Delft University of Technology
Parallel and Distributed Systems Report Series

RTSenv: An Experimental Environment for Real-Time Strategy Games on Multi-Clusters

Siqi Shen, Otto Visser, and Alexandru Iosup
{S.Shen,O.W.Visser,A.Iosup}@tudelft.nl

(To be submitted for publication after revisions.)

report number PDS-2011-002

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Section
Faculty of Information Technology and Systems Department of Technical Mathematics and Informatics
Delft University of Technology
Zuidplantsoen 4
2628 BZ Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.twi.tudelft.nl

Information about Parallel and Distributed Systems Section:
http://pds.twi.tudelft.nl/

Abstract

Today, Real-Time Strategy (RTS) games entertain tens of millions of players world-wide. This growing population expects new game designs and more scalable games every year. However, few tools and environments exist for game designers and implementers; of these, even fewer are available to researchers and game communities. In this work, we introduce `RTSenv`, an environment and associated set of tools for RTS games. Our environment can configure and manage the main aspects of RTS games, such as maps, computer-controlled units, and game scenarios. `RTSenv` leverages multi-cluster systems and reactive fault tolerance mechanisms to perform robust, multi-machine, and multi-instance game experiments. Using our reference implementation of `RTSenv` in DAS-4, a real multi-cluster system, we show that our approach can be used in a variety of scenarios for game performance evaluation and for comparing game design choices. Our results give evicence that several common assumptions made by researchers about game workloads do not hold in general for RTS games and thus warrant a more detailed investigation.

Contents

List of Figures

List of Tables

1 Introduction

Entertainment is an important industry branch in many developed markets around the world. Currently, computer and video gaming (in short, gaming) is the branch of entertainment with faster growth than movies and music, and already one of the largest net incomes[1]. According to the Entertainment Software Association (ESA), gaming revenues have grown from 3.2 billion dollars in 1994 [29] to over 10.5 billion dollars in 2009 [17]. Among the different gaming genres, Real Time Strategy (RTS) games such as StarCraft II (one of the best-selling games of 2010 [10]) are played by millions of players daily. To address the increasing requirements of the gamers and the increasing competitiveness of the market, many RTS game technologies [46, 21] have appeared in the past five years, and a new generation of massively multiplayer RTS games, such as Picaroon [50], is currently under development. The development of a new RTS game can take multiple years to develop [41], and the abundance of challenges in the design, implementation, and testing of RTS games makes experimental tools invaluable. Moreover, few experimental tools are available for RTS game research. To address this situation, in this work we introduce `RTSenv`, an experimental environment for RTS games.

The increased pressure exerted by users on game developers mimics the situation of the general mass-market software. Between 1995 and 2005, the effects of increased competition and the availability of free tools on the software market have made users more conscious of existing choices, and increased requirements such as interactivity [22]. For RTS games, players incur a specific near-real-time constraint that limits the acceptable response times for issued commands to 200-300 milliseconds [61]. Moreover, professional players may require that issued commands are answered in 100 milliseconds or even less. When the near-real-time constraint is not met, RTS game players have poor gameplay experience, and often quit in favor of another game. The near-real-time constraint is difficult to meet continuously for most consumer-grade computers, even today. As a consequence, RTS games rely on complex technology that balances, sometimes equally, the computational, memory, and bandwidth components of the workload on the different system components.

In responding to increased user requirements, the design of mass-market software has greatly benefited from theoretical frameworks, such as the influential user-centered design theory [51], and from software engineering best-practices that include frequent testing and experimental tools. However, these theoretical and practical approaches have not been widely adopted in the development of mass-market games, and in particular RTS games, for reasons including differences in design goals (e.g., game designers have to include challenges, whereas the designers of productivity tools have to exclude them), late or no adoption of outside technology by gaming companies [53, 42], difficulty of developing tools for testing online applications and distributed systems, and a different development process [41] that balances both the software and the entertainment parts of the product. As a consequence, there currently exists a need for new tools and frameworks for developing and studying games, and in particular RTS games.

Conducting experimental studies on RTS games is potentially rewarding yet difficult. Although many RTS players compete online via the Internet, the number of players in one game session remains limited for popular RTS games to only 16 or even fewer players [62]. Many techniques have been proposed to improve the scalability [25, 7, 6] and operation [48] of (massively) multiplayer online games, including RTS games [46, 21]. To compare and improve these techniques, and to gain the insight that may lead to the development of new techniques, scientists need to deploy and observe RTS games in a multi-machine environment, similarly to other online applications. Although simulators may be appropriate for studying RTS games, they represent simplifications of the real systems and games; when their simplifying assumptions do not hold, their results fail to describe reality. Although many tools already exist for testing distributed systems, RTS games have additional, idiosyncratic, performance-affecting configuration parameters, such as the map size and the number

[1] A comparison between the incomes of the incomes reported by ESA (gaming) [63], MPAA (movies) [45], and RIAA (music) [56] in 2007 shows that gaming was by then earning similar amounts as the movies and music industries, but with a faster growth. Since then, income from gaming may have surpassed the income from either the movies and music industries in a number of countries, including UK and the Netherlands; see http://www.davidbordwell.net/blog/2010/12/27/still-cheating-on-video-games/, http://www.videogamesblogger.com/2008/04/09/global-videogame-sales-surpass-movie-industry-in-2008.htm, and http://www.gamedocs.com/blog/wordpress/2010/01/07/uk-game-revenue-surpasses-movies-in-2009/ for more details.

of units.

The limited availability of experimental tools makes it difficult to generalize or even understand the results of previous studies. It is perhaps symptomatic for the state of this research field that a recent study [43] argues against the practical viability of P2P-based games as a conclusion of real-world experiments, thus contradicting several previous simulation-based studies that indicate otherwise [25, 6]. Moreover, while some large gaming companies perform extensive game studies, including comprehensive user studies [53, 37, 69], few others can afford building the tools necessary for such approaches; even the few that do, have little expertise with large-scale experimental environments [41]. Despite recent interest in experimental tools [65, 39, 11], there currently exists no public experimental RTS environment.

In this work we introduce `RTSenv`, which is designed for experimental RTS game studies. `RTSenv` can be used to evaluate the performance of RTS games under a variety of game configurations and scenarios, from traditional performance evaluation to game design. Besides traditional system performance metrics such as CPU, memory, and network consumption, `RTSenv` can assess RTS-specific operational metrics such as player scores, and the number of active and profitable units. `RTSenv` can operate on a variety of physical platforms, from multi-cluster wide-area environments such as DAS-4 [4], to single, multi-core desktop computers. Our main contributions are:

1. We design and implement `RTSenv`, an experimental RTS environment (Section 3);
2. We implement `RTSenv` and show through experiments with a popular open-source game in a real multi-cluster infrastructure how `RTSenv` already supports two classes of scenarios, performance evaluation and game design choice (Section 4).

2 Background on Real-Time Strategy Games

In this section we discuss the background of our work. We first introduce the development process used by most games, with a focus on the stages that can benefit from `RTSenv` and similar tools. Then, we discuss the genre of strategy games, with a focus on real-time strategy games. Last, we introduce a use case for `RTSenv` that we will follow throughout this work.

2.1 The Game Development Process

In 2000, developing a game was already a costly, multi-year process: "In 2000 an established developer in North America would likely receive between $1 million and $3 million in advances paid out over 12 to 36 months for the development of a game." [5, Ch.3, p.15] The development process has become much more expensive and challenging since, and hundreds of people are routinely used today to develop the highest-quality (AAA) games. Due to the high costs and risks associated with game development processes, the market has started to mature, and today most games are developed through the same *basic game development process* [5, Ch.3] [34, Ch.1, p.30–32, and Ch.8, p.207–208] [14, Ch.9] [18, Ch.2] [41, Section 7.3].

The basic game development process may differ when the game is developed as a sequel, when the core technology used in the game can be licensed or already exists from previous projects, when the funding from the game can be obtained without demonstrating, when the development team is distributed around the world [18], when the game is targeted at casual as opposed to hardcore[2] gamers, etc. We describe in the following the main steps of the basic process and emphasize the steps (or stages) where an experimental environment such as that provided by `RTSenv` should be used:

[2]Casual gamers tend to spend little time and short sessions in a game, as opposed to the multi-hour sessions and the 8-20(!) hours spent per week by hardcore gamers [20]. As a consequence, games designed for casual gamers have less strict technological and game-mechanical requirements.

1. **Concept Discovery** is the stage where the main idea for the game to be developed is found and refined into a short description (often, a one-paragraph *game concept*) and a set of "demonstrables": a rough Game Design Document (GDD), inspiring visual representations of the game such as images or video, mockups of the game packaging, etc. The typical Concept Discovery stage includes brainstorming and other typical creative approaches, and no technology testing. The lack of a focus on testing technology means that this step will not typically benefit from an experimental environment such as `RTSenv`. This stage may take 1–2 months [41].

2. **Prototype** is the stage where the game concept and demonstrables are converted into concrete design documents, such as the full GDD, the Technical Design Document (TDD), etc.; later in this stage, the design documents are used to produce a prototype or a set of independent prototypes, each developed independently and focusing on a specific aspect of the game concept or GDD. Most prototypes focus on the experience of players, and in particular on the fun element of the game. Thus, unless the TDD includes technology that is core to the game[3], few or even no prototypes are focusing on testing technology. This stage may take between 2 and 6 months [34, 41].

