UFPE - UNIVERSIDADE FEDERAL DE PERNAMBUCO
CIN - CENTRO DE INFORMÁTICA

MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

“RTSCUP: TESTBED FOR MULTIAGENT SYSTEMS EVALUATION”

Vicente Vieira Filho

Dissertação de Mestrado

RECIFE, AGOSTO DE 2008.

UFPE - UNIVERSIDADE FEDERAL DE PERNAMBUCO
CIN - CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Vicente Vieira Filho

“RTSCUP: TESTBED FOR MULTIAGENT SYSTEMS EVALUATION”

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADOR: Dr. GEBER LISBOA RAMALHO
CO-ORIENTADORES: Drª. PATRÍCIA CABRAL DE AZEVEDO RESTELLI TEDESCO
Dr. CLAUIRTON DE ALBUQUERQUE SIEBRA

RECIFE, AGOSTO DE 2008.

Vieira Filho, Vicente
RTSCup: testbed for multiagent systems evaluation / Vicente Vieira Filho. - Recife: O Autor, 2008.
x, 82 p. : fig., tab., quadros

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2008.

Inclui bibliografia.

1. Inteligência artificial. I. Título.

006.3 CDD (22. ed.) MEI2009 - 143

Dissertação de Mestrado apresentada por **Vicente Vieira Filho** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"RTSCup: Testbed for Multiagent Systems' Evaluation"**, orientada pelo **Prof. Geber Lisboa Ramalho** e aprovada pela Banca Examinadora formada pelos professores:

Flávia de Almeida Barros

Profa. Flavia de Almeida Barros
Centro de Informática / UFPE

Prof. Evandro de Barros Costa
Departamento de Tecnologia da Informação / UFAL

Geber Lisboa Ramalho

Prof. Geber Lisboa Ramalho
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 29 de agosto de 2008.

Francisco de Assis Tenório de Carvalho

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Agradecimentos

Aos meus pais e irmãos, de forma especial, pela presença fundamental em cada momento de minha vida.

Aos meus mestres, orientadores e amigos Geber Ramalho, Patrícia Tedesco e Clauirton Siebra pela oportunidade de trabalharmos juntos e de contar com o seu apoio e dedicação em cada passo da construção deste trabalho.

A todos os meus amigos, dos mais antigos aos mais recentes, pela companhia e momentos vividos durante a pós-graduação. Em especial, a Aércio Filho, Afonso Ferreira, André Lima, Marco Túlio Albuquerque e Vilmar Nepomuceno pela presença sempre constante.

Aos alunos José Carlos, Renan Weber e Victor Cisneiros pelo apoio e contribuição direta no desenvolvimento desse trabalho e construção dos simuladores.

E finalmente, ao Centro de Informática e Universidade Federal de Pernambuco por terem me proporcionado um maravilhoso ambiente de estudo e pesquisa.

Resumo

A avaliação de sistemas computacionais é uma importante fase em seu processo de desenvolvimento, e isso não é diferente para sistemas que utilizam Inteligência Artificial (IA). Para esses sistemas, em particular, existe uma tendência à utilização de ambientes de simulação, conhecidos como testbeds, os quais auxiliam na avaliação desses sistemas em diferentes cenários de teste.

A área de concentração desse trabalho é Sistemas Multiagentes (SMA). Essa área de pesquisa encontra-se em fase de expansão devido aos SMAs estarem sendo empregados em problemas considerados difíceis ou até mesmo impossíveis de serem solucionados por um único agente ou por sistemas monolíticos. Além disso, vários problemas interessantes surgem durante a interação entre os agentes normalmente envolvendo a resolução distribuída de problemas em tempo real.

Atualmente existem vários testbeds utilizados na atividade de pesquisa na área de SMA tais como Trading Agent Competition, RoboCup Rescue e ORTS. Entretanto, a maioria desses testbeds não apresenta as características necessárias para auxiliar os pesquisadores na definição, implementação e validação de suas hipóteses.

Este trabalho apresenta um ambiente de simulação, chamado RTSCup, para ser utilizado como testbed para implementação de aplicações na área de Sistemas Multiagentes. O RTSCup já foi utilizado com sucesso em experimentos práticos durante competições realizadas entre estudantes da Universidade Federal de Pernambuco.

Palavras-chave: Benchmark, Testbed, Sistemas Multiagentes, Estratégia em tempo real, Jogos Eletrônicos

Abstract

The evaluation of any computational system is an important phase of its development process and this is not different to systems that use Artificial Intelligence (AI). For such systems, in particular, there is a trend in employing simulation environments as testbeds, which enable the evaluation of such systems in different test scenarios.

The area of interest of this work is Multiagents Systems (MAS). This area of research is booming because MAS are being used to solve problems which are difficult or impossible for an individual agent or monolithic system to solve. Besides, there are many interesting problems that arise during the interactions among agents which often involve distributed problem solving on the fly (real-time).

There are many testbeds used to evaluate Multiagent Systems such as Trading Agent Competition, RoboCup Rescue and ORTS. However most of these testbeds do not have the appropriate features to help researchers to define, implement and validate their hypothesis. Most features not addressed by simulators are related to usability and software engineering aspects.

The aim of this work is to present a simulator to Multiagent Systems, called RTSCup, which can be used as a testbed for evaluating several AI techniques used to implement teams as Multiagent Systems (MAS). RTSCup was already employed in a practical experiment during a competition among AI students of the Federal University of Pernambuco.

Keywords: Benchmark, Testbed, Multiagent Systems, Real-Time Strategy, Eletronic Entertainment

Content

List of Illustrations

List of Tables

Chapter 1

Introduction

This chapter describes the main motivations for this work, it lists the objectives sought in this research, and finally, it shows how this document is organized.

1.1 Motivation

The evaluation of any computational system is an important phase of its development process. These systems have many characteristics to be evaluated, such as robustness, reliability, efficiency and portability. In those systems the use of testbeds[3] is common to help in the activity of assessing the mentioned requirements.

This is not different for systems that use Artificial Intelligence (AI). For such systems, in particular, there is a tendency (Hanks et al 1993) in employing simulation environments as testbeds, which have the power to highlight interesting aspects of the system performance by enabling both, the conduct of controlled experiments to validate the hypothesis, and the analysis of the results in comparison to the state of the art. Besides that, the experimental control that can be achieved with testbeds can help us explain why systems behave as they do.

Another interesting tendency is the use of such environments in academic competitions (Stone 2003). This trend brings two advantages to AI research. First, competitions motivate the investigation of solutions in specific areas. Second, they integrate the research community, providing an environment to discuss approaches and analyze in a comparative way raised results from contests.

There are some AI areas which already have simulation systems to help in those activities such as Trading Agent Competition (TAC) - designed to promote and encourage research into the trading agent problem – and Open Real-Time Strategy (ORTS) – designed to promote research in the field of AI for computer games. Nevertheless, in other areas where those simulation systems are not yet available, the researchers have no options other than create themselves the environment to develop, evaluate and compare solutions.

The ORTS system brings another tendency in the AI research: the employment of computer game environments as simulation platforms. By definition, the game itself already is a container for the AI (game's logic) formed by many different pieces such as graphic, audio and physics. As a result, there are many other testbeds that use a game environment to simulate different AI problems, such as RoboCup Soccer, which one simulates a soccer game, and the Robocode, which one simulates a combat game among robots.

Despite all differences among them, these systems have a common feature: their focus of activity. Testbeds are usually developed to be used as an environment to simulate some specific

---

[3] Testbeds are the environments in which the standard tasks may be implemented. These tools provide a method for data collection, the ability to control environmental parameters, and scenario generation techniques providing metrics for evaluation (objective comparison) and to lend the experimenter a fine-grained control in testing systems.

AI problems, due the fact that AI area includes a large number of problems, such as planning, learning, perception, deduction, reasoning, knowledge representation, and so on.

The area of interest of this work is Multiagents Systems (MAS). This area of research is booming because MAS are being used to solve problems which are difficult or impossible for an individual agent, or monolithic system, to solve. Besides, there are many interesting problems that arise during the interactions among agents, which often involve distributed problem solving on the fly (real-time).

There are many testbeds used to evaluate Multiagent Systems. However most of these testbeds do not have the appropriate requirements to help researchers to define, implement and validate their hypothesis. Most requirements not addressed by simulators are related to usability and software engineering aspects. For example, the testbeds don't provide both a convenient way for researchers to vary the behavior of the world to generate different test scenarios or a simple procedure for developing interesting agents (Agent Development Kit[4]).

1.2 Objectives

The aim of this work was to develop a simulation environment to enable the creation and analysis of new SMA techniques, enable the evaluation of such systems in different test scenarios, and provide parameters for comparing competing MAS systems.

There are some specific sub-goals which must be achieved in order to reach the main objective presented above.

- Analyze problems involved in the research in SMA.
- Describe main features that a simulation environment must have to be used as testbed for SMA.
- Evaluate current testbeds used for research on SMA.
- Define features of the desired testbed to attend the students and researchers needs.
- Design an architecture to achieve the proposed feature list.
- Produce the planned testbed.
- Experiment and test the developed testbed under real situations.
- Evaluate the testbed applying the same method used to analyze the state-of-art solutions.

---

[4] A set of utility subroutines (programming library) to simplify the procedure for developing interesting agents for the simulation system.

The proposed simulator, named RTSCup, should take into account many requirements related to the benchmark, testbed, artificial intelligence, and software engineering areas, such as development and evaluation facilities Besides that, the choice of a simulation environment for electronic games helps minimizing the barriers to entry because of the natural stimulus of the area. The game area is much more motivating than areas such as rescue in large scale disasters (RoboCup Rescue) or manages multiple simultaneous auctions (TAC).

The electronic games area, however, has a lot of distinct game genres that differ from each other in many aspects, like environment settings, agent architecture and agents organization. Nevertheless, one of these genres stands out due to the large use of AI. This genre is known as Real-Time Strategy (RTS).

The AI systems used in RTS games are some of the most computationally intensive, simply because they usually involve numerous units that must struggle over resources scattered over a 2D terrain by setting up economies, creating armies, and guiding them into battle in real–time. In those games, as the name suggests, the strategy must be continuously defined and applied at every moment, in real time. Some examples of RTS games are Warcraft, Starcraft and Age of Empires.

The RTS games demand complex solutions due to the broad variety of problem types that they can tackle, such as pathfinding, resource allocation, action prediction, tactical and strategic support systems and agent coordination.

1.3 Methodology

The methodology adopted in this work to reach the presented objectives is showed below. The first step is obviously associated to the study of the state-of-art to analyze the testbeds for AI simulation. This analysis produced a detailed list of requirements and features that enables the comparison activity between simulators.

The next step uses the results of the above one to define the features of the desired testbed to attend the students and researchers needs. After the definition of the requirements document and the study of the most used architectures, it's time to start the testbed production.

The third step is the production of the proposed testbed. This step used the agile methods of development, which are composed by two main units of delivery: releases and iterations. A release consists of several iterations, each of which is like a micro-project of its own. Each release was experimented and tested on real competitions among students of the Intelligent Agents Course of the CIn/UFPE.

The forth and final step is the testbed's evaluation. In this phase the developed testbed was analyzed to verify if it meets the proposed requirements.

1.4 Document Structure

The remainder of this document is structured in 5 major parts.

Chapter 2 – Testbeds and Benchmarks in AI: This chapter presents the main difficulties encountered in the activities of research in MAS, shows concrete examples of initiatives in the direction of testbeds for MAS, and finally presents the RTS game genre as a great candidate to serve as a testbed environment.

Chapter 3 – Testbeds in MAS: This chapter presents the most popular testbeds used in the AI research and examines them in a comparative way based on certain parameters.

Chapter 4 – RTSCup: This chapter presents the project designed to create the proposed testbed in this dissertation called RTSCup. The simulator was set up considering the main deficiencies identified in the studied simulators.

Chapter 5 – Experiments and Results: This chapter presents the main experiments conducted with the use of RTSCup for validation of its characteristics and also the results achieved during the experiments.

Chapter 6 – Conclusions: This chapter presents the main project contributions and future work.

Chapter 2

Testbeds and Benchmarks in AI

This chapter presents the main problem addressed in this research work. It examines the difficulties encountered in the activities of research on AI and, more specifically, in the area of multiagent systems. It begins presenting the definition of the testbed and benchmark terms and their importance to the research in science, and more specifically in Artificial Intelligence and Multiagent Systems. Concrete examples of initiatives in this direction are then presented, followed by what is expected in terms of benchmarks and testbed for SMA. Additionaly, it also presents the current trends in the employment of computer game environments as simulation platforms. Finally, it presents the RTS game genre as good candidate to serve as a testbed environment.

2.1 Introduction

The research activity of any research area uses the scientific method to develop experiments in order to produce new knowledge, correct and integrate pre-existing knowledge.

In most science disciplines, the scientific method consists of the (1) definition of the problem; (2) creating a hypothesis; (3) conduct of controlled experiments to validate the hypothesis; (4) analysis of the results in comparison to the state of the art; (5) interpret results and draw conclusions that serves to the formulation of new hypotheses and, finally, (6) publication of the results achieved in scientific work (articles, monographs, dissertations, tests, etc.).

Most of the cited steps are similar to all science fields, except the steps for testing and analysis of the results accomplished. They are typically dependent with more intensity to the research area in question. For example, it is not difficult to imagine that testing in the discipline of Chemistry is different from testing in the discipline of Mathematics.

Similarly, the steps for testing and analysis in research on AI have peculiar characteristics. The evaluation of AI systems is a very difficult task because it involves complex activities, some of which can only be compared to human performance (e.g., music composition and text interpretation).

The experiments in this area involve the creation of intelligent systems (computer programs) for implementing the proposed hypothesis. The analysis of the hypothesis includes the evaluation of performance of the system in both: (1) absolute terms - how well the proposed solution fits the problem under study; and (2) comparative terms - in relation to results achieved by other approaches (state of the art).

The comparative analysis is a complex activity because of the difficulty in recreating the original experimentation’s environment (software and hardware) in an accurate way. In practice, what normally happens is the creation of systems to simulate both the proposed hypothesis and other approaches. Due to the complexity related to the creation and analysis of new AI techniques or systems, lots of research works are prematurely abandoned.

The current tendency to solve the problems above is the use of simulation environments [Hanks 1993], known as testbeds, which provide a way to conduct controlled experiments in order to validate the hypothesis.

2.2 Testbeds and Benchmarks

A benchmark is a standard task, representative of problems that will occur frequently in real domains. In the design of CPUs, for example, matrix multiplication is a good benchmark task because it is representative of an important class of numerical processing problems, which in turn is representative of a wider class of computational problems – those that do not involve significant amounts of I/O. The matrix multiplication problem can be described precisely and rigorously. Moreover, matrix multiplication is informative: it tells the CPU designer something interesting about the CPU, namely, its processing speed.

An early benchmark task for AI planning programs was the Sussman anomaly (the "Three Block Problem") [Sussman 1975]. The Sussman anomaly helped many researches elucidate how their planners worked. It was popular because, like matrix multiplication, it was representative of an important class of problems, those involving interactions among conjunctive sub-goals, and it was very easy to describe. As students and scientists, however, our interests are different. In these roles, we want to understand why a system behaves the way it does.

Understanding a system's behavior on a benchmark task requires a model of that task, so our goals as students and scientists will often be served only by benchmark tasks that we understand well enough to model it precisely. This is especially true for cases in which we expect a program to "pass" the benchmark test. Without a model of the task, it is difficult to see what has been accomplished: we risk finding ourselves in the position of knowing simply that our system produced the successful behavior – passing the benchmark.

Benchmarks ideally are problems that are both amenable to precise analysis and representative of more complex and sophisticated reality. Unfortunately, the current state of the field often elevates these problems to a new status: they become interesting for their own sake rather than as an aid in understanding a system's behavior on larger and more interesting tasks. There are few papers made explicit about connection between the benchmark problems and any other task. Without this additional analysis, it is difficult to say whether these problems are representative of others we presumably care about, and therefore exactly why the reported solutions are themselves interesting.

Benchmarks are problems that everyone can try to solve with their own system, so the definition of a benchmark cannot depend on any system-specific details, nor can the scoring criteria. The advantage of using a benchmark is that comparative analysis of performance is possible; different architectures are applied to the same task and the results obtained by of each one are measured against others.

The problem with benchmarks is that they encourage focusing on the benchmarking problem, instead of the real-world task, and it may be unconsciously affected by their designers.

In other words, the benchmarks should come from people who do not have an investment in the results of systems applied to the benchmark. Another problem with benchmarks, from the standpoint of AI, is that there are really no standard tasks for AI problems. However, several benchmarks have been proposed in AI, based on their recurrence. These include the Yale Shooting Problem [Hanks 1987] and Sussman's anomaly.

The potential mismatch between benchmark scores and performance on real tasks is also a concern for researchers who are developing testbeds. Testbeds are the environments in which the standard tasks may be implemented. In addition to the environment itself, these tools provide a method for data collection, the ability to control environmental parameters, and scenario generation techniques. Thus, the purpose of a testbed is to provide metrics for evaluation (objective comparison) and to lend the experimenter a fine-grained control in testing agents.

Benchmarks and testbeds do not currently bridge the gap between general and specific problems and solutions. The gap exists between the benchmark proposed by Sussman and others, domain-specific problem that others researchers care about.

2.3 Testbeds in MAS

The use of testbeds as evaluation platform for multi-agent systems has become a common practice for AI systems. In fact, this platform is able to evaluate the performance of such systems and provide a basis for comparative analysis. However, some criticisms have been directed against the use of testbeds.

First, such platform offers an empirical type of evaluation users have to distinguish the important evaluation events, as well as interpret the results of such events. Second, there is not a consensus about what a representative testbed domain is. Finally, results from testbed experiments are not general. Rather, they are related to a subgroup of possibilities from the total set of scenarios.

Simulators are a particular type of testbed, in which generation of new states is executed in runtime, and such states depend on the activities performed within the environment. In this way, final results are commonly unpredictable. An interesting trend related to simulators is academic competitions. Besides motivating research and integrating the scientific community, competitions determine deadlines to the creation of functional systems and periodic events. Using the same platform, it enables the improvement of past systems and their approaches.

One of the main competitions related to multi-agent systems is the RoboCup Rescue (RCR) [Kitano 2001]. The intention of the RCR project is to promote research and development in the disaster management domain at various levels involving multi-agent team work coordination, physical robotic agents for search and rescue, information infrastructures, personal digital assistants, a standard simulator, decision support systems, evaluation benchmarks for rescue strategies and robotic systems. In RCR, it is possible to evaluate the performance of a particular solution (hypothesis) using the parameters provided by the simulation, such as number of civilians affected by the disaster and number of damaged buildings.

