RTSCup Simulator Manual

for RTSCup Simulation System version 0.1

1st Edition

written by Vicente Vieira Filho

web page: http://www.rtscup.org

1 Preface

This manual is written for computer science/engineering researchers and students to acquire sufficient knowledge on the RTSCup architecture and mode of function.

In the RTSCup simulation, the RTS agents such as villager, shepherd, miner, priest and many others types (defined by the user) act in large virtual worlds. The RTSCup is a simulator of RTS games defined by users – the user can define all units and buildings types as the tech tree and finish conditions (objectives). This freedom enables the researchers / students to create your own game with your own rules to simulate a specific ambient (an RTS sub-problem) where the user can simulate your solution to many problems related to RTS environment.

The environment called the virtual world is simulated by the *RTSCup Simulation System* (RTSCSS). The RTSCSS consists of several modules, and the *kernel* module manages the whole of the RTSCSS. An RTSCup agent also is one of modules, and communicates with the kernel through a network to act in the virtual world. Necessary for developing an RTSCup agent is understanding of the RTSCSS, the virtual world, and the means by which an RTSCup agent communicates with the kernel.

This manual, at first, describes the RTSCSS and the virtual world. Then it describes how an RTSCup agent acts in the virtual world by communicating with the kernel. Additionally, it shows how to use the RTSCSS.

In this manual, the RTSCSS means the RTSCSS version 0.1 unless otherwise noted.

2 Introduction

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 benchmarks, which enable the evaluation of such systems in different test scenarios. Another interesting trend is the use of such environments in academic competitions. 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 where approaches are discussed and results, raised from contests, can be analyzed in a comparative way.

The aim of this manual is to present a simulator to real-time strategic games, called RTSCup, which can be used as a benchmark 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.

The remainder of this section is structures in three major parts: (1) presentation of the RTS game as a multi-agent environment; (2) identification of the RTS game as benchmark in the multi-agent area, and (3) presentation of the RTSCup simulator project and architecture.

2.1 RTS Game as MAS

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 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 (i.e., foraging, exploration, fight 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, i.e. the movement of the team members and the opponent team is highly dynamic.
- The perception of each player could be locally limited (i.e., fog of war).
- The role of each player can be different (i.e. worker collects resources and tanks attacks).

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 a MAS considering the following main features of a MAS: (1) each agent has incomplete information or capabilities for solving the problem and, thus, it has a limited viewpoint; (2) there is no system global control; (3) data are decentralized; and (4) computation is asynchronous.

2.2 Benchmark and MAS

The use of benchmarks, as evaluation method for multi-agent systems, has become a common practice for AI systems. In fact, this method 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 benchmarks. First, such method is an empirical form of evaluation, its users have to distinguish the important evaluation events, as well, interpret the results of such events. Second, there is not a consensus about what a representative benchmark is. Finally, results from benchmark 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 benchmark, whose generation of new states is executed in runtime and such states depend on the activities performed inside 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, enables the improvement of past systems and their approaches.

One of the main competitions related to multi-agent systems is the *RoboCup Rescue* (RCR). This competition uses a real-time distributed simulation system composed for several modules, all of them connected via a central kernel. The function of this system is to simulate the effects of earthquakes in urban areas. For that 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) provides the initial conditions of the simulated world.

RCR is an appropriate example of benchmark to multi-agent research because it implements several of the requisites that such systems requires. Those are:

i. Agents do not have control on the environment, their actions are not the unique events that can change it;

ii. 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;

iii. RCR environments are complex and each of their objects presents several attributes whose values can affect the simulation progress;

iv. The environment considers communication and coordination among agents as an important simulation issue, there are specific rules to control such communication;

v. There are several ways to measure the efficiency of approaches, for instance, number of victims or total area of fire;

vi. The RCR simulator has a well defined temporal model, which is based on configurable cycles;

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 in such way that it can be more useful during evaluations of some particular aspect.

2.3 RTS as Benchmarks

As discussed in the last section, the use of benchmarks, as an alternative, to evaluate MAS has received several criticisms, 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 represent a limited number of situations that could happen in real scenarios.

There exist cases, however, in which realism is not the main requirement to be considered. In such situations, the goal could be focused on the comparative evaluation among different approaches. For example, benchmarks currently used in the *Planning Systems Competition* and the *Trading Agent Competition* 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 RTS Game AI 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 RTS problems and their investigation.