3. **Approval** is the stage where the game is approved for pre-production and, conditionally, for production, typically by the marketing unit of the game developer or by a game publisher. This is the stage where sales pitches are made, during "green light meetings" [5, p.27]. Experimental environments may be used at this stage to demo the game prototype. The duration of stage depends mainly on the time allocated for obtaining a positive answer (an approval) and for completing necessary business and legal steps, such as agreeing on the ownership of the Intellectual Property that will be created throughout the development of the game; the minimum duration for this stage can be around 2 weeks.

4. **Pre-Production** is the stage where the core game elements are first implemented and tested. This is the stage where some of the game parts that are the most important or raise the highest risks are explored and tested, with a strong focus on functionality and performance, and without much care for aesthetics. Over the last decade, it has become common that the pre-production stage can only finish when "a playable example of what the shipping product should look like" [18, p.14] is produced. As a consequence, a *technical demo* may be required even early in this stage, for example to demonstrate the scalability of a simplified version of the core game engine. Thus, this stage requires a tool such as our `RTSenv` for much of its duration. This stage may take longer than 6 months [41], and both commercial successes such as *The Sims* and failures such as *Daikatana* took longer periods to complete this stage.

5. **(Full) Production** is the stage where the game is actually being built. All components are developed and integrated, and thorough per-component and complete product testing takes place. Thus, this stage requires a tool such as our `RTSenv` for its whole duration. This stage may require at least 12 months to complete [34, 41], but, similarly to the Pre-Production stage, it can take much longer; for example, *The Sims* and *Daikatana* required each over 3 years for this stage. It is customary [18, p.16] to split the project into milestones of a few months each, with up to several weeks allocated at the end of each milestone for Quality Assurance (QA) milestone reviews. These reviews require extensive experimentation and testing, especially for integration milestones, that is, milestones where several components are put together; identified deficiencies must be resolved and demonstrated through more experimentation and testing.

6. **Quality Assurance (Alpha and Beta)** is the stage during which the game is prepared for the launch; the goal of this stage is to identify and eliminate all the discrepancies between the actual game and the design, as expressed in the GDD, TDD, and similar documents. Finding and resolving these discrepancies

[3]The popular and influential game *Doom* (1993) is an example of a technology-driven game—its core game mechanics relied on 3D technology, which was not available at the time for consumer-level computers. The game included a prototype phase during which a novel, realistic 3D graphics engine was developed [38].

requires, similarly to the Production stage, experimentation and testing. Many of the results of this stage are quantitative, which emphasizes the need for tools such as `RTSenv`. An important aspect of this stage is that the team performing this stage is often not the same team that was involved in the Production stage. As a consequence, the experimental tools developed "in-house" by the development team have to be usable by a different team; this further motivates the need for general experimental tools such as `RTSenv`. This stage has two main parts, alpha candidate testing and beta candidate testing. An *alpha candidate* [18, p.22] is a build of the game that includes all of the components and all the representative content of the game, assembled into a playable game that includes most of the complete set of play options [18] [14, p.75]. During alpha candidate testing, game features and options may still be included and tuned. A *beta candidate* is a game build that includes all the features and options, and allows the player to play from the beginning to the end of the game; a beta candidate it is often called a fully playable version [14, p.75]. During beta candidate testing, software bugs and similar low-level issues are identified and corrected; the product is "polished" [41, 18]. Each testing step usually requires 2 weeks [41], but needs to be repeated if the candidate fails; alpha candidate testing of 4 weeks are not uncommon [18, p.21]. This stage requires 3 to 4 months to complete [34, p.31].

7. **Final (the Gold Master)** is the stage where the publisher receives and approves the "Gold Master disc". For console games, the game must also be approved by the console (platform) developer, for example Microsoft for XBox and Nintendo for the Wii. Although the distribution is now mostly digital, the final game build is still called the *Gold version*. The Gold version will be used, after approval, for manufacturing and distribution. Although not common practice today, the large number of online games that had major performance issues in their Gold version indicates that a more thorough approval process, based in part on experimental tools that can stress- and load-test the game, are needed at this stage. This stage requires up to 1 month for single-platform, single-market games [41], but may take up to 3 months for the games developed for multiple platforms and/or markets [18, p.21]; the latter situation is prevalent in the 2010 gaming industry.

8. **Pre-Launch Demo** It is customary [18, p.23] for games to be pre-launched, that is, to be showcased or demonstrated to the publisher, to the various project partners, and even to the public; the industry has numerous meeting venues, for example the Electronic Entertainment Expo (E3, `http://www.e3expo.com/`). Demonstrations can showcase incomplete versions of the game and even incomplete components of the game. Among the different kinds of demonstrations, the technical demos require the demonstration of specific technical features. Technical demos may require the use of experimental tools, and the cooperation of the production and QA teams. Pre-launche demos happen in parallel with other stages of the development cycle.

9. **Launch** is the stage during which the game is launched, that is, released to the market. It is customary to have a unified release date or period for all retailers and platforms [41], which means that all experimentation and testing must finish before this stage. A launch stage taking 1 or 2 months is common.

10. **Post-Launch Support** is the stage during which launched games are maintained and updated by their developers and publishers, and sometimes by a third-party team (the *game operators*[4]). Many of the most popular online games of today offer periodic technological updates and downloadable content, which motivate users to stay involved with the game over long periods of time; a business model based on subscriptions is used at this stage. The development of technological updates and new content requires similar, albeit reduced in size and duration, processes as the development of a complete game; however, special care must be taken at this stage to avoid antagonizing existing (paying) players through poor features and especially poor performance. Thus, experimental tools such as `RTSenv` are required throughout

[4]For example, for technical and legal reasons *World of Warcraft*, which was developed and published by Blizzard, is maintained by third-party game operators in China.

Table 1: Summary of the possible use of experimental tools in game development. The acronym SD replaces "Stade duration". The acronym UET replaces "Use of experimental tools", expressed as a fraction of the total duration of the stage.

Stage name Stage number	Concept 1	Proto. 2	Approval 3	Pre-Prod. 4	Prod. 5	QA 6	Final 7	Pre-Lnch. 8	Launch 9	Post-Lnch. 10
SD [months]	1–2	2–6	0.5	6+	12+	3–4	1–3	0.25+	1–2	12+
UET [%]	-	25*	25	50	75	100	100*	100	-	100

* predicted, not common industry practice between 2000 and 2010.

this stage. The goal of the game developers, publishers, and operators is to extend the duration of this stage for as long as possible; influential games such as *Starcraft* and *World of Warcraft* are very popular over 5 years after their launch.

The use of experimental tools is critical to many stages of the basic game development process. Table 1 summarizes for each stage the fraction of time that experimental tools would be useful, from the total duration of the stage. Experimental tools are particularly useful for the critical production, QA, pre-launch, and post-launch support stages. A large project with one year of production and five years of post launch support would benefit from such tools for over three quarters of its duration, and in particular throughout its money-making post-launch stage. At the other extreme, a short project with one year for the stages from pre-production to launch and no post-launch support (about 15 months in total), would benefit from experimental tools for over half of its duration.

2.2 The Strategy Game Genre

Over time, computer and video game developers have created a broad range of games. Among the many existing genres (a recent study [53] lists over twenty), strategy games represent over a third of the computer games market by units sold [16, p.8].

Strategy games are games where thinking and long-term planning are key skills required from players [57, p.11]. We follow and extend in this work the taxonomy of strategy games proposed by Rogers [57, p.11]:

1. **Abstract Strategy (AS) Games** are strategy games that abstract and/or refine a problem that requires strategic thinking to resolve. *Tic-tac-toe*, *chess*, and *go* are typical games in this class.

2. **Turn-based Tactical (TBT) Games** are non-AS strategy games that focus on very limited scenarios or situations that require strategic thinking to resolve. The *Cannon Fodder* series presents the player with a sequence of independent scenarios, each of which requires mostly tactical (short-term) decisions. The strategic element of the game is the selection of the structure and activity of the in-game squad deployed to resolve a specific scenario, as the squad participants gain experience that can be used in future scenarios.

3. **Turn-based Strategy (TBS) Games** are non-AS strategy games that focus on broad-scale problems that require strategic thinking to resolve. For example, in the *Civilization* series the player has to push a people through the many stages of civilization, from learning how to write to either conquering the other people or the challenges of inter-galactic flight.

4. **Real-Time Strategy (RTS) Games** are non-AS similar to TBT and TBS games, but the time in the game world progresses linearly with the real time experienced by players in the real world. The two main consequences of this difference are emotional (the player becomes more easily immersed in the game world [40]) and technical (the near-real-time requirements are difficult to enforce when the number of simultaneous players increases [61, 25, 6]); other discussions of these consequences exist [66, 24]. RTS games often use an omnipresent model, that is, players can see large portions of the game world and control

Figure 1: A typical OpenTTD screen.

many (even hundreds) of objects simultaneously. In typical RTS games such as the *Age of Empires* [54] and the *Starcraft* series, the player build virtual cities and harvest resources through the use of different mobile units.

2.3 Use Case: OpenTTD, a Real-Time Strategy Game

As main use case for our work we focus on OpenTTD [52], which is an open-source, popular RTS game. OpenTTD has been developed since 2004 by a community of volunteer game developers lead by Remko Bijker as an extension to the commercial game Transport Tycoon Deluxe by Chris Sawyer (MicroProse, 1994). OpenTTD is a client/server business simulation game with a wide appeal: the latest stable release (1.1.1) has been downloaded more than 130,000 times and the game is used as textbook companion for college-level business courses [26].

