1. Introduction

1.1 Problem Definition

Computer game based learning - SimComp

Computer game based learning is a new and exciting area. IDI has through the project Age Of Computers (AoC) learned that the use of games can be very motivating for students in a learning context. The goal of this project is to discuss the possibilities of making a computer game for assisting the teaching in an upper level computer architecture course. The task is to discuss what kind of computer games that may be relevant - and choose one of them for further investigation by developing a prototype.

The game might be inspired by the famous computer game SimCity or other well-known game paradigms. The name SimComp is short for Sim-Computation, and a long-term goal (beyond this project) is to link the game to BSPlab. BSPlab is a virtual laboratory (simulation system) for experimenting with parallel architectures, and the prototype should be designed such that it will be possible in later projects to base the computer processing in the game on simulations executed in BSPlab.

The project is suited for 1-2 students.

1.2 Motivation for the Project

Computer simulations have been used for decades in many different fields to experiment and study the effects and changes in complex systems, without using the real physical objects. A scientist simulates to learn or discover something for his research, but computer game designers found that playing with a simulation can also be entertaining. A game can recreate and simulate a more or less realistic part of the real world and let the player manipulate it. The purpose of the simulator game is primarily entertainment, but if the simulation is well made the player might also learn something.

Computers and information technology are a part of every student's life, both at the university and at home. We communicate, do exercises and read lecture notes via computers and listen to music, play games and watch movies on the computer at home. It is in the context of endless possibilities the computers offer us, Lasse Natvig and the people at IDI started using computer game based learning. Their first project was Age of Computers [Djupdal, Natvig 2004], which taught computer fundamentals through a game in a historic setting. The next is a game based around the simulation of programs running on different types of multiprocessor computers, using the combination of games and simulations to make learning more entertaining and hopefully more motivating.

1.3 Goals

This report is a first step in developing a simulation game for use in teaching computer architecture and has the following goals.

1. Explore different possibilities for making a simulation game in the area of computer architecture.
2. Develop a prototype of one of the possibilities.
3. Create a list of suggestions for further work, based on the experiences from developing the prototype.

1.4 Overview of the Report

Developing a computer architecture simulation game requires knowledge from several areas, shown in figure 1. With so many different areas it is not possible to cover all them in full and I will throughout the report narrow the focus down to the most relevant parts of each area.

Chapter 2 gives an overview of some of the background information for each area. Chapter 3 and 4 covers the development of conceptual ideas for a game, corresponding to goal 1. Chapter 5 and 6 covers the design and implementation of the prototype, corresponding to goal 2. Chapter 7 and 8 covers experiences from developing a prototype and suggestions for further work, corresponding to goal 3.

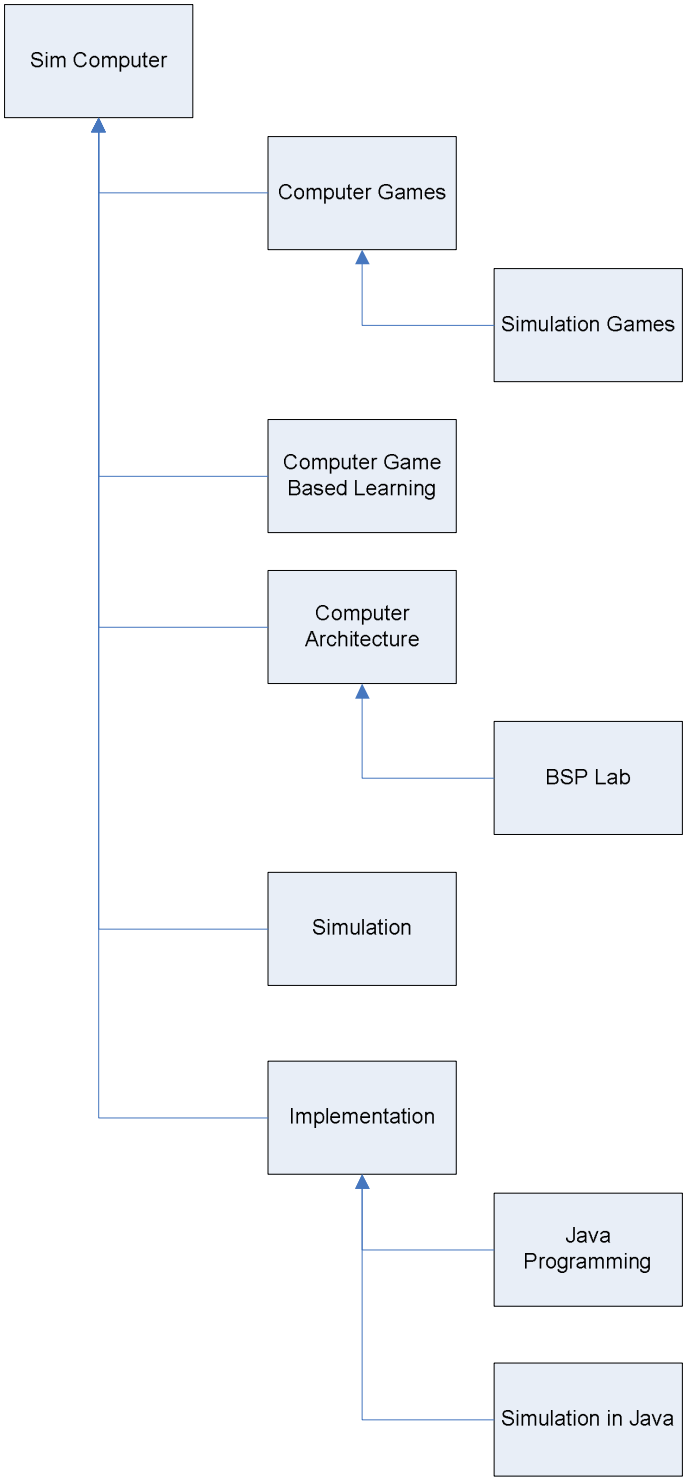

Figure 1: Ares of developing a simulation game

2. Background

In this chapter I present some background information in the important areas in developing a simulation game. First I give an overview of simulators used in computer science and different approaches to doing simulation. Then I give an overview of the most important aspects of computer game based learning. The third part covers the different types of simulation games and looks at some of the best known examples. Computer architectures and Bulk Synchronous Parallel (BSP) programs is the area that is going to be simulated in the game so I give a brief introduction to the basics of computer architecture and the principles of BSP.

2.1. Simulation

Simulations are used in many fields and can be done in many different ways. There are two main types of simulation, analog and computer simulation. Analog simulation involves physical models [Fishwick 1994]. The most relevant type to this project is computer simulation, which is defined by Paul A. Fishwick as.

Computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output. [Fishwick 1995]

The reason for doing simulation instead of just studying a model is that the system can be very complex with many interacting elements, which makes it impossible to just look at the model. Observing the real physical system might also be impossible or very expensive. In the case of computer architecture the ideal would be to build the actual computers and run programs on them, but this would be very expensive and time consuming and not efficient for learning purposes.

An important simulator for computers is the Simics platform [Simics 2002]. It is a simulator framework that supports simulation of a full system.

An area within simulation that is relevant when creating a simulation game is visualization. The Swedish Learning Lab [SweLL 2005] has a project where they use visualization and simulation for higher education in the areas of medicine and archeology.

2.2 Computer Game Based Learning

Why make a simulation game instead of just a simulator for computer architecture? The difference between a simulation game and a normal simulation is the added play and fun factors of the simulation game. When playing a learning game the student does not only have the goal of learning, but also having fun playing the game. Having fun is a very powerful way of motivating students. Ordinary learning methods like lectures and exercises are often viewed as boring by students [Friis, Tollefsrud 2004].

The use of games, play and fun is nothing new in learning when it comes to children. Edutainment and computer game based learning is trying to change learning at all levels and age groups by making it fun and entertaining, but not everyone agrees that it has the desired effect of improving learning. We are however only at the very beginning of development in

this area. E-Learning and computer based learning has only existed for a little over 20 years with most of the work done being aimed at children and company training programs. At college and university levels we have only the last years started to see some projects being implemented, one of them being the Age of Computers game for use in a computer fundamentals course at NTNU [Natvig, Line 2004].

When making a computer based learning game it important to select the right type of game, the right learning method and have clear view of what the students should learn from it [Prensky 2001]. Making those choices is not an easy task and requires knowledge and experience in many areas. Sometimes it just comes down to trying out different approaches and learning from the attempts, like in this project.

The type of game to make depends on who the players are going to be and the purpose of the game. People like different types of games, some love action games and hate strategy games while others don't like computer games at all. So it is usually best to select a type of game that most of the target group likes or at least not hates. Finding a game type that fits the purpose can sometimes be harder, not all areas of education can be mapped directly to a game type. This is where the designer must be creative.

In this project the starting point was an idea that a simulation game might be suited for learning about computer architectures. One can believe that engineering students are interested in playing a technical simulation game. Other than that I have no grounds for saying that this approach is the best, but that is also the purpose of exploring the different possibilities.

2.3 Simulation Games

In a way all computer games are simulations. They try in different ways to recreate a real world situation or setting for the player to enjoy, but far from all are classified as simulation games. There are according to Mark Prensky [Prensky 2001] eight main genres of computer games, one of them being simulation games.