This problem introduces researchers into advanced and interdisciplinary research themes. As AI/robotics research, for example, behavior strategy (e.g., multi-agent planning, real-time/anytime planning, heterogeneity of agents, robust planning, mixed-initiative planning) is a challenging problem. For disaster researchers, RCR works as a standard basis in order to develop practical comprehensive simulators adding necessary disaster modules.

This competition uses a real-time distributed simulation system [Kitano 2001, RCR 2001] composed of several modules, all of them connected via a central kernel. The goal of this system is to simulate the effects of earthquakes in urban areas. For this purpose, each type of event related to an earthquake, such as building collapse or fire spreading, is simulated by a dedicated module, while a Geographic Information System (GIS) [Kitano 2001] provides the initial conditions of the simulated world.

RCR is an appropriate example of benchmark for multi-agent research because it implements several of the requisites that such systems need. They are:

- Agents do not have control on the environment because their actions are not the unique events that can change it. There are lots of units in a RCR simulation acting at the same time within the environment.
- Agents are not able to ensure that a sequence of actions will lead to a specific state, or whether these actions are valid because changes can happen over the environment in between decision and its execution;
- RCR environments are complex and each of their objects presents several attributes whose values can affect the simulation in progress;
- The environment considers communication and coordination among agents as an important simulation issue. There are specific rules to control such communication. There are two different message actions, “say” and “tell”, that can be perceive by agents within a specific radius size.
- There are several ways to measure the efficiency of approaches, for instance, number of victims or total area on fire.

- The RCR simulator has a well defined temporal model, which is based on configurable (time) cycles. In each cycle, the agents must receive their perception, define their actions and then act upon the environment.

A last and important RCR feature is its high level of parameterization, which enables an evaluation of multi-agent systems considering a significant variety of problems and conditions. In this way, RCR users can configure the environment to simulate some particular aspect.

Besides RCR, there are several other testbeds created to foster research in specific areas of AI, like the Trading Agent Competition (TAC) [TAC 2001, Stone 2003] - designed to promote and encourage research into the trading agent problem – and Open Real-Time Strategy (ORTS) [ORTS 2003, Buro 2003] – designed to promote research in the field of artificial intelligence for computer games.

The ORTS brings another tendency in the AI research: the employment of computer game environments as simulation platforms. By definition, the game itself already is a container for the Artificial Intelligence (game's logic) formed by many different pieces such as graphic, audio and physics.

There are many other testbeds that use a game environment to simulate different AI problems, such as RoboCup Soccer [Noda 1996, Stone 2003], which simulates a soccer game, and the Robocode [Robocode 2001], which simulates a combat game among robots.

2.4 Real-Time Strategy Games

Strategy video games [Rollings 2006] focus on gameplay requiring careful and skillful thinking and planning in order to achieve victory. In most strategy video games, the player is given a godlike view (top-down perspective) of the game world, indirectly controlling the units under his/her command.

The origin of strategy games is rooted in board games. Strategy games (Checkers, Chess, Chinese checkers, Go and Masterminds) instantiated on computers generally take one of four archetypal forms [Adams 2006], depending on whether the game is turn-based or real-time, and whether the game's focus is upon military strategy or tactics.

The term "real-time" indicates that the action in the game is continuous, and players will have to make their decisions and act within the backdrop of a constantly changing game state. The turn-based term, on the other hand, is usually reserved to distinguish them from real-time computer strategy games. A player of a turn-based game is given a period of analysis before committing to a game action.

Strategy involves the big picture – the overall plan designed to achieve a particular goal. It is characterized by its nature of being extensively premeditated, and often practically rehearsed. Strategies are used to make the problem easier to understand and solve. In contrast to strategy, tactics involves the art of finding and implementing means to achieve particular immediate or short-term aims.

The word "tactics" is borrowed fromthe military jargon. It originally refers to a conceptual action used by a military unit (of no larger than a division) to implement a specific mission and achieve a specific objective, or to advance toward a specific goal. A tactic is implemented as one or more tasks. These concepts can be defined as a hierarchy (Illustration 2.1).

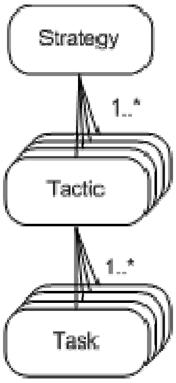


Illustration 2.1: Relationship among strategy, tactic and task.

A Real-Time Strategy (RTS) video game is a strategic game that is distinctly not turn-based and involves both military strategy and tactics. Gameplay[5] generally consists of the player being positioned in the map with a minimal production base capable of creating the basic units and buildings that are needed to start playing. Later, players progress to create increasingly powerful units and buildings, or a small force, the core of which is generally a unit capable of establishing the initial production base.

Thereafter, the game is typically a race of resource gathering, technology research and unit production to claim territory, and suppress and defeat the opponents through force or attrition. Some examples of RTS games are Warcraft [Warcraft 1998], Starcraft [Starcraft 1994] and Age of Empires [Age 2005] (Illustration 2.2).

---

[5] Player experiences during the interaction with games.

Illustration 2.2: Screenshot of the game Age of Empires III .

Each player in a RTS game may interact with the game independently of other players, so that no player has to wait for someone else to finish a turn. The real-time games are more suitable to multiplayer gaming, especially in online play, compared to turn-based games.

Some important concepts related to real-time strategy include combat- and twitch-oriented action. Other RTS gameplay mechanics implied are resource gathering, base building and technological development, as well as abstract unit control (giving orders as opposed to controlling units directly) [Schwab 2004].

The tactic refers to when a player's attention is directed more toward the management and maintenance of his or her own individual units and resources. It involves the use of combat tactics too. This creates an atmosphere in which the interaction of the player is constantly needed. On the other hand, strategy refers to when a player's focus is more directed towards economic development and large-scale strategic maneuvering, allowing time to think and consider possible solutions.

2.4.1 AI in RTS

A RTS game is a specific, but very attractive, real-time multi-agent environment from the point of view of distributed artificial intelligence and multi-agent research. If we consider a RTS team as a multi-agent system, several interesting issues will arise.

In a game, we can have two or more competing teams. Each team has a team-wide common goal, namely, to win the game. The opponent team can be seen as a dynamic and obstructive environment, which might disturb the achievement of this goal. To fulfill the common goal, each team is required to complete some tasks, which can be seen as sub-goals (e.g., foraging, exploration, combat, and resource allocation). To achieve these sub-goals, each team must present a fast, flexible and cooperative behavior, considering local and global situations.

The team might have some sort of global (team-wide) strategies to accomplish the common goal, and both local and global tactics to achieve sub-goals. Furthermore, the team must consider the following challenges:

- **The game environment**

The game environment is obviously highly dynamic due the number of agents in the game. The agent has no control of the environment whose status can change at the same time as the agent acts. For example, at the same time that a miner agent collects resources, another agent could do a movement or attack a building.

Besides that, the number of different states which the environment can reach is immense. A unit in the RTS game can move in different directions and can reach many diverse positions. These features qualify the game environment as continuous.

- **The perception of each agent may be locally limited**

The agent's perception could have two different states. The first one is when the agent perceives the complete environment status at every time moment. In this case the environment is defined as completely observable. The second case happens when the environment is not completely available to the agent's sensors (i.e., fog of war) the environment is defined as partially observable.

- **The agents' coordination**

To accomplish the game goals the agents must coordinate their actions. The coordination is required to determine organizational structure amongst a group of agents and for task and resource allocation. Furthermore, the following characteristics i.e. communication, negotiation and planning have great importance as well.

Agents must communicate with one another to exchange information and knowledge, by which they can solve cooperatively common problems. In addition, negotiation is required for the detection and resolution of conflicts.

In a combat game, for example, it is necessary to mine as many resources as possible to enable the training of military units. The team's units must define a common strategy to reach the victory.

- **Strategic and tactic decisions**

There are lots of sub-goals which must be achieved to reach victory, such as mine resource patches, construct buildings, train units (civil and military) and evolve civilization (tech tree). Each of them to be accomplished needs a strategy and possibly one or more tactics.

For example, the mining activity involves the definition of a manner (strategy) to search for resource patches and to allocate the mineworkers to maximize results. On the other hand, to successful perform the mentioned actions it is necessary to define search (i.e. ant based algorithms) and motion (i.e. pathfinding and collision avoidance) tactics.

- **The role of each agent can be different**

In a RTS game there are lots of specialized units such as priest, villager, swordsman, clubman and bowman with different attributes and characteristics: villager collects resources, swordsman attacks and so on.

In brief, based on these issues, a RTS team can be seen as a cooperative distributed real-time planning scheme, embedded in a highly dynamic environment. Besides, a RTS game can be viewed as MAS considering the following main features of MAS [Weiss 1999]:

- **Each agent has just incomplete information and is restricted in its capabilities**

As already mentioned, the perception of each agent could be locally limited - environment is not completely available to the agent’s sensors. Besides that, each agent has a specific role (i.e. archer, cavalry and so on) with different capabilities. For example, a civil unit has no attack ability; likewise a military unit has no mine faculty.

- **System control is distributed**

There is no central control (i.e. controlling agent) in a RTS game because all agents are autonomous. Each agent receives information from the environment by its sensors, and acts on it, over time, in pursuit of its own agenda. This agenda evolves from drives (or programmed goals). The agent acts to change the environment and influences which is sensed at a later time. It means that agents must interact and cooperate with each other to solve common goals.

- **Data is decentralized**

All the environment information is stored in different independent systems causing the establishment of hardly interconnections among agents. It is difficult for agents to get a complete overview of the content (environment status) and therefore to observe and further analyze their overall situation.

- **Computation is asynchronous**

This means that agents are AI systems whose execution can proceed independently. So they do not need to process information (sense) simultaneously as others do it or receive a message concomitantly as sender transmit it.

It is also important to remember that the research in MAS area is increasing due to the fact that those systems can be used to solve problems which are difficult or impossible for an individual agent or monolithic system to solve.

2.4.2 RTS Games as Testbed for SMA

The use of benchmarks to evaluate MAS has received several criticisms [Hanks 1993], which are mainly based on the fact that such systems are implemented to be used in real situations. In this way, independently of the specification level of a benchmark, it will still present a limited number of situations that could happen in real scenarios.

There are cases, however, in which realism is not the main requirement to be considered. In such situations, the focus could be on the comparative evaluation among different approaches. For example, benchmarks currently used in the Planning Systems Competition (PSC) and the Trading Agent Competition (TAC) corroborate this idea.

Other kind of competition, which is recently receiving more attention from the research community, is related to Real-Time Strategy (RTS) games. Note that the main competition in this area (ORTS Competition) does not have the same maturity than other AI competitions. However, several benefits can already be observed, such as the creation of a community with common interests in the investigation of RTS problems.

One of the main advantages of using RTS environments as testbed is the broad variety of problem types that they can generate [Schwab 2004]. For example, consider a classic RTS battle game between two teams. Some of the AI problems that can be investigated in this case are:

- **Pathfinding**

Teams need to move along the best routes so they decrease time and effort. It is common more elaborated versions of this problem, such as involving routes along unknown lands or facing dynamic obstacles (e.g., enemy teams).

- **Patrolling**

A team can keep a specific area on control, or cover an area to find resources or enemies in an optimized way.

- **Resource allocation** (scheduling)

Each component of a team is a resource that must be allocated in some activity, such as defense or attack. Allocation processes must observe, for example, the *load balancing* of activities, specific resources do not become overloaded while others are free.

- **Action prediction**

A team can try to observe and model the behavior of others; it predicts their behaviors and can anticipate them.

- **Coordination**

Components of a team ideally need some kind of coordination to improve the whole work and they do not disrupt each other. For example, during an attack maneuver the team needs to decide if its members will attack the flanks, or if the infantry should wait by the intervention of the artillery before moving forward.

- **Deliberative and reactive architectures**

Each team needs to deliberate on the strategies to be deployed in a battlefield. However, several reactive actions, such as reactions to eventual ambushes, must also be considered.

- **Strategy and tactic decisions**

Each team must plan and conduct the battle campaign (strategy) in the same way that must organize and maneuver forces in battlefield to achieve victory (tactics).

Then, RTS environments enable a wide set of problems and situations in which it's possible to apply AI techniques. Ideally, these environments must be configurable following the RCR model [Morimoto 2002] and then users can create more appropriate scenarios to each kind of problem.

Besides that, the video game area is notably more motivating than other areas, like search and rescue in large scale disaster (RCR) or manage multiple simultaneous auctions (TAC). There are many studies and surveys indicating that people enjoy video games because they are satisfyed at a fundamental psychological level [Ryan 2006]. This fact breaks the barriers of entry facilitating the admission of students and researchers beyond the creation of communities to discuss about AI problems and benchmarks. The use of the RTS genre as a backdrop to research in IA, and more specifically in MAS, creates the ideal environment to motivate the use of simulation in classrooms. This assumption has already been proved in the many competitions held among students of the Intelligent Agents Course of the CIn/UFPE.

The students could learn various concepts, like agents' type, architecture and environment; multiagent systems and societies of agents (organization, communication, planning and negotiation); and learning in multiagent systems (reinforcement, supervised, unsupervised and semi-supervised learning).

Besides the academic appeal, there is yet the market appeal. A RTS testbed can be applied to research because the AI performance in RTS games is lagging behind developments in related areas such as classic board games. The main reasons are the following:

- **RTS game worlds have many agents, incomplete information, micro actions, and fast-paced action**

By contrast, World–class AI players mostly exist for slow–paced, turn–based, perfect information games in which the majority of moves have global consequences and human planning abilities therefore can be outsmarted by mere enumeration.

- **Market dictated AI resource limitations**

Up to now, popular RTS games have been released solely by games companies, which naturally are interested in maximizing their profit. Because graphics is driving games sales and companies strive for large market penetration only about 15% of the CPU time and memory is currently allocated for AI tasks. On the positive side, as graphics hardware is getting faster and memory getting cheaper, this percentage is likely to increase – provided game designers stop making RTS game worlds more realistic.

- **Lack of AI competition**

In classic two–player games, tough competition among programmers has driven AI research to unmatched levels. Currently, however, there is no such competition among real–time AI researchers in games other than computer soccer. The considerable man–power needed for designing and implementing RTS games, and the reluctance of games companies to incorporate AI APIs in their products are big obstacles on the way towards AI competition in RTS games.

It is possible to observe the direct financial impact on the study and research for solutions to the problems exposed by the existence of a large industry able to absorb and incorporate the positive results achieved.

2.5 Conclusions

This chapter presented the problem of carrying out research on AI without the use of an appropriate environment for simulation. Moreover, it showed the main solutions currently adopted to circumvent this problem. One of the solutions presented is the use of games as simulation environments, because these applications are already a container for artificial intelligence.

However, the area of games is too broad, and has many genres with totally different gameplay. In this case it is essential to define a specific genre to be used as simulation environment to study MAS problems. Among these genres of games, one stands out because of the enormous volume and complexity of MAS problems involved: the RTS genre. The RTS games have several characteristics that make this genre a great candidate for an MAS testbed, and not only to study and research common multiagent problems, but also single agent problems.

Chapter 3

Testbeds in MAS

This chapter presents the most popular testbeds used in the MAS research and examines them in a comparative way, based on certain parameters. These criteria describe the main features in terms of benchmark, testbed, artificial intelligence and software engineering that testbeds should exhibit. For didactic reasons, it is presented first the mentioned comparison parameters. After that, it is presented the main testbeds in the multiagent systems area and more specifically in the RTS area. Finally, these systems are analyzed under the chosen features and compared to each other, to define their strengths and weaknesses.

3.1 Introduction

The current tendency of using simulation environments as a platform to experiment new techniques resulted in the emergence of various systems set up to help in the AI research. Among the best known and used testbeds in the scientific community, we found the following ones:

- **Robocode**

Robocode [Robocode 2001] is a programming game, originally provided by IBM, where the goal is to code a small automated 6-wheeled robot to compete against other robots in a battle arena until only one is left. Robots move, shoot at each other, scan for each other, and hit the walls (or other robots) if they are not careful.

- **RoboCup Soccer**

The soccer game was the original motivation for RoboCup [Robocup 1993]. Besides being a very popular sport worldwide, therefore appropriate to attract people to the event, it brings a significant set of challenges for researchers: (1) collective game, for which more than one agent/robot is required to play; (2) individualistic (each agent/robot must identify relevant objects, self-localize, dribble) and cooperative (passes, complementary roles) elements; and (3) dynamic and adversarial environment, with moving objects, some of them rational agents that are playing a game against your team.

- **RoboCup Rescue**

Motivated by large earthquakes in Kobe (Japan) in 1995 and in Foligno (Italy) in 1997, appeared in the 2001 edition of the RoboCup event a new integrated competition, the RoboCup Rescue [Kitano 2001]. The main objective of the project is to promote research and development of solutions to this area, which leads to an increasing improvement of solutions of multi-agent teams coordination (heterogeneous agents), robotic solutions of search and rescue and so on.

- **Trading Agent Competition**

The Trading Agent Competition (TAC) [TAC 2001] is an international forum designed to promote and encourage high quality research into the trading agent problem. In the TAC shopping game, each AI agent is a travel agent, with the goal of assembling travel packages. Each agent is acting on behalf of eight clients, who express their preferences for various aspects of the trip. The objective of the travel agent is to maximize the total satisfaction of its clients (the sum of the client utilities).

According to the presented description, it is easy to observe the main differences in terms of area of expertise, namely, the problem under study. RoboCup Soccer, for example, tackle problems from the strategy to be used (organization of the team in the field, tactical scheme, among others) to issues such as movement handling and collision avoidance between entities in simulation. In contrast, the simulation in TAC is carried out through auctions which run according to a high-level protocol with specific and well-defined rules.

In spite of the differences among testbeds, it is possible to observe many similar aspects, such as the existence of a viewer to watch the simulation status, the possibility to change simulation's parameter to modify the test scenario and the ease to learn and use (usability). Those aspects are interesting to be used to define the main features which characterize a good testbed and to analyze those ones in a comparative way. The parameters are presented in the next section.

3.2 General Requirements for MAS Testbeds

This section describes the most significant research issues in the development of intelligent systems, and presents corresponding features that testbeds should exhibit.

For didactic reasons, the parameters of evaluation and comparison of testbeds is presented in the following order. Initially it will be introduced the most important parameters of benchmarks which assist in the improvement of its strengths and in the mitigation of the criticisms associate to the benchmarks.

Later it is presented the most significant parameters of a testbed which assist in the research activities particularly in the creation and analysis of new intelligent systems techniques in different test scenarios.