OpenTTD has all the features of a traditional RTS game. Several of these features are depicted in Figure 1 The game world, which may be randomly generated or selected from the many community-created maps, emulates the real world through a combination of realistic geography—hills, plains, water-courses—, economy, and demographics. The player has complete control over a transport company: buying transport vehicles such as buses, trains, boats, and airplanes; building transportation paths such as roads, train tracks, etc.; planning and managing the operation of the company; etc. During the course of the game, a successful player may control several tens to hundreds of vehicles. Rival companies compete against each other to achieve the highest profit by transporting passengers and goods. OpenTTD game sessions (games) can be played against human and/or Artificial Intelligence (AI)-controlled players. Currently, the OpenTTD community has created and made publicly available about 30 different types of AIs.

OpenTTD follows the typical program structure of an online RTS game, in which the game world is maintained on a server, and each player connects and interacts with the game world through a client application running on the computer/device of the player. One of the players often runs the server alongside a client; alternatively, the server is placed on a "neutral" computer for ranked games and competitions. The game server repeatedly executes a main game loop comprised of the sequence "get and process player input", "update the global game world", and "send (small) updates about the game world to each player"; based on the latter step, each client can reconstruct the state of the world, effectively performing the same updates of the global game world as the server.

OpenTTD is highly configurable. Before starting the game, the players can specify the map structure, the maximum number of vehicles, the inflation rate of the economy, etc. Players can select the amount of water (via the sea level), the "hilliness" of the map, and the density of economic resources (via the industry and town density). This level of configuration detail, which is common for RTS games [58], makes modeling such games a difficult and recurrent process.

OpenTTD vs. other RTS games Although a variety of RTS games exist, the AAA game market has focused since 2005 on games such as the *Age of Empires* and the *Starcraft* series, each offering the player large worlds, the possibility to control large numbers (tens or hundreds) of vehicles, and many options to build real-world-like cities. With the exception of in-game combat, OpenTTD has similar features to these games. OpenTTD also employs a non-violent alternative to traditional combat, in that players may use advanced tactical and micro-management[5] skills to restrict the free movement of the vehicles of other players. OpenTTD can also offer sufficient challenge to players of various ability and experience: from the many available AIs, several can give a good challenge to the good human player, while others can win comfortably against the expert human player.

3 The RTSenv Environment

In this section we present the RTSenv environment–in turn, the requirements, the architecture, the experimental flow, the RTS-specific features, and the implementation details.

3.1 Requirements

Starting from the use case, we synthesize three main requirements for an experimental RTS environment:

1. *Control experiments*: the environment must have the ability to control experiments (start, stop, monitor, use results) without input from a human experiment manager. The environment must be able to perform experiments when real users are present, but also when no real users are present; for the latter, the environment must support both the use of traces/replays and the use of AI-controlled players. Last, the environment must support RTS-specific configuration parameters, such as map structure, unit count, number of players (including AIs), etc.

2. *Evaluate and model RTS operation*: the environment must support traditional performance measurements; in particular, it should be able to report metrics concerning resource consumption (CPU, network, disk, memory) over time. Although many traditional online gaming studies have focused exclusively on the network aspects of these Internet-based applications, and in particular on latency and throughput, another study [23] has shown that users may *not* be willing to leave a game even when experiencing high network delay; in other words, that regular users may be insensitive to Quality-of-Service degradation while in the flow of playing [15]. However, players may respond to other sources of performance degradation, such as the server's processing (CPU) speed. The environment must also support RTS-specific operation abstractions and metrics, such as player scores, and the number of active and profitable units.

3. *Compare game design choices*: the environment must have support for comparing the results obtained from different scenarios as support for game design decisions. For example, it is common for game designers to package AI players with commercial games; however, selecting a specific AI type or configuration for a specific (random) map should be based on an in-depth analysis of the AI performance/resource consumption trade-off.

[5] Micro-management of troops, or microing, is an essential skill for the professional player, see http://gaming-shed.com/2011/06/13/starcraft-2-guide-to-microing/ and http://www.starcraftzone.com/forums/index.php?topic=962.0.

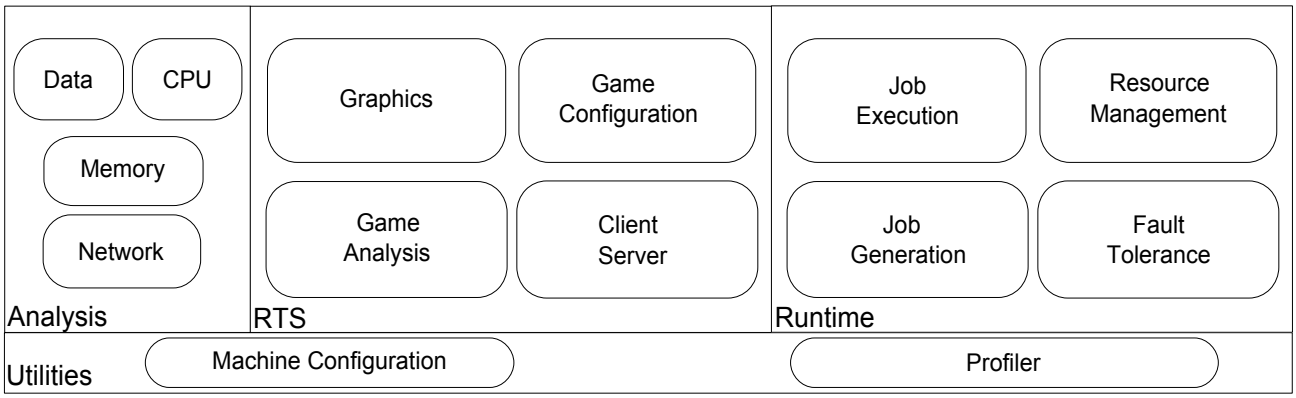

Figure 2: The RTSenv architecture.

We show in Section 4 that an environment satisfying these three requirements can be used in a variety of experimental scenarios, from performance evaluation to game design. We conjecture and leave for future work that such an environment can readily be used for unit testing and debugging (both important parts of game development), and for systems research, for example by comparing algorithms, methods, and techniques under a common and shareable experimental environment.

3.2 Design: Architectural Overview

RTSenv consists of four modules (see Figure 2): Utilities, Analysis, RTS, and Runtime. We describe them in the following, in turn.

The Utilities module packages the common (and trivial) environment operations. It consists of the *Machine Configuration* and the *Profiler* components. The former creates the configuration files for experiments on different platforms. The latter is used to measure resource consumption of CPU, memory, and network.

The Analysis module consists of the Data, CPU, Memory, Network components. The *Data* component collects the raw results of every experiment. The *CPU*, *Memory*, and *Network* components encapsulate analysis specific to corresponding data. New analysis components can be added to the Analysis module.

The RTS module is used to set up and monitor the gaming environment. The RTS module is comprised of four components, which we describe in turn. The *Graphics* component is responsible for configuring the game graphical mode; for example, to the "non-graphical client" mode for compute nodes that do not have graphical cards (i.e., the nodes of the DAS-4 clusters). The *Game Configuration* component automates the RTS-specific configuration of the experiments (see Section 3.4). The *Game Analysis* component collects and analyzes game-specific data for each game session; in particular, this component collects game session (replay) data and savegames. The *Client/Server* component configures the client/server gaming environment; this component controls to which server each running game client should join, and configures the join and exit conditions for each client.

The Runtime module is responsible for provisioning machines for the (multi-cluster) compute environment used by RTSenv. This module is also responsible for organizing, managing, and executing the experimental process, which is based on jobs. The Runtime module consists of four components. The *Job Execution* component is responsible for starting, executing, and stopping jobs on allocated compute nodes. This component receives at start-up various configuration parameters, such as the client and server IP addresses. After setting up the game through the RTS module, it invokes the Profiler module and starts the actual game sessions. The *Resource Management* component is responsible for organizing the execution of experiments; to manage the execution of experimental jobs, it operates an FCFS queue on top of a (multi-cluster) compute environment. This component is also responsible for acquiring and releasing resources, and for invoking the Job Execution component at each allocated machine. The *Job Generation* component parses the experiment description, then generates

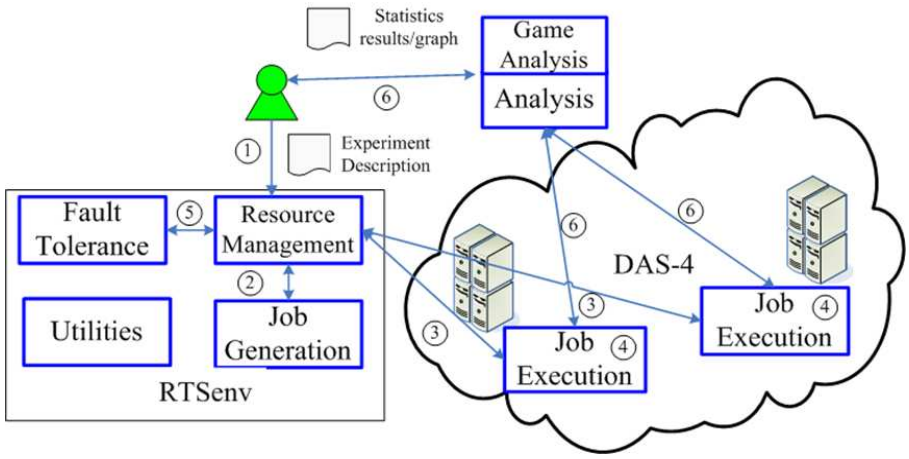

Figure 3: The RTSenv experimental procedure.