- Action games
- Adventure games
- Fighting games
- Puzzle games
- Role-playing games
- Simulation games
- Sports games
- Strategy games

Action games involves as the name suggests action of some kind, often shooting and killing opponents or monsters. They test the reflexes and control handling skills of the player, often referred to as twitch skill.

Adventure games tells a story by letting the player take part in it, exploring a world and facing challenges a long the way.

Puzzle games gives the player a problem to solve, requiring him to use his imagination and intelligence to solve it. The types of problems can range from quizzes to understanding intricate systems.

Role-playing games lets the player take on the role of a character in a setting, making choices on how the character should act. It usually involves making choices on how the character should evolve.

Simulation games focus on recreating a part of the real world and letting the player control how the simulation should run. There is usually a specific goal or purpose that the player must reach.

Sports games are more or less accurate computer versions of different types of sports. They might seem to be very much like simulations at first, but considering that sports are very physical, the difference between pressing keys on a keyboard and actually doing it is huge.

Strategy games revolve around making strategic choices in a given setting.

Only very simple games belong to one genre, usually a game belongs to at least two of them. The most typical thing about a simulation game and what separates them from the others is the degree of realism. A game like X-Wing Alliance [Lucas Art 2005] is unlike Microsoft Flight Simulator [Microsoft 2005] not classified as a simulation game even tough both revolve around piloting different types of aircrafts. In X-Wing Alliance the entire setting is made up, not even the real world laws of physics are present. While in Microsoft Flight Simulator both the aircraft and the environment the player flies in, are made to be realistic.

I found that simulation games can be divided into two main types based on how the player is positioned in the game, either as an *observer* or a *participant*.

In the *observer* type of game the player views the game world from the outside, often from the above. He has access to detailed information about some or all of the elements in the world, often as numbers and textual descriptions. The pace of the game is usually controlled by the player as steps, which gives him unlimited time to think and plan his actions. Strategic planning is more important than the response time and reflexes of the player. Examples of this type of game are Sim City [EA 2005] and Panzer General [Ubi 2005].

In the *participant* type of game the player is placed inside the world, often in the form of a human character or as a driver/pilot of a vehicle. Information is restricted to what he sees and senses in proximity to himself. The representation of information can be in the form of realistic graphical objects or recreations of real world instruments, like gauges on the instrument table in an airplane. Unlike the observer type, the players own response time, reflexes and senses are very important since the game is set in real time instead of player controlled steps. Examples of this type of game are Microsoft Flight Simulator and Gran Turismo [Sony 2005].

For a game that is going to simulate how computer architectures work I believe the observer type to be the best suited. The player needs time to consider different choices and his own physical skills are not relevant to the subject. It would also be difficult to create a realistic setting where the player views the game from the inside. The observer type is more like the way we look at computers in the real world, from the outside, not running around on the inside.

There are many very different games of the observer type, covering many different areas of the real world. We have war simulation games that recreate historic battles, human relations

simulations that deal with how people interact, but I think the three most suited types for a computer architecture simulation are business simulation, competition simulation and building simulation.

Business simulation

The player is put in control of some sort of company or business depending on the area being simulated and his goal is to expand and make money, or at least not go bankrupt. Economy and financial control is essential in this type of game. Examples are games where the player deals on the stock market or a shop simulation like Pizza Tycoon [Pizza 2005].

Competition Simulation

This type of game revolves around competing against one or more opponents in a simulated environment. The opponents can either be computer controlled or other human players. Sport simulations where the player controls a team like in Championship Manager [Eidos 2005] are typical examples.

Building Simulation

In a building simulation game the player is responsible for the planning and building of some sort of simulated system. These games are usually presented very graphically, letting the player see what he is building and how it grows. Typical examples are Sim City where the player builds a city and Railroad Tycoon [Take 2 2005] where he builds railroads.

2.4 Computer Architecture

The area of computer architecture is huge and making a simulation game that covers all of it is a nearly impossible task. With my limited background in the area – only having taken the computer fundamentals course, I have to restrict the area covered by the prototype. From the problem definition I have that the game should offer learning in the area of computer architecture and possibly link with BSPlab. With no specified course or content for the game it is up to me to figure out how to approach the area and create a simulation game out of it. I will in this chapter give an overview of the possible areas to simulate in the game and which relevant courses at NTNU they relate to. I will also give an overview of BSPlab and how it can be used as the basis for the simulation in a game.

Computer architecture is defined to cover three aspects, instruction set architecture, organization and hardware and for a computer architect there are two important measurements, cost and performance [Hennessy, Patterson 2003].

Instruction set architecture is about designing the actual instructions that are available and used by the programs running on the computer. The organization in the architecture is the design of memory, buses and the processors. The hardware is the specific implementation of one machine. Several computers can share the same instruction set and organization but have different hardware implementations [Hennessy, Patterson 2003].

Cost and Performance

When designing a computer architecture, the functional requirements of the computer is of course important, but another important factor is the cost and performance. The cost of a computer can simplified be said to be the sum of how much each component costs, including

things like development and production costs. In an ideal situation a designer would be able to use any component he wants, ignoring the cost, but in most real situation there is a limited amount of resources available. The designer must make choices on how to spend the resources. It can often end up with balancing cost versus performance.

When measuring performance of a computer, one can either measure execution time or throughput [Hennessy, Patterson 2003]. Simplified one can say that execution time is how long it takes for a computer to execute a program and throughput is the total amount of work done in a given time. On most computers more than one program run at the same time so improving one measurement does not necessarily improve the other. Lowering the execution time of one program could result in lower total throughput. Measuring is done by running a program that could either be a set of real applications or some kind of benchmark or kernel program.

Courses at NTNU

At IDI there are several courses covering the area of computer architecture, ranging from basic introduction courses to the more advanced specialized. Table 1 shows a list of the courses and which area of computer architecture they cover.

Table 1: Computer Architecture courses

Course Code	Name	Covered Area
TDT4260	Datamaskinarkitektur	Organization
TDT4255	Konstruksjon av datamaskinsystemer	Instruction sets, hardware
TDT4220	Ytelsesvurdering	Measurement Performance
TDT4200	Parallelle beregninger	Organization, Measurement Performance
TDT4160	Datamaskiner, grunnkurs	Instruction sets, hardware

BSPLab

Leslie Valiant [Valiant 1990] created in 1990 the BSP model for parallel computation. It has three parts, processing elements with its own memory, a communication system and a synchronization mechanism. A BSP configuration is organized in super-steps which are sets of processors running code, doing synchronization between each super-step. There are four main parameters in for a parallel computer in the BSP model; the number of processing elements, processor speed, synchronization latency and communication throughput ratio [Dybdahl, Uthus 1997].

BSPLab is a simulation environment for BSP programs developed at NTNU by Dybdahl and Uthus as their diploma. It is capable of running simulations of BSP on several different types of parallel computer architectures [Dybdahl, Uthus 1997]. The areas of computer architecture covered by BSPLab, is mainly organization and measurement of performance.

The use of BSPLab is based on a configuration file. It has no clearly defined interface or way of communicating with another program. There is however plans to create an interface in the future.

Area to Simulate

I now have five aspects of computer architectures for parallel computers and BSP programs that could be used in a simulation game. Some of them are covered by one or more courses at NTNU.

- Designing instruction sets
- Organizing the components
- Designing hardware components
- Balancing cost
- Optimizing performance

3. Developing a game concept

As shown before, a simulation game can have many different forms and two of the goals of this project are to explore different possibilities and make a prototype of one of them. To achieve that I define the purpose of the game and develop some conceptual ideas for games that might fit the purpose. After that I select one of the ideas to develop further as a prototype.

3.1 Purpose of the game

From chapter 3.4 I have five aspects of computer architecture that could be used as the basis for a simulation game. Which one to focus on depends on what the learning purpose of the game is and what the students are supposed to learn from playing the game. Prensky lists a number of learning types, some of them are [Prensky 2001]:

- Facts
- Skills
- Reasoning
- Communication
- Creativity
- Procedures
- Systems

Learning about computer architecture involves mainly learning facts, reasoning and systems. The facts are details about hardware configurations, instruction sets, architectural patterns, definitions, names, numbers and everything that just needs to be memorized. Reasoning is learning about making strategic and tactical choices, in computer architecture that means making the right choices in configuring and designing a computer. Learning about systems involves learning how things work together and the principles the systems are based on. The systems in computer architecture are the different ways of organizing components. In my opinion a simulation game is not the best suited for learning facts. Reasoning and understanding systems fits the way a simulation game works better.

Ideally the game would cover all five aspects of computer architecture, but with limited time for this project and considering that it is only a first prototype, I will narrow the focus down to three areas. Reasoning and understanding of the systems involved in organizing components will be the main area of the game, with optimizing performance and cost balancing as the motivation for making choices. The reason for this choice is that BSPlab covers organization and performance and the relations BSPlab has to this project. Focusing on simulation of instruction set design or hardware could work too, but it would have no direct relation to BSPlab.

3.2 Three Ideas

From chapter 2.3 I have three suitable types of simulation games, business simulation, competition simulation and building simulation. I have developed three conceptual ideas, one for each type, based on the purpose of the game. They are only described briefly and only one will be developed further.

