can divide coverage algorithms into either:

- Heuristic and Random
- Complete

- Heuristic and Random algorithms imply that the entity or robot does not plan his path ahead, instead it just selects its next movement or direction randomly, this of course is coupled sometimes with some heuristics like avoiding obstacles, moving along or away from walls or other robots. These algorithms rely on purely reactive actions that need no planning and -consequently- need less information which translates into less processing, less hardware/software capabilities (like position sensors, communication devices and GPS systems) but it also -unavoidably - no guarantee on full coverage of the environment.

- Complete Coverage Algorithms work in away that the mobile entity visits every point in the environment (if possible) so that it achieves full coverage of the area, not surprisingly such algorithms require more information and more processing which translates to more sophisticated hardware and software and consequently more expensive robots. Although in terms of quality and guarantee, Complete Coverage algorithms surpass Heuristic and Random, but there could be certain cases where an increase in the quantity of the less expensive robots implementing the Random approach can compensate for the loss in quality and guarantee expected.
2.2.2 A priori information

Another way to classify algorithms is based on whether the planning algorithm can use previous information about the target environment or not, this further more divides algorithms as either off-line (need a priori information to work) or on-line (no a priori knowledge is needed).

2.2.3 Cellular Decomposition

Branding an algorithm as complete or not requires a method to measure its completeness, this can be done by decomposing the space of the environment into cells of certain size that is usually equal or multiple of the size of the robot or its sensors coverage area, and thus the robots movement in the environment can be seen as visiting these cells, and the more the number of cells the robot visits, the higher the coverage percentage. When all the cells in the environment have been visited we consider the coverage complete.

Chooses divides algorithms based on decomposition criteria into three slightly different types:

- **Approximate Cellular Decomposition**: Cells have same size and shape but their union approximates the free space.

- **Semi-Approximate Cellular Decomposition**: In this type of decomposition the cells have the same size but the floor and the ceiling can have any shape.

- **Exact Cellular Decomposition**: In this type of decomposition the union of the disjoint cells is exactly equal to the whole environment.

This classification applies to algorithms that work for both individual robots and multi-robots, next we will demonstrate chosen algorithms for both single and multi-robots classified based on the cellular decomposition technique they use.
2.2.4 Single Robot Algorithms

Approximate Cellular Decomposition: We will illustrate two full coverage algorithms adopting this decomposition criteria:

- **Wavefront algorithm:**

  This algorithm was devised by Zelinsky and others in [20] and is based on the idea of distance transform, where the planner in this algorithm propagates a distance wavefront through all the free space grid cells in the environment from the goal cell (which means in Full Coverage algorithms the cell we want the robot to end in after finishing its trip). The distance wavefront flows around obstacles and eventually through all the free space in the environment. To achieve full coverage the robot follows the path of the steepest ascent, in other words, the robot moves away from the goal keeping track of the cells it has visited. The robot only moves into a grid cell which is closer to the goal if it has visited all the neighboring cells which lie further away from the goal. The following figure illustrates the distance wave transform generated from start to goal cells and the corresponding path followed by the robot.

  ![Distance Wave Transform and Full Coverage Path](image)

  Note that this algorithm was originally meant to achieve start-goal path planning and later was modified to allow full coverage. One of the main advantages of this algorithm is that it can integrate other types of information to be taken in consideration when planning the coverage path to produce coverage paths that have minimum number of turns or that keeps certain distance from obstacles and so on.

- **Spanning Tree Algorithm (STC) [21]:**

  In this algorithm the environment cells are the same size as the robot coverage area (which is assumed to be square) and form a graph of connected cells, the robot is supposed to follow the path that spans this graph so that it passes over each cell precisely once. This algorithm has three versions, the first is an off-line version (i.e. has some *a priori* information about the environment), the second version is an on-line (uses no *a priori* information) but needs O(N) memory location (N: number of cells) and the third (ant-like) is also an on-line version that uses pheromone trails in the environment and only needs O(1) memory locations. In terms of execution time all of the STC version take O(N) to calculate the optimal coverage path.
The off-line version:

The input to this version is a geometrical description of the environment (the *a priori* information) converted into 2D-size grid of cells, the cells form a graph, and finally a certain starting cell. The algorithm works in two basic steps:

- Pre-processing: beginning from the starting cell S a spanning tree for the graph is constructed using any algorithm like DFS (depth first search) for example and during this process each 2D-sized cell is divided into four identical sub-cells of size D (D: is the size of robot coverage area).

- Covering Action: starting from cell S (starting cell) we move between neighboring sub-cells along a path which circumnavigates the spanning-tree along counterclockwise direction and only stop when we reach the starting sub-cell again. see figure 12 below.

The on-line version:

In this version the robot has no *a priori* information about the environment and it has to use its sensory capabilities to detect obstacles and plan its coverage path accordingly.

This version assumes that the robot -at any time- knows its current position and orientation. The robot incrementally constructs a spanning tree for the grid representing the environment, during the spanning tree construction the robot subdivides every cell it encounters into four identical sub-cells of size D, the robot follows a sub-cell path which circumnavigates the incrementally constructs the spanning tree, until the entire environment is covered.

The whole procedure is highly recursive and works as follows:
(we begin from starting cell S)
- Mark the current cell as an old cell (visited).
- For each neighboring obstacle-free and not visited cell, we construct a spanning edge from the current cell to this target cell.
- We move to this target cell and repeat the procedure

This keeps going on until we are back to the starting cell after we have marked all the cells as visited.

Basically, the robot is not doing a very different procedure than the off-line version, but here it is running the DFS algorithm (recursive part) during the spanning tree construction and moving counterclockwise to ensure that it will incrementally circumnavigates the spanning tree. Note also that because we need to mark cells as visited or not we need to have an O(N) storage capacity for this algorithm to work.

The Ant-Like Version:

In this algorithm the same capabilities of the previous version is needed, in addition to the ability to leave markers in the environment (counter part for pheromone in the ant world) and the ability to detect them, note that markers could be color, odor, heat or physical objects, so instead of marking the cell as visited or not in the robot's memory a marker is placed in the environment cell, this will relieve the robot from
the O(N) memory requirement needed in the previous version. Although this algorithm uses a recursive procedure where a DFS is incrementally constructing a spanning-tree of the environment in a counterclockwise manner, it modifies some of the previous version steps, mainly (since there is no graph here) the graph edge needed to determine the next cell to inspect and visit, an alternative method is used for this, namely by identifying the transition between marked and unmarked cells in the environment as the robot proceeds on.

![Diagram](image)

**Figure 12** off-line version of the STC algorithm (a) represents the environment divided as a graph of grid cells (b) the spanning tree of the grid graph (c) the coverage path calculated by the algorithm.
Semi-Approximate Cellular Decomposition:
An example of a full coverage algorithm using this method is:

- **ZigZag Algorithm**

This algorithm was devised by Lumensky and others in [22]. It is an on-line algorithms and of a recursive nature. In this algorithm the robot starts at an arbitrary point in the environment and zigzags along parallel straight lines to cover the whole environment, in fairly complex environments that contain convex cavities (called inlets) the algorithm follows a more complex procedure to cover these inlets, the robot is required to keep track of the points of entry and exit to each inlet to insure that each inlet is only covered once (if possible), each inlet in turn could contains smaller inlets that will be covered in the same manner (hence the recursive nature of the algorithm).

![Figure 13](image)

*(a) The robot starts at point $S_0$ and simply zigzags along the parallel lines to cover the entire area (b) In a more complex environment that contains inlets ($I_1, I_2, I_3$) a more complex procedure is adopted.*

Exact Cellular Decomposition:

This type of decomposition was devised to handle environments that needed to be decomposed into cells and contain obstacles. When a line segment (termed slice) is swept across the environment (in the process of decomposing it into cells) wherever there is a change in the connectivity of the slice (because of the presence of an obstacle) a new cell is formed, and with each increase in the connectivity new cells are spawned, on the other hand, whenever the connectivity decreases cells are merged back into one, as shown in the figure 14.

Using this type of decomposition many algorithms were introduced for full coverage, some are off-line (need to know the position of the obstacles) others are off-line. Most of these algorithms (after the decomposition) use graph searching algorithms to traverse the adjacency graph of each cell in the environment to achieve full coverage.
Figure 14 (a) spawn case, as the slice moves from left to right, its connectivity changes from one to two. (b) merge case, as the slice moves left to right, its connectivity changes from two to one. (c) resultant Boustrophedon decomposition

An example of an algorithm using this technique is Choset’s Boustrophedon decomposition Algorithm.

2.2.5 Multi-robots techniques

Incorporating multiple robots in coverage planning is a natural next step toward better and more efficient coverage. Increasing the number of robot participating in the coverage not only means shorter time to finish this operation, but it also means more robust coverage operation, where if a small number of the participating robots were stalled, stuck or totally malfunctioned the rest of the team will continue the mission, something –definitely- cannot be done in single-robot scenarios.

For this to become true, some sort of coordination mechanism must be present between the robot members of the team, this coordination has been the subject of many research and took many different forms, mainly centralized, de-centralized and semi-centralized approaches. To give a deeper insight into this area we will illustrate some of the coverage algorithms using the multi-robot approach, classified according to the type of the decomposition they utilize.

Approximate Decomposition Algorithms:

One will notice that most of the multi-robot algorithms try to have a decentralized (or at least semi-centralized) approach to coordination, this trend is becoming evident more and more with the introduction of the new distributed technology and pervasive computing. Another thing to notice is that most of the algorithms tries to mimic the communication/coordination mechanisms found in nature and namely between creatures like ants, these creatures drew the attention because they achieve fairly complex tasks and coordinate efficiently via very simple -nearly reactive- decentralized techniques, which means that if these techniques were implemented correctly it will only require very simple (and consequently cheap) robots and can achieve complex tasks and need no centralized coordination.

One of the main communication/coordination mechanisms used by ants and applied in robotics in general and multi-robot coverage algorithms in specific is pheromone based communication, where robots will leave traces (messages) in the environment for other robots to find and receive, as the basic method of coordination. These traces could be heat (by heating the ground under the robot if permitted), odor,
color (leaving colored marks in the environment) or even by physical objects that has certain different properties (color or shape).

Wagner et al have produced several versions of multi-robot coverage utilizing the pheromones mechanism for various communication and coordination tasks. In [23] Wagner used multi-robots for the task of cleaning the floor of a building using traces of chemical odor for communication and coordination, where each robot was provided with odor marking and detection capability, including the ability to evaluate the strength of odor at their current position. This algorithms treats the environments as a graph of vertices, each vertex is considered a tile and is equal to the coverage area of the robot, the edges in this graph is the boundary between two neighboring tiles. The algorithms has three versions:

- **Ant-Walk1**: can be implemented by robots without individual memory component, instead they leave odor traces in the environment, a robot visiting a vertex checks the odor strength on all edges emanating from the vertex, in the direction from inside to outside. Then it goes to an edge that has the smallest trace on it, that is— the edge that was not visited for the longest time. This works because odor fades out slowly with time, so by “smelling” two edges one can say which one of them was visited before the other, by implementing this simple strategy Wagner proved that the algorithm guarantees that two edges emanating from the same vertex will not differ too much (i.e., by more than one) in their respective number of visits, or in other words: the flow from a vertex is fairly distributed among its neighbors, this simple coordination scheme between robots guarantees full and fairly equally distributed coverage of the area. See figure 15.