Then it is presented the important parameters related to research in AI considering the theme of the RTS game adopted. In this case, it is assessed yet the parameters for research specifically in the area of multiagent systems. And finally, the testbeds are evaluated from the standpoint of software engineering.

3.2.1 Benchmark Requirements

Benchmarks and testbeds have a close relationship, as already explained in the chapter 2. On one side, benchmarks are problems that are both amenable to precise analysis and representative of more complex and sophisticated reality; architectures are applied to the same task and the results of each measured against others.

On the other hand, testbeds are the environments in which the benchmarks may be implemented to provide metrics for evaluation (objective comparison) and to lend the experimenter a fine-grained control in testing agents. Both have complementary characteristics and together they create the ideal environment for research in AI [Stone 2003, Cohen 1995, Hanks 1993].

- **Ease to understand problems**

The complexity of the problem influences on the ability to model the problem and propose approaches to solve it. In a simple way, the ease of understanding of the problem is directly related to the ease design and construction of hypothesis and solutions. Thus, the barriers to entry are minimized.

- **Allows to draw valid conclusions**

This means that it is possible to understand why a system behaves the way it does based on information gathered from the testbed.

- **Seductive problem**

The motivational aspects of the problem depend on its type and area. For example, a problem related to a domestic activity (e.g. cooking and dish washing) is probably less motivating than problems found in electronic games. For this reason, the choice of the problem is an important step to ensure the acceptance of the scientific community and to integrate researchers in the investigation of solutions to the proposed problem.

- **Problem can be decomposed into sub-problems**

The problem's ability to decompose itself into sub-problems enables the application of the divide and conquer paradigm in the research activity. The researcher could break down a problem into two or more sub-problems identical (or related), until these become simple enough to be solved directly. The solutions to the sub-problems are then combined to give a solution to the original problem (research hypothesis).

3.2.2 Testbed Requirements

The main purpose of testbeds is to support the implementation of ideas so that they can be evaluated in a useful context. To achieve this objective the testbed should have some particular features to facilitate the creation of new knowledge and techniques of AI in different scenarios of tests. Moreover, the simulator must provide metrics to assist in the activity of comparison between the different approaches adopted.

A testbeds provide the means to implement these ideas, and can provide support facilities to elaborate hypothesis about a particular system characteristic, such as new coordination algorithm or negotiation protocol. With this purpose in mind, most testbeds should provide three classes of facilities extending the analysis structure of multi-agent simulation platforms [Marietto 2002] which are presented below.

Domain Facilities

The testbed should provide a way to represent and simulate the problem being solved.

- **Domain Independence**

A researcher using an abstract simulator (not domain-dependent) may be able to build a system based on what he or she judges to be "general problem-solving principles", but then the difficulty is in establishing those principles apply to any other domain.

In contrast, a researcher using a domain-dependent simulator may be able to demonstrate that his program is an effective problem solver in that domain, but may have difficult to conclude that the architecture is efficient for dealing with other domains.

This feature mitigates the impacts of some critiques about the use of benchmarks such as that the results from benchmarks are not general or that they encourage focusing on the benchmarking problem instead of the real-world task.

- **Supporting Experimentation**

Controlled experiments require problems and environmental conditions be varied in a controlled fashion. A testbed should therefore provide a convenient way (built-in set of parameters that characterize the environment's behavior) for the researcher to vary the behavior of the worlds in which the agent is to be tested (e.g. map editor). In this way, many different test scenarios can be created by varying the world's parameters systematically. This feature means that the simulator enables a flexible definition of test scenarios.

Development Facilities

The testbed should provide an environment and a set of tool for building intelligent agents in a simple and straightforward way to solve the problem under study.

- **Agent development Kit**

The purpose of the Agent Development Kit (ADK) is to simplify the procedure for developing interesting agents for the simulation system. This purpose is two-fold. First, it is to abstract away many of the gritty details of the simulation environment so agent developers can focus on agent development. Second, it is to provide a toolbox for building high-level behaviors.

- **Clean Interface**

It is important to maintain a clear distinction between the agent and the world in which the agent is operating. The natural separation is through the agent's sensors and effectors, so that interface should be clean, well defined, and sufficiently documented. A designer must be able to determine easily what actions are available to the agent, how the actions are executed by the testbed, and how information about the world is communicated back to the agent.

- **Well Defined Model of Time**

Testbeds must present a reasonable model of passing time in order to simulate exogenous events and simultaneous action, and to define clearly the time cost of reasoning and acting. This is a general problem in simulation and modeling. On the other hand the testbed must somehow be able to communicate how much “simulated time” has elapsed. Making experimental results requires a way to reconcile the testbed’s measure of time with that sued by the agent.

Evaluation Facilities

Tools for display and analyze the simulation behavior to understand how well the agents perform.

- **Viewer**

The viewer is an important component of the testbed system which enables the visualization of the current simulation status. This component is the main output of the testbed and shows exactly what is happen in the simulated environment. Without this tool the researcher could not see and analyze the behavior of his solution.

- **Automatic Report**

This feature is important to any simulation tool that is being used as a testbed. The generation of reports enables, for instance, the evaluation of solutions in a post-simulation stage, as well as a better basis for measure the efficiency of approaches. Besides that, it is important that the generated reports could be comprehensible and configurable.

3.2.3 AI Requirements

The testbeds for simulation of intelligent systems, especially multiagent systems, should be able to simulate the different properties of the environment. This section presents these properties.

Environment Features

Despite the broad variety of environments that may arise in AI, it’s possible to identify a reduced number of dimensions to be used in the environments categorization. Those dimensions are presented below [Russel 2003].

- **Completely Accessible vs. Partially Accessible**

An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state. An environment is completely accessible if the agent's sensors percept all relevant aspects needed for the action's choice. Those environments are convenient because the agent do not need to maintain any internal state to control the world.

On the other hand, an environment could be partially accessible due to noise or imprecise sensors or because parts of the state are simply absent in the sensor data. The more accessible an environment is, the simpler is to build agents to operate it.

- **Deterministic vs. Non-Deterministic**

A deterministic environment is one in which any action has a single guaranteed effect – there is no uncertainty about the state that will result from performing an action. In other words, if the next environment state is completely determined by the current state and the action performed by the agent, then the environment is deterministic; otherwise the environment is non-deterministic.

- **Episodic vs. Non-Episodic**

In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performances of an agent in different scenarios. Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to perform based only on the current episode – it need not reason about the interactions between this and future episodes.

- **Static vs. Dynamic**

A static environment is one that can be assumed to remain unchanged except by the actions performed by the agent. A dynamic environment is one that agents do not have control on the environment and their actions are not the only events that can change it.

Besides, agents are not able to ensure that a sequence of actions will lead to a specific state or if these actions are valid because changes can happen over the environment between decision and its execution.

- **Discrete vs. Continuous**

An environment is discrete if there are a fixed, finite number of actions and percepts in it. A chess game is an example of discrete environment, and a taxi driving is an example of a continuous one.

- **Single Agent vs. Multiagent**

The difference between single agent and multiagent environments may seem very simple. For example, an agent solving a crossword puzzle by itself is obviously in a single agent environment; moreover, an agent playing chess is in a multiagent environment with two agents.

The problems related to multiagent environments are very different from the problems related with single agent environments. For instance, the activities like communicate, coordinate or cooperate are associated only with multiagent systems.

The main features that characterize an environment as a multiagent one [Weiss 1999] are described below:

- Multiagent environments provide an infrastructure specifying communication and interaction protocols.
- Multiagent environments are typically open and have no centralized designer.
- Multiagent environment contain agents that are autonomous and distributed and may be self-interested or cooperative.

3.2.4 Software Engineering Requirements

The testbed as a product must have quality and to assess it in the light of contemporary software engineering discipline it's necessary to evaluate several different features in terms of the product and the process of development. Below are presented some of the qualities that can be evaluated [Sanchez 2007].

- **Correctness**

Software needs to work properly. Correct software is one that meets specification (the users and organization requirements) and that has no flaws or errors.

- **Robustness**

The software should be able to prevent unexpected actions from users and to resist these unusual situations without any failures.

- **Reliability**

The reliable and robust software gains the confidence of users since it behaves as expected and not fails in an unexpected situation.

- **Efficiency**

The software should perform its tasks in an appropriate time to its complexity. The use of the resources of hardware (memory, disk, network traffic) should also be done efficiently.

- **Usability**

The software should be easy to learn and use, allowing the user greater productivity, flexibility of use, flexibility of application and provide satisfaction of use. An important issue to reach those requirements is to provide a good documentation and support to the research community.

- **Maintainability**

All software needs maintenance usually to correct errors or meet new requirements. The software should be easy to maintain so that these corrections or updates are made successfully.

Another concept related to this topic is the system's modularity. This software design technique increases the extent to which software is composed from separate parts, called modules. Conceptually, modules represent a separation of concerns, and improve maintainability by enforcing logical boundaries between components.

- **Evolvability**

All software needs to evolve to meet new requirements, to incorporate new technologies or for expansion of its functionality.

- **Portability**

Each researcher has a tendency to use some specific hardware equipment and operation systems on his research activities. For example, a researcher could choose a PC running Windows while another one could choose an Apple Computer running the Mac OS. In addition, these researchers could yet choose different programming language to develop their experimentations.

Consequently, the testbed should be able to run in the widest possible range of hardware equipment and software platforms and support the greatest possible number of programming languages.

- **Interoperability**

The software on different platforms should be able to interact with each other. This feature is essential in distributed systems since the software could be running on different computers and operating systems. It is interesting that different elements of different software can be used in both.

- **Security**

All systems with incomplete information at competitive level on the internet or LAN demand that information can be physically hidden from clients. To see this one just needs to imagine a commercial poker server that sends out all hands to each client. Hacking and losing business is inevitable.

Translated into the RTS world, information security rules out client-side simulations in theory. Regardless of how clever data is hidden in the client, the user is in total control o the software running at his side and has many tools available to reverse engineer the message protocol and information during a game. A testbed system should take care of this issue to avoid hack strategies.

3.3 MAS Testbeds

This section examines the main AI testbeds using the provided criteria. For didactic reasons, it only presents the testbeds which simulate the same AI domain, RTS Games, both for not comparing testbeds of completely different areas as well as for turning this section unnecessarily large.

These testbeds are evaluated against the presented requirements. The performance of the testbed in a specific feature is measured considering the following grades (Table 3.1).

Table 3.1: Available evaluation grades

Grade	Description
1 = Yes	Testbed meets the feature.
2 = No	Testbed doesn't meet the feature.
3 = Partly	Testbed partially meets the feature.

3.3.1 Glest

Glest [Glest 2007] is a free 3D real-time strategy game, in which the user controls the armies of two different factions: Tech, which is mainly composed of warriors and mechanical devices, and Magic, that prefers mages and summoned creatures in the battlefield. Glest is not just a game, but also an engine to make strategy games, based on XML and a set of tools.

The engine can be customized in two different ways:

- **Simulation Configuration**

Every single unit, building, upgrade, faction, resource and all their properties and commands are defined in XML files. Just by creating or modifying the right folder structure and a bunch of XML files, the game can be completely modified, and entire new factions or tech trees can be created

- **Scenario Configuration**

The Glest project provides a map editor to enable the creation of different maps in a simple and ease way.

Therefore researchers can create games using the Glest engine to evaluate their hipothesys in different test scenario by varying the world's parameters (simulation and map) systematically.

3.3.2 Stratagus

Stratagus [Stratagus 2007] is a free cross-platform RTS game engine used to build both multiplayer online and single player offline games. Stratagus is based on peer-to-peer (P2P) technology, which in the context of RTS games means that every player is running the entire simulation and therefore has access to all state information, including enemy locations. So, whatever software the players connect, client-side hacks are possible. Stratagus is configurable and can be used to create customized games and some of these games, such Wargus, are been used as a platform for AI studies.

Wargus is a Warcraft2 Mod that allows users to play Warcraft2 with the Stratagus engine, as opposed to play it with the original Warcraft2 one. There are three main reasons for this. First, it allows users to (1) play Warcraft2 under GNU/Linux and other operating systems not supported by the original Warcraft2 engine. Secondly, it permits users to play over the internet. And finally, it grants users to tweak numerous parameters so it is possible to play around with different unit strength and such. Whereas stratagus is a game engine, the researchers can use the Stratagus in the same way as Glest.

3.3.3 ORTS

Open Real-Time Strategy (ORTS) is a simulation environment for studying real-time AI problems such as pathfinding, reasoning with imperfect information, scheduling and planning in the domain of RTS games. Users of ORTS can define rules and features of a game via scripts that describe several types of units, buildings and interactions. The ORTS server is responsible for loading and executing such scripts, sending the world state to their clients. These clients are applications that generate actions to be performed over the simulation server, according to their objectives and the current state of the world.

The ORTS competition comes up in cooperation with the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE). There are four different contests in this competition:

- **Game 1**

Agents must develop strategies to perform resource gathering. The goal is to collect the maximum amount of resources within 10 minutes;

- **Game 2**

Tank agents must destroy as many opponent buildings as possible within 15 minutes. However, at the same time, they must also defend their home bases;

- **Game 3**

Aside to the resources gathering and battle actions, agents must also deal with resource management, defining if the team should collect more resources or spend it. The decision making process involves, for example, actions to create more units or attack/defend specific positions. The predominant goal is to destroy all opponent buildings within 20 minutes.

- **Game 4**

Marine agents must destroy as many opponent units as possible within 5 minutes.

3.4 Comparative Analysis

This section presents a comparative analysis of the cited simulators, identifying their strengths and weaknesses on both individual and global way. The Table 3.2 shows the requirements of each simulator as well as an indicator of performance that express the percentage of covered items.

Table 3.2: Comparative analysis among testbeds.

Criteria	Item	Glest	Stratagus	ORTS
Benchmark Requirements	Easy to Understand	2	2	3
	Valid Conclusions	2	2	3
	Seductive Problem	1	1	1
	Decomposable Problem	1	1	1
Subtotal		**50%**	**50%**	**75%**
Testbed Requirements	Domain Independence	1	1	1
	Supporting Experimentation	1	2	1
	Agent Development Kit	2	2	2
	Clean Interface	2	2	2
	Well Defined Model of Time	2	2	1
	Viewer	3	3	3
	Automatic Report	2	2	2
Subtotal		**36%**	**21%**	**50.0%**
AI Requirements	Partially Accessible	2	2	1
	Non-Deterministic	1	1	1
	Non-Episodic	1	1	1
	Dynamic	1	1	1
	Continuous	1	1	1
	Multiagent	3	3	3
Subtotal		**75%**	**75%**	**92%**
Software Engineering Requirements	Correctness	1	1	1
	Robustness	1	1	1
	Reliability	1	1	1
	Efficiency	1	1	1
	Usability	2	2	2
	Maintainability	2	2	2
	Evolvability	2	2	2

	Portability	3	3	3
	Interoperability	2	2	3
	Security	1	2	1
Subtotal		**45%**	**45%**	**50%**
Total		**52%**	**48%**	**67%**

The simulators Glest and Stratagus presented very similar grades in their analyses considering the proposed requirements. The only requirements with different evaluation were Supporting Experimentation and Security. The main reason is that these systems were not developed to be used as testbed, but they should be used as game engine to creation of RTS games. This becomes clear when looking at the performance achieved in the Testbed's area, which are 36% for Glest and 21% for Stratagus.

The consequence is that such environments do not assist in the research activity on IA. Furthermore, they will probably never evolve to meet the other Testbed's requirements due to this focus difference.

The ORTS simulator's performance is superior to others because its focus is centered on simulation and research in AI terms. This becomes clear when looking at the performance achieved in the Testbed's area (50%). The performance presented in the AI area is superior to other simulators, but it could be better if the ORTS simulator helps in the development of Multiagent Systems. Nevertheless the ORTS architecture is favorable to creation of monolithic systems instead of MAS.

Moreover, the ORTS presented a weak performance relative software engineering‘s aspects. This is due mainly to the fact that the simulator does not present the necessary documentation to support the research community. Besides the direct impact of the lack of documentation in certain specific items such as usability, the problem of support the research community affects other items such as maintainability and evolvability.

In addition to the specific shortcomings of each simulator it is possible to point out the weakest identified features in general terms (Table 3.3).

Table 3.3: Main testbeds' weaknesses.

Area	Item
Benchmark Requirements	Hard to Understand
	Invalid Conclusions
Testbed Requirements	No Agent Development Kit
	Dirty Interface
	Poorly Defined Model of Time
	No Report
AI Requirements	Completely Accessible
Software Engineering	Poor usability - hard to learn and use

Requirements	Poor maintainability - hard to maintain
	Poor evolvability - no upgrades
	Poor portability - hard to port
	Poor interoperability – hard to integrate with different softwares and hardwares

Besides that, it is possible to conclude that all evaluated simulators have presented similar grades in the Benchmark and AI areas. This is due to the stronger relationship among these features and the simulator domain (i.e. Real-Time Strategy games) than those features and the simulator ones.

3.5 Conclusions

This chapter presented the current state of the art in terms of testbed for research on IA, and more specifically the testbeds which use game environments for simulation. These testbeds were analyzed considering several different criteria.

The result achieved with the comparative analysis is that simulators do not yet have all the requirements expected by the scientific community. The simulator that has achieved the best result obtained a performance's percentage of 50% in the Testbed's features and 67% in general terms.

Chapter 4

RTSCup

This chapter presents the testbed proposed in this dissertation, called RTSCup. The simulator was set up considering the main deficiencies identified in the studied simulators. This chapter begins presenting the Java Real-Time Strategy Simulator (JaRTS). This simulator is a prototype designed to validate the implementation of requirements proposed to solve some of the identified weaknesses, such as usability problems, dirty interface and no ADK. And finally, the theoretical, conceptual and practical foundations of the RTSCup are presented.

4.1 Introduction

The analysis of MAS simulators presented in the previous chapter showed the main problems existing in current simulators used in research on IA, and more specifically in the area of multiagent systems. The outcome of the review identified the need for a more robust simulator to meet the requirements of the area.

The goal of this work was to develop a simulator that meets researchers' needs in the study of Multiagent Systems. To achieve this objective, the weaknesses previously identified will be used for defining the requirements of the simulator.

Table 4.1: Main testbeds' weaknesses.

Area	Item
Benchmark Requirements	Hard to Understand
	Invalid Conclusions
Testbed Requirements	**No Agent Development Kit**
	Dirty Interface
	Poorly Defined Model of Time
	No Report
AI Requirements	Completely Accessible
Software Engineering Requirements	**Poor usability - hard to learn and use**
	Poor maintainability - hard to maintain
	Poor evolvability - no upgrades
	Poor portability - hard to port
	Poor interoperability – hard to integrate with different softwares and hardwares

The Table 4.1 summarizes these acknowledged weaknesses. As already mentioned, the items of the areas of Benchmark and Testbeds are more related to the domain itself (RTS) than the simulator. Therefore, the remaining items belonging to the Testbed and Software Engineering areas were prioritized.