Computer Tycoon

The player is the administrator of a data processing center and must make tactical and strategic decisions on how to solve tasks provided by clients. It covers choosing the right type of hardware, architecture and software for solving a task. There is also a financial aspect of the game where the player must manage resources and make money by solving tasks for clients. Being efficient and providing a good service will draw in new clients. Money can be used to purchase new hardware, architecture plans and software. The game can be played in single player or expanded to a multiplayer game where several players compete for the customers. It is possible with both real time and turn based playing.

Comments: Based on the business simulation type. The setting is realistic and there is a good combination of making the right architectural choices and balancing cost and performance. The financial aspect of running a business gives the extra benefit of possibly learning about economics.

Computer Manager

The player is the manager of a "computer team" consisting of hardware arranged according to different architectures chosen by the player, much in the same way a football manager picks players and sets formations in a football manager game. He then plays matches against competing teams. A match is a given program or set of programs also known as a workload and the team with the best performance running the workload wins. The game can either be played against computer opponents or against other human players in a turn based arrangement.

Comments: It can be a very competitive multiplayer game based on the competition simulation type. The system is simpler and focuses mostly on computer architecture elements and not on other aspects like economics.

Computer City

The player is in charge of building a "computer city", a system of computer hardware components. The basis is a grid-map where the player can place processors, memory, storage etc. and connect them with communication paths. The inhabitants of the "computer city" are different types of software and computational task. Each task will with some degree of randomness choose to occupy a part of the city that is most suited to its need. Tasks can be "taxed" by the player to provide income for the city that can be used for expansion and upgrades. As time goes by the tasks evolve and demand more.

Comments: Based on the building simulation type. The system is a lot more complex than simulating running of programs and involves too many elements for a project of this size. It can be run as turn based or in real time. Seems to be difficult to develop properly and implement in a good way.

3.3 One Prototype

All three ideas from 3.2 are viable for further development, but I have only time to prototype one of them. None of them stand out as the best option so I choose to go with what seems to be the simplest idea, Computer Manager. It also has the feature of possible direct competition between players, which in this case are students. Competing against fellow students in an informal manner in game might serve as a good motivational factor.

The problem definition and the goals of the project say little about the purpose of the prototype. Normally a prototype is made to test certain features of a program, for example user interface, algorithms or system logic. The purpose of this prototype is mainly two things:

- Evaluate the conceptual idea, Computer Manager.
- Evaluate how to develop and implement the idea.

Evaluation of the conceptual idea is concerned with the selection of a type of simulation game and if a simulation game is the right approach at all. The prototype can provide input to making such an evaluation.

Evaluation of development and implementation is concerned with the details of the specific conceptual idea, not whether it is the right thing to do. The prototype can provide input to what works and doesn't work in the conceptual idea and help in improving it by inspiring new ideas.

4. Computer Manager – Conceptual Design

This chapter describes in detail the conceptual design of the game, covering how the game is played, what elements it consists of and how these relate to each other, the computer architectural components that are used and how simulation should be done. It also acts as the requirements specification for the prototype, but is open in many areas on how things should be done since this is a prototype.

4.1 Overall Description

The player is put in the role of the manager of a computer team. The team competes in a league against other teams, playing a series of matches against each other. A ranking system shows how well the teams have done and in the end of the series a winner will be declared. This is similar to a football-league; table 2 shows some of the analogies to football.

Table 2: Football Analogies

Football	Computer Manager
Football Team	Computer Team
Players	Hardware components
Formation and tactics	Architectural organization
Match	Workload (Program)
Goals	Run time
Injured players	Broken hardware

The game play is divided into the following phases:

- Creating a new game
- Managing the team.
- Playing a match.
- Reviewing the results.

Figure 2 shows the phases of the game and how they are connected. A game starts with the create new game phase, before moving on to the manage team phase where the player configures his hardware and architecture. The play match phase involves taking the players' architectural configurations and simulating running workloads on them in a set of matches. The results of the simulation are presented in the review results phase. After reviewing the results the player returns to the manage team phase. Going around that cycle once, is defined as a turn and the flow of the game is turn based. The details of each phase are described in chapter 4.3.

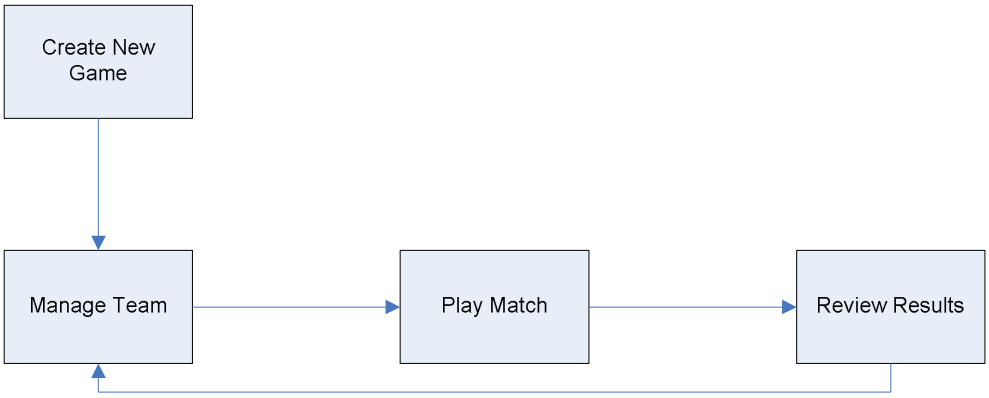

Figure 2: Game Phases

4.2 Game Elements

The computer manager game consists of elements that cover different areas of the game. They are the objects the player interacts with during the phases of the game and the foundation for the game play.

The game consists of the following elements:

- Computer Teams
- The League
- Series of Matches
- Resources
- Simulation
- Results and Ranking

4.2.1 Computer Teams

A computer team is similar to a football team, but consists of hardware and an architectural organization instead of players and tactics. It can be as simple as a single processor personal computer or as complex as a parallel processing computer system. The purpose of the computer team is to run different types of software as fast and efficient as possible in competition against another computer team.

Attributes

A computer team has a set of attributes describing it; they are listed in table 3.

Table 3: Computer Team Attributes

Attribute	Description
Team name	Just like football teams have names a computer team must have one too. It can be decided by the player when he starts playing.
Manager name	This is the name of the player. It can be his real name or made up.
Hardware components	The components the computer system is built from. Depends on how complex the simulation is, which is covered in 4.2.5.
Architectural organization	The organization of the hardware components.
Resources	A team has limited resources to spend on hardware and architecture. Described in 4.2.4

Computer Opponents

There are two types of computer teams, player managed and computer managed. The computer managed teams act as artificial opponents for the player and take part in the league just like human players. To give a more realistic playing experience when using multiple computer opponents and only one human player the computer opponents should vary in how well they play. Some of opponents should be easy to beat while others should be almost impossible. A game that is either too hard or too easy is not fun to play, so it is important that a balance is found between easy and hard opponents.

4.2.2 The League

The league consists of a set of computer teams, both player-managed and computer opponents. The human player will meet each computer opponent in a series of matches (see 4.2.3), meaning that he will face the same type of match several times. This allows him to learn from mistakes and adjust his architecture from what he learned in previous matches. The computer opponents will in a similar way play against each other. The results from each match are used to create a ranking and result table.

With four teams and one match type there would be three matches per team going over three rounds and a total of six matches played; shown in table 4.

Table 4: Rounds in the League

	Team 1	Team 2	Team 3	Team 4
Team 1		Round 1	Round 2	Round 3
Team 2	Round 1		Round 3	Round 2
Team 3	Round 2	Round 3		Round 1
Team 4	Round 3	Round 2	Round 1	

4.2.3 Series of Matches

The purpose of the game is to play through a series of matches to decide who the best team is. A match is a computational task, a workload that the computer team must run. Each match type represents a different workload. Types of workloads can range from simple ones that involve sorting a data set or doing calculations to complex ones consisting of multiple programs and huge amounts of data.

For learning purposes the series of matches can be organized so that the player faces increasingly more difficult problems and must use his experience from the previous ones to solve the next in the best way.

Workloads

A workload can basically be any type of program or set of programs that can be run on a computer system. There are two approaches to creating workloads for use in matches; predefined ones or generation from templates. Using predefined workloads means that the player faces the exact same matches every time he plays and it will be possible to find configurations that are the best for each type. This reduces the replay value of the game. The other option is to generate the matches during run time from a set of predefined templates. This will give the player a slightly different experience every time.

Hennessy and Patterson suggest three different classes of workloads for use in understanding how the characteristics of a workload affect the performance of a multiprocessor computer [Hennessy, Patterson 2003]. They are shown in figure 3.