![Figure 15](image.png)

**Figure 15** (a) part of the environment divided into square tiles. The cleaning robots are shown with their directional smell traces. Note that the traces degrade with time. (b) state of Four tiles after they have been traversed by robot, the corresponding directed graph with the smell-labels is shown on the left. Note that there are two labels on each edge of the graph, to designate the trace intensity in each direction of the edge.
-Ant-Walk2:
This version eliminates some of the unnecessary redundant work done by the first version (due to visiting certain vertices more than once) on the expense of more sophisticated hardware, namely memory. This is adopted by following a multi-level DFS algorithm where a DFS is constructed but on multi-levels and after each search level is done, the robot backtracks to the previous one -if possible- until the end. This means that the robot needs to remember which search level it is in, since it only considers a vertex not visited if it was marked in the current level of search only, hence the need for the extra memory components compared to the previous version, however this will result in a better performance.

-Vertex-Ant-Walk: This method, is similar to the first one, but assumes that smell traces are laid on the vertices rather than the edges.

-Dirt Trails: Wagner et al in [24], employs the idea of pheromones in a new way, for a more specialized application of the coverage area which is cleaning the dirt off a building floor. They take advantage of the dirt itself as the medium for communication between robots, i.e. instead of leaving pheromone trails on the floor the dirt itself is used as a trace (by cleaning it or not which is equivalent to dropping a pheromone trace or not). Cleaning the dirt on the tiles has to be done in a special manner that preserves the connectivity across all tiles in the region until no other non-critical tiles (tiles that connect different dirt sub-graphs to the rest) exists.

Figure 16 dirty, critical and non-critical tiles
- **MAC Algorithm (Mark and Cover):**

In [25] a method for covering a continuous unknown planar space by group of robots leaving marks (pheromones) behind them is presented, it is an on-line algorithm, that is also robust against addition/removal of robots, more over the robot tool here is a circle.

The group of robots have no mean of knowing their location or communicating directly, their only mean is short lived trails on the ground that means a location has been visited before. The algorithm is a very simple set of reactive rules followed by the group of robots, as follows:

- The robot marks its current location (Z1)
- The robot finds a point within its sensing range that has not been marked yet and moves to it, as it arrives there it marks the new location (Z2), note that the mark intensity at Z2 is stronger than on Z1 (because of the passage of time), this will be used to indicates the direction the robot traveled.
- If no unmarked point was found within sensing range then the robot backtracks to its previous point following the previously marked path Z1-->Z2 in reverse order.
- This is repeated until no more unmarked points are found nor backtracking is possible. See figure 17.

![Figure 17](image-url) Mac Algorithm after seven steps, starting at Z1.
Note that this algorithm needs no further modification nor extra hardware to work for multiple robots because marked lines of one robot never intersects neither itself nor other robots lines (because all robots treat all marked points the same regardless of which one has left it). See Figure 18.

Figure 18 (a) an early stage of the MAC algorithm (b) Final stage of MAC notice that each of the four paths of the four robot doesn't overlap

But still, the initial position of the robots and the geometry of the environment control the balance of work distribution on robots which in turn affects the degree of speed up gained by adding more robots.

- **PC (Probabilistic Coverage):**

The authors of the MAC algorithm acknowledged that it cannot be considered the ultimate solution for the coverage problem because of the expected noise and failures in the sensors and effectors, the expected change in the environment during operation and the short life-span of the marks which limits the amount of area that can be covered in a single shot, all this motivated a second algorithm of a random nature they called Probabilistic Coverage or PC for short. Because this algorithm is random it doesn’t need any sensory input, at each step the robot selects a neighboring point to visit randomly. Although this algorithm doesn’t suffer from errors of measurement and fading markers (because it uses none!) but it is much slower to cover the area (because of the redundant work) and cannot determine when it has finished. So, the authors suggested a third hybrid algorithm called it MAC-PC that combines both algorithms in away such that the robot follows the random PC procedure( select a new point at random each step) but the radius of coverage is extended, this is done using the markers to cover a circle of such extended radius (they have to introduce the markers because the PC cannot keep track of whether it finished covering a certain radius or not without the help of the markers), put another way, the algorithm applies MAC locally in an extended coverage radius and when it finishes, it chooses the next neighboring area to work on randomly. This resulted into a much better performance than the PC algorithm (but still less than MAC) and avoided the problems that resulted from errors of measurement.
- **MSTC (Multi-Robot Spanning Tree Coverage)**[27]:

This algorithm is a recent extension to the single robot STC algorithm (see 2.2.3 STC), it is an off-line algorithm. The authors of this algorithm designed it with robustness and efficiency in mind, they discovered that - counter-intuitively - non-backtracking (making sure that each point of the working area is covered only once) and efficiency are independent, i.e. non-backtracking algorithms could be less efficient than backtracking ones. They introduced two version of their MSTC algorithm, one is non-backtracking and the other allows backtracking, as follows:

**- Non-Backtracking MSTC:**

Since the algorithm is off-line, each robot is assumed to have a complete map of the working-area, its boundaries and all the obstacles. Each robot is square of size $D$, the area is divided into square cells of size $4D$ (discarding cells that are partially covered by obstacles). The algorithm works in two stages:

- a spanning path is built in the same way as in the single robot STC algorithm (see 2.2.3 STC for a detailed description). This is done exactly the same way at the same time in each robot of the team or can be done by one robot and then communicated to the rest of the team

- each robot then uses its own copy of the STC path and its initial position to traverse its sub-path that is allocated to it.

To guarantee robustness each robot keeps sending I am alive messages and if after a period of time this message is not received, the robot is assumed to have failed and the robot behinds it will continue to cover the failed robot sub-path when it finishes its own sub-path (remember that the robots are topologically moving in a circle along the STC path), which means that as long as there is at least one robot left that hasn't failed, full coverage will be achieved.

In terms of efficiency, it is highly dependent on the initial position of the robots, the best case is when the robots initial start positions are distributed evenly apart along the STC part, so that their sub-paths are approximately of equal length, on the other hand, the worse case will happen if all robots initial position are right next to each others on adjacent cells which means that only the first robot will have almost all of the STC path allocated to it (while the others will move one step and finish), put another way, the worse case running time is almost equal to that of a single robot.
**Backtracking MSTC:**

In a later paper [28] the authors have demonstrated that the choice of the initial Spanning Tree has great consequences on the coverage time and that if the tree was constructed appropriately it will reduce the coverage time considerably, based on this they presented a polynomial time tree construction algorithm to achieve such spanning trees.

**Exact Cellular Decomposition:**

As an example of a multi-robot coverage algorithm under this category is Butler et al.’s

**DCR (Distributed Coverage of Rectilinear environment) Algorithm [26]:**

This algorithm is designed for a cooperative team of square robots working in a rectilinear environment, and only requires that the robots have contact sensing (know when they have came across another robot or wall). This algorithm is an extension of a previous algorithm they designed for single robots that was called CCRM, the latter algorithm covers the environment by incrementally building a cellular decomposition of the environment and storing the composition of the environment and its state in a memory component C, the interesting thing is how the DCR algorithm is implemented as extra layers on top of the CCRM, where certain components in the robot (feature handler) has the task of communicating with other robots in the team to determine their relative position and then another component (overseer) integrates the information collected from the other robots into the memory component C holding the state of the environment, and since the CCRM layer only deals with C component the latter communication and coordination are
done transparently without the CCRM knowing (and consequently there is no need to change the way the CCRM is working regardless of the number of robots involved).

Figure 19 an example of adding integrating a new area by the overseer, (a) the current state of C component (b) the collected incoming cell decomposition information (c) the incoming information is re-adjusted to conform with local perception of the environment (d) C component after the integration.
3. The Simulation Platform

3.1 Introduction

Building a game-based simulation platform and evaluating the viability of such a new approach in simulation is one of the main tasks in this project. We believe that this cannot be done properly and objectively unless we go through a real, complete process of building a simulation, running it and extracting the results to experience all the little and important details that a quick look or review cannot reveal. On the other hand, this made us try to create an -as generic as possible- set of tools that not only make it easy to start and configure the simulation but to debug it and see what is happening in real-time i.e. visualizing the simulation environment. In this chapter we try to give the architecture of our simulation platform, its features and details of the implementation, we also try to give the reader a brief introduction to some of the technical issues he needs, to understand how the platform (and similar game-based platforms) work.

(Before reading on, please make sure you have read section 2.1.10, the following sections assume the reader did so)

3.2 The Visualization

The QASE[17] SDK -already- provides a comprehensive set of tools that achieve part of what we need, to start the simulation, the SDK hides away all the gory details of communication between the agent (bot) and the quake2 server(environment) and handles all these details of sending and receiving the perceptions and actions between the agent and the environment. This is done by providing java classes of agent templates implementing all the various functionalities, the user only needs to extend these classes and add his/her own code and data that implements the specific behavior of his/her agent(s), compile the corresponding java classes and start the simulation from the command prompt and will have got himself/herself an agent running in the environment. But in reality a serious simulation needs more than that, most importantly we need a facility to see what is happening every instant in the environment, because a log-file or command prompt output -although useful to plot a graph or extract a trend- is not helpful enough in giving you a clear picture of what is happening in the environment, the latter can only be fulfilled by the ability to see the agents move around and interact in real-time within the whole environment or parts of it.

To understand how and what to visualize we need to go to a lower level than the one the QASE SDK offers, and deal with the Quake2 Game engine itself, we have to understand how a level (the physical environment in the simulation) is built and how it can be visualized.

3.2.1 Quake2 Level Bsp Files

The level is the physical landscape where the agent (or the bot as it is called in Quake2) moves about, climbs ladders, pickup items, shoot an enemy bot or help a friend one, it can be one big room, a complex of rooms, a building of many stories, etc (see the figure)
The geometry of the level is very important to the way the Quake2 engine works, on one hand, this geometry will decide how the player is supposed to move and interact with the environment, for example he can wander around until he stumbles into a wall or cannot exit the space of a room until he finds the door and so on, this means that for each action the player wants to perform, the engine has to check if this action is allowed by the physical geometry of the level where the player is, for example, if the agent wants to move a step forward the engine has to check the level if there is anything that hinders this action (collision detection) like a wall, if none is found the player will move a step forward and his position will be updated in the environment, all this means that the geometry of the level has to be queried by the engine for each player 10 times per second (the refresh rate of the Quake2 Game) which poses a non-trivial performance issue for even the smallest of levels unless a clever technique is deployed (as we will see later).

On the other hand, the geometry of the level has to be drawn so that the players can see the level, the floors to walk, the doors to pass, the items to picks and the enemies to shoot, and not only this has to be in a 3-dimensional high detailed manner, but also, it has to be done 30 times per second -minimum- to create the illusion of continuous movement and to reflect the quick update of the position and status of the items and players in the environment, all this poses an even more serious performance issue on the engine.

The solution to both issues was by adopting a special technique for representing the geometrical information inside the Quake 2 engine, namely a BSP Tree (Binary Space Partitioning Tree) approach. A BSP is a binary tree data structure resulting from
recursively sub-dividing all of the polygons in the level into two lists with respect to a certain plane, all of the resulting polygons in each list is sub-divided further into two lists based on another plane and so on, creating a binary tree until the leaves of the tree are single polygons that cannot be subdivided any further. The union of the

![Recursive division of a 2D polygon by 3 planes](image)

**figure 21:** Recursive division of a 2D polygon by 3 planes [35].

How can this help to speed up drawing (rendering)? The main problem (performance wise) in rendering 3d polygons is depth sorting, i.e. drawing far objects on the screen before near objects so that their overlap is correct, this operation is very expensive especially that it has to be done 30 times per second, BSP Trees provide a very efficient method for doing this by performing a back to front tree traversal on the polygons in the BSP tree. The procedure is simple, we begin at the root node and classify the eye point of the viewer with respect to its partition plane (the plane that was used to partition the list of polygons into two), draw the sub-tree at the far child from the eye, then draw the polygons in this node, then draw the near sub-tree. Then we repeat this procedure recursively for each sub-tree[33]. The result is that we are drawing the polygons in the correct order without the need for sorting them, and since the geometry of the environment is static we only need to create the BSP tree once (for dynamic environments a more complicated method is used).