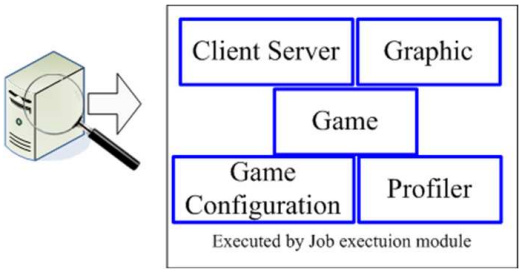

Figure 4: RTSenv job execution on a single machine.

configuration files and experiment jobs using the RTS module and the Machine Configuration component. Jobs can be single-machine clients, single-machine servers, or multiple-machine instances comprising one server and multiple clients. The *Fault Tolerance* component uses a reactive, retry-based fault tolerance mechanism in which failed jobs are re-submitted for execution until they succeed or the number of failures exceeds a user-defined threshold; failures are detected at the end of the job execution by checking the job output.

Experiment Control RTSenv allows users to run single and multi-machine experiments using multi-clusters. For gaining confidence in experiment results, users may specify the number of repetitions for single and multiple-instance experiments. RTSenv also allows its users to specify a maximum experiment runtime; it will stop overdue or user-selected experiments. RTSenv assumes that the environment in which it runs achieves synchronization between compute nodes; for example, time synchronization through the use of the NTP protocol, machine start synchronization through multi-machine allocation and co-allocation, etc.

Meeting the Requirements RTSenv meets the three requirements formulated in Section 3.1. For Requirement 1, RTSenv combines experiment control with replay-based testing abilities (see Section 3.4). For Requirement 2, RTSenv can evaluate and create simple statistical models of traditional performance evaluation and RTS-specific metrics (see Section 3.4). For Requirement 3, RTSenv can be used to test and compare the performance of different AIs under various game settings (map structure, etc.)

3.3 Design: Experiment Procedure

Figure 3 shows the experimental procedure of RTSenv , the process of running experiments has five main steps:

1. The user prepares an experiment description file and submits it to RTSenv.

2. The Resource Management component receives the request for experiments and invokes the Job Generation component to generate the experiment jobs.

3. The Resource Management component parses the job descriptions, allocates resources for the jobs, and prepares the necessary configuration files and binaries to be copied to the allocated compute nodes. After configuring the environment via the RTS and Utilities modules, the Job Execution component is invoked on the allocated compute nodes.

4. The Job Execution module copies and configures the game executable on each remote machine, starts the Profiler, then starts the game session. This procedure is shown in Figure 4.

5. Performance and game results are collected and stored after the gaming process finishes. The Fault Tolerance component checks the experiment output for errors, and reacts to failures.

6. If the experiment completes successfully, the results are analyzed. Analyzed results and statistics graphs are sent to user.

3.4 Design: RTS-Specific Features

Many RTS games, such as StarCraft II, provide a in-game recording mechanism (*replay*), which saves all the game commands issued by players. If the game world simulation is deterministic, replays can be used to reproduce the recorded game. Replays are commonly used by game operators to debug games (*replay-based testing*), and by players to share their gaming achievements and to learn from more skilled players. `RTSenv` supports *replay-based testing*.

`RTSenv` already supports many of the abstractions present in modern RTS games. First, `RTSenv` can control the virtual world's geography by changing the map size and structure (such as density of water or hills). Second, `RTSenv` can control the level of challenge proposed by the virtual world through parameters such as the number of resources present on the map, the amount of starting resources for each player, the types of resource collectors, etc. Third, `RTSenv` can control the maximum allowed player presence, for example by limiting the number of units each player can control. Fourth, our tool can control the in-game duration of a game session, for example one year.

Besides traditional performance metrics, `RTSenv` can measure various RTS-specific metrics, such as the scores registered by each player, the number of (profitable) units for each player, etc.

3.5 Implementation

Our reference implementation of `RTSenv`, which is coded in Python, is portable and extensible. We have tested our implementation on various Windows 7 and Ubuntu systems; on single desktops and on the DAS-4 [4] multi-cluster/grid computing environment. Our implementation is extensible in the sense that a user can add more experiments, more performance metrics, and more statistical tools by simply adding Python components to the source directory.

Adding in `RTSenv` support for `OpenTTD` required from us several modifications to the game packaging; no changes were made to the game core or design. First, we have modified the visualization of `OpenTTD` and add a non-graphical client mode into `OpenTTD`; this is needed for experimenting in cluster environments. Second, we have extended the original in-game limitation to the number of companies(one player per company) from under 16 to 250. Third, we have changed the operation of the `OpenTTD`'s AI module to allow AI-controlled players to operate remotely from the game server; this effectively enables multi-machine game sessions and various scalability tests. Fourth, we have made use of the original debug utilities of `OpenTTD` to implement the replay functionality. Fifth, we have used the load map function of `OpenTTD` to verify the correctness of completed games and to obtain game-specific scores and data.

4 Experimental Results

In this section we present the experiments we have performed using a reference implementation of `RTSenv` on the multi-cluster system DAS-4 [4]. Our experiments show evidence that `RTSenv` can be used in a variety of experimental scenarios; a comprehensive evaluation of `OpenTTD` or building a performance model for RTS games are both outside the scope of this paper, albeit being made possible by `RTSenv`.

Overall, we have conducted using `RTSenv` over 20,000 game sessions, which amount to over 30,000 in-game years and over 7,000 real operational hours. We find that some common game models, such as the linear dependency between the amount of network traffic and the number of players, do not hold for the case of `OpenTTD` and thus warrant a future, in-depth study. We also show evidence that the scalability of a game with the number of players is not limited only in terms of system performance, but also in terms of game experience. Last, we show how `RTSenv` can be used to compare and even indicate improvements for `OpenTTD` AI types.

4.1 Experimental Setup

`RTSenv` records the performance of CPU and memory load twice per seconds, and uses `wireshark` to record the network traffic generated by `OpenTTD`. Except for the experiments done for measuring network traffic, each experiment is repeated at least 30 times.

Environment We ran our multi-machine experiments using the DAS-4 six-cluster, wide-area system. A typical compute node of DAS-4 has a dual quad-core 2.4GHz Intel E5620 CPU and 24GB memory; the intra-cluster network is 1GBit Ethernet. DAS-4 uses the Sun Grid Engine batch queueing system to manage its resources; compute nodes are synchronized through the NTP protocol. The job execution model of DAS-4 assumes exclusive resource use. To avoid interference effects, each game instance is allocated a single node of DAS-4, when possible. We ran single-machine experiments on a Dell workstation T3500, with Intel Xeon W3680 six-core 3.3GHz CPU and 4GB memory.

Game Configuration All the experiments use Artificial Intelligence (AI)-controlled players; the AI algorithms (the AIs) are real, high quality, open-source, and OpenTTD community-provided. Unless otherwise specified, the game sessions are configured as follows. Each game session's configuration is set to the default configuration of `OpenTTD`, with the following exceptions. The starting in-game time is set to the first day of 1998 and the initial bank loan is set to be high. This setup, which is commonly used in `OpenTTD` AI competitions[3], allows more initial investment from players but leads to higher than average resource consumption. Each game starts with 16 random AI players. Each game instance is scheduled to run one in-game year by default, which leads to a *nominal execution time* of about 825 seconds on a typical DAS-4 node. Longer execution times, which result from server overloads, are noticeable by players only if they exceed 105% of the nominal execution times (the *playable execution time range*). Although we have scheduled game instances to run multiple in-game years, most of the results in this section is obtained from one in-game year game instance.

4.2 Performance Evaluation Results

We focus in this section on two aspects of the game that can be quantified through performance evaluations, the system and the user experience. We conduct three experiments for the former, and one for the latter; the results are described in the following, in turn.

Network Measurements, System (Figure 5): We conduct multi-player game experiments to assess network consumption. Each AI player connects to the server remotely, first to download the game map, then to issue commands and to receive small updates (for example, each command issued by the each other player)—this scenario emulates real gamers playing on a commercial game server. The server is executed on desktop computer while the clients are executed on cluster's computer node. Figure 5 shows the latter type of network traffic, as observed for this scenario when the number of remote players is varied from 1 to 25. The statistical properties of the network consumption (Figure 5, left) indicate that the median network traffic size increases

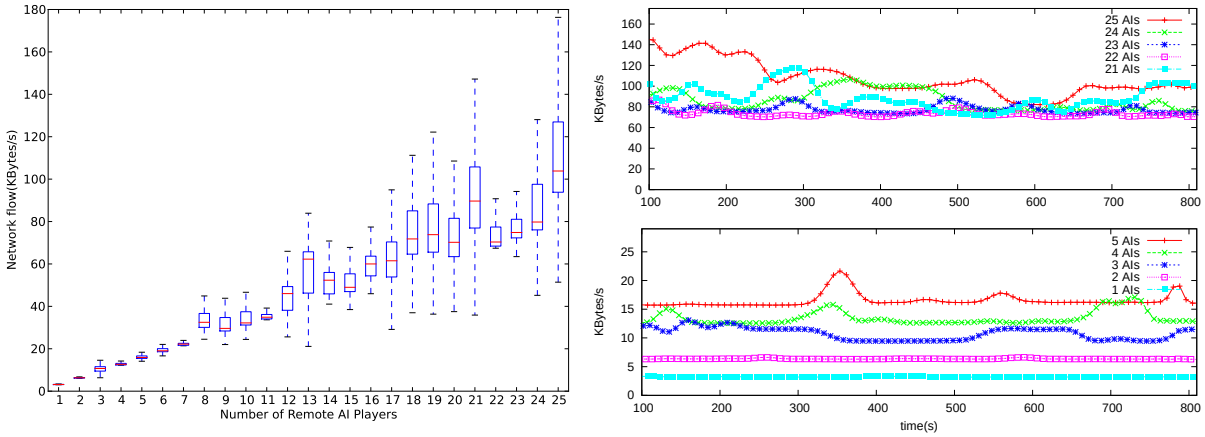

Figure 5: Network traffic of the `OpenTTD` server for various remote player counts: (*left*) basic statistics, depicted as box-and-whiskers plots; (*right*) consumption over time.