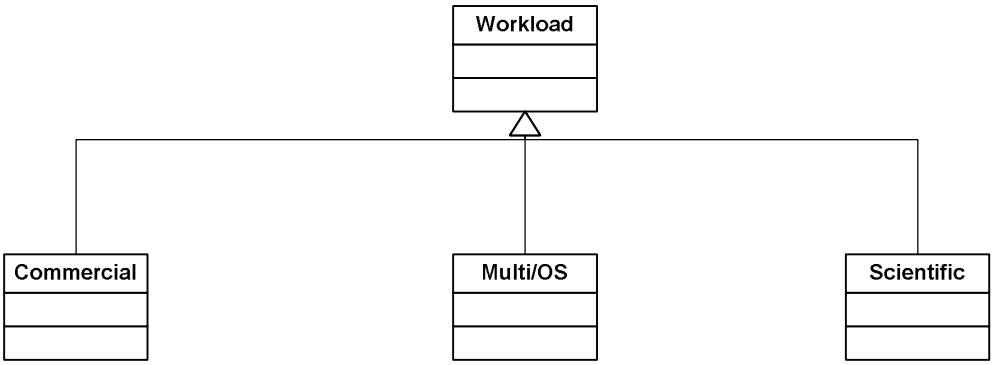

Figure 3: Classes of Workloads

Commercial

This type of workload consists of typical commercial applications with several processes running in parallel, many users and huge amounts of data. Three possible applications of the commercial type are shown in figure 4. [Hennessy, Patterson 2003]

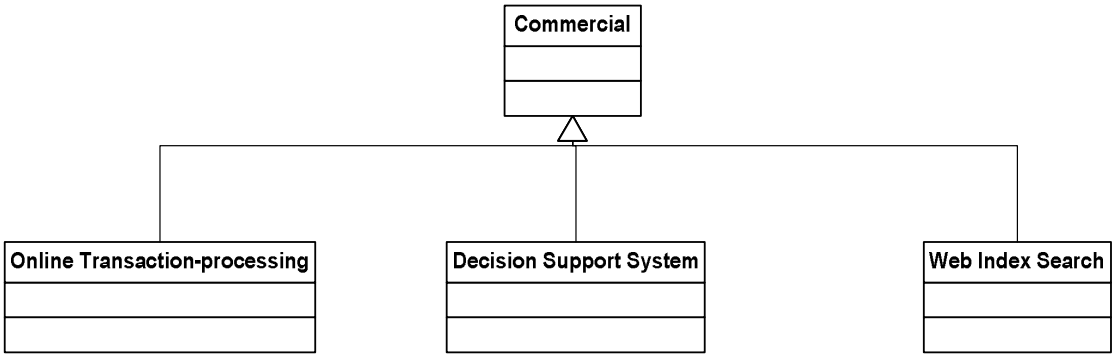

Figure 4: Commercial Workloads

Multi/OS

A multiprogramming and OS workload consist of user activity and operating system activity running on a smaller scale multiprocessor system.

Scientific

These types of workload are individual parallel programs from the scientific domain. Hennessy and Patterson suggest using the computational kernels fast Fourier transformation (FFT) and LU decomposition and the two applications Barnes and Ocean; shown in figure 5 [Hennessy, Patterson 2003].

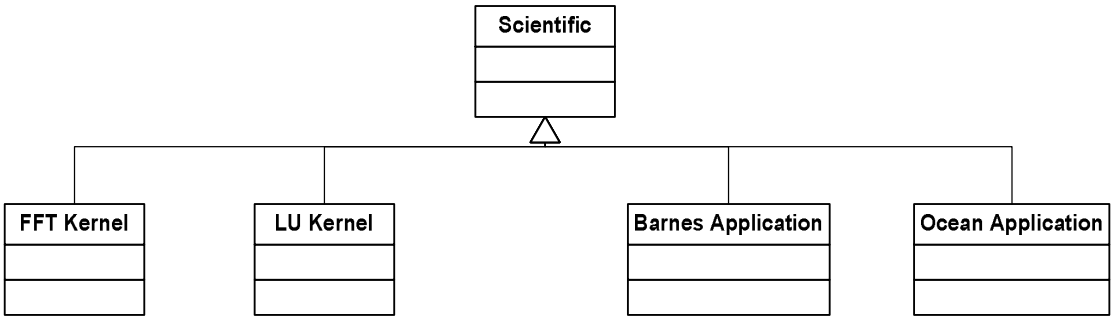

Figure 5: Scientific Workloads

The mentioned classes of workloads can either be used as predefined workloads or be the basis for templates that workloads can be generated from. Generation from templates can be done in several ways:

- The template is a set of basic workloads that is combined to create a workload for a match.
- The basic workloads can have defining parameters that are set to create different variations of the workload.
- Develop a modeling language for the characteristics of workloads and use it to generate different workloads for matches.

The best solution for the prototype is using predefined workloads.

BSP Workload
The special structure of a BSP workload can be used in the game to show the progress of a match. Simplified a BSP program consists of a set of super-steps and synchronizations. When two computer teams play a match against each other the current super-step on each computer can be used to determine who is in the lead. This feature of a BSP workload opens up the possibility of graphically visualizing a match for the player and allowing him to adjust his configuration in the middle of a match, just like a football team can change players and tactics during the match.

4.2.4 Resources

An important element in many simulation games is resource management. Resources can be many things; money, building materials, energy, time, capacity in an elevator or space. The challenge for the player is to make the best choices on how to spend his resources. Without a resource management aspect in a simulation game it looses some of the game feeling and becomes more like simulation tool.

The resource management aspect of computer architecture is balancing cost versus performance. Purchasing hardware components costs money, developing new architectural organizations costs money and maintaining and running computer systems costs money. Money is the main resource to manage for the player. I will only give a brief description of a resource management system for the game, since this aspect is of low priority for the prototype.

Making money

When the player starts a new game he is given a small sum of money to get started. It is enough to purchase some basic hardware and build a simple computer system. He can then make money from playing matches. How much each match pays off depends on how well the player does and the type of match. A system where only the winner gets paid would not work since it could quickly get unbalanced by one team winning several matches and becoming harder and harder to beat. Instead both teams in a match get paid and the winner gets a bonus.

Maintenance

All hardware components have a maintenance cost that must be paid every turn. The player must make strategic choices on which components to keep and sell and if he wants to keep a lot of hardware it is going to affect his ability to purchase new components.

Buying and Renting

There are two ways of acquiring new components, either buy them or rent them. Buying a component is usually more expensive than renting it. Maintenance cost must be paid for both rented and bought components. Renting a component lets the player use that component for one turn and is often cheaper than buying it if he only needs it for one turn.

Selling

Bought components can be sold back to the shop or to other players. Shops buy used components at a lower price than they sell them, while players can make a more varied offer. Selling older components can be wise to avoid having to pay maintenance for them.

Cost of Components

The cost of components depends on their quality. Higher quality components usually last longer and have lower maintenance cost but a higher price. Lower quality components are cheaper to buy but may turn out to be an expensive investment over time.

4.2.5 Simulation

The simulation element in the game is one of the most important parts. It covers the details of architectural configurations and how they are simulated. Figure 6 shows the three important parts of the simulation element; the simulator, the architecture it simulates and the workload running on the system. Workloads have already been covered in 4.2.3 but I will give a description of the two other parts.

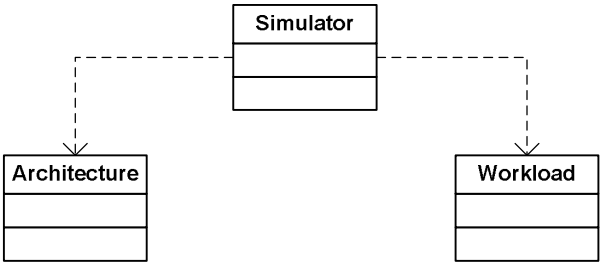

Figure 6: Overview Simulation

Architecture

The computer architecture components in the game are based on those that are supported by BSPlab and are taken from Dybdahl's and Uthus' report [Dybdahl, Uthus 1997]. Figure 7 shows an overview of the architecture components in BSPlab.

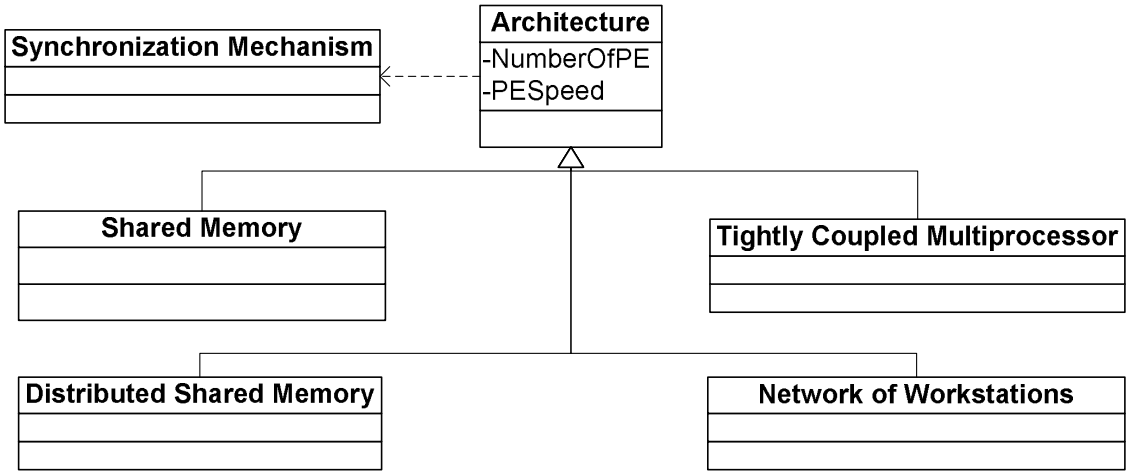

Figure 7: Computer Architectures in BSPLab

There are two important parameters for all architecture classes, the number of processing elements (PEs) and the speed of the processing elements. All BSP architectures also have a synchronization mechanism. Network of workstations is not included in the prototype.

Synchronization Mechanism

BSP uses barrier synchronization to synchronize processing elements. Barrier synchronization can either be hardware or software implemented. In a hardware implementation the processors use circuits to signal and are faster than software synchronization but have an extra cost. The software synchronization uses the communication system to pass messages.