How can this help to speed up geometry collision detection? When a player is checked against the polygons constituting the geometry of the level, the player is approximated by a bounding box or cylinder and is checked against the BSP tree, at each step it is checked against the partitioning plane of the current node if the bounding box or cylinder is found completely on the left or completely on the right of the plane then it is further checked recursively for the sub-tree on that side, if on the other hand parts of the bounding box or cylinder are found on the left of the plane and parts are on the right, then there is a chance of a collision and it is checked against the polygons that is located on the partitioning plane, if there are any a collision has occurred, otherwise no collision has occurred. Note that at each step of the checking the number of level polygons to check against is reduced by half, i.e. the search is logN Where N is the number of polygons in the leaf of the BSP tree.
All of this (and many other advantages) lead to the adoption of BSP trees as the
technique of choice in Quake2 engine (and many other engines for that matter) and
consequently the level editors (programs that builds the geometry of the level) for
the Quake2 game exports the levels designed using them into .bsp files that the
Quake2 engine can load, query and render easily, and so, in order to design levels
that we can use in our quake2 based simulation platform we have to design them
using these very same quake2 level editors and more importantly if we want to
visualize these levels (the environment) we have to be able to draw (render) .bsp file
inside our platform.

3.2.2 The Level Editor – GtkRadiant [34]

Quake2 engine and all the derivative games based upon it are widely popular and
have a large community of developers and users, which means that there exists a
plethora of related tools dealing with all aspects of the games and the engine, among
which are level editors. Level Editors are architectural-like software where the user
draws the geometry of the level in a 3D environment and determine the locations of
players, items, their numbers and types, in addition to other numerous game-related
details. A large number of level editors that are available for the Quake2 engine were
reviewed and we have chosen GtkRadiant for the following reasons:

- Available under Windows platforms, GNU/Linux, MacOS X.
- Available under GPL licensing (free)
- Written by the same people who wrote the Quake2 Engine
- Has a large community and plenty of help resources

![figure 22: a screen shot of the GtkRadiant while designing one of the levels of the simulation.](image)
Using this level editor the simulation environment is constructed, the geometry is built using Top/Side/3D views for the user to see the building blocks he/she creates, after that textures (images) are applied to the walls, floors, then the location of the players and the items are specified within the level and finally the level is compiled to produce the .bsp file (see previous section), this file will be loaded by the server (for physical simulation of the environment) and by the clients (agents) for visualization.

3.2.3 Simulator Platform Visualization

Due to the standard nature of the bsp file format that is outputted by the level editor, the task of visualizing the environment becomes easier, rather than writing a bsp viewer from scratch, we could use some of the available libraries that make this task easier, this is true because of the large –if not huge- Quake2 community out there, but -still- we needed to find a library that is written in java (for compatibility with the QASE SDK), a task that was very difficult in the past, due to the bad reputation java used to have performance-wise which made most 3D rendering developers prefer to write their applications in other languages like C/C++ or any other compiled language for speed, this began to change since the time Sun introduced the new Java3D API technology which provided a set of object-oriented interfaces that support a simple, high-level programming model -you can use- to build 3D applications that is optimized for maximum speed and efficiency. So after reviewing and testing a number of available java3d rendering SDK’s we found one that is both quick and free from a company called New Dawn Software[36], and after some work we were able to integrate this visualizing SDK to work with our Simulation Platform and using it we were able to load bsp files compiled by the level editor and render it inside the viewing pane of our platform.

![figure23](image)

*figure23:* the viewing pane in the platform showing on of the simulation levels.
3.3 Platform Architecture and Features

3.3.1 Introduction

Because the simulator is built upon the QASE SDK which is built upon the Quake2 engine, the architecture of the simulator will depend heavily on the way Quake2 works. (please refer to section 2.1.10 for a detailed description of the Quake2 engine) Mainly to startup a simulation the Quake2 server application should be started after specifying the bsp file for the level to load and (optionally) the port number for which the server will listen to. Now for each agent we spawn, we have to specify the IP address and the port number of the machine where the server runs, and as soon as the agent (bot) is connected to the server it starts receiving perceptions and sending actions according to each agent implementation. To automate this process and add improved capabilities for debugging, communication, data collection and visualization our simulation platform provides many features, we will try to highlight most of them in the following sections.

3.3.2 The Probe Agent

Because of the distributed way the simulation works where each agent is run in a separate space (that could be different workstations or separate threads in the same application), it is not a trivial task to present the user with a collective view of what is going on in the simulation with all agents at the same time. We found that this could be done by either restricting all the agents to be run as threads within the same application and this way the needed collective view is simple to get because the application can access any thread easily (since they are running in the same space) or the other option (which doesn’t impose the former restriction but is more sophisticated) is using what we called a “probe agent”. A probe agent is a special type of agent that connects to the server and gets into the simulation for only one purpose, to report the status of all the other agents and items in the environment. Based on the probe agent, the simulation platform can provide one of the most important features which is the visualization. The moment the probe agent connects to the server it starts reporting the names and locations of all agents in the environment (in addition to the types an locations of all items), this information is updated 10 times per second (the refresh rate of the Quake2 engine) and is used by platform to represent the position of each agent in 3D space and is overlaid on top of the 3D visualization of the bsp file that represents the current level to form a truly live and dynamic window into what is happening in the environment in real time, a matter that helps to a great extent in debugging, analyzing and understanding the simulations. To take this feature to its maximum potential the platform gives the user the option of recording every point the agents follows along its path for later analysis or to visualize this path in real-time, further more each agent and its path can be given different colors to distinguish between them instantly and to help realize quickly the nature and location of each agent path or area of coverage. (see figure 24).
3.3.3 Probe Limited Visibility Problem

Because the Probe Agent (like any other agent under the QASE SDK) is perceived by the server as a player, all the rules and limitations that apply on players will be enforced upon it, and one of those rules is the radius of perception, which means that each player receives the notification and events for items or other player that fall within a certain radius of this player, outside this radius nothing is reported to the player. This poses no problem inside the Quake2 game, because players cannot make use of events that are far away from their current position, but for our probe agent this is bad! Because regardless of the position of the probe agent, it has to receive events and notifications of the position and status of all other entities in the environment, which means that it has to receive this information even for entities outside its radius of interest. This was a serious obstacle facing the simulator platform, because it is related to a matter specific to the way the Quake2 engine was implemented and to fix it one needs to understand, modify and compile the actual source code of the Quake2 engine, luckily enough the source code of the Quake2 engine has been released for quite some time and the tools to compile and build it were available too, and because we were familiar with the way game engines in general and Quake2 in specific work, we were able (after a non-trivial effort) to locate the related code in Quake2 and modify it to allow the probe agent (after exchanging a specific command with the server) to receive events from all over the environment regardless of the probe’s position or its perception radius.

figure 24: Probe Panel manages visualizing the whole environment including all the agent and their paths.
3.3.4 KQML Communication Layer

One of the most important features a Simulation Platform needs is providing a mechanism for communication between the agents, this is a basic requirement if the simulation platform is to support multi-agent systems. QASE SDK doesn’t provide such a mechanism explicitly, but because the QASE provide access to the basic client/server functionality supported by the Quake2 engine one can always take advantage of the facilities provided by the Quake2 game. The Quake2 engine supports a simple broadcasting messaging system where a player (client) can send a text message to the server which broadcast it to all the clients connected to the server and they can see it (usually this happens when a player want to greet all the other players or when the server wants to notify all the players with a certain event like when a new player is connected or disconnected for example).

The QASE SDK supports storing and reading all the text messages the server broadcasts for later processing, we took this simple text-based messaging system into a more sophisticated level, by building a KQML messaging system on top of it, we have written a small and efficient KQML parser java class that can be used for sending, receiving and processing KQML messages between the agents in our platform, this opens the door for simulating many multi-agent protocols and communication scenarios and adds a great value to our platform and to the diversity of the settings it can handle.

Figure 26: KQML messages dumb inside the simulator.
Due to the nature of the Game Server in Quake2, a limit is imposed on the frequency of posting messages per second by each client, another limit is imposed on each message size, finally a limit is imposed on the maximum allowed capacity of the buffer in each client for storing incoming messages which if exceeded the client will overflow and be disconnected from the server, the first issue can be amended by a special command to the server to increase the allowed frequency, while the second issue can be solved easily by either changing the maximum allowed buffer size for client messages in the Quake2 engine or simply fragmenting large message into smaller ones and third issues can be solved by either increasing the maximum buffer size for all messages in the memory space dedicated for each client in the server code or better ignore the message buffer overflow and simply flush this buffer whenever its full, this will work because the messages are received by all the clients and processed immediately and there is no need to store them in the server, notice that we were able to solve the first and second problems without modifying the Quake2 source code, but the third problem cannot be solved without that, so we had to go again to the Quake2 source code and modify it again and it worked.

3.3.5 Agent Management

The simulation platform provides other features to facilitate initiating and controlling agents, in addition to visualizing each agent perception of the environment, the current percentage of coverage for each agent, the total coverage by all agents (further explanation of these terms in the coming section) and a status window for live display of simulation messages.

![Agent Panel manages adding, removing, controlling agents and visualizing each agent perception of the environment.](image)

*Figure 26:* Agent Panel manages adding, removing, controlling agents and visualizing each agent perception of the environment.
4 Simulation Setup

4.1 Introduction

The problem of area coverage and exploration is one of the most important topics in many research areas that involve applications like de-mining, painting, cleaning and mapping. Further more, the nature of this research problem suits -to a great extent - our simulation platform. The capabilities and facilities that are needed to simulate the environment in which the area coverage problem is usually applicable, matches what our platform is capable of.

In the remaining part of this report we are going to examine and evaluate a number of different exploration strategies under many varying parameters like implicit and explicit cooperation, range of coverage, number of participating agents and other factors. In addition we will be studying how our simulation platform -in particular- and game-based simulation platforms -in general- are really suitable for such a task.

We will start in this chapter by introducing how we are going to represent the agent perception of its environment, how to calculate each agent’s percentage of exploration and finally and most importantly we will demonstrate the different exploration strategies we will study and the different environment (levels) they will be working in.

4.1.1 Agent’s Environment Representation

Agent perception of the environment will be one that best serves the coverage problem. The coverage problem in its simplest form classifies the environment into units that is either “walk-able” (the moving entity was able to visit it), “obstacle’ (the moving entity was able to reach it but couldn’t visit it because of the presence of an obstacle) and finally “unknown” (is not visited yet, for one reason or another). Following this classification the agent will first represent the environment as being completely unknown, as the agent advances in the environment, it starts identifying the units it reaches as either walk-able or obstacle, depending on whether it was able to visit the unit or was obstructed and not able to do so. We have chosen to adopt a bitmap approach to implement this representation rather than other approaches like grids or waypoints (although we have used waypoints for other purposes) for many reasons: memory wise, it is more efficient and less consuming than other methods and needs allocation only once. Strategy wise, one of the strategies we use (namely frontier exploration) depends on analyzing the environment representation to extract certain points called frontiers (more on this topic in the coming sections), this procedure is basically an image processing task which works naturally with the bitmap-based representation. Visually wise, because the representation is already in a bitmap format, it will be straight forward to visualize each agent representation in the simulation platform.

Thus, each different type of the units in the environment will have a different color to distinguish it from the rest, unknown units will be gray, walk-able (space) units will be black and obstacles will be represented by the green color (see figure 27). The procedure for categorizing the environment units is simple, when the agent starts all the units of the environment will be gray (unexplored) then, each simulation tick, whatever the current unit the agent exists in, is colored black
(because the agent is able to visit it) and whenever the agent is trying to move in a
certain direction and it cannot (for a number of simulation ticks) a green line is
drawn in the intended direction of movement to signify the existence of an obstacle
in that position. Note that the environment size of the agent is a 2048x2048 units in
the Quake2 system (which is quite large!) and is represented by (1024x1024) pixel
bitmap.