In particular, the highlighted items in the table were considered first. All these items are related to the usability, allowing the researcher greater productivity, flexibility of use and flexibility of application. The objective is to maximize the community contribution so that certain items such as maintainability and evolvability will be developed over time through the participation of the scientific community.

4.2 JaRTS

Java Real-Time Strategy Simulator [JaRTS 2006, Vieira 2006, Vieira 2007b, Vieira 2007c] is a RTS game simulator that enables users to focus the development uniquely on the agents' behavior. This means that users don’t have to worry about the simulation and evolution of the environment. The approach of JaRTS is very similar to the Robocode simulator [Li 2002], where there are basic classes that implement the behavior of their agents, such as “move to”, “twirl around”, or “shoot in”. The task of programmers is to extend such classes to create more complex behaviors.

4.2.1 Main Requirements of a MAS Simulator

The main requirements described in this section are related to some problems identified in the comparative analysis on Chapter 3 (Section 3.4).

- **Simplicity**

The Robocode values the simplicity aspect. The simulation of a specific environment is done in a simple way by defining the agent’s behavior in a java file and then importing it in the simulation system. This importing system was one of the main features adopted by the JaRTS to simplify the startup process and reduce the learning curve.

Besides that, the agent’s implementation is a very simple process. The Robocode provides an ADK (Java API) with a *Robot* class which is the abstract representation of the robots. This class has a *run()* method in which must have the agent’s behavior. This approach isolates the simulator architecture and implementation from the agent’s behavior. Thus users don’t have to understand how simulator works, but to learn how to develop an agent using the available ADK.

- **Configurability**

The JaRTS presents many different properties, such as world environment and simulation type (resource gathering or tank combat), which enable the definition of different test scenario to support experimentation

- **Visibility**

The visibility aspect is related to the domain’s accessibility. The agents receive the sensory information based on their field of vision. In the Robocode, the agents only perceive what is in front of them. In the ORTS, the agents perceive everything that is located within a certain range (circular field of vision). The JaRTS adopted the ORTS’ visibility strategy.

4.2.2 Architecture

The objective of the proposed architecture is to reduce the time spent in the learning process - a common factor when adopting a new tool. Therefore the structure and simulation method are based on the diffused Robocode system. Thus users already familiar with the Robocode will not find difficulty to carry out simulations in JaRTS. The adopted structure is known as monolithic architecture (Illustration 4.1).

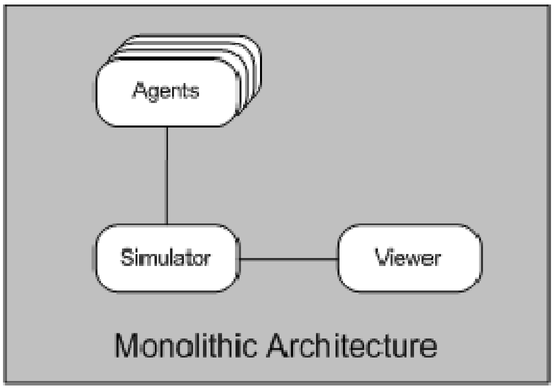


Illustration 4.1: JaRTS' architecture.

In a monolithic architecture the processing, data and user interface resides on the same system. However it is possible to identify the sub-components inside the architecture.

- **Simulator**

It is responsible for the simulation of the actions sent by agents, and it updates the environment status.

- **Viewer**

The viewer is the graphical interface used to visualize the simulation status.

- **Agent**

Each agent in the simulation controls a unit in the simulated world. The approach of JaRTS is very similar to the Robocode simulator [Li 2002], where there are basic classes that implement the main behaviors of their agents, such as move to, twirl around, or shoot in. The task of programmers is to extend such classes to create more complex behaviors.

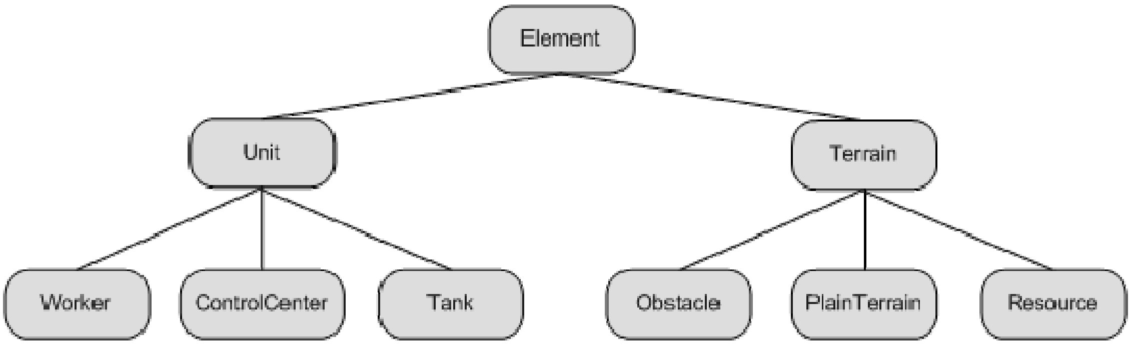


Illustration 4.2: Structure of elements in JaRTS simulator.

JaRTS presents two basic elements type: Unit, that represents existing agent's types, and Terrain, which describes the available tiles' types (Illustration 4.2).

- **Unit**

The Unit type represents the elements controlled by user. It can be of three distinct types: Worker, Tank and ControlCenter. The Worker describes a person who works in a mine, also called mineworker. The main function of this entity is to collect resources and deliver them in ControlCenter. The ControlCenter represents a place to deposit collected resources and is also the target (to protect or to destroy) of Tank agents. And the Tank represents soldiers whose objective is to attack or defend specific positions around the environment.

- **Terrain**

It represents the scenario elements such as Obstacle, Resource and PlainTerrain. The Obstacle describes a tile on the map where agents can not pass. On the other hand, the PlainTerrain represents an empty tile. The last one describes the entities used by Workers to collect resources.

Each one of these elements, units and terrains, has a different representation to enable the identification of each entity type in the graphical interface (Table 4.2).

Table 4.2: Graphical representation of elements.

Units	Terrains
● Worker	▣ Obstacle
🟩 Tank	▲ Resource
▢ ControlCenter	■ PlainTerrain

The environments are based on tiles, which means that the map area is divided into rectangular tiles forming a large square mesh. Each one of these tiles can only be occupied by a single unit.

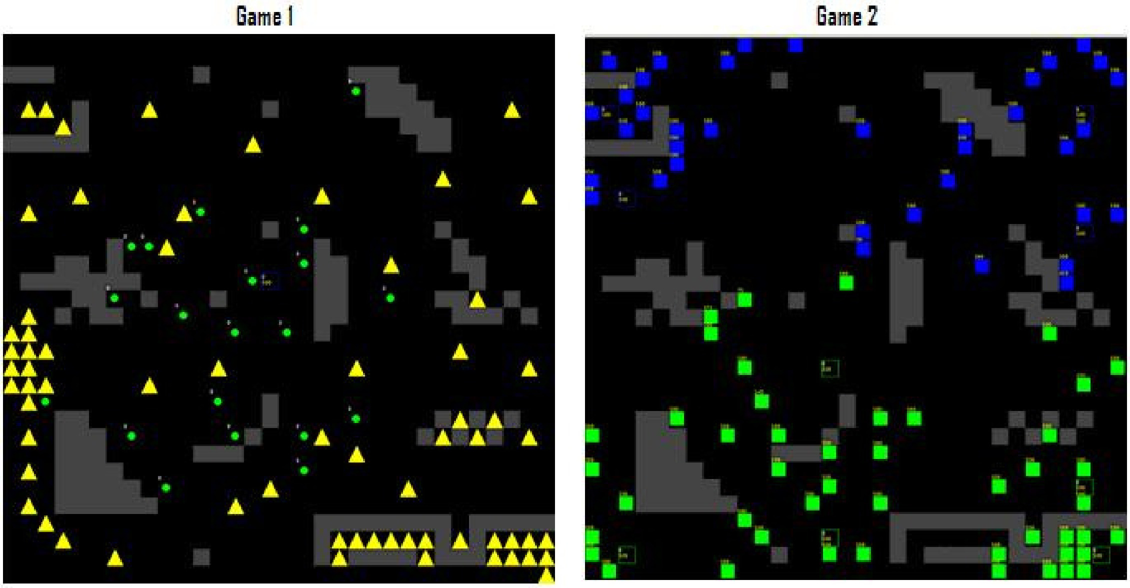


Illustration 4.3: JaRTS' simulation environment.

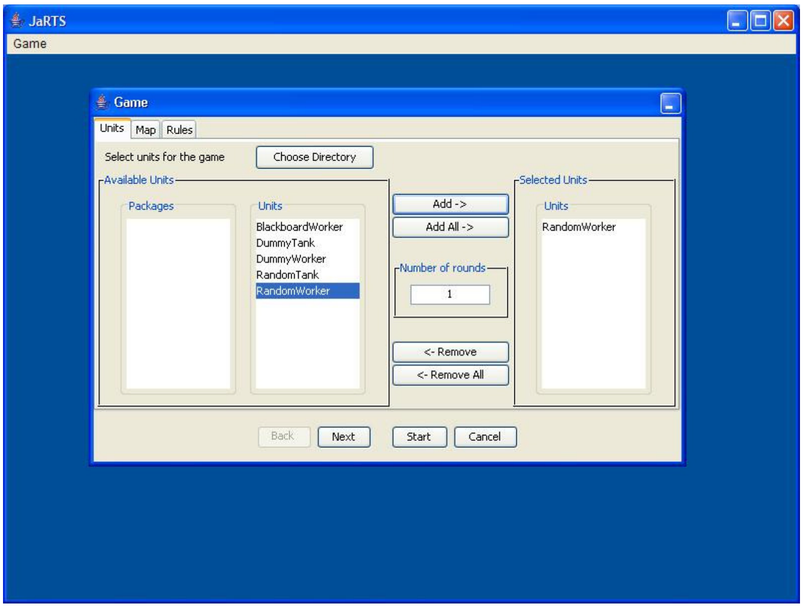
JaRTS simulates two different game types similar to the challenges presented by ORTS simulator. In Game 1, agents must develop strategies to perform resource gathering. The goal is to collect the maximum amount of resources within 10 minutes. In Game 2, the objective is to destroy as many opponent buildings as possible.

Progress of the Simulation

The RTSCup simulation occurs in two steps: Initialization and Progress. The Initialization is the first step and occurs only once. Differently, the Progress step is repeated until the end of the game.

- **Initialization Step**

The user configures the game simulation in three steps. In the first step, the user chooses the units that will act in the environment. All agent implementation has a specific .class file which is used to load its behavior by choosing it in the Units tab (Illustration 4.3).



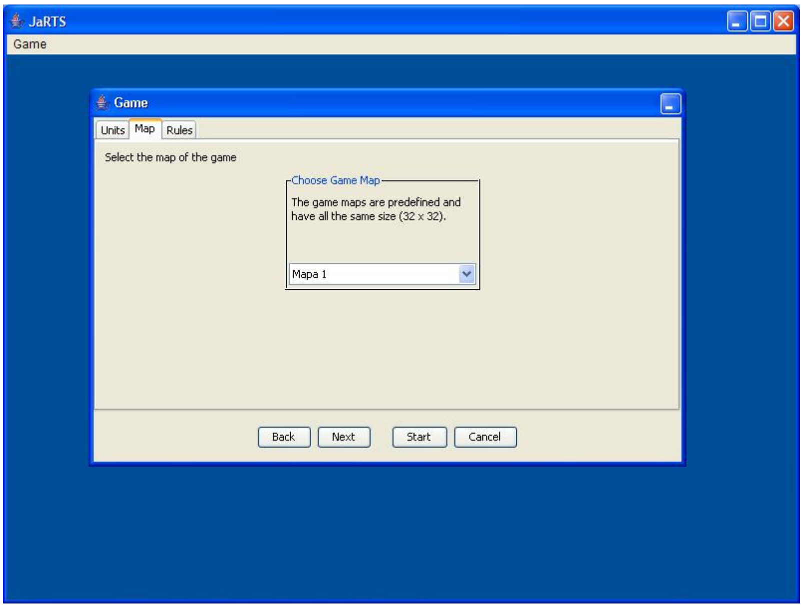
Units: BlackboardWorker, DummyTank, DummyWorker, RandomTank, RandomWorker

Number of rounds: 1

Selected Units: RandomWorker

Illustration 4.4: New simulation window.

The second step is the choice of the map. The JaRTS already provide some maps to facilitate the startup process, but it is possible to create different map configurations (Illustration 4.4).

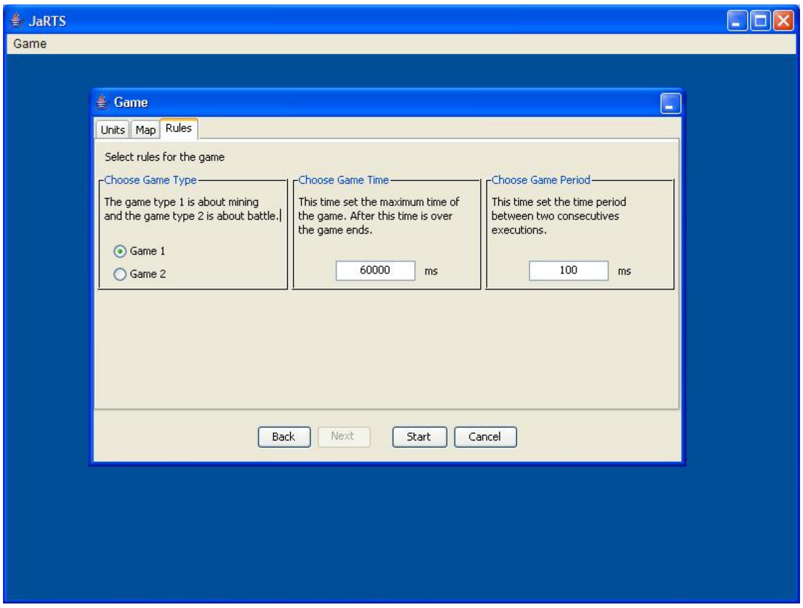


The game maps are predefined and have all the same size (32 x 32).

Mapa 1

Illustration 4.5: Map chooser window.

The third and last step is the definition of the game's rules. In this phase, it is possible to define the game type, game period and maximum game time (Illustration 4.5).



Select rules for the game

Choose Game Type

The game type 1 is about mining and the game type 2 is about battle.

● Game 1
○ Game 2

Choose Game Time

This time set the maximum time of the game. After this time is over the game ends.

60000 ms

Choose Game Period

This time set the time period between two consecutives executions.

100 ms

Illustration 4.6: Rules configuration window.

After all these steps, the simulator already knows all the necessary information to start the simulation. Therefore, JaRTS creates all necessary agents and bind them with the existing entities in the map. There is one agent to each entity in the simulated environment.

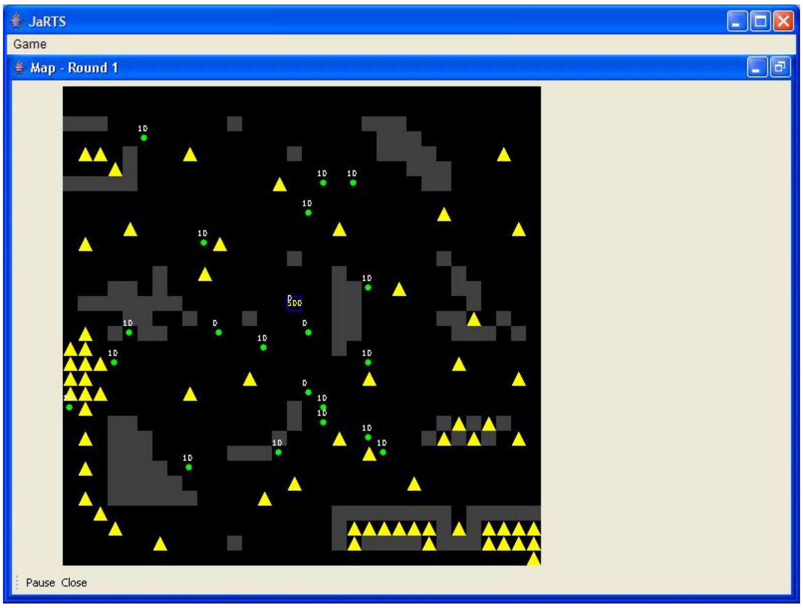


Illustration 4.7: Simulation window.

- **Progress Step**

When all unit types and buildings in the virtual world have been assigned to JaRTS agents, the simulator finishes the initialization step. Then, the simulation starts. The simulation proceeds by repeating the cycle represented in the Illustration 4.8.

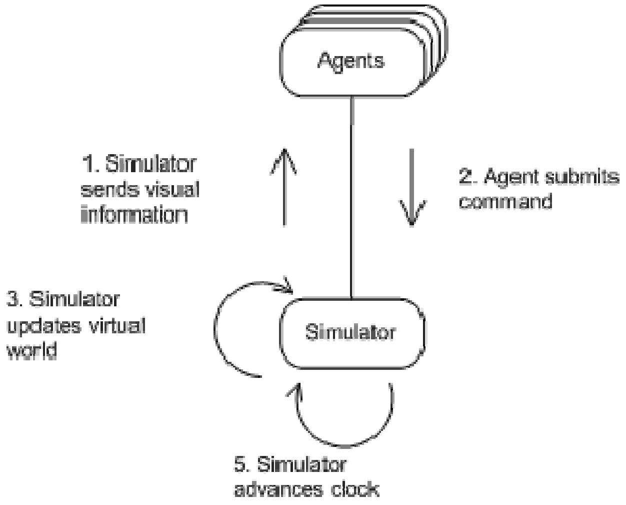

Agents
1. Simulator sends visual information
2. Agent submits command
3. Simulator updates virtual world
Simulator
5. Simulator advances clock

Illustration 4.8: JaRTS' simulation cycle.

4.2.3 Implementation

This section presents the main project decisions in technical terms. The Java programming language was adopted due to its popularity (highly diffused) and the character inherent in the multi-platform technology allowing the user to focus only on the problem: agents' implementation.