I have defined three types of synchronization for use in the game; shown in figure 8. The BSP synchronization is more advanced than the basic barrier synchronization and is implemented using one or more barrier synchronizations.

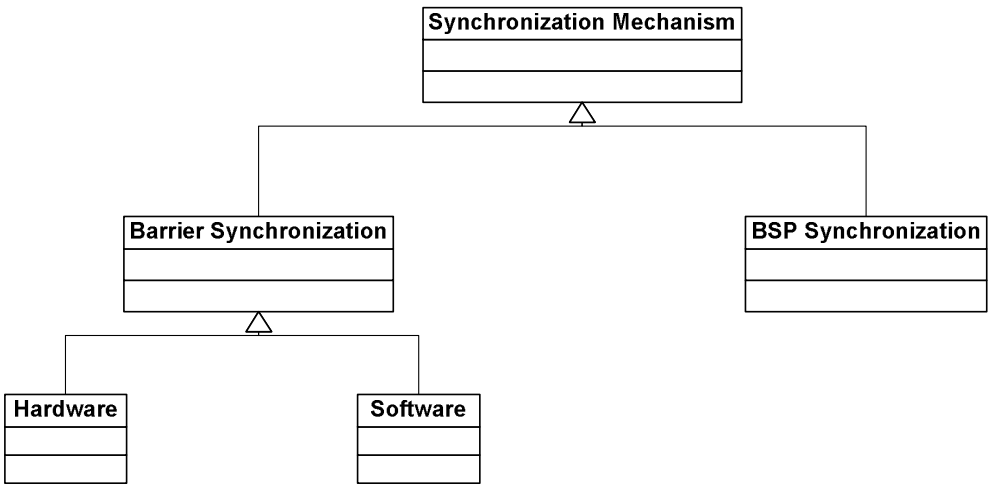

Figure 8: Synchronization Mechanisms

Shared Memory

A shared memory architecture consist of a set of processing elements and a centralized shared memory connected by an interconnection device. Figure 9 shows the components of a shared memory architecture. Each processing element is characterized by speed, cache size and block size. An interconnection device has parameters for band-with and latency. Memory modules are defined by size and speed [Hennessy, Patterson 2003].

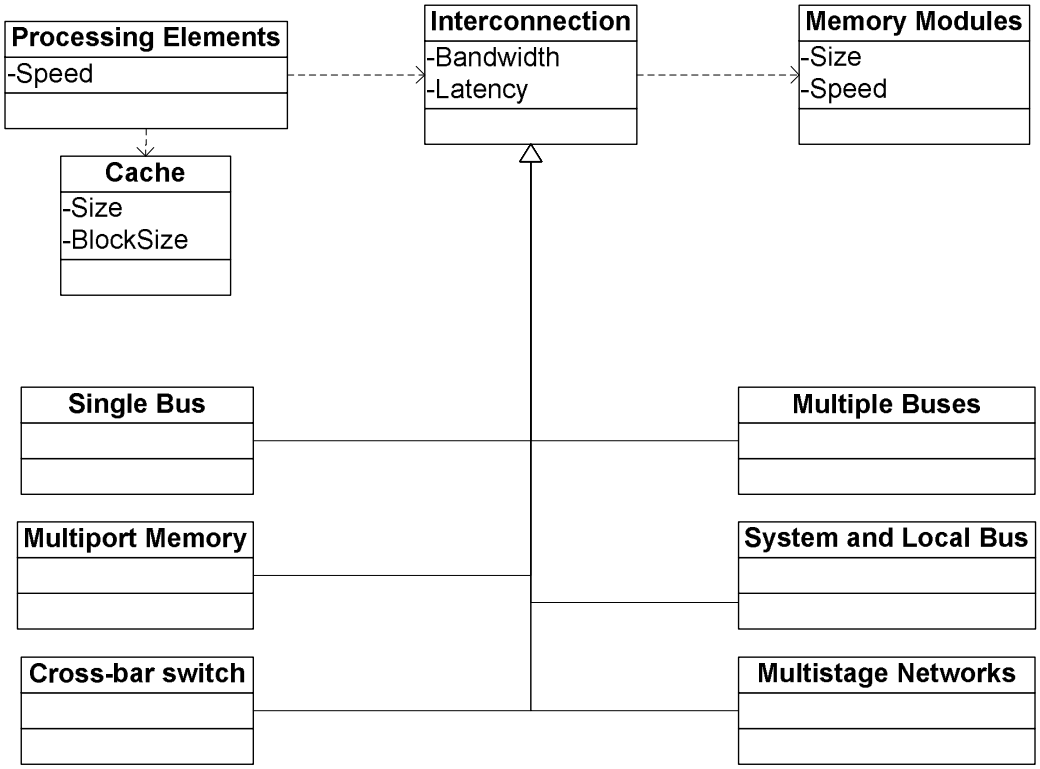

Figure 9: Shared Memory

Distributed Shared Memory (DSM)

This architecture is functionally similar to shared memory, with a shared memory address space. Physically the memory is distributed among the processors to support the band-with demand of a large number of processing elements. Figure 10 shows the components of a DSM.

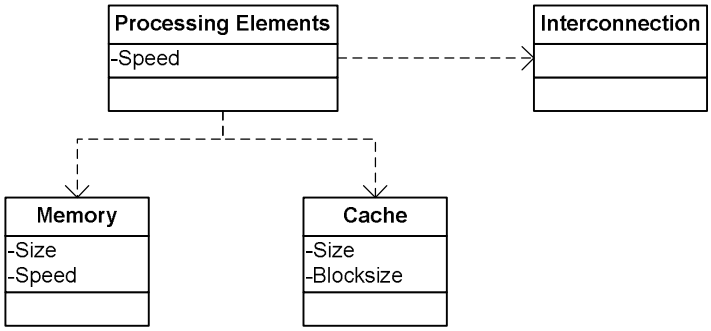

Figure 10: Distributed Shared Memory

Tightly Coupled Multiprocessor (TCM)

This type of architecture is a set of processing nodes connected by a network and is characterized by a routing mechanism and the topology of the network; shown in figure 11.

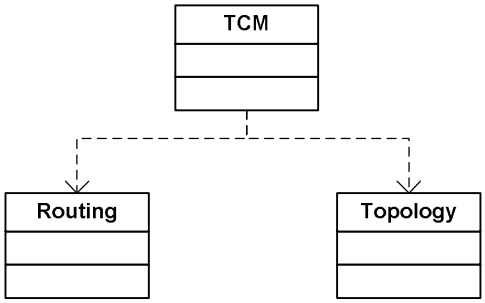

Figure 11: Tightly Coupled Multiprocessor

Figure 12 shows some possible topologies for a TCM [Dybdahl, Uthus 1997].

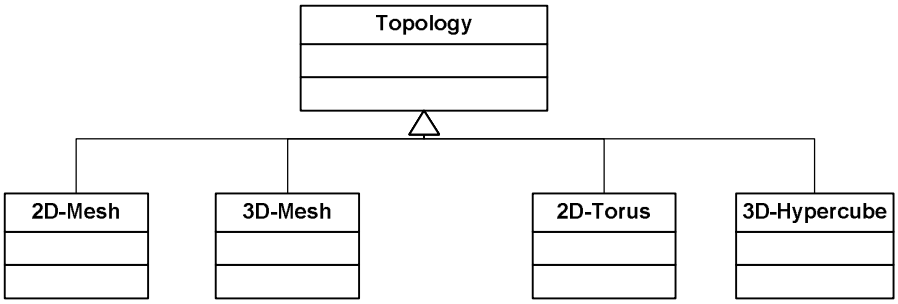

Figure 12: Topologies in a TCM

Routing is the mechanism for passing messages between processing nodes on a topology where not all nodes are connected directly to each other. Figure 13 shows some possible routing mechanisms [Hennessy, Patterson 2003].

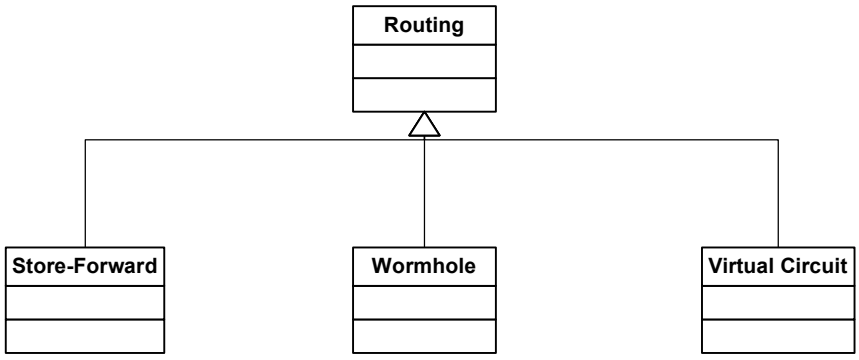

Figure 13: Routing Mechanisms

Simulator

The purpose of the simulator is to take an architectural configuration and a workload and simulate the execution of the workload and provide the results for determining the outcome of a match. The results from a simulation are the total runtime if the workload isn't of a special type.