4.1.2 Exploration Percentage Calculation

Since our simulation will be involving area coverage, one of the most important
things we need to know at any instant during the simulation is the current
percentage of exploration (percentage of -so far- covered space out of the total
possible coverable space) achieved by agents both individually and collectively, this
is necessary because the exploration percentage is the prime factor we will be
evaluating each strategy upon.

The exploration percentage calculation depends on the method we have adopted to
represent the environment, for example if the method for representing the
environment was by adding waypoints every certain distance, then the exploration
percentage calculation would be by dividing the number of the added waypoints so
far by the maximum number of waypoint that can be possibly added in the
environment (which has to be calculated somehow off-line for each different
environment setting). Since we have adopted a bitmap approach to the environment
representation our exploration percentage calculation will have to adopt a compatible
approach as follows, for each different environment (level) we design in the level
editor (see previous sections) we create a bitmap of a top-view of this level, we color
every pixel of the bitmap according to our representation color convention, -that is-
we color every walk-able point in black, every obstacle point (e.g. Wall edges) in
green and every inexplorable points (points inside walls, outside the level where no
agent can reach) in gray, the result will be a template bitmap representing how the
environment would have been represented inside an agent that have covered it
completely. Now the rest is straight forward, at the simulation startup, the bitmap
template corresponding to the current environment (level) is loaded, the maximum
possible exploration is calculated by dividing the number of black pixels by the total
number of pixels in the bitmap, the number obtained will be stored, at any time later
in the simulation when the simulator needs to calculate the current exploration
percentage of any agent two steps are needed, the first step is to divide the number
of black pixels by the total number of pixels in the agent’s environment
representation bitmap, then we divide the resulting number by the number we
stored in the beginning, the result will be the current exploration percentage of the
agent, this method will work regardless of the difference in dimensions of the
template bitmap relative to each agent’s environment representation bitmap.
It is needless to say that each agent doesn’t know its current percentage exploration because it does not possess the bitmap template of its environment, because the assumption of all our simulations is that the agents are exploring unknown environments, rather it is the simulation that needs to calculate this information for later data collection, analysis and evaluation of strategies.
4.2 Exploration Strategies

The second main goal of this project is to study how the different strategies adopted by the agents affect the coverage process. We are going to study two types of strategies, individual ones (where the agent works alone without regard to other agents working around him) and collective (cooperative) ones where some sort of cooperation (or awareness at minimum) between the agents exist, whether it was explicit or implicit. Furthermore, the effect of increasing the number of participating agents and the effect of the nature of the different environments is also studied for each strategy.

Please note that all the strategies we will be looking at are on-line (please see section 2.2 for more detail) in the sense they assume no prior information about the environment. And also we assume in all of these strategies that the agent knows its current position in the environment (using some sort of GPS system), and in some strategies a limited communication capabilities between the agents is also assumed.

4.2.1 Individual Strategies

The agent in these exploration strategies is not aware of other agents (if they exist) working in the same environment. We will demonstrate two strategies in this category one is random and the other involves a sophisticated procedure.

The Random Strategy

The Random strategy is the simplest of the strategies we have examined, in which the agent will wander around in the environment changing direction every certain amount of time (40 simulation ticks). Note that this strategy doesn't require any sort of intelligence or computation, also the agent cannot know when it has finished its task (covered all the accessible space).

Figure 28: agent following the random strategy after 280 simulation ticks (changed direction 7 times)
Frontier-Based Exploration Strategy (non-cooperative)

Frontier-based exploration technique was introduced in [29]. This exploration strategy introduces the notion of frontiers, regions on the boundary between already explored (accessible) space and unexplored regions of the environment. The main idea behind this strategy as the author explains: it tries to answer the following question “Given what you know about the world, where should you move to gain as much new information as possible?” the answer is frontiers, so the central concept is that: “To gain the most new information about the world, move to the boundary between open space and uncharted territory that is Frontiers”. This way you are guaranteed that as the agent progress in its exploration it is going to keep learning new information about its environment and the explored region will continue to expand, something the random strategy -for example- cannot guarantee since the agent could end up visiting the same places it has visited before. Because of this guarantee, this strategy can be considered Complete (see section 2.2) in the sense that in a finite time and assuming perfect position information the agent will cover and explore all the region of its environment accessible from it initial position. This algorithm assumes the agent knows its current position in the environment (using a GPS-like system).

![Figure 29: (a) a partially explored environment bitmap representation (b) detected frontiers cells colored in white.](image)
The original technique uses evidence-grids for environment representation, which are cartesian grids containing cells, and each cell stores the probability that the corresponding region in space is occupied, note that our bitmap representation of the environment is basically a form of evidence grids if we considered each pixel (that signifies a certain points in the environment) a cell, and the color value or that pixel is another form of representing the probability of occupancy, that is, if the color is black it is similar to a high probability of the cell being not occupied and if that color is green then it is occupied and so on. In the original frontier-based procedure the author used probability because he used laser-based and sonar-based sensors for obstacle detection and this introduced a significant margin of error, so the probabilistic representation was needed to cope with this situation, In our case we didn’t find that probabilistic representation necessary because we are not using any type of sensor to identify obstacles, instead we are using the fact that if the agent has reached a certain point then it means it is not occupied, and because we are assuming an error-free GPS-like system for identifying each agent position.

**Frontier-Based Exploration Implementation**

Implementing this strategy was not a trivial task, it involves a set of sub-tasks that need to be implemented to realize this technique, first we have to find a way to extract frontier cells, then we need to navigate from our current position to the frontier cell position following the shortest (if possible) obstacle-free path.

**Extracting frontiers**

The first problem of extracting the frontier cells is basically an image processing task (taking into account the bitmap-based representation of the environment) in which we search for pixels that exist on the border of space/unknown regions, that is, we search for black pixels that have immediate gray neighboring pixels, after that we start collecting these frontier pixels to form contiguous lumps of cells where each lump contain frontier cells that are continuously connected, each lump is called a frontier, only frontiers that have enough frontier cells are reported back to the agent. Because the agent’s environment representation bitmap is relatively large (1024x1024) the search for frontier cells is first conducted within a smaller rectangular area surrounding the agent (say 128x128) pixels, if not enough number of frontier lumps is found, the area is expanded (becomes 256x256) and the procedure is repeated, until enough number of frontiers is found or the area reaches the maximum possible area which is equal to the size of the whole environment, if this area is reached and no frontiers are found, then the agent knows that it has completed its task of covering the entire accessible space in the environment and the bitmap it holds can be considered a complete map of its environment (the accessible parts at least).

At the end of each search stage, all the frontier lumps (if found) are presented back to the agent so that it selects on of them, selecting one of the lumps can be done according to more than a criteria, the agent can choose the closest frontier lump to its current position, or it can chose the lump with the wealthiest number of frontier cells because it means that this lumps is on the border of a large unexplored area, another criteria can take into account the other agents and their choices (more on that later). We have experimented with both the first and the second criteria and found that the second is better, actually the second one is implicitly a hybrid of the two criteria, because it is selecting the frontier lump that is the wealthiest of the
nearby lumps, this is true because the first stage of the extraction of frontier cells start by collecting the cells within a small area surrounding the agent's position, which means that the frontier lumps reported will be the ones nearest to the agent position. After a lump is selected the position of the cell in the middle (the median) is reported back to the agent so that the next stage of the strategy starts.

**Navigating towards Frontiers**

This stage is concerned by finding a path from the current position of the agent to the target frontier point, to implement this the correct way, a complete navigation system has been created, and it was one that best suits our environment, we have chosen to implement a waypoint navigation system, the reason for this is that, it is a well known method that was implemented successfully in many applications and games, -more importantly- it is the method used by the Quake2 (see section 2.1.10) engine itself for bot navigation (although we are not going to use the Quake2 navigation system because it is off-line generated and is not accessible to the clients of the game because it is implemented internally inside the game engine and used by its own controlled bots or characters, but it is comfortable to know that the Quake2 game uses the same method we are using, so it will be suitable for our platform and environments).

Adopting the waypoint navigation system means that the agent has to place (by place we mean store in its own memory) a data structure called a Waypoint describing a position in the environment every time the agent moves to a new point within its environment, we have chosen to place a waypoint every 32 units, so each simulation tick the agent checks to see if he has got farther than 32 units from the last waypoint it has places or any neighboring waypoint, if so he places a new waypoint at that place, otherwise he just waits until he moves further, in addition to placing the waypoints the agent stores the connectivity information between waypoints by indicating in each waypoint structure pointers to the immediate neighboring waypoint it is connected to, this generates a big graph of connected nodes (waypoints) that covers the whole explored area of the environment and provides a map of the possible paths the agent can follow from any point to the rest within the explored area (see figure 30).
Figure 30: Illustration of the waypoint graph of a partially explored environment for an agent.

Now when the previous stage of detecting frontier lumps and choosing the median cell of the wealthiest nearby frontier lump presents the navigation stage with the median frontier cell, the position of the median cell will be the destination and the current position of the agent will be the source for the trip, two steps need to be done now, first the nearest known waypoint to the destination position is retrieved by querying the waypoint graph, finding this waypoint refines the problem into the following: find the optimal path between the current waypoint (source) to the waypoint we just retrieved (destination) across the graph of waypoints. This is a well-defined and understood problem that has many algorithmic solutions, like depth first, best first and A*, we have chosen the A* algorithm because it guarantees to return the optimal path and it uses a heuristic to speed things up. We have implemented the A* algorithm into our navigation system and it proved to be both robust and lightening fast, despite the large complex graph of waypoints we had for all environments. The A* algorithm returns a list of the waypoints starting from the current waypoint and ending at the waypoint nearest to the destination frontier, the position of each waypoint in this list is added to a navigational queue and the agent enters to path following mode where it starts traveling from point to point in the queue, and for each point it reaches, it is removed from the queue and the next one is set as a destination until no more points exist in the queue which means that the agent has reached its destination, which is the frontier, and then the random behavior kicks in, to allow the agent to explore this new unexplored area, this lasts for a certain number of ticks after which the whole new procedure is repeated and a new frontier is extracted and the agent heads for it. This continues until no more frontier points are detected which means the whole accessible space was explored.
Figure 31: consecutive shots illustrating frontier-based strategy in progress, red point is the agent, each two large white points represent source and destination waypoint and the dashed line represent the chosen path between them, the graph of waypoints was shown in some of the shots and hidden in the rest for better viewing (left to right, top to bottom)
4.2.2 Collective Strategies

In this category of strategies, the agent is aware of other agents working around him, this awareness is the basis of the cooperation and coordination between the agents, this cooperation and coordination will vary from a simple, implicit and reactive one like in the Repel and ant-like strategies to an explicit and complex one as in the cooperative-frontier-based strategy. Three strategies were studied, each representing a different approach to the task of collective exploration and coverage.

Agent-Repel Exploration Strategy

This strategy is random but –still- it uses a heuristic to help it achieve better coverage and exploration results. The heuristic tries to stretch the agents as much as possible by allowing them to repel from each other when they come within a certain range of each other, this prohibits more than an agent to be at the same place at the same time, which decreases the chance of two agents covering the same area redundantly, of course it does not guarantee that the agent will not come back.
(driven by its random motion) later to explore the same area when the other agent is gone, nor does it guarantee that the agent will not explore an area that it did explore before because -as we said- it is still a random motion, but it helps to keep agents as dispersed and stretched in the environment as possible.

For this strategy we assume that the agent is able -in some way- to detect the presence of other agents within a limited range, this could be done using simple visual sensor or a limited transmitter/receiver for example.