Besides, the Java programming language has an advanced feature known as Reflection. It gives runtime information about objects, classes, and interfaces. Reflection allows the manipulation of objects and the execution of actions like:

- Constructing an object using a given constructor
- Invoking an object's method using such-and-such parameters
- Assigning a value to an object's field

This feature enabled the creation of the loading structure of java classes in the simulator. With the definition of class, the system creates all class instances required for simulation. It is these class instances which control the units in the environment of simulation.

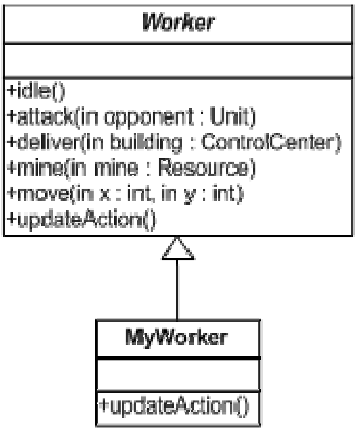

Worker
+idle()
+attack(in opponent : Unit)
+deliver(in building : ControlCenter)
+mine(in mine : Resource)
+move(in x : int, in y : int)
+updateAction()
MyWorker
+updateAction()

Illustration 4.9: Definition of a new agent type based on the unit Worker.

The users just extend one of the unit classes (*Worker*, *Tank* or *ControlCenter*) to specify a particular behaviour. For example, users can create the class *MyWorker* extending the class *Worker* (Illustration 4.9).

4.2.4 Evaluation

This section analyzes JaRTS considering the requirements described in Chapter 3, in order to verify its performance and compare it to other simulators. The Table 4.3 summarizes the evaluation.

Table 4.3: JaRTS evaluation.

Criteria	Item	Evaluation	Description
Benchmark Requirements	Easy to Understand	3	The RTS problem is not easy to understand, however it's possible to understand its subproblems. (decomposable problems).
	Valid Conclusions	3	It's hard to understand the behavior of a complete RTS simulation and why the system (game) behaves the way it does.
	Seductive Problem	1	The RTS problem as already discussed is a very motivating one.
	Decomposable Problem	1	This system enables the simulation of specific RTS game sub-problems.
Subtotal		**75%**	
Testbeds Requirements	Domain Independence	1	The solutions adopted in RTS domain such as pathfinding and patrolling can be applied in other domains.
	Supporting Experimentation	1	The system is configurable.
	Agent Development Kit	1	The system provides an API for help in the development of new agents.
	Clean Interface	1	There are no modules.
	Well Defined Model of Time	1	It's possible to adjust the time's model (game cycle value).

	Viewer	3	Viewer doesn't show all environment properties.
	Automatic Report	2	There is no computer generated report.
Subtotal		**79%**	
AI Requirements	Partially Accessible	2	Every unit receives complete sensory information.
	Non-Deterministic	1	There is no certainty about the state that will result from performing an action.
	Non-Episodic	1	The agent needs to think ahead to determine its actions.
	Dynamic	1	There are many other agents acting and modifying the environment.
	Continuous	1	There is no fixed and finite number of actions and percepts in it
	Multiagent	1	All agents are autonomous.
Subtotal		**83%**	
Software Engineering Requirements	Correctness	1	This system meets its specification as a game engine, not an AI testbed.
	Robustness	1	-
	Reliability	1	-
	Efficiency	1	-
	Usability	1	The system focus in the usability. Plug-and-play theory.
	Maintainability	3	
	Evolvability	2	The production is stopped for several months.
	Portability	3	The portability is restricted to hardware and software not to programming language (only Java).
	Interoperability	3	The system runs in all platforms but has no modules (monolithic system).
	Security	2	The simulation and agent implementation run in the same JVM, and therefore it is possible to hack.
Subtotal		**70%**	
Total		**77%**	

The JaRTS presented a better overall performance (77%) when compared to other testbeds: Glest (52%), Stratagus (48%), and ORTS (67%). Nevertheless, there are several required upgrades to fit the researchers' needs. These improvements were developed in this research and are described below.

- **Client-Server architecture**

The first improvement is the modification of the current architecture to a client-server style. The most common architecture in commercial RTS games is based on the client-side simulation. A more detailed description of this approach can be found at (Boson 2005). Clients simulate the game actions and they only send the player actions to a central server or to their peers (*peer-to-peer*). An alternative approach is the client-server architecture. In this approach a central server simulates the RTS game and only sends, to its clients, the game state that the client must know. This approach is safer because the simulation is carried out into the server so that part of the information can be hidden from clients.

- **Open communication protocol**

A second improvement is the definition of an open communication protocol between server and clients. The client-server approach, allied to a well-defined and documented communication protocol between clients and server, enables that different clients can be developed using different programming languages. In this way, the researcher can use his/her previous work without worrying about the programming language that the simulator uses.

- **Multiplatform application**

A third improvement is the definition of the server as a multiplatform application, enabling the server performance over several operational systems. To guarantee this feature in our new simulator version, we are using Java as programming language and a communication strategy based on sockets.

- **Flexible configuration**

A forth improvement is the definition of several configuration parameters to enable a flexible definition of test scenarios. Besides the fact that RTS games have several similar features, they also present some particular features. The main features that differ one game from other are: building types, unit types, terrain types, tech-trees types and finish conditions. The objective is to enable the customization of these and others options, so that users can simulate different game scenarios. This feature is being implemented as a configuration file, which indicates the features (value of parameters) of a particular game. In this way, users can have a library of games, each of them customized to test a particular feature of their approaches.

- **Map Editor**

A fifth improvement is the creation of a map editor to enable the building of different environments. The map editor is a desirable feature in a simulation environment used as benchmark. The map edition enables the creation of different test scenes that can be also used to verify the adequacy and generalization of the proposed solution.

- **Automatic Report**

A last improvement is the automatic generation of reports about the simulation. This feature is important to any simulation tool that is being used as benchmark. The generation of reports enables, for example, the evaluation of solutions in a post-simulation stage, as well as a better basis for comparative evaluations.

The JaRTS architecture does not support the inclusion of the above requirements, such as client-server architecture and open protocol. Therefore, it was necessary to create a new simulator with a different architecture to meet these changes.

4.3 RTScup

The RTSCup Project [RTSCup 2007, Vieira 2007a, Vieira 2007d] is designed to maximize the community contribution and attain high-throughput in research itself. From the research perspectives, RTSCup was designed (1) to enable users to focus their efforts on AI research; (2) to be used as testbed in the multi-agent area; and (3) to integrate the MAS research community. The Illustration 4.10 shows the simulation environment.

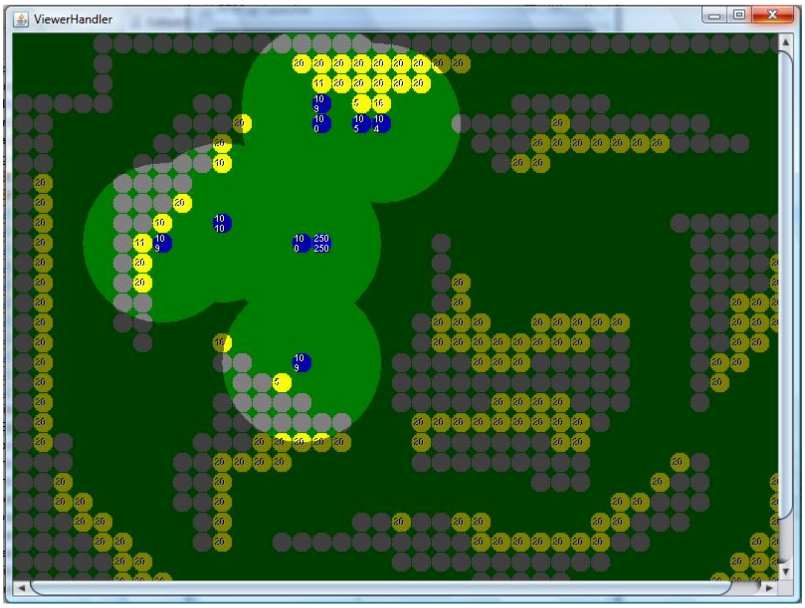


Illustration 4.10: RTSCup's viewer.

4.3.1 Main Requirements

The main simulator requirements are related to the identified problems during the previous comparative analysis of other simulators.

- **Free software (FS)**

The RTSCup is released under the GNU Lesser Public License (LGPL), which means that anyone can download the source code at no cost in order to learn how the system works and to contribute to the project by submitting bug fixes and adding new features. It also means that projects that incorporate RTSCup code need to release their source code as well. The benefit of LGPL’ed software releases to the community is tremendous as witnessed by the success of the Linux, KDE, and Mono projects.

- **Website (WS)**

All this documentation will be available in the website [RTSCup 2007] with all the information that users could need in order to develop their strategies. This website will also contain some strategy ideas with their source codes, which users can compare with their ones. Our intention is to create a community in this site, with a forum where users can discuss about strategies, exchange experiences, download other client modules (in Java, C, C++, etc) and talk about their doubts. And more than that, the proposal is to encourage the community to participate and collaborate by improving the testbed and also creating support tools (add-on projects).

- **Client-server architecture (CS)**

The most common architecture in commercial RTS games is based on the client-side simulation. A more detailed description of this approach can be found at [Boson 2005]. Clients simulate the game actions and they only send the player actions to a central server or to their peers (*peer-to-peer*).

The commercial solutions is based on client-side simulations (peer-to-peer) where the AI code for all players runs on all peer nodes to save bandwidth and to keep the simulations synchronized. This creates unwanted CPU load. Moreover, user configurable AI behavior is limited to simple scripts because there are no APIs to directly connect AI systems that run on remote machines.

An alternative approach is the client-server architecture. In this approach a central server (kernel) maintains the entire world state, simulates the RTS game and only sends visible information to players which connect from remote machines. This approach is safer because the simulation is carried out into the server so that part of the information can be hidden from clients. Therefore map–revealing client hacks that are common in commercial client–side simulations are therefore impossible.

- **Open communication protocol (OP)**

The client-server approach, allied to a well-defined and documented communication protocol between clients and server, enables that different clients can be developed using different programming languages. In this way, the researcher can use his previous work without worrying about the programming language that the simulator uses.

Furthermore, despite today’s commercial RTS games confine users to single view graphical user interfaces, fix low–level unit behavior, and sometimes use veiled communication, the use of an open communication protocol allows AI researchers and players to connect *whatever* client software they like—ranging from split–screen GUIs to fully autonomous AI systems.

- **Multiplatform server (MP)**

A multiplatform application enables the server to be executed over several operational systems. This feature guarantees that researchers can use their previous works without worrying about the platform. To ensure this feature in our new simulator version, we are using Java as programming language and a communication strategy based on sockets (UDP).

- **Parameterized game description (PGD)**

This feature means that the simulator enables a flexible definition of test scenarios. Besides the fact that RTS games have several similar features, they also present some particular ones. The main features that differ from one game to another are: building types, unit types, terrain types, tech-trees types and finish conditions. Our intention is to enable the customization of these and others options, so that users can simulate different game scenarios.

This feature is implemented as a configuration file, which indicates the features (value of parameters) of a particular game. In this way, users can have a library of games, each of them customized to test a particular feature of their approaches. The community can help here because RTSCup is free software.

- **Agent Development Kit (ADK)**

The purpose of the RTSCup ADK is to simplify the procedure for developing interesting agents for the simulation system. This purpose is two-fold. First, to abstract away the details of the agent-kernel communication protocol so agent developers can focus on agent development. And second, to provide a toolbox for building high-level behaviors.

These goals are to be achieved while providing maximum flexibility to the agent developers. It is an important goal of the ADK not to be biased towards any particular solution to the agent development problems. The toolbox should only contain well-known tools for addressing only the simplest problems inherent in the RTS environment.

For example, the basic commands such as move and attack are basic tools that this kit includes, but an implementation of a specific agent communication protocol is not, since this is an area of ongoing research. Currently the ADK is only available in Java [Vieira 2008].

- **Viewer (VW)**

The viewer should present all important information demanded by researchers to enable the understanding of what is happen in the simulated environment.

The Table 4.4 summarizes the relationship among each of the cited requirements and the required ones. The main RTSCup's requirements are listed in the table's columns and the requirements used in the testbed's comparison are listed in the rows (see Table 4.4). A filled cell indicates that the testbed's requirements described in the row are presented in the RTSCup due to the requirement indicated in the column.

Table 4.4: Relationship between RTSCup requirements and evaluation ones.

Area	Item	FS	WS	CS	OP	MP	PGD	ADK	VW
Benchmark Requirements	Easy to Understand								
	Valid Conclusions								
	Seductive Problem								
	Decomposable Problem						▨		
Testbeds Requirements	Domain Independence								
	Supporting Experimentation						▨		
	Agent Development Kit							▨	
	Clean Interface		▨					▨	
	Well Defined Model of Time						▨	▨	
	Viewer								▨
	Automatic Report								
AI Requirements	**Partially Accessible**								
	Non-Deterministic								
	Non-Episodic								
	Dynamic								
	Continuous								
	Multiagent								
Software Engineering Requirements	Correctness								
	Robustness	▨							
	Reliability	▨							
	Efficiency	▨							
	Usability	▨	▨						
	Maintainability	▨							
	Evolvability	▨	▨			▨			
	Portability	▨		▨	▨	▨			
	Interoperability			▨	▨	▨			
	Security			▨					

4.3.2 Architecture

The software architecture of a computing system is the structure of the system, which comprises software components, the externally visible properties of those components, and the relationships between them. It is in the software architecture's definition that earlier problems of complexity were solved by choosing the right data structures, developing algorithms, and by applying the concept of separation of concerns.

And precisely because of the importance to establish a robust and efficient architecture, several architectures have been studied and implemented before reaching the present stage of maturation. Initially the architecture was based on the architecture used in Robocode. This happened mainly due to the ease of use of this platform allowing the use of the system without the need of help. After the implementation of JaRTS, several points of improvement were identified in addition to the need for improvement in terms of the cited requirements.

- **Client-server architecture**

Although the peer-to-peer architecture is the most common in commercial RTS games, it is necessary to adopt the client-server architecture style. A more detailed description of this approach can be found at (Boson 2005). Clients simulate the game actions and they only send the player actions to a central server or to their peers (*peer-to-peer*). An alternative approach is the client-server architecture. In this approach a central server simulates the RTS game and only sends, to its clients, the game state that the client must know. This approach is safer because the simulation is carried out into the server so that part of the information can be hidden from clients.

- **Open communication protocol**

The client-server approach, allied to a well-defined and documented communication protocol among clients and server, enables that different clients can be developed using different programming languages. In this way, the researcher can use his/her previous work without worrying about the programming language that the simulator uses.

After the detailed analysis of the points presented and also the study of the architectures used in other testbeds, the architecture chosen was the RoboCup's one. This design decision was based on the following aspects:

- **Accumulated Expertise**

The RoboCup is an international robotics competition founded in 1993. The architecture of the proposed system has been revised and redesigned several times to reach the current status. That is, many problems have been already addressed and resolved in the RoboCup, Therefore, the choice of the RoboCup's architecture grants a similar level of maturity to the RTSCup.

- **Well known architecture**

The RoboCup is already widely used by the scientific community. This means the following: the researcher who has used the RoboCup will have a smaller learning curve of the RTSCup's platform. This leads to lower the barriers to entry.

- **Architecture standardization**

Despite the progress in the field of AI with the advent of testbeds, there is still a problem to deal with. It is difficult to switch from one to another because the need to learn a new structure and way of functioning. If the other testbeds follow the example of RTSCup all simulators would be similar in structure, although they simulate different environments and problems.

Despite the similarities, these two testbeds have some requirements differences resulting in the need to study and change the RoboCup's architecture at specific points to adjust to the area of RTS.

Structure of the Simulation System

The simulation project involves the development of a comprehensive simulator that enables the integration of all above features in a multi-agent environment in a large scale. The simulator should be able to combine all above cited features and present them as a coherent scene. The current version of the simulator architecture is shown in Illustration 4.11.

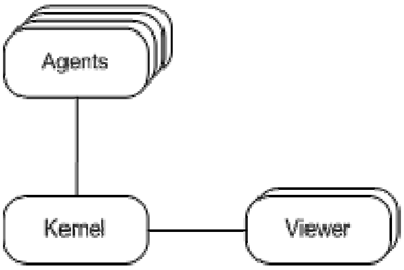


Illustration 4.11: RTSCup's architecture.

The RTSCup simulator is a real-time distributed simulation system that is built of several modules connected through a network. Each module can run on different computers as an independent program, so the computational load of the simulation can be distributed to several computers. The architecture is divided into the following modules.

- **Agent**

The agent module controls an intelligent individual that decides its own action according to situations. Villagers, swordsman, archer, priest and so on are agents. Individuals are virtual entities in the simulated world. Their "will" and actions are controlled by the corresponding agents, which are the agent modules (software program). . The user is responsible for the development of the agent

The agent modules are the client programs in the client-server architecture presented that communicates with the kernel through a network to act in the game environment. The agent module is responsible for receive the sensory information, decide the actions and then send a command to the kernel. The kernel modules check the actions and update the virtual world.

- **Viewer**

Another RTSCup simulator module used to visualize the game environment status.

- **Kernel**

The kernel controls the simulation process and facilitates information sharing among modules. This module is responsible for combining all commands and updates the game world.

Progress of the Simulation

The RTSCup simulation occurs in two steps: Initialization and Progress. The Initialization is the first step and occurs only once. Differently, the Progress step is repeated until the end of the game.

- **Initialization Step**

The kernel reads the configuration files to load the initial condition of the game world. Now the kernel is ready to receive connections from viewer and clients. The viewer connects to the kernel before all the RTSCup agent assignments have been carried out. RTSCup agents connect to the kernel with their agent type. The kernel assigns each RTSCup unit and building in the virtual world to an RTSCup agent developed by user, sending the initial condition related to each agent's cognition. This flow is represented in Illustration 4.12.

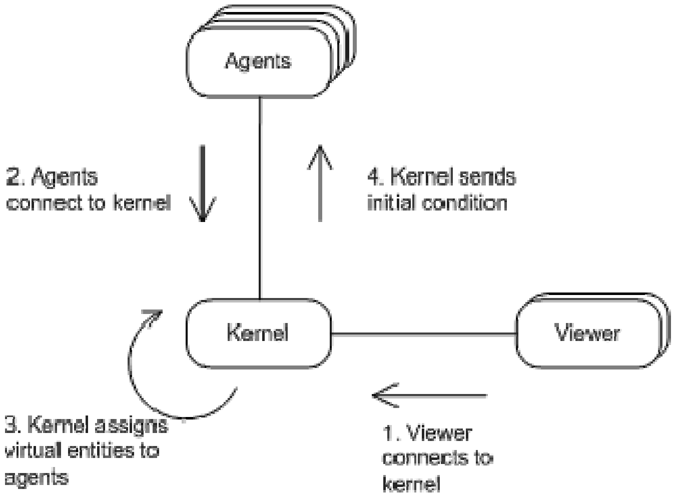

Agents
2. Agents connect to kernel
4. Kernel sends initial condition
Kernel
Viewer
3. Kernel assigns virtual entities to agents
1. Viewer connects to kernel

Illustration 4.12: RTSCup's initialization step.