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

i. Pathfinding: teams need to move along the best routes so that 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);

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

iii. 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;

iv. Actions prediction: a team can try to observe and model the behavior of others, it predicts their behaviors and can anticipate them;

v. 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;

vi. 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;

vii. 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 enables 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 users can create more appropriate scenarios to each kind of problem.

2.4 The RTSCup Project

The RTSCup Project is designed to maximize the community contribution and attain high-throughput in research itself. From the research perspectives, RTSCup is designed; (1) to enable users to focus their efforts on AI research; (2) to be used as benchmark in the multi-agent area; and (3) to integrate the research community.

3 RTSCup Simulation System

This section presents the simulator and its structure.

3.1 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 Figure 1.

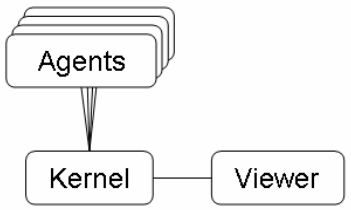

Figure 1 RTSCup Structure

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.

- Kernel: combines all information and updates the game world. In the kernel is presented all the main systems mentioned in the above section.
- Agent: one of the RTSCup simulator modules that communicates with the kernel through a network to act in the game environment. The user is responsible for the development of the agent.
- Viewer: another RTSCup simulator module that is used to visualize the game environment status.

3.2 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.

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. At the first cycle of the simulation, steps 1 and 2 are skipped.

i. The kernel sends individual vision information to each RTSCup agent.
ii. Each RTSCup agent submits an action command to the kernel individually.
iii. The kernel sends action commands of RTSCup agents to all sub-simulators.
iv. Sub-simulators submit updated states of the virtual world to the kernel.
v. The kernel integrates the received states, and sends it to the viewer.

vi. 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.

3.3 World Modeling

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, mainly, the following ones.

Unit: A unit is a general object that is able to represent all RTS units like villagers, swordsman, archer, priest and so on.

Building: A building is a general type that represents all construction types in the virtual world. This type can be customized, by modifying its properties, to represent different building types: house, barracks, storage pit, tower, and so on.

3.4 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 (Table 1).

header
body length
body ...

Table 1 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 (Table 2). 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

Table 2 packet of the RTScup protocol

The following table (Table 3) shows headers related to RTSCup agents. An 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.

Value	Header	Use
	To the kernel:	

1	AK_CONNECT	To request for the connection to the kernel
2	AK_ACKNOWLEDGE	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

Table 3 Headers of the RTSCup protocol

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 following sections. Here we describe the data types and structures of the elements constructing a body field.

ID_1	size	type	$object_1$
ID_2	string	id	$object_2$
...	...	properties	...
ID_n		...	$object_n$
0			TYPE_NULL

Table 4 IDs, string, object and objects representation in the RTSCup protocol

int An *int* element is a 32-bit signed integer. It represents coordinate values (x, y), an ID for the position property, etc.

IDs An *IDs* element consists of zero or more IDs and 0 as a terminator (Table 4). It represents the entrances, edges, etc.

String A *String* element consists of a byte size of the string and an ASCII character string. The string is aligned to a multiple of 4 bytes, but the byte size is set the original size. This element is used for inter-agent communications etc.

Object An *Object* element represents an object in the virtual world, and consists of the type, ID, and properties of the object (Table 4). The type is the object type name, and the properties consist of zero or more properties <*type, value*> and a *PROPERTY_NULL (0x00)* as a terminator:

$$\{< \text{type}_1, \text{value}_1 >, \cdots, < \text{type}_n, \text{value}_n >, \text{PROPERTY\_NULL}\}.$$

Objects An *Objects* element consists of zero or more Object elements and a terminator *TYPE_NULL (0x00)* (Figure 13). The kernel sends information of objects in the simulated world to an RTSCup agent by the Objects element: however, it does not necessarily contain all objects and properties; that is, the kernel sends only differences from those that it sent last time.

3.4.1 Commands

The agent programs make decision of their own behaviors according to diverse objectives.

3.4.2 Connection Commands

Here we describe the body of RTSCSS protocol blocks used in the initialization process.