After an agent detects the other agents within its range a simple vector mathematics is applied to calculate the direction the agent needs to move to in order to repel from all the other agent and disperse as much as possible: first, a vector is calculated from each agent’s position towards the current position of the agent, then a resultant vector is calculated out of all the calculated vectors of the previous step, this resultant vector represents the direction the agent will move to, note that even after we decide the direction of movement a random angle is introduced relative to that direction to preserve the random behavior of the strategy. A suitable weight that is proportional to the distance between the agents is assigned to each vector in the first step to, so that agents located near affect in a larger extent compared to agents located farther, this system resembles a gravitational repulsion of equally weighted masses. Note that the same procedure is at work instantly inside all the other agents which creates this collective stretching and expansion in the environment.
Figure 33: (a) 4 agents applying the repel-agent strategy right after the start of the simulation
(b) analysis of the previous screen shot from the view of Ag1, the blue circle shows the range of Ag1
detection, the white vectors are vectors Ag1 used to estimate its resultant direction vector (the small red
vector). The rest of the red vectors (long ones) are the resultant direction for each of the three agents
(Ag2, Ag3, Ag4). Notice that because Ag2 was the nearest agent to Ag1 at the time of repulsion it nearly
balanced the effect of the two farther agents Ag4 and Ag3.

During the simulation we have used two values for the detection range, 256 unit and
512 units, both of which exhibited slightly different results under different
circumstances as we will see in the results chapter.
**Pheromone-Based Exploration**

This strategy takes a different approach for coordination between agents, namely by leaving pheromone trails in the environment, this way we are using the environment itself as a medium for communication between agents, usually pheromones are used for either attracting agents or detracting them, we are here using the second case, where each agent drops a pheromone trail every certain amount of units (32 units), this pheromone trail fades out as time passes until it vanishes, the trails are used in the following manner:

By default the agent will move randomly in the environment, while it moves it tries to detects any nearby pheromones, if none is founds it continues its random movement, if some pheromones trails are found, it tries to direct its random movement towards the area with the least pheromone trail concentration, this will mean the area with no pheromone trails (if possible) or the area with pheromones that faded out the most. This way this strategy makes sure that the agent will not visit an area visited previously by another agent and will prefer always to explore new areas that have no or weak pheromones which means areas that haven’t been explored or has been explored, but not recently. Needless to say that this algorithm doesn’t eliminate the redundancy, because after a certain time the pheromone trails will vanish, but it will help to keep the agent away from already visited areas for a relatively long time forcing them to explore new areas.

![Figure 34: screen shots for agents applying pheromone strategy in consecutive steps, notice how using the pheromone forces the agents to spread out in the environment away form the pheromone trails.](image-url)
Cooperative Frontier-Based Exploration

This strategy is an extension of the frontier-based exploration strategy discussed above, it tries to increase the efficiency of exploration by introducing an element of cooperation between agents. This strategy was introduced by the same author of the frontier-based strategy in [30], its advantages include that it follows a decentralized approach to the coordination and is robust to the loss of robot failures. This strategy works as follows: At the beginning the same procedure as the frontier-based strategy is applied; the agent scans for frontiers (regions between explored space and unexplored areas) selects one of the frontiers, retrieves the path from its current location to the frontier location and follow this path until it reaches the frontier then it start exploring randomly for a certain amount of time after which it repeats the whole procedure. The only difference in the cooperative version is that as an agent approaches another one and becomes within range each one, it communicates its information to the other, the exchanged information includes two elements: each agent’s environment representation bitmap and its waypoint graph.

After receiving both elements the agent first, integrates the other agent’s bitmap into his, this is done by learning all the points that was explored and marked as either walk-able or obstacle by the other agent and making them so in the agent’s own bitmap, note that this is possible because we are assuming perfect position information which makes matching the two bitmaps straight forward.

Not only do we need to share the bitmaps, but we have to share the waypoint graphs as well, because it encodes all the information on how to reach any point within the explored space, which is necessary for the correct functioning of the frontier-based exploration strategy especially the part where the path to the frontier is to be determined. Basically the waypoint graph of both agents have to be merged into one containing the knowledge of them both, this has to be done while eliminating any kind of redundancy that might stem from the fact that some waypoints in both graphs correspond to the same location in the environment but are different with a unit or two due to the difference in movement between the two agents, when merging into one graph these two waypoints must converge into one. This whole procedure can be considered a special case of a graph merging problem, which we have implemented fairly successfully. To avoid over-exchange of information, each time the agent exchanges information with another agent, it records the time (simulation time) of this exchange, and when a later encounter with this same agent happens, the exchange occurs only if a certain period of time has passed since the last exchange, this restriction is imposed because in real-life situation the exchange of information could be an expensive (time-wise, power-wise, etc) operation that must be done only when the outcome is worth it, that is, significant amount of information is exchanged.
4.3 Simulation Environments (Levels)

To evaluate each one of the different strategies, they had to be tested against more than one environment, because the characteristics of the environment plays a significant role in the degree of success or failure of a certain coverage strategy. To guarantee a subjective analysis and evaluation we have tried to provide reasonably different types of environment, each having a certain characteristic that we expected to have an interesting effect on the way the strategies work.

First, we have the Rectangle level which can be considered a big empty roam, a yard or a plain landscape. The second level is called Complex, which has roughly the same layout but we have introduced a couple of obstacles and dividers all over the level to create some kind of complex space that is divided into sub-spaces, the third level is called the Connected, where it is a set of relatively small rooms and passages connected to each other, and finally, the Jagged level that is the most complex one and has an edgy nature with very narrow passages linking the different parts.
5 Results & Discussion

5.1 Introduction

In this chapter we are going to illustrate the results of the simulation runs and try to discuss them relating the task of evaluating the different coverage and exploration strategies. After that an evaluation of our platform and the Game-based simulation approach is presented.

We have conducted a large number of simulation runs, each of the five strategies was tested against the four different environments (some was tested only on three of them because of the limited time we had), and the simulation run for a certain strategy in a certain environment was repeated for each different number of agents working at the same time (1, 3, 5 and 7 agents) and some of the strategies were repeated each time we changed a certain parameter (e.g. changing the detection range for the repel strategy and so on).

Each simulation run would end when, it achieves full coverage of the environment or lasts for more than a specific number of simulation cycles that is adjusted slightly for each environment (usually around 2 hours run-time). We had to impose this limit because of the large number of simulation we had to do within the relatively short time we had, nevertheless, this limit was long enough to study and evaluate how good the strategy is, and most of the strategies finished coverage long before this limit ended (except for most of the strategies in the Jagged environment), and finally it is reasonable to impose a limit on coverage tasks in practical situations, where surpassing this limit is not allowed, because of time or operating cost constraints.

We will be evaluating each strategy both individually and comparatively to other ones, after that a comprehensive over-all comparison of all strategies will follow at the end. A number of characteristics about each strategy is studied, mainly how adding more agents affect the exploration time and the effect of different environments on the performance of each strategy. Other characteristics are considered also -when possible-, like the effect of communication and varying the detection range for some strategies.

5.2 The Random Strategy

Counter-intuitively this strategy performed well on all the environments, except for the jagged environment. Here is the table summarizing our simulation results.

<table>
<thead>
<tr>
<th>Num of Agents</th>
<th>Cycles needed to achieve full coverage of the environment</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rectangle</td>
</tr>
<tr>
<td>1</td>
<td>54400</td>
</tr>
<tr>
<td>3</td>
<td>22050</td>
</tr>
<tr>
<td>5</td>
<td>9550</td>
</tr>
<tr>
<td>7</td>
<td>9400</td>
</tr>
</tbody>
</table>

Table 1: Random strategy simulation results
Here is a visual representation of the result, please note that we have measured the
time (in simulation cycles) needed to achieve full coverage of the environment, also
notice that the jagged level is discussed separately later.

![Random Strategy - Exploration Time](image)

**figure 36:** Random Strategy Exploration Time in cycles v.s number of agents.

**Number of Agents**

We notice that increasing the number of agents decreases the time needed to
achieve full coverage, which is expected since you have more workers covering more
space at the same time, this is true until a certain point is reached, after which
adding more agents becomes of no much use, this is caused by the redundancy in
coverage, since nothing in the random strategy forbids an agent from covering an
area that was covered by another agent, so we expect that part of the work done by
each agent is redundant. This is evident in the graph where the rate of decrease in
coverage time becomes less as we increase the agents, for example the decrease in
coverage time when introducing three agents compared to one and five agents
compared to three is quite significant (from 75% to 40%) while the decrease
obtained when introducing 7 agents compared to 5 is almost negligible.

**Different Environments**

The Random strategy performed basically in the same manner in all of the
environments (except for the Jagged environment), this similarity is expected
because of the nature of the random movement which assumes nothing about the
environment and has the same probability of moving in any direction at all times.
After this steady performance of the random strategy we tried to design a certain
environment that has certain characteristics that forms a challenge to the Random
strategy, so we designed the Jagged environment (see figure 35-d) which has jagged
edges especially in the narrow passages connecting different parts, these hook-like edges were introduced to trap in the agents when they try to cross from one part to the other, unless their direction is parallel to the passage direction. This was devised to alleviate the effect of sliding which is caused by the way the Quake2 engine treats collisions, that is, whenever an agent collides with a smooth wall (wall with no edges) it will slide along this wall until either the agent changes the direction or it bounces off an edge or an obstacle on the wall later, this means that the agent doesn’t need to have the correct direction to pass through a narrow passage between two connected part, all that is needed is to pump into the wall and have any direction that is slightly tilted towards the passage and this will be enough to slide along the narrow passage and pass the second part, although this could happen in special cases with real exploration robots (if they are designed to slide along rectilinear geometry) but we wanted our results to be as general as possible, so we introduced these edges -to counter the sliding- in the Jagged level. And as we expected the random strategy was not able to completely cover the Jagged level regardless of the number of agents used within the threshold time, (although the trends shown in other environments still applied) but in this environment the agents needed more time (more than the threshold) both because of the edgy nature of this environment and its complexity.

<table>
<thead>
<tr>
<th>Num of Agents</th>
<th>Coverage percentage at threshold time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33.30%</td>
</tr>
<tr>
<td>3</td>
<td>63.30%</td>
</tr>
<tr>
<td>5</td>
<td>64.22%</td>
</tr>
<tr>
<td>7</td>
<td>79.80%</td>
</tr>
</tbody>
</table>

Table 2: Random strategy simulation results for Jagged environment

5.3 The Agent-Repel Exploration Strategy

Basically this strategy is similar to the Random strategy with the exception that it uses the heuristic of repelling form other agents to help expand further in the environment. We have conducted two sets of simulations changing the radius of the detection range of nearby agents from 256 to 512 units.

<table>
<thead>
<tr>
<th>Number of Agents</th>
<th>265 Unit Range</th>
<th>512 Unit Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycles needed to reach full coverage percentage</td>
<td>Cycles needed to reach full coverage percentage</td>
</tr>
<tr>
<td></td>
<td>Rectangle</td>
<td>Complex</td>
</tr>
<tr>
<td>1</td>
<td>54400</td>
<td>71550</td>
</tr>
<tr>
<td>3</td>
<td>15300</td>
<td>26150</td>
</tr>
<tr>
<td>5</td>
<td>8650</td>
<td>16600</td>
</tr>
<tr>
<td>7</td>
<td>5550</td>
<td>7750</td>
</tr>
</tbody>
</table>

Table 3: Agent-Repel strategy simulation results.

Here is a visual representation of the result, please note that we have measured the time (in simulation cycles) needed to achieve full coverage of the environment, also notice that the jagged level is discussed separately later.
**Number of Agents**

The same trend that showed up in the Random strategy applies here too, as we increase the number of agents the exploration time needed to achieve full coverage decreases, this is always true only in the 256 unit case, in the 512 units case, after increasing the number of agents to five and more in the connected environment the agents get stuck, similarly after increasing the number of agents to five and more in the complex environment the time needed starts to increase instead of decreasing. Both becoming stuck and the increase in time instead of decrease are caused by the same factor, which is the high repel range relative to the geometry of the environment (more on that later).