BSP Simulation

With a special type of workload like a BSP program the results of a simulation are not only the total run time but also the runtime of each super-step and synchronization. Figure 14 shows the flow of a BSP simulation. Super-steps and synchronizations are repeated until the program is done, producing results every cycle. Table 5 shows an example of results from running a simulation of a BSP workload with two super-steps.

Table 5: BSP Results

	Time
Super-step 1	10ms
Synchronization 1	5ms
Super-step 2	20ms
Synchronization 2	7ms
Total Time	42ms

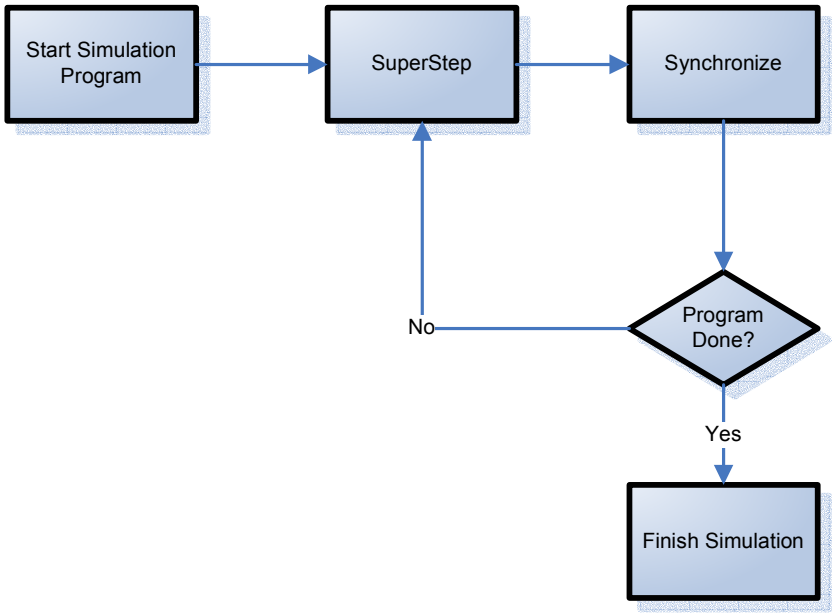

Figure 14: BSP Simulation

How to Simulate
Simulating a computer system is not an easy task, creating an accurate and realistic simulation is even harder. Fortunately a simulation for use in a computer game is not necessarily required to be very accurate and realistic. There must be a certain degree of realism but as long as the results from the simulation aren't completely illogical, for example an Intel 286 being faster than an Intel Pentium 4, the game can still be played.

There are four approaches to creating a simulator module for the game, with varying realism and accuracy.

- Real simulator
- Tables
- Formulas
- Dummy file

Real simulator, this approach is as the name suggests the only approach that is actually doing any simulation. It involves connecting the game to a simulation system like BSPlab and run real simulations of the matches. Running simulations in real time during the game might not be possible, so a kind of staged simulation could be used [Sirer, Walsh 2004].

Tables, an alternative to connecting the game directly to a simulator is to run a large number of different workloads on a simulation system and store the results in a table. Results can then be looked up in the table and interpolated to fit the configuration of a match.

Formulas, an even simpler approach than using tables is to define some formulas for calculating the results of a match from the parameters of the architectural configuration.

Dummy file, this approach is totally unrealistic and has nothing to do with simulation. It is just a set of made up numbers. The purpose of such a simple approach is to test other parts of the game in for example a prototype.

4.2.6 Results and Ranking

After each match a result from the simulation is generated and a winner of the match is decided depending on the type of match played. In most cases the criteria will be time spent on running the workload.

The teams are ranked according to a point system similar to a football league. The winner is given three points, the looser zero points and if it is a draw both teams get one point. A ranking table consists of points, number of played matches, number of wins and losses and total computing time. Table 5 shows an example of how it might look.

Table 6: Ranking Table

Position	Team	Played	Won	Lost	Time	Points
1	Norway	2	2	0	3000	6
2	Sweden	2	1	1	6790	3
3	Denmark	2	0	2	14200	0

4.3 Game play phases

The game play consists of one startup phases and three main phases that the player goes through for each match in the series. I will here describe what the player does in each of them.

4.3.1 Create New Game

When a player starts the game he has two choices, either start a new game or continue playing a saved game. The game is meant to be played over several sessions so the player must be able to save it so he can continue playing later.

Starting a game

When the player selects to start a new game he must enter the name of the team and his manager name. The game then sets up a new league, a series of matches and a number of computer opponents are created.

Loading and Saving

Saving a game involves storing all information about the state of the game; the league, the series of matches, the state of computer opponents and the state of the human player with his hardware components and architectural configurations. Loading the game means creating a game that has the same exact state as the one that was saved.

4.3.2 Manage Team

This phase is where the player configures his architecture and selects the hardware components he is going to use in a match. It involves managing resources; buying, renting and selling components, configuring his computer team and generally making strategic and tactical choices on how to beat his opponents.

Managing Resources

The activity here involves the elements described in 4.2.4. The player is presented with a list of available hardware and architectural components he can either buy or rent. Depending on his financial situation and strategy he must decide on how to spend his limited resources. The purpose of this is to learn how to balance the cost to get the best possible performance. Figure 15 shows a use case diagram of what the player does in this part of the phase.

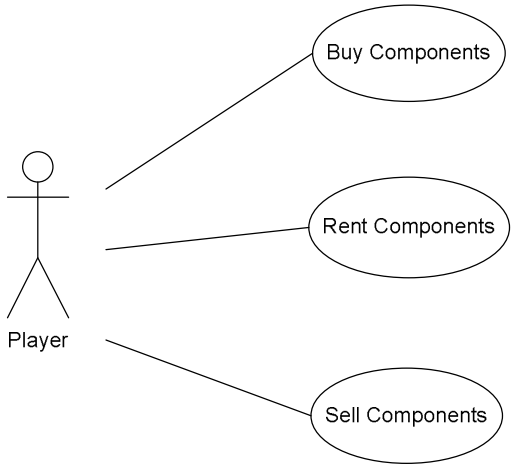

Figure 15: Use Case, Managing Resources

Configuring Architecture

The activities here involve configuring his architecture, setting parameters of components and trying to create a configuration that is able to win a match. Figure 16 shows a use case diagram of what the player does in this part of the phase.

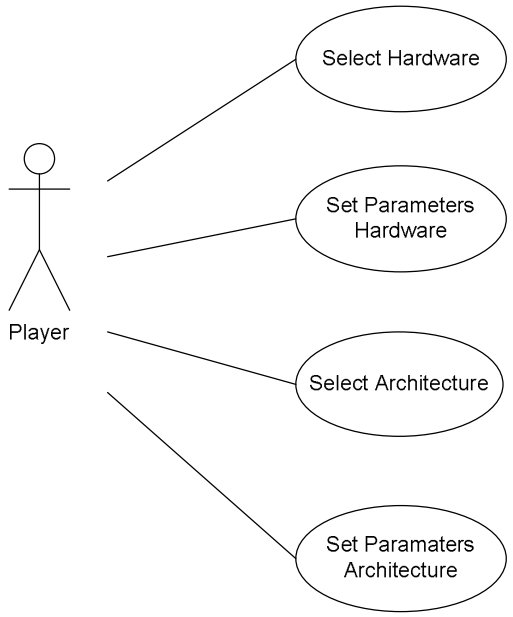

Figure 16: Use Case, Configuring Architecture

4.3.3 Play Match

In this phase the player has little interaction with the game. The game plays the scheduled matches for the round, generates results and updates the league table. Figure 17 shows how a match is played. The architectures of the two teams playing are sent to the simulator along with the match workload and results are generated from the simulation.

An alternative to just playing the matches without player interaction is to visualize the match for the player and allow him to adjust his configuration during it. I will not go any deeper into how this could be done, but it is something that should be considered for a more advanced version of the game.

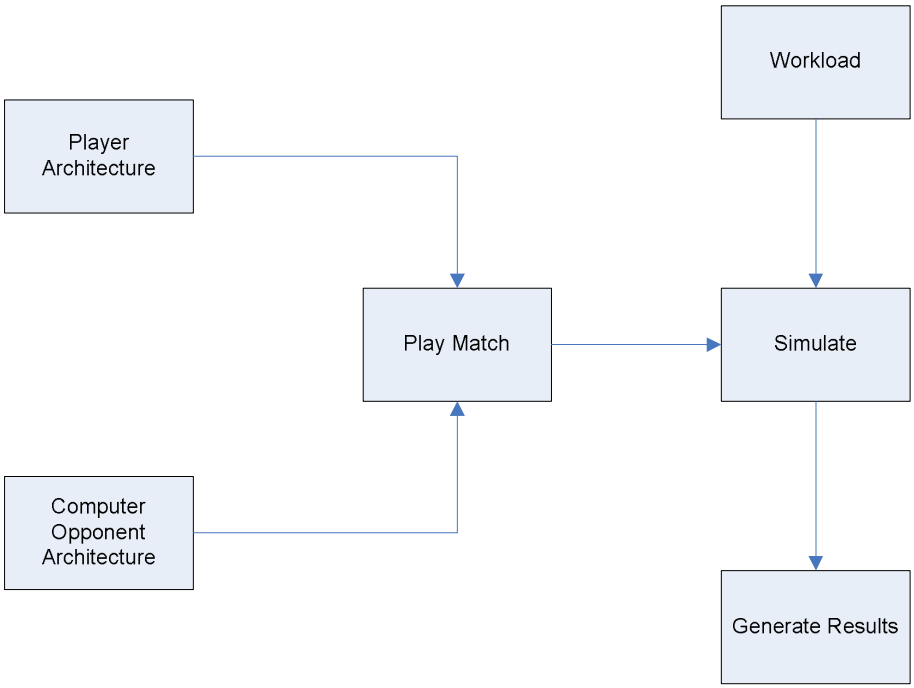

Figure 17: Match Flow

4.3.4 Review Results

In the review phase of the game the results from the played round are presented to the player. He can view the results from his own match and from the matches that only involve computer controlled players. A more advanced way of doing this is to allow the player to not only see the results but also view the architectural configurations of the computer controlled teams. This would give him some idea of how the opponents play and how to beat them.