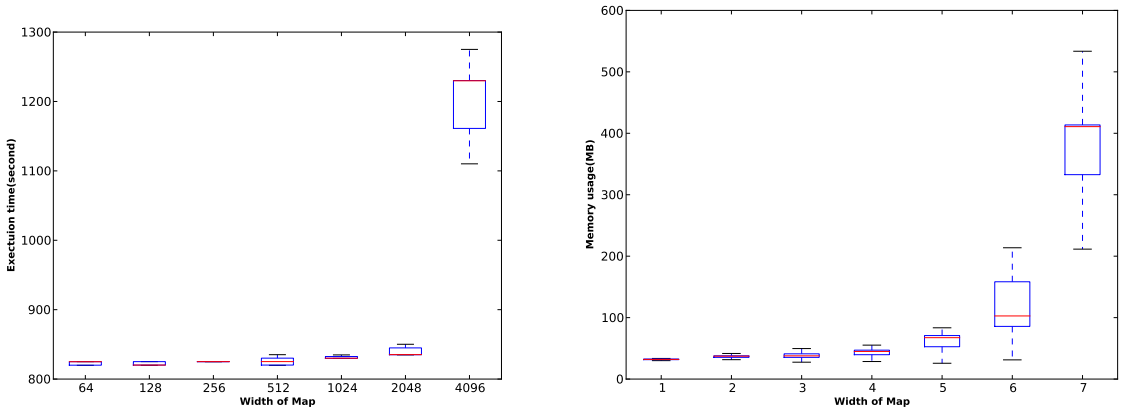

Figure 6: Basic statistics, depicted as box-and-whisker plots, of system performance metrics for the `OpenTTD` server for various sizes of square game map. (*left*) game execution time; (*right*) memory consumption.

nearly linear to the number of players (the *linear model* commonly assumed for games [60, 68]), but the relatively large traffic size ranges may cause problems for players with limited bandwidth. We observe the traffic over time in Figure 5, right–the network consumption is stable when the number of players is low (up to 5 players), but increases and becomes more variable as the number of players increases. Noticeably, minute-long periods of much higher traffic than expected from the linear model occur even for 5 players (see corresponding curve between 300 and 400 seconds). Periods of such length cannot be hidden from even beginning players. *Main Finding: The linear model does not hold for RTS games in general* (we have just shown an example where it does not hold)*, and periods of high variability become common as the number of players increases.*

Impact of map, System (Figure 6): We assess the impact of in-game map size by increasing the size of map from 64 × 64 to 4096 × 4096 in-game tiles. The basic `OpenTTD` engine logic is a infinite-loop over each tiles and units, the game engine simulates some events on each tiles every loop; more CPU operations are needed for larger game maps. As it is expected, when the system is under loaded, the execution time and memory consumption increase a bit with the increase of map size. The `OpenTTD` limited maximum size of map to be 2048 × 2048 tiles, in order to evaluate the performance when the `OpenTTD` is overloaed by large size of map; We exceed this limitation and set the size of map to be 4096 × 4096. As it is expected the system is overloaded, the

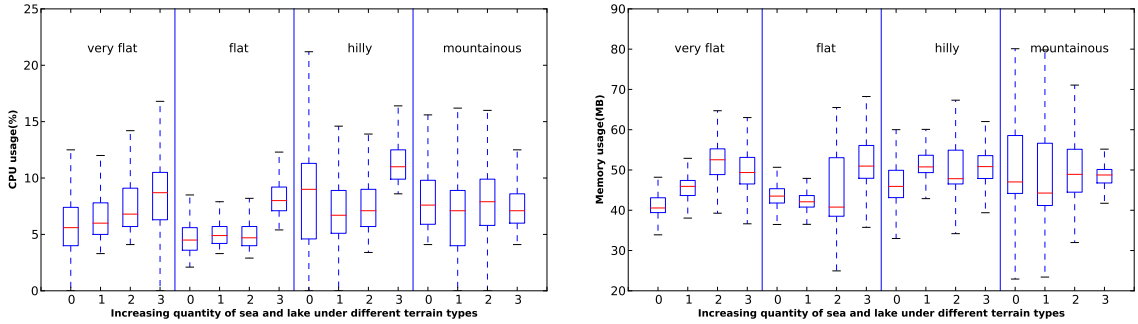

Figure 7: CPU usage and memory consumption under different types of terrain.

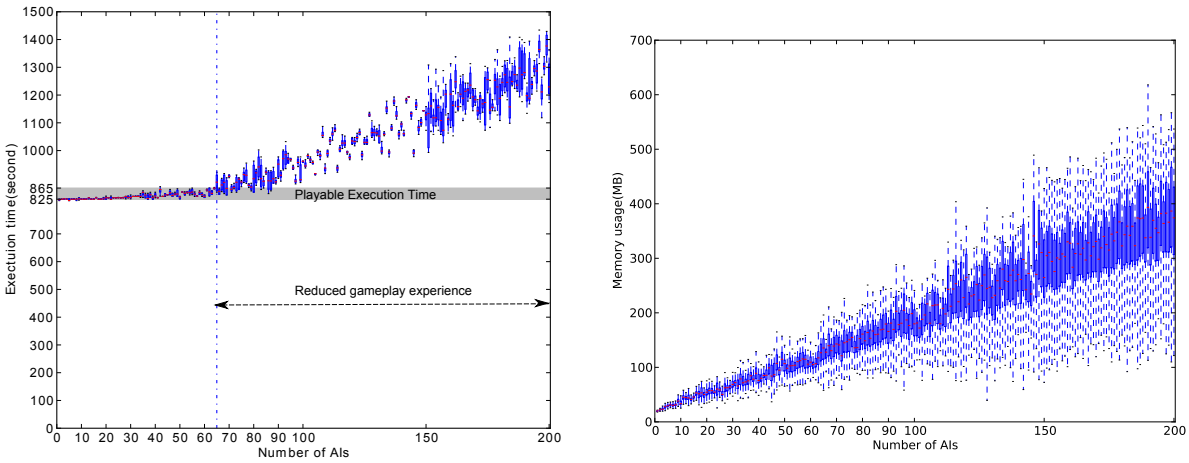

Figure 8: Basic statistics, depicted as box-and-whisker plots, of system performance metrics for the `OpenTTD` server for various local player counts. (*left*) game execution time; (*right*) memory consumption.

execution time and memory consumption increase a lot. Such a map configuration is not playable. (Figure 7): Further we assess the impact of map structure by changing the map terrain from very flat to mountainous and in each terrain we increase the sea level from 0 to 3. Figure 7 shows the box-plot of cpu usage of `OpenTTD` sessions with different quantity of sea and different terrain. When the terrain is less mountainous, the CPU usage increases with the sea level; but when the terrain is mountains, sea level doesnt take a significant affect. This is cause by most AI uses A* algorithm to build path between cities and resources, when the sea level increase, it needs more calculation to find a proper path to build bridge; when the terrain is mountainous, the map is already divided into many patches, so sea level doesnt affect system performance that much.

Scalability, System (Figure 8): We assess the scalability of the game server by increasing the number of players from 1 to 200, that is, to an order of magnitude more players than in today's commercial games. All the AIs are running on the game server, thus consuming CPU and memory resources—this is the typical mode of operation for current commercial games [60, 48, 43]. The results are depicted in Figure 8. When the number of players is below 30, the game execution time is stable and very close to the nominal execution time (see Section 4.1). The playable execution time range is exceeded at about 60 players; afterwards, the game execution time increases quickly in both median value and range, which results in reduced gameplay experience.