Comparing how each of the two different radii of the repel range has affected the decrease in full coverage time (ignoring the two abnormal cases discussed above) there is only a slight difference between the two cases, which suggests that the 256 unit is the optimum value because it has worked for all the environments unlike the wider 512 unit value. (comparison with other strategies later).
Different Environments

The Agent-Repel strategy is sensitive to its environment more than the random strategy, even though it assumes nothing about its environment, but because the direction and movement of the agent becomes dependent on the position of other agents and how close or far they are from it, and since the position of those agents depends -momentarily-on the geometry of the environment (for example the agent cannot move backward because there is a wall) the agent movement becomes dependent indirectly on the geometry, this has introduced some problems in certain circumstances, for example -while moving randomly- an agent could come near a corner while another agent -at the same time- is at the opposite corner and both become in the repel range of each other at that instant, so both of them will be pushed further toward the corner and get stuck, we have found that the chance for this and similar scenarios to happen increases when the number of agents involved is increased, the repel range is wider or the geometry becomes more confined, this justifies clearly why the agents became stuck in the connected environment as the number of agents was increased to five and above and the repel radius was 512 units, because in that scenario all the conditions will apply; the connected environment is a series of confined small room-like parts, the range of the agent was relatively large compared to these small rooms and the number of agents was high for that size of the room. On the other hand, in the complex environment after increasing the number of agents to five and more and the repel radius to 512 although the conditions didn’t apply completely enough to cause the agents to stuck, but they were bad enough to obstruct and limit the movement of the agents and cause the exploration time to increase sharply instead of decreasing, this is true because although the Complex environment has many edges and corners it is still not confined enough to trap the repelling agents, at least not until the number of agents is large and the repel range is wide relative to this environment.

Because the Agent-Repel is still a random strategy it suffered from the same problems of the Random strategy in the Jagged environment, in addition to the problems introduced by the Repelling behavior.

<table>
<thead>
<tr>
<th>Num of Agents</th>
<th>Coverage percentage at threshold time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32.17%</td>
</tr>
<tr>
<td>3</td>
<td>73.10%</td>
</tr>
<tr>
<td>5</td>
<td>77.24%</td>
</tr>
<tr>
<td>7</td>
<td>stuck</td>
</tr>
</tbody>
</table>

Table 4: Agent-Repel strategy simulation results for Jagged environment.

In the cases where the agents where not stuck, the agents suffered from the complexity of the environment and its edgy nature, the time required to achieve full coverage was more than the threshold time, but still we notice that the coverage percentage achieved by the time the threshold was reached was slightly higher than that in the Random strategy (which is a general trend as we will see next). Because the jagged level can be considered highly confined with lots of corners and edges the number of agents didn’t need to reach a large number to exhibit the “stuck condition”, in the 512 unit case just after the agent are raised to three they become helplessly stuck, the 256 unit case manages to avoid being stuck until the number of agents reaches seven then they are overcrowded enough to become stuck.
Comparison

Comparing the performance of the Agent-Repel strategy to the Random strategy, we want to see if using the repel heuristic has made an improvement in respect to coverage time. The following graph compares the exploration time for the two strategies in two different environments (The Rectangle and Complex).

![Graph comparing exploration time for Repel and Random strategies in Rectangle and Complex environments.](image)

We note that there is a slight improvement in the exploration time for the Agent-Repel strategy over the Random one, especially when the number of agents start to increase because only then the repelling behavior starts to kick in between agents, though the improvement is not drastic, because although the repelling reduces the chance of redundant work by expanding out the agents from each other (at least temporally), but it will not eliminate it completely. Note that we have compared the Random with the 256 unit case of the Repel-strategy which is the overall better case of the two (256 and 512). The above analysis applies -also- to the other environments, including the jagged environment, as it shows in the following table.
comparing the Agent-Repel with the Random in context of the Jagged environment (ignoring the stuck case in the agent-repel seven agents case).

<table>
<thead>
<tr>
<th>Num of Agents</th>
<th>Coverage percentage at threshold time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Repel</td>
</tr>
<tr>
<td>1</td>
<td>32.17%</td>
</tr>
<tr>
<td>3</td>
<td>73.10%</td>
</tr>
<tr>
<td>5</td>
<td>77.24%</td>
</tr>
<tr>
<td>7</td>
<td>stuck</td>
</tr>
</tbody>
</table>

Table 5: Comparing Agent-Repel and Random strategies simulation results for the Jagged environment.

5.4 Pheromone-Based Exploration

This strategy is basically a Random strategy that uses another technique for improving the exploration process, it uses a pheromone-trail to reduce the redundancy by directing the agent to areas that have no pheromone or the pheromone that has fades the most. The following table summarizes the results for this strategy in the Rectangle and Complex environments (the Jagged will be discussed later).

<table>
<thead>
<tr>
<th>Num of Agents</th>
<th>Cycles needed to reach full coverage percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rectangle</td>
</tr>
<tr>
<td>1</td>
<td>43850</td>
</tr>
<tr>
<td>3</td>
<td>24050</td>
</tr>
<tr>
<td>5</td>
<td>15600</td>
</tr>
<tr>
<td>7</td>
<td>15400</td>
</tr>
</tbody>
</table>

Table 7: Pheromone-based Exploration strategy simulation results.

Here is a visual representation of the result, please note that we have measured the time (in simulation cycles) needed to achieve full coverage of the environment.

**Figure 40:** exploration time for Frontier-based (cooperative/non-cooperative) in the Jagged environment.
Number of Agents and Different Environments

The trend is clear for this strategy, as we increase the number of agents the time needed to achieve full coverage is decreased, although the rate of decrease gets less as we add more agents, this is evident in the Rectangle environment where the improvement is meager -if any- when raising the number of agents from five to seven, this is caused by the inevitable redundancy at the large number of agents working together (although this redundancy is kept at a minimum because of the pheromone trails).

Since the Pheromone-based strategy is a random strategy it suffered from the same problem of all other random strategies in the Jagged environment, it was not able to achieve full coverage before the threshold time, here is a table of the coverage percentage achieved at the threshold time for the Pheromone-Based strategy.

<table>
<thead>
<tr>
<th>Num of Agents</th>
<th>Coverage percentage at threshold time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>51.97%</td>
</tr>
<tr>
<td>3</td>
<td>73.47%</td>
</tr>
<tr>
<td>5</td>
<td>80.11%</td>
</tr>
<tr>
<td>7</td>
<td>82.59%</td>
</tr>
</tbody>
</table>

Table 8: Random strategy simulation results for Jagged environment

Comparison

Comparing this strategy with the random strategy we want to know if the introduction of the Pheromone trails has brought any improvement to the procedure of exploration. The following table and graph summarize the comparison.

<table>
<thead>
<tr>
<th>Number of Agents</th>
<th>Random Cycles needed to reach full coverage percentage or percentage achieved at threshold time (Jagged)</th>
<th>Pheromone-Based Cycles needed to reach full coverage percentage or percentage achieved at threshold time (Jagged)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Rectangle</td>
<td>Complex</td>
</tr>
<tr>
<td>1</td>
<td>54400</td>
<td>71550</td>
</tr>
<tr>
<td>3</td>
<td>22050</td>
<td>25300</td>
</tr>
<tr>
<td>5</td>
<td>9550</td>
<td>12250</td>
</tr>
<tr>
<td>7</td>
<td>9400</td>
<td>13250</td>
</tr>
</tbody>
</table>

Table 8: Comparison of exploration time for the Random and Pheromone-based strategies

Here is the graph of the comparison of exploration time in the Rectangle and Complex environments.
We note that the Pheromone-based strategy achieves a slightly better performance over the Random in some cases and sometimes the Random performs better, taking in consideration the Jagged environment results in the table (where the Pheromone-based is much better than the Random), it seems that the more the space in the environment is smaller and more confined (like in the Complex and Jagged environments) the better this strategy performs, this makes sense because when the space is smaller and more confined the agent takes less time to move from an area to another randomly which increases the chance that the pheromone trails didn’t fade out yet and the agent will detect them and move away reducing the chance for redundancy, while if the space is large and open (like in the Rectangle environment) the agent will move a farther distance by its random movement and consequently take a longer time, and by the time it visits a place it has visited before, the pheromone trails it had dropped previously will have faded out and the agent will explore this area if its random movement drives it there.

Despite this explanation and after we started analyzing the results, we still believed that this strategy should have performed better, so we observed it thoroughly during execution in all the environments and found out that it suffered from a problem, we have found that the agent -sometimes- gets near a corner that has pheromones trails on both its left and right, when the agent reaches the corner and tries to decide its next direction of movement it detects the nearby pheromone trails, when the agent tries to find the direction that enables it to move away from these trails it will be towards the corner and its stuck (because nothing in its strategy tells it what to do if the direction it moves into is blocked with an obstacle) until the pheromone trails (or at least one of them) fades out and it can move. We noticed that this case happens fairly frequently in all the environments and we believe that it is the cause of the under-achievement of this strategy. By the time we noticed this, we had no time to modify the strategy (to deals with such situation) nor the time to repeat the simulation runs.
5.5 Frontier-Based Exploration Strategy (Cooperative and non-Cooperative)

We have conducted two sets of simulations for this strategy, a non-cooperative one (where agents working at the same time are not aware of each other—and so they are working independently) and the other is cooperative (where agents share their knowledge periodically as they get near each other). The following table summarizes the results.

<table>
<thead>
<tr>
<th>Number of Agents</th>
<th>Non-Cooperative</th>
<th>Cooperative</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycles needed to reach full coverage percentage</td>
<td>Cycles needed to reach full coverage percentage</td>
</tr>
<tr>
<td></td>
<td>Rectangle</td>
<td>Complex</td>
</tr>
<tr>
<td>1</td>
<td>43950</td>
<td>51300</td>
</tr>
<tr>
<td>3</td>
<td>37200</td>
<td>48150</td>
</tr>
<tr>
<td>5</td>
<td>28350</td>
<td>42600</td>
</tr>
<tr>
<td>7</td>
<td>32550</td>
<td>37350</td>
</tr>
</tbody>
</table>

Table 6: Frontier-based Exploration strategy simulation results.
Non-Cooperative Frontier-Based Exploration Strategy

Here is a visual representation of the result, please note that we have measured the time (in simulation cycles) needed to achieve full coverage of the environment.

![Frontier-Based Exploration Time (non-cooperative)](image)

**Figure 43:** Simulation results for the non-cooperative frontier-based exploration strategy.

### Number of Agents and Different Environments

This strategy exhibited a slightly different behavior for different environment, although we could note a general trend where the time needed to achieve full coverage is decreases as we increase the number of agents until we reach a certain number of agents (five) after which the time in both the Rectangle and the Jagged environments starts to increase instead of decreasing, while in the Complex environment it continue to decrease, we believe that the later increase in coverage time in both the Rectangle and Complex is caused by the fact that since this strategy is not cooperative an agent can work on a region that was explored by another agent, which introduces redundancy, and because of the way the frontier-strategy chooses the frontier to work on (it chooses the nearest and wealthiest frontier) it is very likely that when two agents are near each other they will choose the same – nearby- frontier to work on, which means, the more the agents are clustered near each other the more they will do redundant work. The characteristic of the environment plays an important role in this situation; because the Rectangle environment is just one big space all the agent will be working in the same area. As the number of agents become large (more than five) and because they are working in the same space some of them will start choosing similar nearby frontiers to work on and the redundant work starts to become higher and the time needed for full coverage starts to increase (which explains the curve of the Rectangle environment), on the other hand, because the complex environment has many dividers and corners that –divides the total area into sub-regions, the chance that different agent will work on relatively separate parts (sub-regions) is increased and so the amount of redundant will -consequently- decrease, (which explains why the curve in the Complex environment continues to decrease when adding agents). According to this
explanation we can also understand the curve of the Jagged environment; because this environment is divided naturally into interconnected rooms or sub-regions, the chance for different agents to move into different parts is increased, which in turn decreases the chance for choosing the same frontiers and doing redundant work (this explains the sharp decrease in exploration time when adding three and five agents), but this will not continue indefinitely, as the number of agents increases to seven, agent will -inevitably- start to cluster and do redundant work, note that this didn’t happen in the Complex environment even when the agent number reached seven because the latter has more walk-able (open) space than the narrow edgy Jagged environment, and so the Complex sub-regions can accommodate more agents without them being squeezed near each other than the Jagged environment’s narrow room-like sub-regions.