5. Prototype Design

This chapter describes the design of the prototype, which modules it consists of, what they do and how they work together. The first part is a discussion on how to implement a simulation game for use in learning.

5.1 Implementation

A computer game can range from a simple text based program to a complex system of three dimensional graphics, sound and animation. With the power of today's personal computers the players expect much more from games than before. This puts extra demands on the game designer; a text based interface is not enough. Even two dimensional graphics can disappoint the players. This has led to higher development costs for game companies and only those with a great idea and enough resources to implement it properly succeed in the highly competitive market. The same demands do not necessarily apply for a learning game, unless it is aimed at the commercial market where the competition is strong even for these types of games.

As with most software today users at least expects an interactive graphical user interface, which is what I will use as a basis for evaluating different ways of implementing the simulation game.

There are three main approaches to developing a learning game [Friis, Tollefsrud 2004]. I will briefly discuss each of them and how they fit this project.

- Custom made
- Modification
- Commercial off the shelf (COTS)

In the custom made approach most of the game is made from scratch using only the libraries of the programming language and basic components. This approach requires experience in programming and designing user interfaces and is the most time demanding approach of the three. The advantages are that the designer is free to make the game the way he wants it to be and there is a certain value in having developed something brand new.

The modification approach is based on taking an existing game and modifying it to fit the intended learning purposes. Several games come with level editors, modification tools or content managers that allow the user to add or modify content, both learning content and game play. This approach requires less skill in programming and graphical design and could be less demanding in time, but not necessarily in cost.

The third approach involves purchasing an existing learning product. This approach is the simplest provided there is an existing product that fits the purpose.

One of the goals of this project is to develop a prototype of a simulation game. With that in mind all three approaches are theoretically possible. There is as far as I know no existing simulation game that covers computer architecture so that rules out the third approach. Taking an existing simulation game of some kind and modifying it to simulate computer architectures is a very viable option but requires the cooperation of the developers of that simulation game and would probably be a very expensive way of developing something that is only meant to be the first prototype. If someone at a later stage were interested in continuing the project the

modification approach should be explored further. A custom made approach is in my opinion most suited for developing a prototype in a project like this. It puts no restriction on how things are implemented and what kind of content it can have.

Implementing a simulation game from scratch requires some kind of development or programming skills. Since I am the only one working on this project I can only consider options I believe are viable with my experience of programming and development. There are two alternatives.

- Developing a stand alone program using the Java programming language.
- Developing a web based application, using a browser interface and an environment like Flash [Macromedia 2005].

Considering that the game is primarily a prototype, my preferred choice would be to develop it as a stand alone program in Java. I have a lot more experience with programming in Java and making graphical user interfaces than I have with making web based games. One alternative approach would be a combination of the two, making a web based Java application, but it would mean putting unnecessary restrictions on the game.

Java

Java is an object oriented programming language that supports the development of platform independent code. Instead of generating native platform dependant code during compilation it creates code that can be interpreted by a virtual machine. Implementations of the virtual machine are available for several different platforms including Windows, Linux and Mac.

The core of Java is the Java 2 Platform, Standard Edition (J2SE) from Sun Microsystems [Sun 2005]. It comes with libraries for creating graphical user-interfaces, playing sound and video, handling I/O against different types of devices and a set commonly used data-structures and mathematical functions.

An excellent book on game programming in Java is “Developing Games in Java” by David Brackeen [Brackeen 2004]. It covers the basics of drawing graphics, animation, input from players, sound, threads and 3D graphics.

Simulation in Java can be done using the standard libraries or a specialized simulation package like Desmo-J [Desmo-J 2005]. It is a framework for discrete-event modeling and simulation and it is free. An alternative is interfacing against a major simulation package like Simulink and Matlab [MathWorks 2005].

When developing a program and writing code, it is important to have the right tools for making the task easier. An integrated development environment (IDE) is one of those tools. Instead of writing code in a simple text editor an IDE can add several very useful features like color coding of text, in-line error checking of syntax, organizing files, debugging and generating code. For the implementation in this project I use Eclipse [eclipse.org 2005].

5.2 Top Level Design

In chapter 4 I describe a game containing several elements and both a competitive and financial aspect. For the prototype I have chosen to focus on the competitive aspect and leave out the financial aspect of managing resources. There are also some other minor elements that might not be fully included in the prototype. The reason for this is the limited time available for development and the need to get at least some parts of the game working in the prototype.

The prototype is divided into modules that provide different types of functionality. Figure 18 shows the modules and how they are connected. They use a shared datastructure package to exchange information. One important thing to notice is that there are two types of Player modules. A detailed description of each module is given later in this chapter.

The reason for dividing the prototype into modules is to make modification easier. Modules can easily be replaced by new implementations and different approaches can be tested without having to modify the entire system.

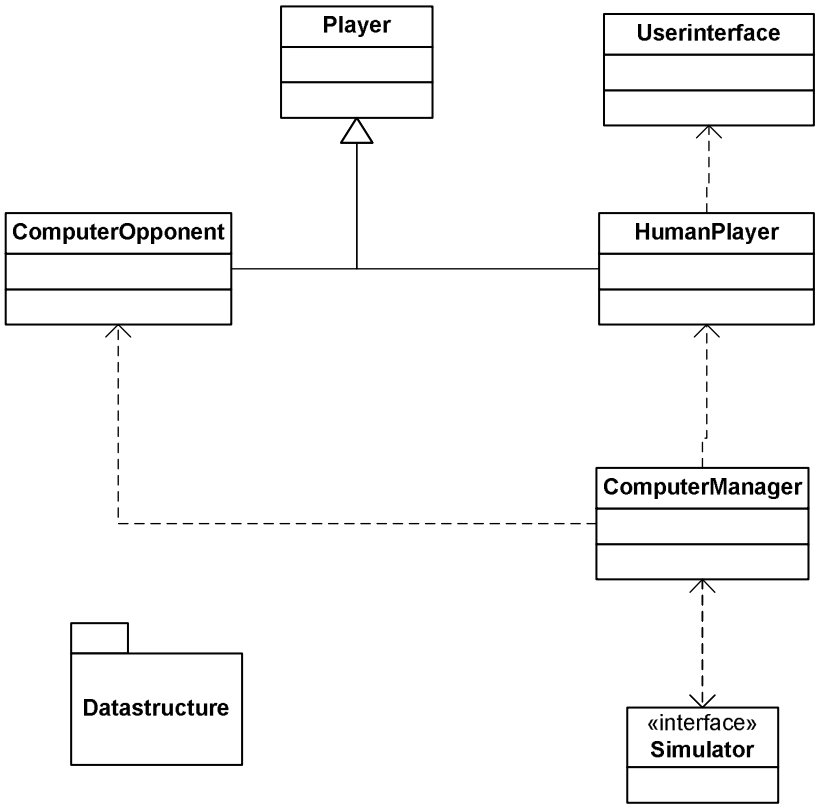

Figure 18: Top Level Modules

5.3 Data structure

The data structure package is a collection of classes that are used for storing and passing information between modules in the game. There are three structures in the package; game play, hardware and architecture components and constants.

5.3.1 Game Play

This structure covers the game play elements related to the league and matches. Figure 19 shows the structure.

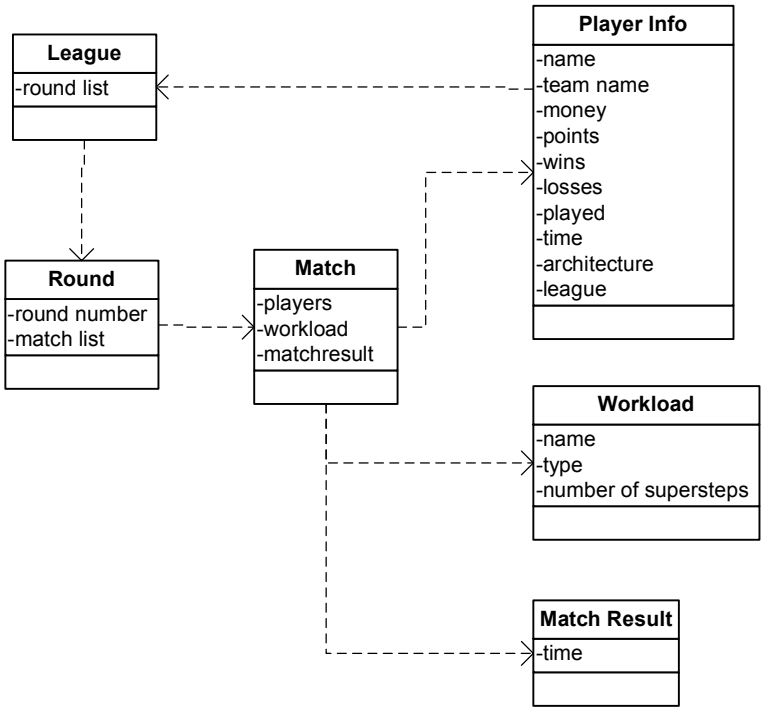

Figure 19: Game Play Structure

League
This class represents the league of computer teams in the game. It has one attribute.