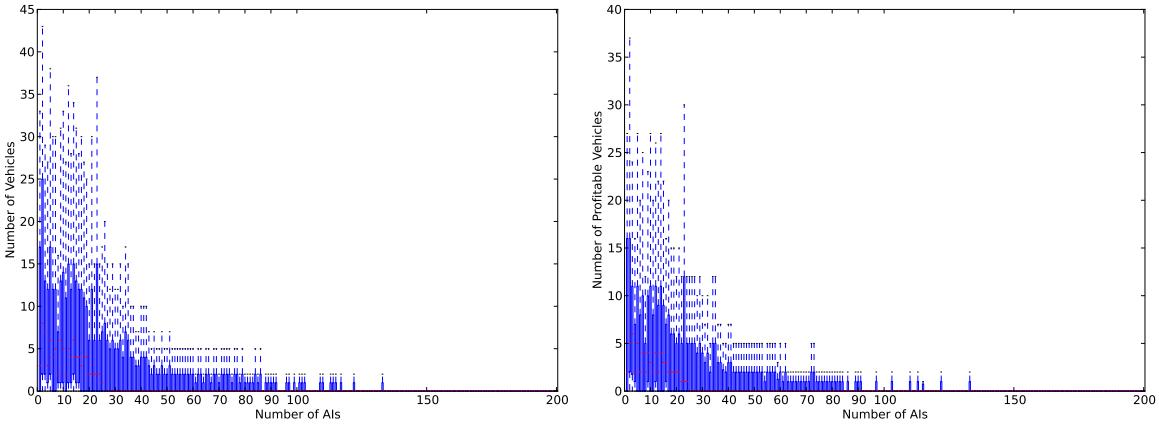

Figure 9: The performance of AI-controlled players in multi-player game sessions.

The game execution time growth is mainly due to CPU consumption; as shown in Figure 8, right, the maximum memory consumption is below 700MB (`OpenTTD` is not memory-bound). *Main Finding: `RTSenv` can be used to analyze the scalability of RTS games.*

Scalability, Gameplay Experience (Figure 9): We further analyze the results of the previous experiment from a gameplay experience perspective. Since in our tests we cannot ensure the presence and follow a group of human testers that adhere to the standards of gameplay experience evaluation [53, 37], we use instead a proxy metric for gameplay experience, the median number of profitable vehicles. For scalable gameplay experience, the median number of (profitable) vehicles should be scalable, that is, it should at least not decrease when the number of players increases. When the number of vehicles is not scalable, due to the low number of vehicles regardless of the player ability the players do not feel a correlation between their skill and experience a reduced feeling of mastery. When this scalability metric reaches zero, the median player does not have even a single profitable vehicle, and thus no possibility to earn in-game money. Similarly, when the maximum number of vehicles observed for a setting is low (for example, below 10), the number of vehicles to control may not be challenging enough for players to enjoy the game. A comprehensive user study focusing on the relationship between the number of vehicles and the players' enjoyment falls outside the scope of this paper. Figure 9 shows the statistical properties of the number of (profitable) vehicles, as a function of number of competing players. As the number of players increases, it becomes increasingly difficult for players to have profitable vehicles. Over 25 players, the median value of the number of profitable vehicles is 0 and the maximum value drops quickly below 5. The observed gameplay scalability limit (25 players) is below the system scalability limit (60 players for our platform) and does not depend on the platform; thus, *Main Finding: for (RTS) games, system scalability needs to be analyzed and improved in conjunction with gameplay experience scalability.*

4.3 Comparing Game Design Choices

AI in-game performance vs real-world resource consumption (Figure 10) An important RTS game design choice is the AI—good and/or human-like AIs make RTS games more challenging and interesting. While the ability of an AI to perform in-game depends on the quality of the Artificial Intelligence techniques it employs, the game designer has another important aspect to consider: the real-world resource consumption. RTS games have a limited amount of resources to offer to the real-time decision-making process of an AI, for example 15% of the CPU cycles and 5% of the allocated memory. We compare in this experiment the performance and the resource consumption of 17 real, open-source, community-provided `OpenTTD` AIs. Figure 10 shows the basic statistical properties of the total score (in-game achievement) and CPU usage (real-world resource consumption)

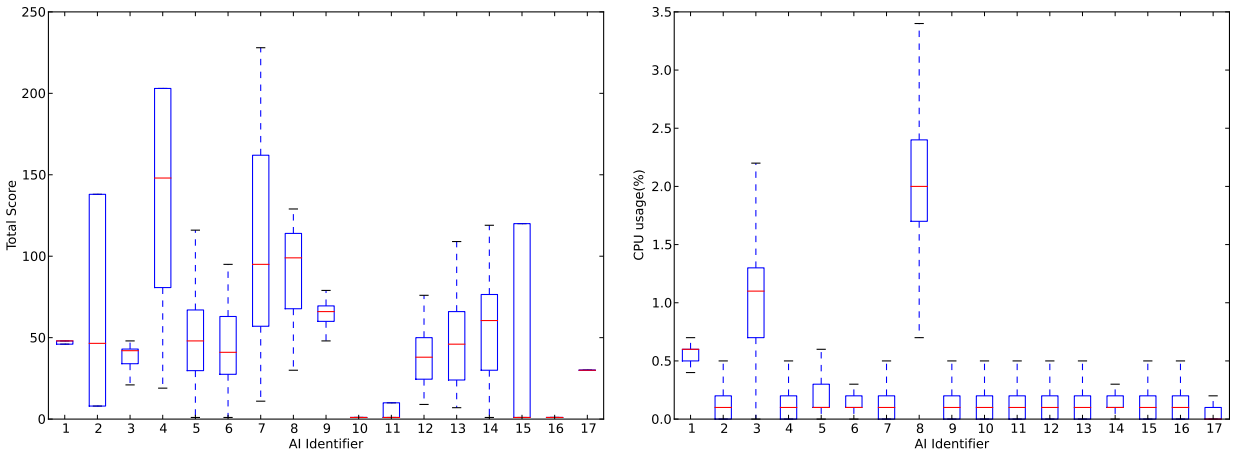

Figure 10: AI comparison experiments: (*left*) in-game score; (*right*) CPU usage.

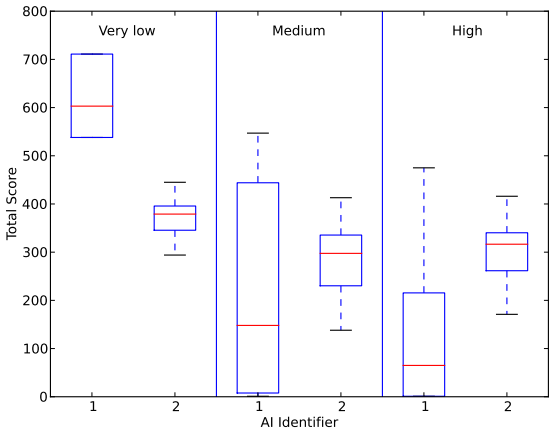

Figure 11: In-game performance of two AIs with increasing water presence on flat terrain.

achieved by each AI. *Main Finding: `RTSenv` can support the game design choice of AIs.*

Selecting AIs in different map settings (Figure 11): An AI technique that achieves good results in one RTS setting may work less effectively in another. We consider a five-year game in which two AIs compete on various map structures—we vary the density of bodies of water. AI_1 is good at building roads between industries and cites and transporting goods in land, while AI_2 put more focus on aeroplane. In `OpenTTD`, land transportation is more profitable than air transportation. As it is shown in the figure, the performance of AI_1 decrease with increasing level of water, that is because it becomes more difficult for AI_1 to build a profitable road when the land is divided into many patches by water. The performance of AI_2 is stable because its commercial strategy does not heavily affected by water. This result further emphasizes the need for an automated AI comparison tool such as `RTSenv` as support for AI selection. *Main Finding: `RTSenv` can support selecting AIs in different types of maps.*

Dissecting AI operation (Figure 12): In `OpenTTD`, AI players have a limited amount of operations (10,000) they can perform during an in-game day. An `OpenTTD` tick ends for a player either because 10,000 operations have been executed, or because the AI has issued a command that changes the game world (such as building a station). To address this situation, the OpenTTD community has created "rules of thumb" for AI development;