- **Progress Step**

When all unit types and the buildings in the virtual world have been assigned to RTSCup agents, the kernel finishes the integration and the initialization of the kernel. Then, the simulation starts.

The simulation proceeds by repeating the following cycle (Illustration 4.13). At the first cycle of the simulation, steps 1 and 2 are skipped.

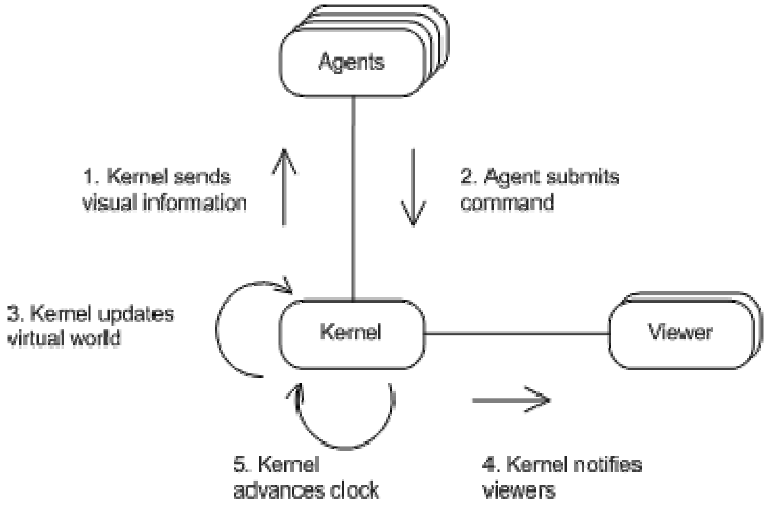

Agents
1. Kernel sends visual information
2. Agent submits command
3. Kernel updates virtual world
Kernel
Viewer
5. Kernel advances clock
4. Kernel notifies viewers

Illustration 4.13: RTSCup's simulation cycle.

1. **The kernel sends individual vision information to each RTSCup agent**

At the beginning of every cycle, the kernel sends sensory information to each agent. This sensory information consist of information that individual (controlled by the agent module) can sense in the simulated world at that time.

2. **Each RTSCup agent submits an action command to the kernel individually**

Each agent decides what actions the individual should take, and send it to the kernel.

3. **The kernel receives all action commands from RTSCup agents and updates the state of the virtual world**

The kernel gathers all messages sent from agent modules. Commands are sometime filtered. For example, commands sent by an agent module whose corresponding individual is already dead are discarded. Since the simulation proceeds in real time, the kernel ignores commands that do not arrive in time.

The kernel computes how the world will change based upon its internal status and the commands received from the agent modules.

4. **The kernel notifies the viewers about the update**

The kernel sends the current status of the virtual world to the viewers. The viewers need this information to updates the visual information that is being displayed.

5. **The kernel advances the simulation clock of the simulated world**

One cycle in the simulation corresponds to a predefined time in the simulated world. This time can be editable by user to simulate different cycle's time – it's useful when the RTSCup agent has a long processing time or when the network has significant time delays.

World Modelling

The kernel models the world as a collection of *objects* such as buildings and units. Each object has *properties* such as its position and shape, and is identified by a unique *ID*.

The kernel doesn't send all the properties of each object every time; sending only necessary or limited part of objects properties. For example, sensory information sent by the kernel to each agent contains only information that the individual can sense visually. However, all properties are only broadcast on the beginning of the simulation.

Objects in the RTSCup simulator are represented by general types. Those types can be customized, by modifying its properties, to represent different object's instances. The existing types are presented below.

- **Unit**

It is a general type that is able to represent all RTS units such as villagers, archer, swordsman and priest, and so on.

- **Building**

It is a general type that represents all construction types in the virtual world such as house, barracks, storage pit, tower, and so on.

- **Resource**

It is a general type that represents all resources types in the simulation such as mines and trees.

- **Obstacle**

It is a type that represents all obstacle types in the simulation such as stones, hills, walls, and so on.

For more details, see the RTSCup Manual [Vieira 2008].

Protocol

The protocol specifies communication between modules – kernel, viewer and agents. A data unit for the protocol is called a block which consists of a header, body length, and body field (Illustration 4.14).

header
body length
body ...

Illustration 4.14: Block of the RTSCup protocol.

And a packet of the protocol consists of zero or more blocks and a HEADER NULL (0x00) as a terminator (Illustration 4.15). The format of the body field depends upon the header. The body length field is set the byte size of the body.

block_1	block_2	…	block_n	HEADER_NULL

Illustration 4.15: Packet of the RTSCup protocol.

The following table (Table 4.5) shows headers related to RTSCup agents. A protocol block having some header H is called an H block, and a body of an H block is called an H body for short. Moreover, a block issued to submit the will to act such as an AK_MOVE and an AK_REST block is called an action command.

Table 4.5: Headers of the RTSCup protocol.

Value	Header	Use
	To the kernel:	
1	AK_CONNECT	To request for the connection to the kernel
2	AK_ACK2WLEDGE	To acknowledge for the KA_CONNECT_OK
3	AK_MOVE	To submit the will to move to another position
4	AK_BUILD	To submit the will to build a building
5	AK_FIX	To submit the will to fix a building
6	AK_COLLECT	To submit the will to collect resources from a resource font
7	AK_DELIVER	To submit the will to deliver resources to an element (unit or building)
8	AK_ATTACK	To submit the will to attack an element (unit or building)
9	AK_CURE	To submit the will to cure a friend unit
10	AK_CONVERT	To submit the will to convert a enemy unit
11	AK_REST	To submit the will to stop current activity and rest.
12	AK_TRAIN	To submit the will to train a given unit type
13	AK_RESEARCH	To submit the will to research a given technology
	From the kernel:	
14	KA_CONNECT_OK	To inform of the success of the connection
15	KA_CONNECT_ERROR	To inform of the failure of the connection
16	KA_SENSE	To send vision information

The body field consists of 32-bit integers such as a time and an ID, strings, and objects serialized into binary data. The information is encoded using the UTF-8 format. The UTF-8 format was chosen because is able to represent any character in the Unicode standard, yet the initial encoding of byte codes and character assignments for UTF-8 is backwards compatible with ASCII. Body field formats will be defined in the RTSCup Manual [Vieira 2008].

4.3.3 Implementation

This section presents the technical aspects associated with the development of RTSCup. The purpose of this section is to show the main project decisions in technical terms.

Programming Language

The programming language adopted in the RTSCup, differently from that used in the RoboCup, is the Java language. The choice of the java language is closely related to the main benefits of this language which help in the achievement of some of the testbed's parameters cited. The main java features are presented below as well the associated benefits.

- **Simple, Safe and Object-Oriented**

These characteristics, among others, have helped to disseminate the Java programming language in the world and now this technology is widely used in both academia and industry. Thus the local community, particularly students and researchers from UFPE, have a strong base of knowledge in the Java language. The evidence of this is that the project RTSCup already counted with the participation of about 10 students from this university.

- **Platform Independent**

The Java programming language was designed to not only be cross-platform in source form like C, but also in compiled binary form. Since this is frankly impossible across processor architectures Java is compiled to an intermediate form called byte-code. A Java program never really executes natively on the host machine. Rather a special native program called the Java interpreter reads the byte code and executes the corresponding native machine instructions. Thus to port Java programs to a new platform all that is needed is to port the interpreter and some of the library routines. Even the compiler is written in Java. The byte codes are precisely defined, and remain the same on all platforms.

This feature ensures the use of RTSCup on different platforms. The simulator is not restricted only to PC platform, like most other simulators, but can still be used in Apple computers and even in portable devices such as cell phones and palms. Moreover, there is no limit in terms of operating systems (Illustration 17).

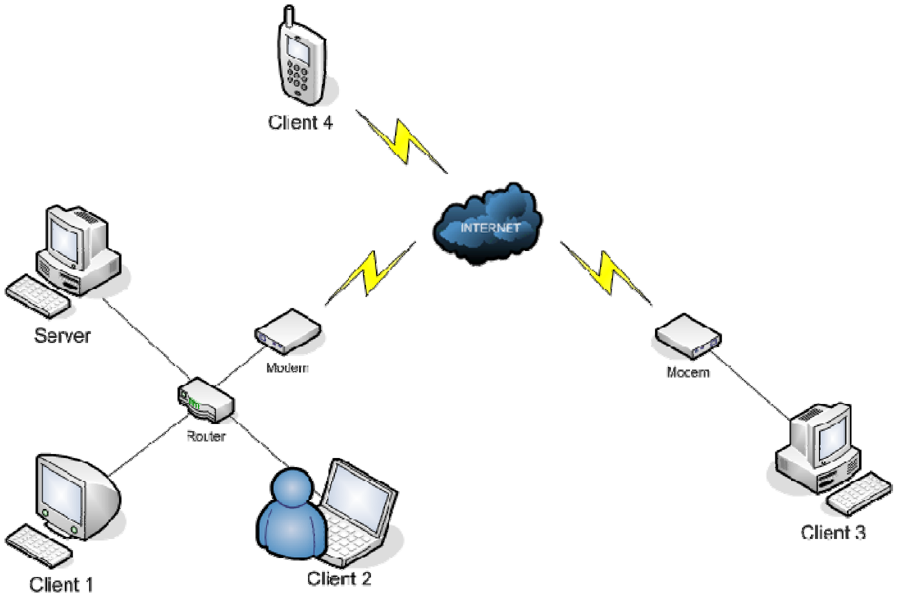


Illustration 4.16: Detailed strcture of the RTSCup system.

This feature ensures more freedom in the researchers' activities because of the possibility of use of the desirable platform and operating system. The researcher can choose the operating system that best suits him. If the researcher has a preference for the Linux operating system, for example, he can use this system without the need to change the configuration of his workstation. This is an important feature in order to not limit or excludes researchers with different preferences such as Windows, Linux, Mac OS and FreeBSD.

Moreover, the research area can be directly linked to the platform. For example, the research may be related to the use of multiagent systems in portable systems. In this case, the possibility of create and experiment applications embedded on portable devices approaches the research to its real platform. Thus the researchers can draw valid conclusions from their experiments as well as having more chances to apply the research results in real environments.

Communication Protocol

Communication between kernel and other components are done mostly by UDP. The data send from kernel is a big one, and its length may be larger that 64kb that UDP can not handle. The RTSCup protocol provides LongUDP protocol to transmit a big packet. LongUDP divides the big packet into small parts, adds 8 byte-header to each part, and send them by UDP. The Table 4.6 shows a LongUDP header format. The data is represented in network byte order.

Table 4.6: Representation of a LongUDP header.

Offset	Data	Comment
0	0x0008	Magic number
2	ID	ID number of the LongUDP packet
4	number	This integer shows where this UDP packet is in LongUDP (0 ~ total – 1)
6	total	Total packet numbers in LongUDP packet

LongUDP uses IP address and port number as well as UDP. The both IP address and port number are the same ones as UDP. ID number in the LongUDP is assigned not to be the same as the other LongUDP's ID. The long packet is unified by:

1. Collecting LongUDP packets with the same ID number
2. After receiving total packets, sorting them in number ascending order.
3. Concatenating them without header part.

The length of divided small parts except the last one (its *number*=*total*-1) is a multiple of four. The length of divided small parts is bigger than eight, i.e., the may have data besides header part. When *total* packets are not read for some period, some UDP packets may be lost. It is recommended to stop receiving the LongUDP packets. Otherwise a wrong LongUDP may be constructed later, because kernel creates ID number cyclically.

- **LongUDP Packet Format**

A LongUDP packet has block more than 0. The block is composed of *header* and *body*. The length of body may be 0xFFFFFFFF in the Table 4.7 when kernel received the message. In this case, the length of body is calculated from remaining packet's length.

Table 4.7: LongUDP packet format.

Offset	Hex Bytes	Comment
0	00 00 00 01	Header is 1
4	00 00 00 10	Length of body is 16
8	00 00 02 56	Body
12	00 00 01 34	
16	00 00 01 6F	
20	00 00 00 00	Header is NULL (block's end)
24	00 00 00 0E	Header is 14
28	00 00 00 10	Length of body is 16
32	00 00 02 57	Body

36	00 00 01 55	
40	00 00 01 A8	
44	00 00 00 00	Header is NULL (block's end)
48	00 00 00 00	Header is NULL (packets's end)

The body's content varies as its header. Connecting modules with different version may add extra data the body as explained later. The modules are recommended to neglect them. Or broadcasting data causes receiving packets for other module will cause similar situations. It is also recommended to neglect the packet for which the value of the header may be out of the specified values.

4.3.4 Evaluation

This section presents the evaluation of the RTSCup considering the features discussed in Chapter 3. The Table 4.8 summarizes the evaluation.

Table 4.8: RTSCup's evaluation.

Area	Item	Evaluation	Description
Benchmark Requirements	Easy to Understand	3	The RTS problem is not easy to understand, however it's possible to understand its subproblems. (decomposable problems).
	Valid Conclusions	3	It's hard to understand the behavior of a complete RTS simulation and why the system (game) behaves the way it does.
	Seductive Problem	1	The RTS problem as already discussed is a very motivating one.
	Decomposable Problem	1	This system enables the simulation of specific RTS game sub-problems.
Subtotal		**75%**	
Testbeds Requirements	Domain Independence		The solutions adopted in RTS domain such as pathfinding and patrolling can be applied in other domains.
	Supporting Experimentation	1	The system is configurable.
	Agent Development Kit	1	The system provides an API for help in the development of new agents.
	Clean Interface	1	There are no modules.
	Well Defined Model of Time	1	It's possible to adjust the time's model (game cycle value).
	Viewer	1	Viewer doesn't show all environment properties.
	Automatic Report	2	There is no computer generated report.
Subtotal		**86%**	
AI Requirements	Partially Accessible	3	Every unit receives complete sensory information (totally accessible).
	Non-Deterministic	1	There is no certainty about the state that will result from performing an action.
	Non-Episodic	1	The agent needs to think ahead to determine its actions.

	Dynamic	1	There are many other agents acting and modifying the environment.
	Continuous	1	There is no fixed and finite number of actions and percepts in it
	Multiagent	1	All agents are autonomous.
Subtotal		**93%**	
Software Engineering Requirements	Correctness	1	This system meets its specification as a game engine, not an AI testbed.
	Robustness	1	-
	Reliability	1	-
	Efficiency	1	-
	Usability	1	The system focus in the usability. Plug-and-play theory.
	Maintainability	1	
	Evolvability	1	-
	Portability	1	The RTScup is able to run in all platforms and using all programming languages.
	Interoperability	1	Open communication protocol
	Security	1	Client-server architecture model.
Subtotal		**100%**	
Total		**89%**	

The RTSCup presented the best overall performance (89%) when compared to other testbeds: Glest (52%), Stratagus (48%), ORTS (67%), and JaRTS (77%).

4.4 Conclusion

This chapter presented the effort spent developing this work. Firstly, it was created a prototype system named JaRTS which address some weaknesses founded in the testbeds' evaluation to examine the feasibility of the development of a robust testbed.

Event though the JaRTS is a prototype version of the desired testbed, it presented a better performance (77%) when compared to other testbeds Glest (52%), Stratagus (48%), and ORTS (67%). The detailed evaluation of all testbeds, including JaRTS, presented some directions to the development of a better testbed.

Thus, the RTSCup simulator was developed to address most of the weaknesses presented by other simulators. The RTSCup was evaluated and its performance (89%) is better than all other testbeds. Although the achieved performance is good, the RTSCup presented some flaws too.

Nevertheless, the RTScup was designed to integrate the research community and maximize their contribution in the development of the project. The proposal is that the research community help to improve the testbed.

Chapter 5

Experiments and Results

This chapter presents the main experiments conducted with the use of RTSCup for validation of its requirements and also the results achieved during the experiments. For didactic reasons, the experiments are initially presented in details through the description of the domain used, users involved, platform used and types of experiments performed. Next, it is presented the results achieved with the experiments. And finally, an analysis is conducted to see whether the results meet the initial objectives.

5.1 Experiments

The evaluation experiments involved about 200 students of the Intelligent Agents course in the Informatics Centre – UFPE at both undergraduate and postgraduate levels. The final project was the implementation of multiagent teams that could run and be evaluated via RTSCup.

After the conclusion of their projects, students should report the agents' issues/features that were evaluated via simulator. Furthermore, the reports were also important to clarify the approaches used for each team. The experiments have two different goals:

- **Test the implemented features in real applications**

The experiments is one of the steps (test phase) associated with the production of software. In such tests are evaluated many items from the perspective of engineering software such as correctness, robustness and efficiency.

- **Evaluate the testbed performance**

The experiments also served to assess the performance of the simulator from the perspective of testbeds for AI. In this case the points of evaluation are different from those presented in the above item. In this case, the testbed must provide certain characteristics that help the users in their research activities such as Agent Development Kit, Clean interface and Supporting Experimentation.

The presented goals are complementary, and both help in building a better simulation environment with focus on productivity and efficiency.

5.1.1 Platform

The experiments were performed using the computer laboratory of the CIn / UFPE. The computers in question had the following configuration:

- Pentium 4 (1.7 GHz)
- 1GB RAM
- Windows XP
- Java Development Kit 1.5.0_13

In addition, students used the Agent Development Kit [Vieira 2008] developed to assist in the construction of agents to be used in the RTSCup.

5.1.2 Data Set

The experiments were split up in three parts. First, groups of students should focus on approaches to the pathfinding problem. After that, the focus was on the resource gathering. And finally, the focus was on approaches to the tank battle problem. These three problems are detailed below.

Game 0

This game configuration was created to deal with the pathfinding problem. The pathfinding problem treats the problem of ploting the best route from point A to point B. Used in a wide variety of computer problems, and more specifically in games, it refers to AI solution to find a path around an obstacle, such as a wall, door, or bookcase. In recent games, pathfinding has become more important as players demand greater intelligence from their own units (in the case of real-time strategy games) or their opponents (as in the case of first-person shooters).

In the RTS case, the games have to deal with a larger, more open terrain, which is often simpler to find a path through, although they almost always have more agents trying to simultaneously travel across the map. This situation creates a need for different and often significantly more complex algorithms to avoid traffic jams. In these games the map is normally divided into tiles which act as nodes in the pathfinding algorithm.