Connect *version*

This message is used to start the connection process and indicate the version of the protocol used.

Connect_ok

This message is the response of the *connect* message indicating that the *connect* message had reached its destiny and no error has occurred.

Connect_error

This message is the response of the *connect* message indicating that the connection process had finished and an error has occurred.

Acknowledge

This message is the response of the *connect* message indicating that the connection process has finished successfully.

3.4.3 Action Commands

After receiving a KA_SENSE block, an RTSCup agent submits an action command as the will of its controlling object at each cycle. Although an RTSCup agent may submit two or more action commands at one cycle, the kernel adopts only the last one. As mentioned in §2.2, an RTSCup agent needs to submit an action command before taking half second after receiving a KA_SENSE block (considering a one second cycle), or the kernel regards that the RTSCup agent does not want to act at the cycle. Moreover, delayed action commands are ignored.

Move *position, speed*

Move a unit to the destiny position at this speed.

Build *position, type*

Create a new building in this position.

Fix *building*

Fix a building.

Collect *resource*

Collect resources from a resource patch.

Deliver *building*

Deliver the collected resources in the building..

Attack *building* | *unit*

Attack an enemy building or unit.

Cure *unit*

Cure (heal) a friendly unit.

Convert *unit*

Convert an enemy unit to become a friendly unit.

Rest *unit*

Stop current unit activity.

Train *unit*

Create a new unit in the simulated world.

Research *technology*

Research a new technology defined in the tech tree.

3.4.4 Sensory Information

An RTSCup agent has to construct a model that represents the virtual world in its interior based upon vision information contained in a KA_SENSE block at each cycle.

Sense *self, map*

Inform the agent modules of visual information sensed by the self unit. This message contains all self and map properties.

4 Objects in the simulated world

This section describes objects in the simulated world. Kernel models the simulated world as a collection of objects.

4.1 World

The world is the structure in the kernel responsible for the objects management and consistency.

Property	Type [or Unit]	Comment
width	[px]	The width of the world
height	[px]	The height of the world

Table 5 Properties of the world

4.2 Building

The building represents the house, barracks and another construction in a RTS game.

Property	Type [or Unit]	Comment
x, y	[px]	The x-y coordinates of the representative point.
type		The type of the building.
area	[px^2]	The area of the ground floor.
cost		The cost of the building.
hp	[health point]	The building resistance to enemy attacks.
attack	[health point]	The attack force defined for the life quantity that is removed from enemy unit.
defense	[health point]	The defense force defined for the life quantity that is absorbed from enemy attack.
load		The maximum load supported by the building.

Table 6 Properties of a building object

4.3 Unit

The unit represents the archer, worker and another unit types in a RTS game.

Property	Type [or Unit]	Comment
x, y	[px]	The x-y coordinates of the representative point.
type		The type of the unit.
area	[px^2]	The area of the unit.
cost		The cost of the unit.
hp	[health point]	The unit resistance to enemy attacks.
attack	[health point]	The attack force defined for the life quantity that is removed from enemy unit.
fieldOfAttack	px	The field of attack of the unit defined by the circumference

		ray.
defense	[health point]	The defense force defined for the life quantity that is absorbed from enemy attack.
load		The maximum load supported by the unit.
speed		The maximum speed of the unit.
maintenance		The cost of maintenance (in food) of the unit.
fieldOfView	px	The field of view of the unit defined by the circumference ray.
agility		The speed of the unit to dodge from enemy attacks and to attacks successive times.
cure		The cure percentage of the unit.
fieldOfCure		The field of cure of the unit defined by the circumference ray.
cureCooldown		The time between two successive cures.
convert		The convert percentage of the unit.
fieldOfConversion		The field of convert of the unit defined by the circumference ray.
convertCoolDown		The time between two successive cures.

Table 7 Properties of a unit object

4.4 Resources

Represent the resource patches found in RTS maps like gold, wood and food.

Property	Type [or Unit]	Comment
x, y	[px]	The x-y coordinates of the representative point.
load		The remaining load (resources quantity) in this patch.
area	[px^2]	The area of the resource.