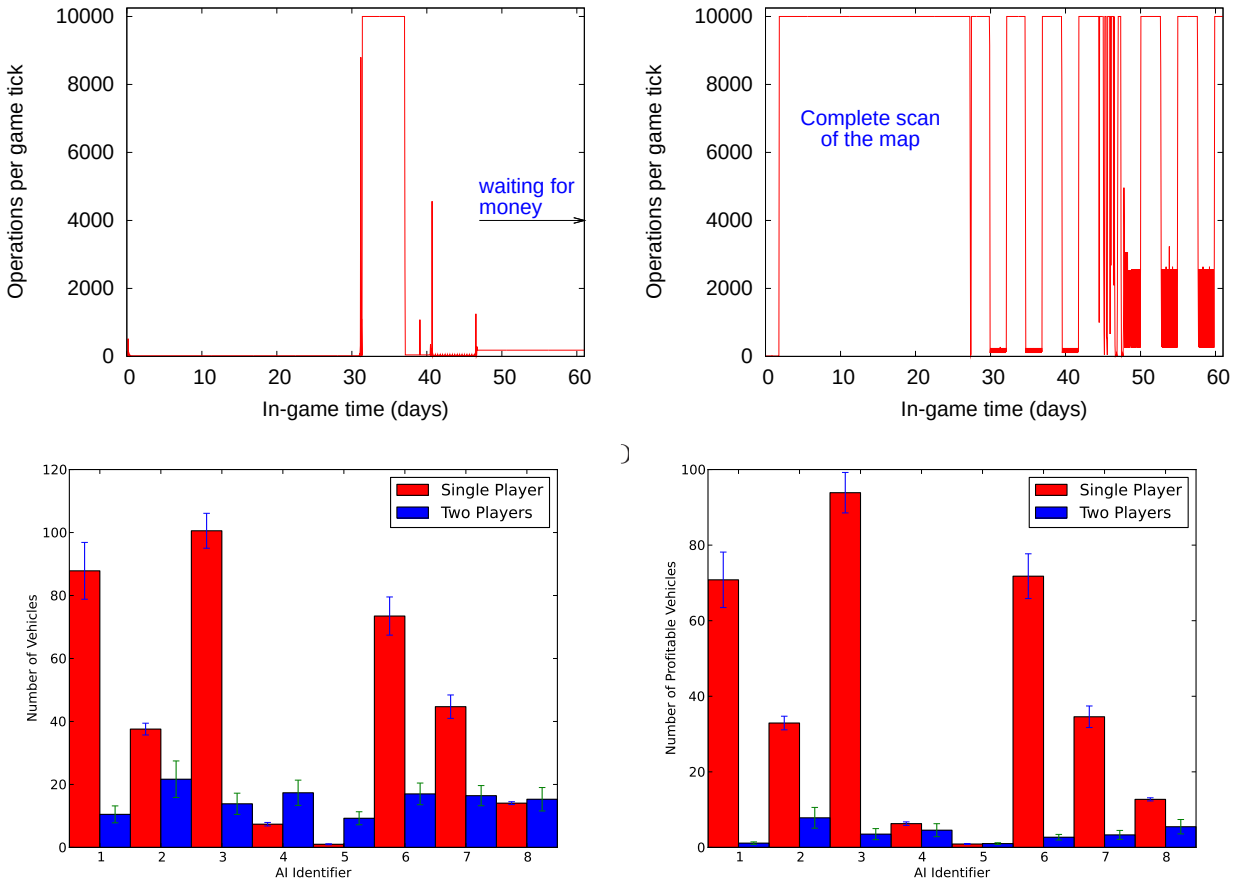

Figure 13: AI performance in single-player and two-player games. Vehicles (*left*) Profitable Vehicles (*right*).

for example, an AI should not take too much time (a whole in-game year) to create its first profitable route. In this experiment we dissect the operation of two `OpenTTD` AIs. The AI "OtviAI"[1] (in-game performance similar to that of a good human player) builds its own route and then waits for enough money to be available before building the next route. In contrast, the AI "Rondje om de Kerk"[2] (winner of the 2008 TJIP competition [3]) tries to make use of the roads build by other players, and alternates between scanning for new routes and building/maintaining vehicles. Figure 12 compares the use of allocated operations by these two `OpenTTD` AIs. OtviAI has several idle periods; the period starting at roughly in-game day 47 corresponds to waiting for money to build a new route. The contrasting behaviour of Rondje is depicted in Figure 12 (right). Both OtviAI and Rondje have unused allocated operations. This could be exploited better by Rondje, because it fits better with its design philosophy: many activities of few operations. *Main Finding: `RTSenv` enables AI developers to make better use of the available resources and evaluate their design decisions.*

Impact of competition(Figure 13) We evaluate the effect of competition in `OpenTTD`. In this experiment, we schedule different types of AI in a single-player game and in a two-player game with the introduction of a strong competitor. Figure 13 shows the in-game performance of players in single/two-player game. It is interesting to see that with the introduction of strong competition, most of AIs perform significantly worse because their profitable route may be occupied/blocked by opponent. The phenomenon indicates that game

engine should employ some form of free-riding prevention technique to provide a fair game environment.

5 Related Work

We contrast in this section RTSenv with previous experimental RTS/game studies, experimental RTS/game platforms, RTS/game simulators, and general large-scale experimental platforms and tools. In comparison with this body of related work, RTSenv has different scope (system and game-specific experiments), focus (RTS games), and application (a *real* game, OpenTTD, and a variety of high quality artificial players.)

Few *experimental RTS game studies* exist. Closest to the experimental part of our work, a study on the prototype RTS game Rokkatan [46] tests scalability with about 400 clients, but the clients are controlled by prototype instead of production, that is, complex and CPU-intensive, AIs. Several studies [61, 12, 55, 37, 69] assess with the help of real users the Quality of Service and of Experience for different game genres; our work complements this body of work.

Experimental RTS platforms have been used as test-beds for AI research [11, 35]. In these environments, scientists can study and design various techniques and algorithms for tactical and strategic decision-making [67]. OpenNero[35] is a game platform for AI research and Education that provides packages to build games for AI research. In contrast, RTSenv focuses on game scalability and system bottlenecks; our results indicate that both in-game performance *and* real-world resource consumption should affect AI selection.

Experimental game platforms exist [64, 65, 39], but they do not have RTS-specific support and the studied resource is mainly the network. Mammoth [64, 36] is a massively multiplayer game research framework that has be used to compare complex game state management techniques. Several environments [65, 39] focus on experimental studies for P2P gaming, and in particular on the effects of the network on performance. Our results indicate that CPU consumption, due in RTS games to world simulation and artificial player control, is an important source of performance concerns.

Simulation-based studies of online games Many simulators have been developed for the study of games, in particular by the community studying peer-to-peer gaming, but few of these simulators are shared or even validated with real-world games. All these simulators represent a trade-off between the representativity and accuracy of the results, and the difficulty of development and use. Simulators developed using standardized, industrial-grade simulation platforms such as HLA [28] and the older DIS [27] and FIPA [19], have been used to investigate online game-like scenarios [44] and are used for army training. A simulator co-developed by an author of this work was used [47, 32, 49] to study how data centers and cloud computing can support online gaming. A variety of peer-to-peer online game simulators exist; the studies of the VON [25], the Mercury [8], the pSense [59], and the Donnybrook [9] systems each developed a new simulator. One of the earliest simulators that allows the comparison of different gaming architectures is NGS [68], which focuses on the high-level definition of games, and models peer-to-peer, centralized, and hybrid architectures.

General large-scale experimental platforms and tools have been proposed [13, 31, 33], most recently in the context of grid and cloud computing. RTSenv complements these tools with RTS-specific functionality, from configuration to results analysis, and focuses on interactive applications that are not part of common grid computing workloads [30].

6 Conclusion and Future Work

The increasing popularity of RTS games fosters demand for new designs and technical solutions, which emphasizes the need for experimental environments. In this work, we have introduced RTSenv, an experimental environment for RTS games. Our environment can control and measure many RTS-specific aspects, and enables a variety of experimental scenarios–from performance evaluations to taking game design decisions. RTSenv can operate in several types of computing environments, from single desktop computers to wide-area multi-clusters, and leverages reactive fault tolerance techniques to perform robust, multi-machine, multi-instance RTS game

experiments. We have used RTSenv in DAS-4, a real multi-cluster environment, to conduct an extensive study of the popular RTS game OpenTTD. The experimental results show that RTSenv can test major RTS-specific features and can help comparing different game design choices. They also show that RTSenv can lead to new findings, such as:

1. Common assumptions made by researchers about game workloads, including the linear dependence between network traffic and the number of players, do not hold for RTS games in general;
2. The scalability of an RTS game should be evaluated not only from the performance but also from the gameplay experience perspective;
3. Game design choices such as artificial intelligence selection require a careful evaluation of both in-game capability and real-world resource consumption.

We conclude that RTSenv is a useful tool for both RTS game researchers and designers. We are currently working on operating RTSenv in cloud computing environments, and on integrating a network emulator into the environment.

7 Acknowledgements

The first author is supported by a joint China Scholarship Council and TU Delft grant, and by the National Basic Research Program of China (973) under Grant No.2011CB302603. The authors thank Remko Bijker, OpenTTD Project Lead.