This game is named "Game 0" because the treatment of this problem is essential for the development of any research to solve complex problems will require this algorithm such as foraging (Game 1) or combat (Game 2). The detailed rules of this challenge are presented in Table 5.1.

Table 5.1: Description of the game 0 rules.

Section	Description
Objective	Reach the resource path in the shortest time possible.
Setup	Single player. Perfect information. Random map (small regular static obstacles). One worker. One resource patch. Some small mobile obstacles ("sheep") moving randomly.
Actions	move(x,y,s): start moving towards (x,y) with speed s. stop(): stop moving.
Tournament Settings	Each program will play k games depending on available time. The player with the shortest time wins this category.

Game 1

The problem of collecting resources, also known as foraging, is the first problem that the player faces in a RTS game, because this task is the basis for the player to acquire resources to make the next steps, build buildings, train units, develop technologies and fight the enemy army.

The foraging is a complex problem in which it is necessary to use the agents in a coordinate and cooperative way, to make them load a maximum amount of resources, and to collect the maximum amount of resources in the shortest time.

The foraging problem has several characteristics that describe it [Pyke 1984].

- **Single Agent vs. Multiagent**

In the single agent, as the name says, a single agent is responsible for the collection, while in the multiagent a group of agents is responsible for the collection.

- **Central Place vs. Various Place**

It refers to the amount of existing deposit points. In the case of central place, everything that is collected must be taken to a single point. If the players have more than one point of deposit, then it is Various Place foraging.

- **Recurrent vs. One Shot**

One Shot means that the agent needs to visit a particular point only once, while in Recurrent the same point has to be visited several times.

In RTS environments, the foraging problem can be described as Multiagent Various Place Recurrent Foraging. Multiagent due the fact that RTS games are multiagent systems. Central place because the collected resources should be taken to a central point and then be deposited. Recurrent due the fact that the agent needs to go in the mine several times to collect resources until the mine is exhausted. The detailed rules of this challenge are presented in Table 5.2.

Table 5.2: Description of the game 1 rules.

Section	Description
Objective	Gather as much resources as possible within 5 minutes.
Setup	Single player. Perfect information. Random map (small regular static obstacles). One control center, 20 workers nearby. Several resource patches . Physical limit number of workers on single resource patch. Some small mobile obstacles ("sheep") moving randomly.

Actions	move(x,y,s): start moving towards (x,y) with speed s stop(): stop moving and mining. mine(obj): start mining minerals, need to be close to mineral patch; mining 1 mineral takes 1 game cycle; a worker can hold at most 10 minerals at any given time. deliver(obj): drop all minerals instantly, need to be close to control center.
Tournament Settings	Each program will play k games depending on available time. The player with the highest total mineral count wins this category.

Game 2

RTS games are commonly defined as "Real Time Strategy games are simulators of war where various factions struggling in a virtual world" [Laursen 2005]. From that definition, it is easy to understand the importance of combating in this style of game.

The combat activity is one of the most challenging factors in the RTS genre because the victory in the game depends on it. Therefore the combat should receive a special treatment from the viewpoint of Artificial Intelligence.

It is necessary to take into account several factors at the time of decision-making (what tactics should be used to counter-attack, for example) that happens in real time to address the combat problem. For human players the decision-making process is done on a very intuitive and fast way. On the other hand, this is a very complex problem for computers with restrictions of time and processing. The detailed rules of this challenge are presented in Table 5.3.

Table 5.3: Description of the game 2 rules.

Section	**Description**
Objective	Destroy as many opponent buildings as possible within 10 minutes.
Setup	Two players. Perfect information (apart from simultaneous actions). Random terrain (small regular static obstacles). For each player: 3 randomly (but symmetrically) located control centers with 10 tanks each nearby. Some small mobile obstacles ("sheep") moving randomly.
Actions	move(x,y,s): start moving towards (x,y) with speed s attack(obj): attack object stop(): stops moving and attacking
Tournament Settings	Round-robin style. This evaluation scheme encourages to destroy many buildings quickly. Each pair of players will play a 2k games where k >= 1 depending on available time. The outcome of each game pair is evaluated as

follows:

- When all buildings of a player are destroyed the game ends instantly.

s = number of destroyed opponent buildings in game 1 and game 2
r = s(player 1) - s(player 2)
If r > 0, player 1 wins.
If r < 0, player 2 wins.
If r = 0: (tiebreaker)
  s = - elapsed time when last opponent building is destroyed in game 1 - elapsed time when last opponent building is destroyed in game 2
  r = s(player 1) - s(player 2)
  If r > 0, player 1 wins.
  If r < 0, player 2 wins.
  If r = 0, game is a tie

5.2 Results

This section presents the solutions adopted by the teams during the competition at CIn / UFPE. These solutions are described in details to show the developed applications using RTSCup. For didactic reasons, the description of the results is divided according to the proposed challenges.

Game 0

In this challenge, most teams used the same solutions with minor modifications. The general solution adopted, and the small differences among solutions will be presented. The first step to implement a pathfinding solution is to define a good way to represent and store the map and its structures, such as obstacles and buildings. The adopted choice was the Quadtree structure.

A Quadtree [Samet 1988] is a tree data structure in which each internal node has up to four children. Quadtrees are most often used to partition a two dimensional space by recursively subdividing it into four quadrants, or regions (Illustration 5.1). There are many ways to implement a Quadtree, but all forms of Quadtrees share some common features:

- They decompose 2-dimensional space into adaptable cells.
- Each cell (or bucket) has a maximum capacity. When the maximum capacity is reached, the bucket splits.
- The tree directory follows the spatial decomposition of the Quadtree.

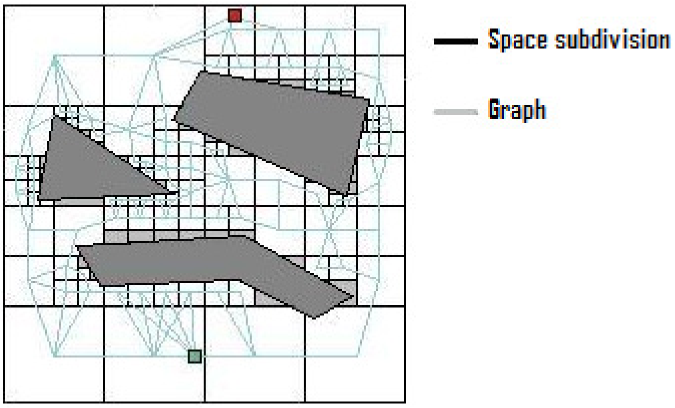


Illustration 5.1: Quadtree application - subdivision of the continuous space.

After this step, it is possible to apply a search algorithm to find the best path between two points in the map (Illustration 5.2). The teams have chosen the A* algorithm to find the paths. The A* is a best-first, graph search algorithm that finds the least-cost path from a given initial node to one goal node (out of one or more possible goals).

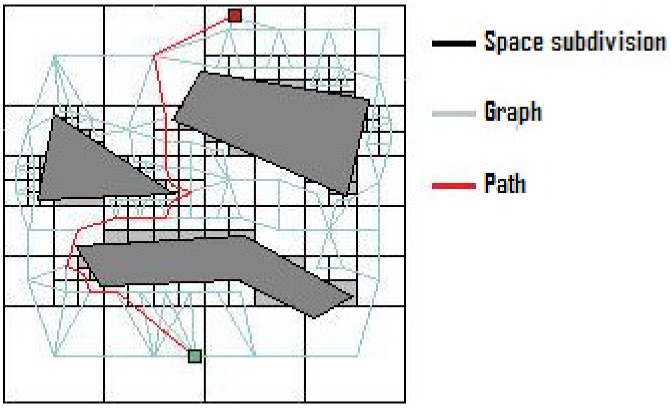


Illustration 5.2: Pahfinding application on a quadtree structure.

After the representation of the map and the search for a path, the agents can move around the map. However, the environment has many agents moving at the same time to different positions, and therefore they may collide with each other. To solve this issue it is necessary to define a collision avoidance method [Johnson 2004]. It is precisely at this stage that the teams differ. The two most used methods to address this problem are presented below.

- **Friendly-collision avoidance**

Units which are friendly to one another typically need some method of avoiding collisions and continuing toward a goal destination. One effective method is as follows: Every game cycle or so, make a quick map of which tiles each unit would hit over the next two seconds if they continued on their current course.

Each unit then checks to see whether it will collide with any other unit. If so, it immediately begins decelerating, and plans a new route that avoids the collision (Illustration 5.3). It can start accelerating again once the paths no longer cross. Ideally, all units will favor movement to the right side, so that units facing each other won't keep hopping back to the left and right (as people often do in life). Still, units may come close to colliding and need to be smart enough to stop, twirl to the right, and take a step backward if there's not enough room to pass, and so on.

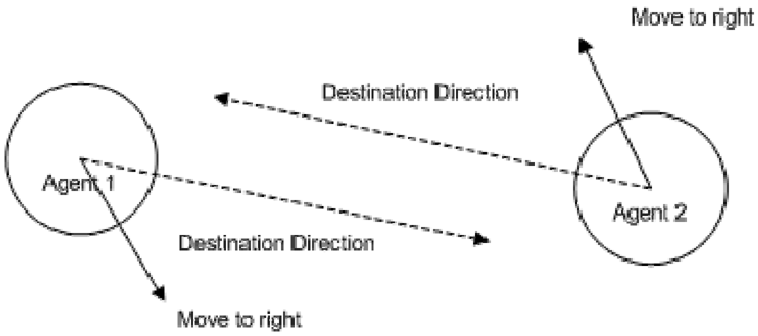


Illustration 5.3: Frienly-collision avoidance method.

- **Artificial Potential Field Force**

In this algorithm, a goal attracting force is defined in order to avoid the collision among agents. When the distance between two agents becomes smaller than a pre-defined effective avoidance distance, an artificial repulsion force is generated as a function of the distance resulting in repulsion between the two closing objects. The commanding force, which combining goal attracting force and the repulsion force, drives the agent toward the goal position without colliding with other objects in the work space (Illustration 5.4).

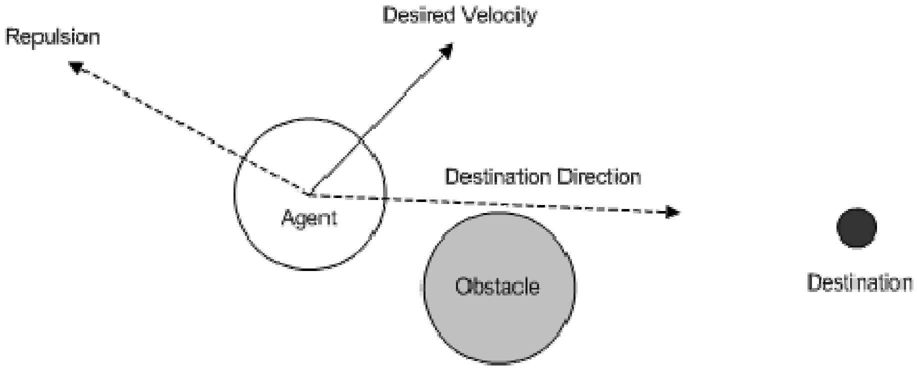


Illustration 5.4: Artificial potential field force method.

Game 1

Now we consider the solutions implemented by two student teams to Game 1. The first team has chosen a multiagent approach where one agent, the control center, accounts for calculating the distance between the mines and the control center. This happens in the initial part of the simulation. After that, workers start to negotiate with each other to define which agent will go to which mine. The negotiation can differ depending on the distance between agents to mines.

Besides the use of a multiagent system, this approach uses a blackboard structure that facilitates the negotiation between agents.

The second team has used the initial simulation cycles to perform a long initialization process, which maps the mines and calculate best routes to them. Mines store some information about their capacity and its distance to the control center, and workers are implemented as finite state machines.

In both solutions, they used a similar state machine to represent all agents' states. These states are presented in the Illustratoin 23.

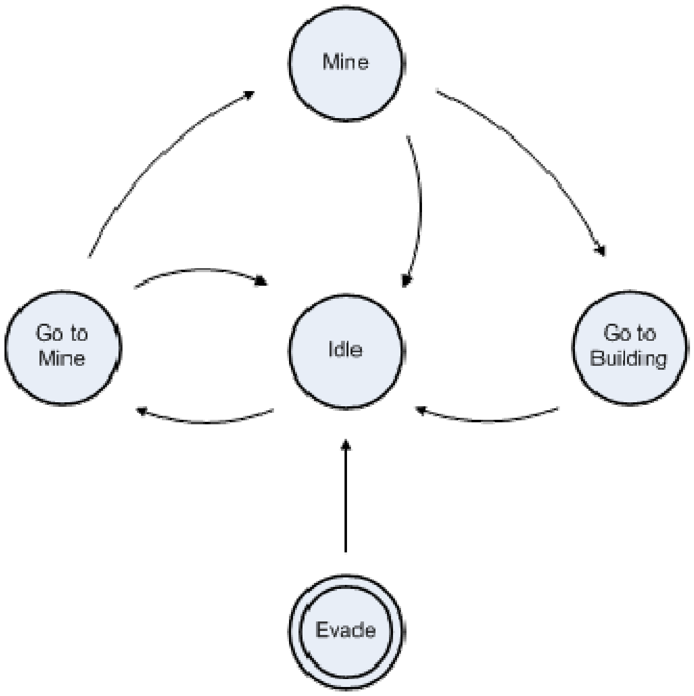


Illustration 5.5: Mineworker's state machine.

- **Evade**

This is the initial state of the Finite-State Machine. In this state the agent starts a random motion to avoid collision among agents and to spread agents over the map. After that, the agents' state is set to *Idle*.

- **Idle**

In this state, the agent search for the closest resource patch to mine. After the mine choice, the agents' state is set to *Go to Mine*.

- **Go to Mine**

The agent moves from its position to the resource patch location. When the agent reaches its destination, its status will change to *Mine*.

- **Mine**

The agent collects the resources until the mine exhaust or agent's maximum permitted load has been reached. In the first case, the agent's state is set to *Idle*; otherwise the agent's state is set to *Go to Building*.

- **Go to Building**

In this state, the agent calculates the best route to the building and move to its destination. When the agent reaches the building, it delivers all collected resources to the building. After that, its state is changed to *Idle*.

Game 2

Here we consider the approaches implemented by two student teams to Game 2. The first team has chosen an approach where the agents define a suitable spot to meet close to the average tank position. Then each tank moves to the combined position. When joining is finished, the tanks start hunting and attacking the closest enemy tank with the entire group. When all tanks are destroyed, bases are attacked. Weakest targets are attacked first while minimizing overkill.

In the second solution, the student team opted for the definition of two different roles: leader and soldier. The leader is responsible for plans path to the nearest base. And soldiers follow the leader. When a target is encountered, line or wedge formation is produced. The general algorithm is presented below.

1. Locate closest enemy base and move towards it in snake formation.
2. If during this traversal, enemy tanks are encountered, attack the weakest of all tanks in range.
3. Tanks move towards the weakest target while firing at the weakest target in range.
4. If the base is destroyed, locate a new base and move towards it.
5. If no more enemy tanks are in sight, resume formation and travel to the enemy base and attack it.

Despite the first solution appear to be too simple, it showed superior performance compared to the second one.

5.2.1 Discussion

The challenges presented instigate the research and creation of practical solutions using the concepts learned in the classroom, such as planning, decision making theory and multiagent systems. Besides the application of conceptual knowledge in practice, the offered challenges assist students in developing an analytic process for problem's investigation by enabling the identification of sub-issues involved in each of the challenges, and the application of the divide-and-conquer paradigm to reach a solution to the original problem.

In the first problem (Game 0), for example, it is necessary to identify that there are three main sub-problems: representation and decomposition of the space (map); evaluation of the best path; and avoid collisions with other moving objects. Thus, the student can handle the sub-problems separately and achieve an efficient solution to the problem.

We can highlight some points about the evaluation of the students' teams in the scenarios. In the second problem (Game 1), resource gathering, we have noticed that the main tactic used by the teams was to collect mineral from the mines closer to the control centre. In this way, a significant issue of these approaches was a good specification of the pathfinding algorithm.

An interesting observation was that some teams had a very good performance in some game maps; however such performance did not come up again when other maps were employed. Thus we could clearly conclude that particular features of the environment, such as mines among obstacles, had significant impact on the algorithms because such features could configure "logical traps". Such fact was important to stress the usefulness of RTS environments as benchmarks, which can identify strengths and weaknesses of an algorithm in specific configurations and scenarios.

This issue is more critical in the third scenario (Game 2). As it happens in real world, an attack/defense strategy is strongly dependent on features of the environment. For example, a combat strategy for an open battlefield cannot be used in a scenario with several obstacles (e.g., trees), which can make difficult the moving of military divisions. Again, the use of a RTS environment as benchmark enables the evaluation of strategies, such as the advantages and disadvantages that each strategy can offer for specific configurations of the environment.

Unfortunately the experiments have also stressed some problems previously discussed. The process of choosing maps, for example, has not used a proper methodology, so that the maps only cover a small part of the existent possibilities. In addition, the analysis of events is an exclusive task of evaluators, so that they own need to infer, for example, if a specific approach is (not) efficient in a specific configuration of the simulation scenario.

5.2.2 Improvements

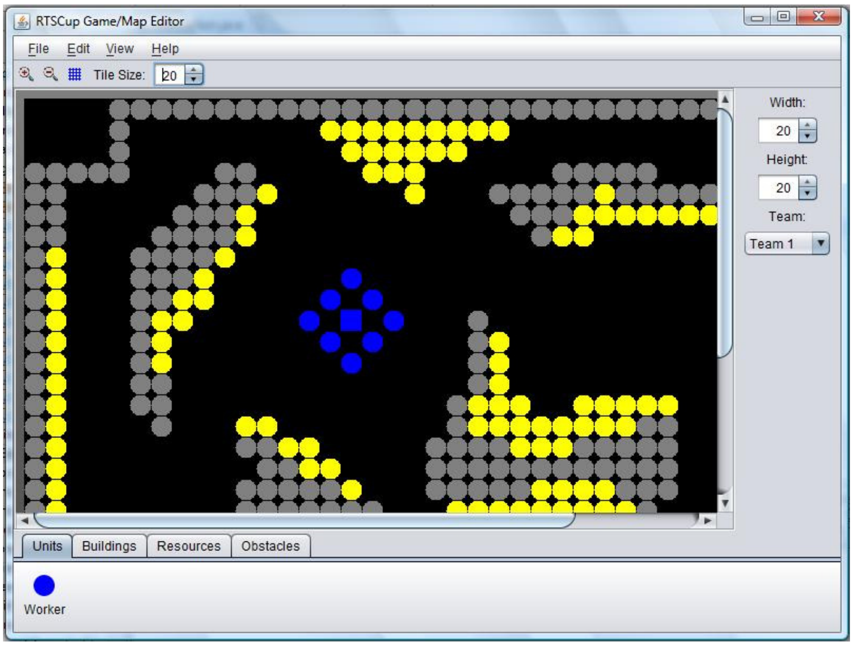
During the experiments several points of improvement, both in terms of testbed and software engineering, were identified. The main ones are listed below:

- **Agent Development Kit**

The ADK [Vieira 2008] has been completely rebuilt to fit the needs of the user in terms of usability. In addition, the related documentation was created and made available at the project site [RTSCup 2007]. Currently the creation of a simple agent (e.g. dummy agent) is a very simple process and consumes only a few minutes.