Table 8 Properties of the resources

5 UDP

The RTSCup protocol assumes the *UDP* as the communication protocol between clients, viewer and server. The UDP is one of the core protocols of the internet. Using UDP, the clients and server can send short messages (using Datagram Sockets) to one another. UDP is sometimes called the Universal Datagram Protocol or Unreliable Datagram Protocol.

The UDP protocol was chosen because without the overhead of checking whether every packet actually arrived, UDP is faster and more efficient for many lightweight or time-sensitive purposes.

6 Connection kernel

As explained in section 3.2, kernel plays an important role in exchanging data. Starting simulation, kernel connects to viewers and agents in order. Before connecting agents, other modules are connected to kernel. Kernel starts simulation after all objects are connected.

During simulation kernel repeats the following steps.

i. The kernel sends individual vision information to each RTSCup agent.
ii. Each RTSCup agent submits an action command to the kernel individually.
iii. The kernel sends action commands of RTSCup agents to all sub-simulators.
iv. Sub-simulators submit updated states of the virtual world to the kernel.
v. The kernel integrates the received states, and sends it to the viewer.
vi. The kernel advances the simulation clock of the simulation.

6.1 Connection to Viewer

The viewer requires connection with the kernel using the default port 9998. The viewer and kernel doesn't need to run in the same machine. The connection process uses the following messages.

VK_CONNECT body
A VK_CONNECT block is used for an RTSCup viewer to request for the connection to the kernel. Its body consists of one int element: version (Table 9).

int version	This must be 0.

Table 9 VK_CONNECT body

KV_CONNECT_OK body

A KV_CONNECT_OK block is sent from the kernel, and the RTSCup viewer can get information of the whole of the simulate world. Its body consists of temporaryId and map (Table 10).

Objects map …	The information about the whole of the world.

Table 10 KV_CONNECT_OK body

KV_CONNECT_ERROR body

A KV_CONNECT_ERROR block is sent from the kernel in order to tell failure of the connection. Its body consists of the error's reason (Table 11). There are two reasons mainly:

- “unknown version” means that the given version by the VK_CONNECT block is invalid, and
- “connection not accepted” means that the viewer tried to connect after the game simulation had started.

string reason …	The reason why the connection failed.

Table 11 KV_CONNECT_ERROR body

VK_ACKNOWLEDGE body

A VK_ACKNOWLEDGE block is used for the RTSCup viewer to acknowledge the KV_CONNECT_OK block. Its body is empty (Table 9).

-	Empty body

Table 12 VK_ACKNOWLEDGE body

6.2 Connection to Agents

The viewer requires connection with the kernel using the default port 9998. The viewer and kernel doesn't need to run in the same machine. The connection process uses the following messages.

AK_CONNECT body

An AK_CONNECT block is used for an RTSCup agent to request for the connection to the kernel. Its body consists of three int elements: temporaryId, version, and agentType (Table 13). The temporaryId field is used to identify the RTSCup agent submitting an AK_CONNECT block from the socket from which other RTSCup agents may also submit a block. In this case, a map field of a KA_CONNECT_OK is not sent in order for many RTSCup agents to establish the connection as fast as possible.

int temporaryId	An arbitrary ID.
int version	This must be 0.
int agentType	The type of the agent. This is described on the configuration file.

Table 13 : AK_CONNECT body

KA_CONNECT_OK body

A KA_CONNECT_OK block is sent from the kernel, and the RTSCup agent can get information of its controlling object and the whole of the virtual world. Its body consists of temporaryId, id, self, and map (Table 14).

int temporaryId	The same value as that of the AK CONNECT block.
int id	The ID of the *self*.
Object self …	The object which the RTSCup agent controls.
Objects map …	The information about the whole of the world.

Table 14 KA_CONNECT_OK body

KA_CONNECT_ERROR body

A KA_CONNECT_ERROR block is sent from the kernel in order to tell failure of the connection. Its body consists of the temporaryId and the reason (Table 15). There are two reasons mainly:

- “unknown version” means that the given version by the AK_CONNECT block is invalid, and

- “no more agent” means that all controlled objects whose type is the same as the given agentType by the AK_CONNECT block have already been assigned to other RTSCup agents.

int temporaryId	The same value as the AK CONNECT block’s
string reason	The reason why the connection failed.
…	