References

[1] OtviAI, 2008. http://www.tt-forums.net/viewtopic.php?f=65&t=39707. 19
[2] Rondje om de Kerk, 2008. http://www.tt-forums.net/viewtopic.php?f=65&t=39756. 19
[3] TJIP OpenTTD Challenge, 2008. http://www.tjip.com/tjip-challenge.html. 14, 19
[4] ASCI. Distributed ASCI Supercomputer-Version 4, 2010. http://www.cs.vu.nl/das4/. 5, 13, 14
[5] E. Bethke. *Game development and production*. Wordware Publishing, Inc., New York, NY, 2003. 5, 6
[6] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In *SIGCOMM*, pages 389–400, 2008. 4, 5, 8
[7] A. Bharambe, J. Pang, and S. Seshan. Colyseus: a distributed architecture for online multiplayer games. In *NSDI*, pages 12–12, 2006. 4
[8] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. In *SIGCOMM*, pages 353–366, 2004. 20
[9] A. R. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: enabling large-scale, high-speed, peer-to-peer games. In *SIGCOMM*, pages 389–400, 2008. 20
[10] Blizzard Inc. StarCraft II: Wings of Liberty Becomes Biggest PC Game Launch of the Year, 2010. http://us.blizzard.com/en-us/company/press/pressreleases.html?100803. 4
[11] M. Buro. RTS Games as Test-Bed for Real-Time AI Research. *JCIS*, pages 481–484, 2003. 5, 20
[12] K.-T. Chen, P. Huang, and C.-L. Lei. How sensitive are online gamers to network quality? *Commun. ACM*, 49(11):34–38, 2006. 20
[13] B. Claudel, G. Huard, and O. Richard. Taktuk, adaptive deployment of remote executions. In *HPDC*, pages 91–100, 2009. 20
[14] D. S. Cohen and S. A. Bustamante II. *Producing Games: From Business and Budgets to Creativity and Design*. Focal Press, New York, NY, 2009. 5, 7
[15] M. Csikszentmihalyi. *Flow: The Psychology of Optimal Experience*. Harper Perennial, New York, NY, March 1991. 10
[16] Entertainment Software Association. 2011 essential facts about the computer and video game industry. Technical Report. [Online] Available: http://www.theesa.com/facts/pdfs/ESA_EF_2011.pdf, 2011. 8
[17] Entertainment Software Association. Video games in the 21st century: The 2010 report. Technical Report. [Online] Available: http://www.theesa.com/facts/pdfs/VideoGames21stCentury_2010.pdf, 2010. 4
[18] T. Fields. *Distributed Game Development: Harnessing Global Talent to Create Winning Games*. Focal Press, New York, NY, 2010. 5, 6, 7
[19] FIPA. FIPA Agent management specification, 2000. http://www.fipa.org/spect/fipa00023/. 20
[20] T. Fritsch, B. Voigt, and J. H. Schiller. Distribution of online hardcore player behavior: (how hardcore are you?). In *NETGAMES*, page 16, 2006. 5
[21] F. Glinka, A. Ploss, J. Müller-lden, and S. Gorlatch. RTF: a real-time framework for developing scalable multiplayer online games. In *NetGames*, pages 81–86, 2007. 4
[22] J. Grudin. The human-computer interaction handbook. chapter Introduction, pages 883–906. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003. 4
[23] T. Henderson and S. Bhatti. Networked games: a qos-sensitive application for qos-insensitive users? In *Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS: What have we learned, why do we care?*, RIPQoS '03, pages 141–147, New York, NY, USA, 2003. ACM. 10
[24] M. Hitchens. Time and computer games or "no, that's not what happened". In *Proceedings of the 3rd Australasian conference on Interactive entertainment*, IE '06, pages 44–51, Murdoch University, Australia, Australia, 2006. Murdoch University. 8
[25] S.-Y. Hu, J.-F. Chen, and T.-H. Chen. VON: a scalable peer-to-peer network for virtual environments. *IEEE Network*, 20(4):22–31, 2006. 4, 5, 8, 20
[26] H. Huang. *Logistics Operations Management Simulation Practice Guide (In Chinese)*. Higher Education Press of China, ISBN 978-7-04-028496-6, 2010. 9
[27] IEEE. IEEE 1278-1993 Standard for Distributed Interactive Simulation, 1993. http://standards.ieee.org/downloads/1278/. 20
[28] IEEE. IEEE 1516-2010 Standard for Modeling and Simulation High Level Architecture, 2010. http://standards.ieee.org/downloads/1516/. 20
[29] Interactive Digital Software Association. State of the industry report (2000-2001). Technical Report., 2000. 4

[30] A. Iosup and D. Epema. Grid computing workloads. *IEEE Internet Computing*, 15:19–26, March 2011. 20
[31] A. Iosup and D. H. J. Epema. GrenchMark: A framework for analyzing, testing, and comparing grids. In *CCGrid*, pages 313–320. IEEE, 2006. 20
[32] A. Iosup, V. Nae, and R. Prodan. The impact of virtualisation on the performance and operational costs of massively multiplayer online games. *IJAMC*, 4(4):364–386, 2010. 20
[33] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *IEEE Trans. Parallel Distrib. Syst.*, 22:931–945, June 2011. 20
[34] D. Irish. *The Game Producer's Handbook*. Course Technology PTR, New York, NY, 2005. 5, 6, 7
[35] I. Karpov, J. Sheblak, and R. Miikkulainen. Opennero: A game platform for ai research and education. In *AIIDE*, page demo, 2008. 20
[36] J. Kienzle, C. Verbrugge, B. Kemme, A. Denault, and M. Hawker. Mammoth: a massively multiplayer game research framework. In *FDG*, pages 308–315, 2009. 20
[37] J. H. Kim, D. V. Gunn, E. Schuh, B. Phillips, R. J. Pagulayan, and D. R. Wixon. Tracking real-time user experience (TRUE): a comprehensive instrumentation solution for complex systems. In *CHI*, pages 443–452, 2008. 5, 17, 20
[38] D. Kushner. *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture*. Random House, New York, NY, 2004. 6
[39] M. Lehn, T. Triebel, C. Leng, A. Buchmann, and W. Effelsberg. Performance evaluation of peer-to-peer gaming overlays. In *P2P*, pages 1–2, August 2010. 5, 20
[40] C. A. Lindley. The semiotics of time structure in ludic space as a foundation for analysis and design. *Game Studies*, 5(1), 2005. 8
[41] G. McAllister and G. R. White. Evaluating user experience in games. chapter Video Game Development and User Experience, pages 107–128. Springer Verlag, London, UK, 2010. 4, 5, 6, 7
[42] B. Medler, M. John, and J. Lane. Data cracker: developing a visual game analytic tool for analyzing online gameplay. In *CHI*, pages 2365–2374, 2011. 4
[43] J. L. Miller and J. Crowcroft. The near-term feasibility of P2P MMOGs. In *NETGAMES*, pages 12–17, 2010. 5, 16
[44] P. Morillo, S. Rueda, J. M. Orduña, and J. Duato. Ensuring the performance and scalability of peer-to-peer distributed virtual environments. *Future Generation Comp. Syst.*, 26(7):905–915, 2010. 20
[45] Movie Picture Association of America. US entertainment industry: 2006 market statistics. Annual Report. [Online] Available: http://www.mpaa.org/, Mar 2007. 4
[46] J. Müller and S. Gorlatch. Rokkatan: scaling an rts game design to the massively multiplayer realm. *Computers in Entertainment*, 4(3), 2006. 4, 20
[47] V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. H. J. Epema, and T. Fahringer. Efficient management of data center resources for massively multiplayer online games. In *SC*, page 10, 2008. 20
[48] V. Nae, A. Iosup, and R. Prodan. Dynamic resource provisioning in massively multiplayer online games. *TPDS*, (3):380–395, 2010. 4, 16
[49] V. Nae, R. Prodan, A. Iosup, and T. Fahringer. A new business model for massively multiplayer online games. In *ICPE*, pages 271–282, 2011. 20
[50] Nice Technology. Picaroon, massively multiplayer strategy game, 2011. [Online] Available: http://www.picaroonthegame.com/, Accessed Jan 2011. 4
[51] D. A. Norman. *The Design of Everyday Things*. Basic Books, Sept. 2002. 4
[52] OpenTTD team. OpenTTD, 2010. http://www.openttd.org. 9
[53] R. J. Pagulayan, K. Keeker, D. Wixon, R. L. Romero, and T. Fuller. The human-computer interaction handbook. chapter User-centered design in games, pages 883–906. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003. 4, 5, 8, 17
[54] M. T. Paul Bettner. 1500 archers on a 28.8: Network programming in Age of Empires and beyond, 2001. http://www.gamasutra.com/view/feature/3094/1500_archers_on_a_288_network_.php. 9
[55] D. Pittman and C. Gauthier. A measurement study of virtual populations in massively multiplayer online games. In *NETGAMES*, pages 25–30. ACM, 2007. 20
[56] Recording Industry Association of America. 2006 year-end shipment statistics. Annual Report. [Online] Available: http://www.riaa.com/, Apr 2007. 4
[57] S. Rogers. *Level Up!: The Guide to Great Video Game Design*. Wiley, New York, NY, 2010. 8
[58] A. Rollings and E. Adams. *Fundamentals of Game Design*. Prentice Hall, 2006. 10

[59] A. Schmieg, M. Stieler, S. Jeckel, P. Kabus, B. Kemme, and A. P. Buchmann. psense - maintaining a dynamic localized peer-to-peer structure for position based multicast in games. In *Peer-to-Peer Computing*, pages 247–256, 2008. 20

[60] A. Shaikh, S. Sahu, M.-C. Rosu, M. Shea, and D. Saha. On demand platform for online games. *IBM Systems Journal*, 45(1):7–20, 2006. 15, 16

[61] N. Sheldon, E. Girard, S. Borg, M. Claypool, and E. Agu. The effect of latency on user performance in Warcraft III. In *NETGAMES*, pages 3–14, 2003. 4, 8, 20

[62] D. Spohn. Top 15 Multiplayer RTS Games for the PC, 2010. http://internetgames.about.com/od/strategygames/tp/tprts.01.htm. 4

[63] The Entertainment Software Association. Essential facts about the computer and video game industry: Sales, demographics, and usage data. Technical Report. [Online] Available: http://www.theesa.com, Nov 2008. 4

[64] The Mammoth team. Mammoth, a multiplayer game research framework, 2007. http://mammoth.cs.mcgill.ca/. 20

[65] S. Tolic and H. Hlavacs. A testbed for P2P gaming using time warp. In *Transactions on Edutainment II*, volume 5660 of *LNCS*, pages 33–47. 2009. 5, 20

[66] A. Tychsen and M. Hitchens. Ghost worlds - time and consequence in mmorpgs. In *TIDSE*, pages 300–311, 2006. 8

[67] R. van der Krogt, N. Roos, M. de Weerdt, and C. Witteveen. Multiagent planning through plan repair. In *AAMAS*, pages 1337–1338, 2005. 20

[68] S. D. Webb, W. Lau, and S. Soh. Ngs: an application layer network game simulator. In *Proceedings of the 3rd Australasian conference on Interactive entertainment*, IE '06, pages 15–22, Murdoch University, Australia, Australia, 2006. Murdoch University. 15, 20

[69] D. R. Wixon and R. J. Pagulayan. That's entertainmnt - Halo 3: the theory and practice of a research-design partnership. *Interactions*, 15(1):52–55, 2008. 5, 20