- **Map Editor**

The RTSCup enables the customization of the simulation parameters to create many different problem configurations. Although those parameters are described in a XML file (human-legible format), it was not easy to configure a specific test scenario. Therefore a tool was designed to facilitate the map and game edition (Illustration 24).

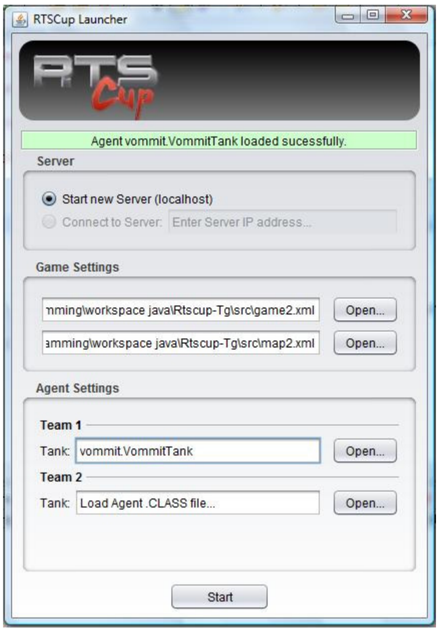


Tile Size: 20
Width: 20
Height: 20
Team: Team 1

Illustration 5.6: RTSCup map editor.

- **Initiallization Step**

The startup of the simulation requires the execution of all modules involved. This includes the server, the viewer, and all client applications involved (one per team). With the use of the testbed by the students this activity proved to be tremendously boring. Thus an application was developed to facilitate the boot stage. This RTSCup Launcher enables the startup of the testbed in a single click (Illustration 25).



Agent vommit.VommitTank loaded sucessfully.

● Start new Server (localhost)
○ Connect to Server: Enter Server IP address...

nming\workspace java\Rtscup-Tg\src\game2.xml
amming\workspace java\Rtscup-Tg\src\map2.xml

Team 1 Tank: vommit.VommitTank
Team 2 Tank: Load Agent .CLASS file...

Illustration 5.7: RTSCup launcher application.

- **Efficiency**

The efficiency of the implementation has been improved in all editions of the competition. As a result, several corrections and improvements were made showing improvements of both server (entire simulation) and network (exchange of messages among the modules) processing speed.

- **Collision System**

The collision also proved to be a problem during the first experiments carried out because of the complexity involved in treating the collision of the many entities in the simulation. The algorithm used in the collision was redesigned to both prevent failure and speed up detection of collisions.

5.3 Conclusions

The RTSCup has been used in many different opportunities. In this chapter was presented the experiments involving the students of the Intelligent Agents course in the Informatics Centre – UFPE at both undergraduate and postgraduate levels. Furthermore, the RTSCup has been used in scientific productions by both undergraduate and postgraduate students [Moura 2006, Cisneiros 2008, Sette 2008]. These experiments served to validate the operation of the testbed and its main features.

One of the most important features is the focus on research and productivity. This feature involves both the ease of use, with a very low learning curve, and the possibility for advanced customization of the experiments in a transparent way. As a result, the students could create different test scenario for validation of their solutions.

This feature added to the software engineering's characteristics such as maintainability, evolvability and interoperability allows both the insertion of the community in developing the RTSCup testbed and the continuation of the project.

The evidence of this is that many students have been involved, directly or indirectly, in the creation and improvement of both the simulator and auxiliary tools. This is an important feature in a research project to prevent lost time and effort involved in creating the research project.

Chapter 6

Conclusion

This chapter provides some closing comments on the main topics discussed in this dissertation, including the contributions made and directions dor future work.

6.1 Contributions

This dissertation started out with a set of ambitious tasks. It has reached most of its initial objectives, but more work needs to be done if we are to have a better testbed to facilitate the research in MAS area. Let's start with its main accomplishment, before turning to its shortcomings and setting out some avenues for future research.

The main contribution of this dissertation is that it developed a new simulation environment to enable the creation and analysis of new MAS techniques and the evaluation of such systems in different test scenarios. In brief, the RTSCup objective is to enable that users keep the focus on their research rather than on understanding the simulator operation, or learning a new programming language

A great number of testbeds have been proposed such as RoboCup Rescue, Trading Agent Competition and ORTS. While these constitute critical contributions to the research in MAS area, they lack in many important aspects and do not have the appropriate requirements to help researchers to define, implement and validate their hypothesis. Part of the reasons for writing this dissertation was to avoid the development of such simulation environments by researches as happen when testbeds does not satisfy their needs.

The RTSCup provide support facilities to elaborate hypothesis about a particular system characteristic such as a new coordination algorithm or negotiation protocol. Some of these facilities are described below.

- **Supporting experimentation**

The RTSCup provide a convenient way (built-in set of parameters that characterize the environment's behavior) for the researcher to vary the behavior of the worlds in which the agent is to be tested (e.g. map editor). In this way, many different test scenarios can be created by varying the world's parameters systematically.

- **Agent Development Kit**

The RTSCup's ADK simplify the procedure for developing interesting agents for the simulation system. This purpose is two-fold. Firstly, it is to abstract away many of the gritty details of the simulation environment so agent developers can focus on agent development. Secondly, it is provide a toolbox for building high-level behaviors.

- **Usability**

The RTSCup is easy to learn and use allowing the user greater productivity. The RTSCup provide provide a good documentation and support to the research community through the project website.

6.2 Applicability

The RTSCup has been used in many experiments involving the students of the Intelligent Agents course in the Informatics Centre – UFPE at both undergraduate and postgraduate levels. The proposed experiments instigate the research and creation of practical solutions using the concepts learned in the classroom such as planning, decision making theory and multiagent systems. Besides the application of theoretical knowledge in practice, the offered challenges assist students in developing an analytic process for problem's investigation by enabling the identification of sub-issues involved in each of the challenges and the application of the divide-and-conquer paradigm to reach a solution to the original problem.

Furthermore, the RTSCup has been used in scientific productions by both undergraduate and postgraduate students. These experiments served to test the implemented features in real applications and evaluate the testbed performance. The result of the experimentations is that the testbed enables the research of many different AI problems. Some of the problems that can be investigated in this case are:

- **Pathfinding**

Teams need to move along the best routes so they decrease time and effort. It is common more elaborated versions of this problem, such as involving routes along unknown lands or facing dynamic obstacles (e.g., enemy teams);

- **Patrolling**

A team can keep a specific area on control, or cover an area to find resources or enemies in an optimized way;

- **Resource allocation** (schedule)

Each component of a team is a resource that must be allocated in some activity, such as defense or attack. Allocation processes must observe, for example, the *load balancing* of activities, specific resources do not become overloaded while others are free;

- **Actions prediction**

A team can try to observe and model the behavior of others; it predicts their behaviors and can anticipate them;

- **Coordination**

Components of a team ideally need some kind of coordination to improve the whole work and they do not disrupt each other. For example, during an attack maneuver the team needs to decide if its members will attack the flanks, or if the infantry should wait by the intervention of the artillery before moving forward;

- **Deliberative and reactive architectures**

Each team needs to deliberate on the strategies to be deployed in a battlefield. However, several reactive actions, such as reactions to eventual ambushes, must also be considered;

- **Strategy and tactic decisions**

Each team must plan and conduct the battle campaign (strategy) in the same way that must organize and maneuver forces in battlefield to achieve victory (tactics).

6.3 Outlook

A future goal is to create a public contest to support the community interaction to motivate the investigation of solutions in specific areas (Game 0, Game 1 and Game2) and integrate the research community, providing an environment where approaches are discussed and results raised from contests, can be analyzed in a comparative way. The propostal is to create competitions, where any people who wants to participate, would send his/her source code and a brief explanation about his/her strategy.

Despite its accomplishments, this proposed testbed has its weaknesses. Allow me to mention a few of them, with some elements of solution for future research:

- **Automatic Reports**

One of the most important testbeds' features is the ability to compare different competing systems. The automatic report ability can help in the evaluation of solutions in a post-simulation stage, as well as provide a better basis for measure the efficiency of approaches.

- **Fog of War**

The fog of war is a term used to describe the level of ambiguity in situational awareness experienced by participants in military operations. In computer games the fog of war is related to the more limited sense of enemy units or buildings being hidden from the player. Often this is done by obscuring sections of the map already explored by the player with a grey fog whenever they do not have a unit in that area to report on what is there. The player can still view the terrain but not any enemy units on it. The fog of war ability enables the configuration of the environment properties to simulate a partially accessible world.

References

[Adams 2006] Adams, D. (2006). *The State of the RTS*. Available from: http://pc.ign.com/articles/700/700747p3.html [Accessed 26 Aug 2007]

[Age 2005] Age of Empires, (2005). *Age of empires site*. Available from: http://www.ageofempires.com [Accessed 10 Jul 2008].

[Aha 2004] Aha, D. and Molineaux, M. (2004). Integrating learning in interactive gaming simulators. *In: D. Fu & J. Orkin (Eds.) Challenges in Game AI: Papers of the AAAI'04 Workshop. San José, CA: AAAI Press.*

[Boson 2005] Boson, (2005). *Boson Homepage*. Available from: http://boson.eu.org [Accessed 25 Aug 2007].

[Buro 2003] Buro, M. (2003). Real-Time Strategy Games: A new AI Research Challenge. *In: Proceedings of the 2003 International Joint Conference on Artificial Intelligence, Acapulco, Mexico.*

[Cisneiros 2008] Cisneiros, V. (2008). Aperfeiçoamento do Simulador de Jogos de Estratégia em Tempo Real RTSCup. *Thesis, (Graduate Project).* UFPE.

[Cohen 95] Cohen, P. (1995). Empirical Methods for AI. *MIT Press*, 1995

[Glest 2007] Glest, (2007). *Glest: A Free 3D RTS Project*. Available from: http://www.glest.org [Accessed 25 Aug 2007].

[Hanks 1987] Hanks, S. and McDermott, D. (1987). Nonmonotonic logic and temporal projection. *Artificial Intelligence*, 33(3): 379-412.

[Hanks 1993] Hanks, S., Pollack, M., Cohen, P. (1993). Benchmarks, Testbeds, Controlled Experimentation and the Design of Agent Architectures. *In: AI Magazine 13(4), June 17, 1993.*

[JaRTS 2006] JaRTS, (2006). *JaRTS: A Java Real-Time Strategy Engine*. Available from: http://www.cin.ufpe.br/~vvf/jarts [Accssed 25 Aug 2007].

[Johnson 2004] Johnson, G. (2004). Avoiding Dynamic Obstacles and Hazards. *AI Games Programming Wisdom 2*. Charles River Media, Hingham, Massachusetts, pp. 161-170.

[Kitano 2001] Kitano, H. and Tadokoro, S. (2001). RoboCup Rescue: A Grand Challenge for Multiagent and Intelligent Systems. *In: AI Magazine*, 22(1):39-52.

[Laursen 2005] Laursen, R., Nielsen, D. (2005). Investigating small scale combat situations in real time strategy computer games. *Thesis, (Masters)*. University of Aarhus, Department of computer science.

[Marietto 2002] Marietto, M.B., David, N., Sichman, J.S. and Coelho, H. (2002). Requirements Analysis of Agent-Based Simulation Platforms: State of the Art and New Prospects. *In: Proceedings 3rd. International Workshop on Multi-Agent Based Simulation (MABS'02), Bologna, Italy.*

[McDermott 2000] McDermott, D. (2000). The 1998 AI Planning Systems Competitions. *In: AI Magazine, 4(2):115-129.*

[Morimoto 2002] Morimoto, T. (2002). How to Develop a RoboCupRescue Agent. Available from: http://ne.cs.uec.ac.jp/~morimoto/rescue/manual/manual-1_00.pdf [Acessed 19 Aug 2007].

[Moura 2006] Moura, J. (2006). Uma Estratégia Eficiente de Coleta Multiagente para Jogos RTS. *Thesis, (Graduate Project)*. UFPE.

[Noda 1996] Noda, I., Matsubara, H. (1996). Soccer Server and Researches on Multi-agent Systems. *In: Proceedings of the IROS-96 Workshop on RoboCup, pages 1-7, November 1996.*

[ORTS 2003] ORTS, (2003). *ORTS Homepage*. Available from: http://www.cs.ualberta.ca/~mburo/orts [Accessed 19 Aug 2007].

[Ponsen 2005] Ponsen, M., Lee-Urban, S., Muñoz-Avila, H., Aha, D. and Molineaux, M. (2005). Stratagus: An open-source game engine for research in real-time strategy games. *In: Proceedings of the IJCAI Workshop Reasoning Representation, and Learning in Computer Games. Edinburgh, UK.*

[Pyke 1984] Pyke, G. (1984). Optimal foraging theory: a critical review. *In: Annual Review of Ecological Systems, November 1984.*

[RCR 2001] RoboCup Rescue, (2001). *RoboCup Rescue Homepage*. Available from: http://www.robocuprescue.org [Accessed 25 Aug 2007].

[Robocode 2001] Robocode, (2001). *Robocode Homepage*. Available from: http://robocode.sourceforge.net [Accesses 25 Aug 2007].

[Robocup 1993] Robocup, (1993). *Robocup Homepage.* Available from: http://www.robocup.org/ [Accessed 17 Aug 2007].

[Rollings 2006] Rollings, Andrew; Ernest Adams (2006). Fundamentals of Game Design. Prentice Hall.

[Russel 2003] Russell, S., Norvig, P. (2003). Artificial Intelligence: A Modern Approach. 2nd Edition. *Prentice Hall*, 2003, pp 41-44.

[RTSCup 2007] RTSCup, (2007). *RTSCup Homepage*. Available from: http://www.rtscup.org [Accessed 25 Aug 2007].

[Ryan 2006] Ryan, R., Rigby, C., Przybylski, A. (2006). The Motivational Pull of Video Games: A Self-Determination Theory Approach. *In: Motivation and Emotion*, Vol 30, No 4, pp. 344-360. December, 2006.

[Samet 1988] Samet, H. (1988). An Overview of Quadtrees, Octrees, and Related Hierarchical Data Structures. *NATO ASI Series*, vol. F40, 1988.

[Sanchez 2007] Sanchez, J., Canton, M. (2007). Software solutions for engineers and scientists. *CRC Press*, 2003, pp 726-730.

[Schwab 2004] Schwab, B. (2004). AI Game Engine Programming. *Charles River Media*, 2004, pp 97-112.

[Sette 2008] Sette, S. (2008). Um Estudo de Estratégias para Coleta de Recursos em Ambientes Multiagentes. *Thesis, (Graduate Project)*. UFPE.

[Starcraft 1998] Starcraft, (1998). *Starcraft site*. Available from: http://www.blizzard.com/starcraft [Accessed 10 Jul 2008].

[Stone 2003] Stone, P. (2003). Multiagent Competitions and Resarch: Lessons from RoboCup and TAC. *In: Gal A. Kaminka, Pedro U. Lima, and Raul Rojas, editors, RoboCup-2002: Robot Soccer World Cup VI, Lecture Notes in Artificial Intelligence*, pp. 224–237, Springer Verlag, Berlin.

[Stratagus 2007] Stratagus, (2007). *Stratagus: A Real-Time Strategy Engine*. Available from: http://stratagus.sourceforge.net/ [Accessed 19 August 2007].

[Sussman 1975] Sussman, G. (1975). A Computer Model of Skill Acquisition. *In: American Elsevier, New York, 1975.*

[TAC 2001] TAC, (2001). *TAC Homepage*. Available from: http://www.sics.se/tac [Accessed 19 Aug 2007].

[Vieira 2006] Vieira, V., Siebra, C., Weber, R., Moura, J., Tedesco, P. and Ramalho, G. (2006). JaRTS: Java RTS Simulator. *In: Proceedings of the 5th Brazilian Symposium on Computer and Digital Entertainment, 8-10 November 2006 Recife, Brasil.*

[Vieira 2007a] Vieira, V. (2007). RTSCup: Ambiente de Simulação de Jogos RTS com Foco na Inteligência Artificial. *In: Proceedings of the 6th Brazilian*

Symposium on Computer Games and Digital Entertainment, 7-9 November 2007 São Leopoldo, Brazil.

[Vieira 2007b] Vieira, V., Siebra, C., Weber, R., Moura, J., Tedesco, P. and Ramalho, G. (2007). A Proposal of Strategic Game Simulator and Benchmark for Multiagent Systems. *In: Proceedings of the 10th International Computer Games Conference, 21-23 November 2007 Louisville, USA.*

[Vieira 2007c] Vieira, V., Siebra, C., Weber, R., Moura, J., Tedesco, P. and Ramalho, G. (2007). Evaluation of Multiagent Teams via a new Approach for Strategy Game Simulator. *In: Proceedings of the 8th Annual European GAMEON Conference, 20-22 November 2007 Bologna, Italy.*

[Vieira 2007d] Vieira, V., Siebra, C., Weber, R., Moura, J., Tedesco, P. and Ramalho, G. (2007). RTSCup Project: Challenges and Benchmarks. *In: Proceedings of the 6th Brazilian Symposium on Computer Games and Digital Entertainment, 7-9 November 2007 São Leopoldo, Brazil.*

[Vieira 2008] Vieira, V. (2008). *RTSCup: Agent Development Kit*. Available from: http://www.rtscup.org/images/stories/documentation/adk_0_1.pdf [Accessed 22 Aug 2008].

[Warcraft 1994] Warcraft, (1994). *Warcraft site*. Available from: http://www.blizzard.com/war1 [Accessed 10 Jul 2008].

[Weiss 1999] Weiss, G. (1999). Multiagent Systems. *MIT Press*, 1999, pp. 3.

[Wellman 2001] Wellman, M., Wurman, P., O’Malley, K., Bangera, R., Lin, S., Reeves, D. and Wash, W. (2001). A Trading Agent Competition. *In: IEEE Internet Computing*, 5(2):43-51.