Table 15 KA_CONNECT_ERROR body

AK_ACKNOWLEDGE body

An AK_ACKNOWLEDGE block is used for the RTSCup agent to acknowledge the KA_CONNECT_OK block. Its body consists of the ID of its controlling object (Table 16).

int id	The ID of its controlling object

Table 16 AK_ACKNOWLEDGE body

6.3 Sending Sensory Information to Agents

The kernel sends the KA_SENSE message to all agents every cycle. A KA_SENSE body consists of id, time, self, and map (Table 17). The time may be regarded as the number of cycles from the start of the simulation. The self and the map contain only differences from the last cycle. The map contains information of objects within a radius defined by the user for each unit (field of view attribute).

int id	The id of the Self.
int time	Current time in the virtual world.
Object self	The object which the RTSCup agent controls.
…	
Objects map	The vision information.
…	

Table 17 KA_SENSE body

7 Configuration Files

This section explains the configuration files used in the RTSCup simulation. The main configuration files are the game.xml and the map.xml described below.

7.1 Game.xml

The *game.xml* file describes the game properties to be used in the simulation. The kernel loads this file to read the game configuration and prepare to start the simulation. This file contains the game main properties: game name, map name (must be the same used in the *map.xml*), time (maximum game time, in milliseconds), game tick (time between two progress steps, in milliseconds), and teams (indication of the teams' quantity and names). Besides those properties, this file describes the building, units, and resources to be used in the map and in the simulation.

```
<game>
  <name>Game1</name>
  <map>Map1</map>
  <time>300000</time>
  <tick>100</tick>
  <teams>
    <team>
      <name>Team 1</name>
    </team>
  </teams>
  <buildings>
    <building>
      <name>Control Center</name>
      <cost>300</cost>
      <hp>1000</hp>
      <attack>15</attack>
      <defense>10</defense>
      <currentLoad>0</currentLoad>
      <maximumLoad>1000000</maximumLoad>
    </building>
  </buildings>
  <units>
    <unit>
      <name>Worker</name>
      <cost>50</cost>
      <hp>90</hp>
      <attack>15</attack>
      <attackRange>50</attackRange>
      <defense>10</defense>
      <maximumLoad>10</maximumLoad>
      <currentLoad>0</currentLoad>
      <speed>20</speed>
      <maintenance>5</maintenance>
      <fieldOfView>20</fieldOfView>
```

```
        <agility>80</agility>
        <cure>10</cure>
        <fieldOfCure>15</fieldOfCure>
        <cureCooldown>2</cureCooldown>
        <convert>80</convert>
        <fieldOfConvertion>15</fieldOfConvertion>
        <convertCooldown>3</convertCooldown>
        <miningRange>25</miningRange>
        <mineSpeed>1</mineSpeed>
        <deliveryRange>45</deliveryRange>
      </unit>
    </units>
    <resources>
      <resource>
        <name>Mine</name>
        <currentLoad>1000</currentLoad>
      </resource>
    </resources>
</game>
```

7.2 Map.xml

The *map.xml* describes the map properties and position of the objects in the simulated world on the beginning of the simulation. The kernel loads the file after reading the *game.xml* file. This file can contain linked information, such as the name of a building or team previously defined at the *game.xml* file. This file and the game.xml file, together, complete all the information required to start the simulation.

```
<map>
   <name>Map1</name>
   <width>1000</width>
   <height>1000</height>
   <buildings>
      <building>
         <team>Team 1</team>
         <name>Control Center</name>
         <x>100</x>
         <y>300</y>
         <width>50</width>
         <height>50</height>
      </building>
   </buildings>
   <units>
      <unit>
         <team>Team 1</team>
         <name>Worker</name>
         <x>300</x>
```

```
        <y>100</y>
        <width>20</width>
        <height>20</height>
      </unit>
    </units>
    <resources>
      <resource>
        <name>Mine</name>
        <x>80</x>
        <y>80</y>
        <width>20</width>
        <height>20</height>
      </resource>
    </resources>
    <obstacles>
    </obstacles>
</map>
```

8 Install of prototype version

The prototype version is available at URL: http://www.rtscup.org.

8.1 Unix Version

Requirements:
Java Development Kit 1.5

How to run:
Extract files in a folder
Execute batch file in the root folder.