This clustering of agents in the Jagged environments creates another problem, it causes the agents -sometimes- to get stuck when they come in each other’s way in narrow passages, this would happen when two agents are heading towards two frontiers located on two opposite sides of a narrow passage, if they meet in the middle of the passage neither one can move towards its destination and become stuck, luckily we have handled this case in our implementation, where if the agent becomes stuck when traveling towards a frontier for a certain amount of the time, it tires first to move randomly for a limited period of time and then checks again if it can now move freely, if it discovers that it still cannot reach its destination it cancels the current frontier and tries to find another one, this would solve the problem, but will cost both agents valuable time, this case happened a lot in the Jagged environment (because it is rich with narrow passages) and contributed to the increase in coverage time observed at large numbers of agents, in addition to the other factors mentioned above (see figure 44).

**figure 44:** two agents following frontier-based strategy trying to cross a narrow passages into opposite directions (not to mention the third one stuck behind).
Cooperative Frontier-Based Exploration Strategy

Here is a visual representation of the result, please note that we have measured the time (in simulation cycles) needed to achieve full coverage of the environment.

![ Frontier-Based Exploration Time](cooperative)

Figure 45: Simulation results for the cooperative frontier-based exploration strategy.

Number of Agents and Different Environments

Although the introduction of information exchange between the agents in the cooperative version of the frontier strategy, has significantly reduced the time needed to achieve full coverage (as we will discuss in more detail next) the trend regarding adding agents and the effect of different environments is still the same: as we increase the number of agents time needed to achieve full coverage will decrease until we reach a certain number of agents (five) after which the time in both the Rectangle and the Jagged environments starts to increase instead of decreasing, while in the Complex environment it continue to decrease (for a full detailed explanation for this behavior please read our analysis for the non-cooperative version of the strategy above) we have -basically- attributed this to redundant work done by the agents when they cluster near each other and start working on the same frontiers other agents worked or will work on, having mentioned this, a reader might wonder why this effect is still present in the cooperative version, the reader will reason that since the agents -now- share information about parts of the environments they have explored, it means that the redundancy should be eliminated because the agents will not attempt to revisit those parts again, which should eliminate the redundancy. Well, this analysis is partly true, yes, the agents will not revisit parts already explored by others, but in the period between information exchanges between the agents, no coordination exists, which means that after two agents exchange their information about the environment and have the same perception of the environment, they will decide independently which frontier to explore next, which means that they -still- might choose the same frontier and work on it redundantly until the next exchange comes between both of them and they discover that they are working on the same frontier. Put another way, the cooperation has reduced the redundancy significantly (and this explains the better
performance by the cooperative version as we will see next) but still it didn’t
eliminate it completely, to eliminate it we need to include cooperation on the level of
frontier selection, something the current strategy doesn’t provide.

Comparison

We want to compare the performance of the Frontier-Based strategy (both
cooperative and non-cooperative versions) to the Random strategy in order to
evaluate their performance relative to each other. The following graph compares the
performance of the three strategies in two different environments.

![Comparison Graph]

**Figure 46:** Comparison of exploration time for Frontier-based (cooperative/non-cooperative) and Random strategies in (a) Rectangle (b) Complex environments.

Cooperative V.S Non-Cooperative

It is clear from the graph that the cooperative version of the Frontier-based strategy
is significantly better on all aspects in all the environments (this applies for the
Jagged also, more on that later), which is what we expected, since the exchange of
information between agents will reduce (but not eliminate) the redundancy and -
consequently- reduce the time needed to achieve full coverage, the improvement in
the Rectangle and Complex environments is around 50%.

Cooperative V.S Random

Comparing the better of the two versions of the Frontier-based strategy which is the
cooperative to the Random strategy we –first- look at the Rectangle and Complex
environments depicted in the graph above, we notice that in the range from one to
three agents the Cooperative Frontier strategy is the better one, after the number of
agents is increased beyond three, the Random becomes the better one, this is not
the case always, in the Complex environment where the frontier-based strategy is
allowed to work effectively because of the characteristic of the environment (see
analysis of non-cooperative frontier-based strategy above) the Cooperative regains
its previous advantage over the Random strategy when the number of agents
reaches seven, while on the other hand, in the Rectangle environment where the
characteristic of the environment increases the redundancy (see analysis of non-
cooperative frontier-based strategy above) the Random maintains its advantage over the Cooperative. Another -even more important- reason for the Frontier-bases being behind the Random in some cases is because the Frontier-based strategy works in away that makes it “Slow” compared to the Random, because finding frontiers and moving towards them is a crucial part of this strategy, a large amount of time is spent by the agent trying to reach the frontier where it is supposed to explore, and because the agent is following the graph of waypoints to reach the frontiers it is walking on an already explored regions, which means that a large amount of time is spent without exploring new parts, this makes the Frontier-Based strategy slow relative to the Random strategies that moves in every direction and has a good chance to move into unexplored areas every time (especially at the start), luckily for the frontier-based strategy, this effect is alleviated by the fact that the agent tries to move towards the nearest frontiers available which reduces the distance traveled and enables it -sometimes- to outperform the “fast” Random strategy.

After this result one might wonder why should we consider the Frontier-based strategy that requires extra information and sophisticated calculations (which translates into expensive hardware) when it doesn’t show a significant advantage all the time over the simple Random strategy that requires no information nor calculation? The answer to this question will be clear when we discuss the result of the simulations of the third environment, the Jagged.

Here is the graph of the exploration time for the Jagged environment for both the cooperative and non-cooperative versions of the Frontier-based strategy.

![Exploration Time Graph](image)

**Figure 47**: exploration time for Frontier-based (cooperative/non-cooperative) in the Jagged environment.

Besides showing that the cooperative version has performed better than the non-cooperative version (which we knew from the previous discussion), this graph -also- shows that the Frontier-based strategy was able to achieve full coverage before the threshold time, actually to be exact the single agent using the frontier-based strategy was not able to achieve full coverage by the threshold time; it needed 103100 simulation cycles (the threshold is around 65000 cycles), but when the number of agents is increase to three and more the Frontier-based exploration starts to achieve full coverage within the threshold time. This is where the Frontier-based strategy as a complete strategy outperforms the random based strategies,
remember that none of the random based strategies (Random, Repel and Pheromone) were able to achieve full coverage before the threshold time.

5.6 Overall Strategies Comparison

We will try now to compare the performance of all the strategies together for the different environments.

---

**figure 48**: Comparison of the four strategies exploration time for the Rectangle environment

**figure 49**: Comparison of the four strategies exploration time for the Complex environment.
Comparing the Random-based strategies together, The Agent-Repel strategy has performed either the same or better of the Random strategy in most cases, this is expected as it is allowed to use more information (position of nearby agents) than the Random strategy which uses none, following this we would expect that the Pheromone-Based strategy performs the best, because not only it uses more information than the Random strategy, but this information is more than what is allowed to the Agent-Repel, because the Pheromone-based strategy knows the places visited by the agents some time in the past (depending on the life time of the pheromone) while the Agent-Repel knows the places visited by the nearby agents (from the position of those agents) in the current instant of time only. This is reflected clearly in the Jagged and Complex environments, where it outperforms both the Random and the Agent-Repel strategies, but in the Rectangle environments it only outperforms them in the single agent case and after that (for higher number of agents) the Random and Repel performs better, we have tried to explain this previously and attributed it partly to the nature of this environment and most importantly to a certain frequent problem faced by the Pheromone strategy at corners and obstacles edges (see our discussion on Pheromone-bases results above for more details), we believe that if this problem were resolved the Pheromone-based strategy will outperform the other random-based strategies in all cases.

The previous discussion leads to an important result or trend, which is that the more information we allow an exploration strategy whether about other agents or the environment the more it will perform better and reduce the time needed for exploration.

Finally, comparing the random-based strategies with the Frontier-bases strategy, we notice that because of the “slow” nature of the Frontier-based strategy where a large amount of time is spent for the agent walking in already explored paths trying to
reach the frontiers to explore, we find that although it starts well, it is outperformed later by the random-based strategies in the Rectangle and Complex environments, but this all come to an end in the Jagged environments, where all random-based strategies fail to achieve full coverage within threshold time, and the Frontier-based is able to achieve it in much less than the threshold time, showing clearly the meaning of a “Complete” strategy that achieves full coverage in all environments regardless of its characteristics. This justifies the extra information and sophisticated calculation it performs relative to the simple, but not suitable-for-all random-based approaches.

5.7 Simulation Platform and Game-based Approach Evaluation

We will start by evaluating the game-based simulation approach in general, then an evaluation of our simulation platform will follow.

Game-based Approach Evaluation

Designing our simulation platform and using it extensively in the simulation for evaluating the different coverage and exploration strategies gave us a clear and practical insight into the game-based simulation approach. We have found many advantages and disadvantages for this approach, starting with the advantages, first, building or using a platform based on a game, means that you will find a huge amount of functionality there for you, for example, in our platform, the physical simulation engine was built for us, which is the Quake2 game engine, with all the needed functionality for collision detection, players movement management, items management, in addition to a full working communication layer, taking care of all the details of sending and receiving perceptions and actions, and working on more than a different network configuration. This all comes with a suit of tools and applications supporting this engine, you have many level editors (most of them are free) that can be used to build the environment for your simulation, some other tools help you customize all aspects of settings in the game. Add to all this, that all these tools from the engine, the communication layer to the level editors are tested for robustness by millions of users (players, level designers, ..) that have worked with it under hundreds of different configuration, systems and scenarios, so, you expect these tools to be as bug free as possible judging by the success of the product, this leads us to the next point in favor of this approach, which is that because of that success and popularity, you have a huge community of users and developers that have produced countless tutorials, bug fixes, extra tools supporting the game engine, this has proven to be very helpful in the process of using or developing a simulation platform, for example when we tried to design the environment using the level editor we have found -literally- hundreds of tutorials showing you how to do that, also when we needed a library to help visualize the environment, we found one because of the popularity of the game and consequently the large number of developers out there who written tools, extensions and libraries to supporting it, making some of the proprietary formats used by this game become a standard (like the bsp format that represents the geometry).

We believe that if all these facilities are to be written from scratch, it will take a considerably long time and huge effort, and even if that happened, it will not be as robust and efficient as games, which are written by a group of professional programmers working in this field for years, and squeezing every bit of performance out of the software and hardware.
These were the advantages of the game-based approach, now we will talk about the disadvantages, first and as we said before, the problem is that for this approach to work many condition has to be met: first, not all games are suitable for supporting simulations, the game has to be available for the community and open for customization, this means two things, one is that the tools used in game content creation like level editors have to be available and free (or at least reasonably priced), the other -more important- thing is that the game source code (or at least a script language controlling the inner workings of the game) has to be released also, this is important because -inevitably- a moment is going to come when a certain functionality (or an extension of an already existing one) is going to be needed, like when we needed to remove the restriction on the probe agent perception range or when we wanted to handle the message buffer overflow problem, if the source code of the game and the tools to compile it are not available, the whole simulation platform will run into a serious problem. This will lead us into another main issue, because one will -eventually- need to modify the source code, a substantial knowledge of the inner working of game engines is needed, or otherwise the person who wants to modify a certain functionality will find himself/herself in amidst thousand (if not millions) of lines of code that is written for speed and efficiency not for readability, the community here can help, since there exist a large number of sites, discussing the source code and providing tutorials on how to modify it, but -still- the effort is not trivial.