8.2 Windows Version

Requirements:
Windows 98 or older
Java Runtime Environment 1.5 (or corresponding Microsoft Java VM)

How to run:
Extract files in a folder
Execute batch file at the main folder.

8.3 Configuration Files

The configuration files are located at the main folder. With replacing files, the simulated game and map will change accordingly.

8.4 Log Files

The server outputs logfiles of execution by deleting # marks in config.txt.

9 Programmer's Guide

This section helps the user to learn how to send actions to the kernel. For more information see the RTSCup site to download the API create to help you to develop an client program and connect it in the kernel.

9.1 Agent

This section describes how to use commands in section ### when you program agents. (Code samples are available at RTSCup site).

AK_MOVE body
A unit can move in the virtual world by submitting an AK_MOVE block (Table 13). The message must contain the destiny position and the movement speed.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.
int x	The x coordinate of the destiny position.
int y	The y coordinate of the destiny position.
float speed	The speed of the movement.

Table 18 AK_MOVE body

AK_BUILD body
A unit can build a construction in the virtual world by submitting an AK_BUILD block (Table 6). The message must contain the position of the building and its type.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.
int x	The x coordinate of the destiny position.
int y	The y coordinate of the destiny position.
int type	The building type.

Table 19 AK_BUILD body

AK_FIX body
A unit can fix a construction in the virtual world by submitting an AK_FIX block. The message must contain the building id of the building.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.
int buildingId	The building id.

Table 20 AK_FIX body

AK_COLLECT body

A unit can collect resources from a resource font in the virtual world by submitting an AK_COLLECT block (Figure 25). The message must contain the id of the resource font. The unit can only collect the resource if the unit is located near the resource font.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.
int id	The resource font id.

Table 21 AK_COLLECT body

AK_DELIVER body

A unit can deliver resources in the virtual world by submitting an AK_DELIVER block (Figure 25). The message must contain the id of the building where the unit wants to deliver. The unit can only deliver the resource if the unit is located near the building.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.
int id	The building id.

Table 22 AK_DELIVER body

AK_ATTACK body

A unit can attack enemy units and buildings in the virtual world by submitting an AK_ATTACK block (Figure 25). The message must contain the id of the unit or building which the unit wants to attack. The unit can only attack the resource if the unit is located within his field of attack.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.
int id	The building or unit id.

Table 23 AK_ATTACK body

AK_CURE body

A unit can cure friendly units in the virtual world by submitting an AK_CURE block (Figure 25). The message must contain the id of the unit which the unit wants to cure. The unit can only cure the friendly unit (target) if the unit is located near the friendly unit (target).

int time	Current time in the virtual world.
int id	The unit id.

Table 24 AK_CURE body

AK_CONVERT body

A unit can convert enemy units in the virtual world by submitting an AK_CONVERT block (Figure 25). The message must contain the id of the unit which the unit wants to convert. The unit can only convert the enemy unit (target) if the unit is located near the enemy unit (target).

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.

int id	The unit id.

Table 25 AK_CONVERT body

AK_REST body

A unit can stop itself in the virtual world by submitting an AK_REST block (Figure 25). The message must contain only the current time, in cycles, in the virtual world.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.

Table 26 AK_REST body

AK_TRAIN body

A building can train a unit in the virtual world by submitting an AK_TRAIN block (Figure 25). The message must contain only the current time, in cycles, in the virtual world.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.
int type	The unit type.

Table 27 AK_TRAIN body

AK_RESEARCH body

A building can research a new technology in the virtual world by submitting an AK_RESEARCH block (Figure 25). The message must contain only the current time, in cycles, in the virtual world and the technology to be researched.

int unitId	The unit id controlled by this user.
int time	Current time in the virtual world.
int technology	The technology to be researched.

Table 28 AK_RESEARCH body

References

[1] FILHO, V., ET AL. 2007. *A Proposal of Strategic Game Simulator and Benchmark for Multiagent Systems. In: Proceedings of the 12th International Computer Games Conference*, Louisville, Kentucky, USA.

[2] FILHO, V., ET AL. 2007. *RTSCup Project: Challenges and Benchmarks. In: Proceedings of the VI Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital*, Porto Alegre - RS, Brazil.

[3] RTSCUP, 2007. *RTSCup Homepage* [online] Available from: www.rtscup.org/ [Accessed 15 October 2007].