The latter discussion comes to context as a problem if we knew that not all games have their tools and most importantly their code released, at least not before a long time has to passed that the technology behind the game becomes relatively obsolete (market wise) and is released for the public.

Another condition that has to be met for a game to be used to support simulation, is that the nature of the game has to match that of the simulation field, for example we have used the Quake2 game engine as a basis to simulate mobile robotics strategies, It was not a coincidence that the Quake2 game involves players moving in an environment, pumping against walls, collecting items, planning their paths, coordinating their attacks, etc. which are all, things that is similar to what robots do, actually it is a must to have this match, because otherwise instead of supporting the simulation by providing the facilities needed, the game will start to impede the progress of the simulation by requiring the developers/users to implement the missing functionalities an facilities they need. This leads us into acknowledging that the resulting simulation platform based on a game is going to be of use only within the same field, we are not saying that this matching requirement and consequent limited usability is a disadvantage in its own right, because even if we used other approaches like building the platform from scratch it will only be useful for simulations within a certain field only, like the social field or the biological field and cannot have a platform that can be used for robotics and social simulation at the same time. Rather, what we are trying to say is that this matter adds another restriction on the games we are able to work with in simulations, since now it has to be open source, relatively cheap and matching the field we want to simulate. Taking the previous factors in consideration, one starts to realize that -regardless of the advantages and disadvantages of this approach- it is a limited one, in the sense that, we are not going to find a game that meets all our conditions for every field of
research we want to simulate, and only those fields that match the current genres of
games are candidates for simulation using this approach.

**Simulation Platform Evaluation**

To evaluate our platform we need first to evaluate the QASE[17] SDK which our
platform is built upon, after that we evaluate our simulation platform.

QASI is a powerful agent simulation platform, it provides quite rich functionalities
and utility tools providing great flexibility -when needed- especially because of its
open source policy. We believe that it opens the door for lots of opportunities in this
field. We think - though - it still misses some features that if added, will make it –
yet- more useful, please note that we have implemented some of these feature when
we have built our platform.

Because it is a relatively new API not much examples of usage and applications are
available yet, though the API SDK provides a couple of templates for many of the Bot
classes implementation, but still this is not enough, we have felt frustrated
sometimes because of the lack of tutorials, community support, base of users, which
makes the only possible way to understand some features or how to use them, is to
start reading the source code and try to reason about it.

The API doesn't provide visual rendering for the client to see what is happening in
the environment in real time, to do so the user has to either go to the server
machine and make it run in graphical mode and watch what is happening from there,
or start an instance of the Quake2 game on any other machine (including the
client’s) and join the server as a spectator or participate as a player if he wishes to.
We have addressed this issue in our platform, by implementing this functionality
through the bsp environment viewer and the Probe agent.

The QASI documentation doesn't show how the user could build his own level and
assign bots locations and populate the map with items, i.e. some knowledge about
the Quake2 game and its accompanying utilities is assumed for anyone who is
supposed to use this SDK. We have partially addressed this issue in this report,
where an extensive background and some of the necessary inside knowledge on the
Quake2 game (and all similar games) were presented to make the reader familiar
with this topic, and –at least- know where to start learning.

The QASI API does all its magic without touching the game source code, a lot can be
added if it was combined with a modification of the actual game engine to tune it
specifically for the task of being an agent simulation framework. We had to touch the
code ourselves to implement some of the functionality we needed, if more time
where allowed other things could have been added to support simulations, which
indicate the possibilities if the game code was modified more extensively.

Finally, we will be evaluating our simulation platform and the simulation runs we did
using it. We think that we have been all that time talking about the advantages and
good aspects of our simulation platform, and so we will now show the disadvantages
and the difficulties we have faced developing and using it for our simulations.

One of the main difficulties we have faced in this project is the long time each
simulation run takes, because the platform is using the Quake2 game engine as the
basis for the physical simulation of the environment, the speed of receiving
perceptions and sending actions is that of the engine, the rate of this exchange is 10 times per second, this is relatively slow (compared to what modern computers are capable of) and is this way because the engine has lots of things to calculate so it needs some time for processing on each information exchange, and even if the amount of processing was done faster, the game needs not do so, because the game is dealing with players who are humans, having reactions even slower than this rate of exchange, not to mention that the exchange occurs across the network, which imposes a limit on the maximum speed your can reach eventually. All this made the simulation runs occur at near human reaction speed, and an hour of simulation is approximately equal to an hour in real life, which is very slow and takes a long period if you have a large number of simulation runs like the one we had to do. We believe that platforms not based on a game, like the ones built from scratch will not suffer from this, and using them an hour in real life could be simulated in minutes or seconds.

The second issue is that, because the QASI SDK is written in Java, we had to write our platform in Java, this had the advantage of making our platform portable working under many systems, but the bad side of using Java for simulation is that, first, java is not very favorable performance-wise, especially in graphics, and sense one of the main features of our platform is the visualization, this could have been a serious issue, but luckily, it was not, because of the introduction of the Java3d and the library we have used (see section 3.2 for more details) but still the visualization was slower than what it could have been if it was implemented by C++ for example, which is much more efficient than java.

The second, issue was memory, because of the way the QASI SDK work each agent (being a thread) allocates a considerable amount of memory, in addition to the memory allocated by the visualization of the environment (both the environment visualization and the bitmap representation for all agents), and finally the memory allocated by some of the strategies like the frontier where it works on copies of the environment representation bitmap for each agent and also the memory needed for the graph of waypoints. All this have posed a memory problem, because the memory management is automatic in Java, we had to do lots of optimizations to make the huge platform memory requirement manageable, and we still needed more. We think that if we have implemented it in a language like C++, where the programmer controls memory management explicitly and where memory representation of objects is smaller than what it is in Java, we would have required less memory than the one currently required.
6 Conclusion

We believe that this project has achieved most of the objectives it had set out to achieve to a reasonable extent, we have illustrated a new approach in Multi-agent based simulation that is Game-based simulation, presenting all the necessary background needed to understand all the theoretical and practical aspects related to this field, where we believe that the lack of such information was a deterrent for anyone in the research field trying to take advantage of this new approach in simulation. Further more, we have enriched our theoretical presentation of this approach with a practical hands-on knowledge by putting all this into practice and building a simulation platform, documenting all the necessary technical skills, describing all the needed steps in detail and providing the best assessment possible for a platform which is using it in reality, doing a large number of simulations, under different circumstances and for long periods, revealing the true power or weakness of this platform in specific and the game-based approach in general. We believe that anyone reading this report can assess confidently the viability of this approach to his/her needs, and know certainly the type of skills he/she needs if he decides to use it.

The second objective that was achieved is implementing and evaluating a number of coverage and exploration strategies. We have chosen three random-based exploration strategies and one complete strategy, we have studied the effect of varying a number of parameters on the coverage time; we have found that as we add more agents, the coverage time decreases, until a certain number of agents is reached after which the improvement becomes insignificant or even the coverage time starts to increase, this trend was observed in all the strategies (the random and the complete) with various degrees. We have found that the characteristics of the environment plays a great role, where although it doesn’t change the previous observed trend but it fine tunes it in a different way for each different strategy.

We have found that surprisingly the random-based strategies were able to achieve full coverage of all the environments except for one that needed more time than was allowed, but we believe (judging from the simulation results) if time was not an issue they will achieve full coverage eventually. We have also found that although the complete strategy we have adopted is “slow” (due to the processing it needs and the way it works) it was able to compete with the “quick” random-based strategies in some (but not all) scenarios and beat it sometimes. In a certain environment -that has certain characteristics impeding the random-based strategies- only the complete strategy was able to achieve full coverage within the time limit we have imposed, which showed clearly that even though the random-based strategies performed well in most environments, but there could always be a certain environment that will make it difficult for a random strategy to finish within feasible time.

We have also found that the more the information allowed to the strategy, (whether about the other agents or the environment), the more it performs better, this was evident for both random-based and complete strategies. Same result applies for the cooperation between the agents, this cooperation whether it was a mere awareness, like in the repel strategy or the pheromone-based or an explicit cooperation and exchange of information like in the cooperative frontier-based will definitely increase the performance of the whole system.

Finally, we have provided an -as objective as possible- evaluation of our platform
and the game-based simulation approach in general based on the experience we have gained in this project.
7 Future Work

Due to the short time allowed for this project, we were not able to implement many of the ideas we wanted to include in the simulator, nor were we able to continue some of the ones we have implemented to the end, we -also- wanted to further study the strategies in more detail and conduct more simulations.

Regarding the platform implementation, we opted for the agent’s internal representation of the environment to be displayed in a distributed way, so that regardless of where the agent is running across the network, the simulator platform is able to view its internal representation, this would have been done using the KQML communication layer which we implemented, but we didn’t have enough time to use it for this purpose, this didn’t pose a problem since all the simulations were executed on the same machine. Further more, one of the tools we wanted to implement was one that simplifies the level (environment) construction, so that instead of building the level using the level editor, which can be a complex task for some, because it uses a 3Dimensional approach, the user of this tools could draw a simple 2D sketch of the environment, indicating open space areas and obstacles by coloring them differently, then our tool would take this simple 2D bitmap and convert it into a format used by the level editor. We believe that implementing this tools and continuing the previous implementation of the representation should be a straight forward task if more time was allowed.

Regarding the strategies we examined, we -also- wanted to study the effect of some of the parameters involved in more depth, for example, in our analysis of the pheromone-based strategy, we wanted to study further the effect of varying the life time of the pheromone-trail on the coverage time. Also, in the frontier-based exploration we wanted to study the effect of varying the interval after which the agents would exchange their information, on the total coverage time, further more, we wanted to see if we can find an alternative method for exchanging the reachability information between the agents, by replacing the waypoint graph with a simpler technique, we also thought of incorporating the pheromone-trails in the repel strategy and the frontier-based strategy to see if this could improve the performance. The problem is that the platform building took a significant time and the basic simulations for the strategies evaluation took what was left, and we had to stop at some point in order to be able to finish by the allowed time, nevertheless, we believe that since the platform is working now, these extra simulations shouldn’t have took us more than a week or two if we had that time.

One of the direct extensions for the frontier-strategy we where to implement, was taking the cooperation to the next level, the cooperation now was at the level of information exchange, this reduced a great deal of redundancy but didn’t eliminate it completely, eliminating more redundancy would have been possible by taking the cooperation to the frontier selection level, where each time the agent detects the current frontiers, it coordinates which frontier to go to after consulting the other agents (at least the ones in its range) so that if more than a frontier exist each agent selects a different one (for example the one nearest to its location), this will eliminate the possible case of two agents working on the same frontier if no coordination existed. This method was suggested as an extension to the original strategy in [31].
8 Bibliography

[1] Paul Davidsson, **Agent Based Social Simulation: A Computer Science View** *Journal of Artificial Societies and Social Simulation* vol. 5, no. 1.


[12] Alexander Repenning, Andri Ioannidou and John Zola, **AgentSheets: End-User Programmable Simulations**


[34] http://www.qeradiant.com/


[38] http://www.brook.edu/es/dynamics/models/ascape/default.htm


[40] http://www.gis.usu.edu/swarm/