- ***round list***, a list of the rounds that are going to be played.

Round
This class represents one round of matches. It has the following attributes.

- ***round number***, the number of the round.
- ***match list***, a list of the matches in that round.

Match
This class represents a match played by two computer teams. It has the following attributes

- ***players,*** a Player Info object for each of the team taking part in the match.
- ***workload,*** a Workload object describing the type of workload being used in the match.
- ***matchresult,*** a Match Result object for each team describing how the team did in the match.

Player Info
This class holds all the information about a player. It has the following attributes.

- ***name,*** the name the player chose.
- ***team name***, the name the player chose for the team.
- ***money***, how much money the player has available to spend on components.
- ***points***, total amount of points gained from playing matches.
- ***wins***, how many matches the player has won.
- ***losses***, how many matches the player has lost.
- ***played***, how many matches the player has played.
- ***time***, total time from all played matches.
- ***architecture***, the players current architectural configuration defined by objects from the hardware and architecture components structure.
- ***league***, reference to the top League object in the structure.

Workload
This class represents a workload used by a match. It has the following attributes.

- ***name***, a describing name for the workload.
- ***type***, the type of workload defined by a constant from the Workload class in the. Constants structure
- ***number of super-steps***, the number of BSP super-steps the workload consists of.

Match Result
This class represents the results of running a workload and has one attribute.

- ***time***, the time it took to run the workload.

5.3.2 Hardware and Architecture Components

This structure covers the hardware and architecture components that are used for configuring a computer team. Figure 20 shows the structure.

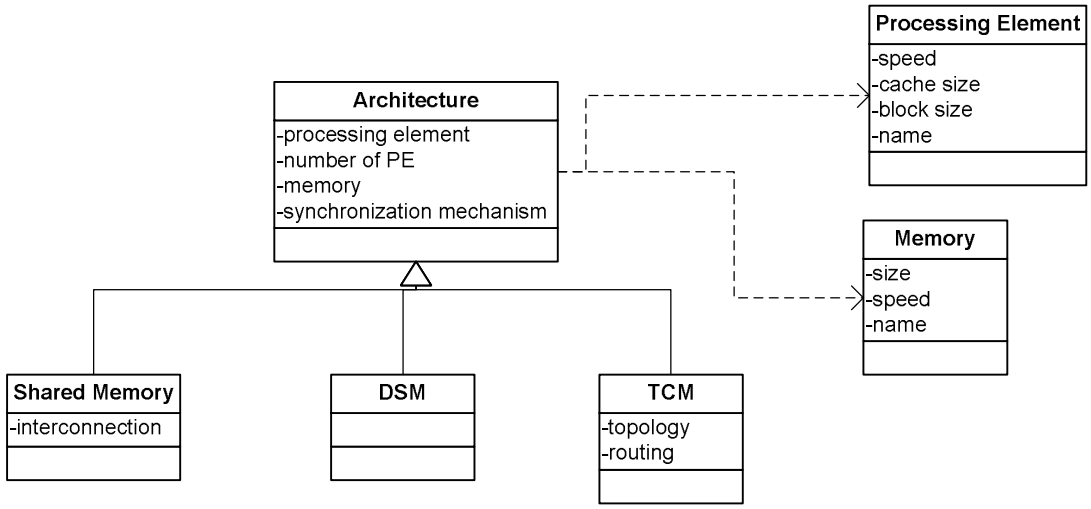

Figure 20: Hardware and Architecture Components

Architecture

This class is the super-class for all types of architectural configurations. The following attributes are shared among them.

- ***processing element***, a Processing Element object describing the parameters of the processing elements used in the architecture.
- ***number of PE***, the number of processing elements.
- ***memory***, a Memory object describing the type of memory used by the architecture.
- ***synchronization mechanism***, a constant defining the type of synchronization mechanism. The different types are defined in the Synchronization class in the Constants structure.

Shared Memory

This class represents a shared memory architecture and is a subclass of Architecture. It has one defining attribute.

- ***interconnection***, a constant from the Interconnection class in the constants structure, defining the type of interconnection.

DSM

This class represents a distributed shared memory architecture and is a subclass of Architecture.

TCM

This class represents a tightly coupled multiprocessor architecture and is a subclass of Architecture. It has the following defining attributes.

- ***topology***, a constant defining the type of topology. The different types are defined in the Topology class in the Workload structure.

- ***routing***, a constant defining the type routing. The different types are defined in the Routing class in the Workload structure.

Memory
This class represents the memory modules in an architecture and has the following attributes.

- ***size***, the memory size.
- ***speed***, the speed of the memory.
- ***name***, a name describing the type of memory.

Processing Element
This class represents a type of processing elements and has the following attributes.

- ***speed***, the speed of the processing element.
- ***cache size***, the size of the cache.
- ***block size***, cache block size.
- ***name***, a name describing the type of processing element.

5.3.3 Constants

This structure consists of a set of classes with constants defining types of architectural components. Figure 21 shows the structure.

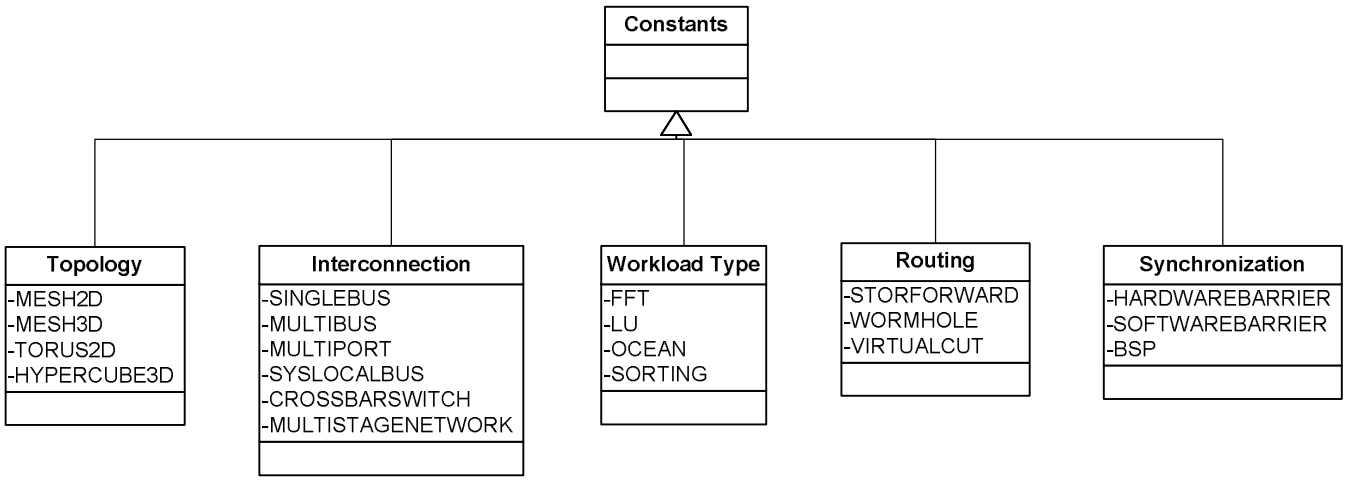

Figure 21: Constants

Topology
Class containing constants for defining different types of topologies in a tightly coupled multiprocessor architecture.

- ***MESH2D***, a two dimensional mesh topology.
- ***MESH3D***, a three dimensional mesh topology.
- ***TORUS2D***, a two dimensional torus topology.

- ***HYPERCUBE3D***, three dimensional hybercube topology.

Interconnection
Class containing constants for defining different types of interconnections in a shared memory architecture.

- ***SINGLEBUS***, a single bus interconnection.
- ***MULTIBUS***, a multiple buses interconnection.
- ***MULTIPORT***, a multi-port interconnection.
- ***SYSLOCALBUS***, a system and local bus interconnection.
- ***CROSSBARSWITCH***, a cross-bar switch interconnection.
- ***MULTISTAGENETWORK***, a multi staged network interconnection.

Workload Type
Class containing constants for defining different types of workload. For the prototype I have only included four types.

- ***FFT***, a fast Fourier transformation.
- ***LU***, LU kernel.
- ***OCEAN***, Ocean application.
- ***SORTING***, workload involving a typical sorting application.

Routing
Class containing constants for defining different types of routing in a tightly coupled multiprocessor architecture.

- ***STOREFORWARD***, a store-forward routing mechanism.
- ***WORMHOLE***, a wormhole routing mechanism.
- ***VIRTUALCUT***, a virtual-cut routing mechanism.

Synchronization
Class containing constants for defining different types of synchronization mechanisms.

- ***HARDWAREBARRIER***, a hardware implemented barrier synchronization.
- ***SOFTWAREBARRIER***, a software implemented barrier synchronization.
- ***BSP***, a special BSP synchronization.

5.4 Simulator

The simulator module is responsible for generating the time used in the results of matches. It takes an architecture configuration and a workload and returns the time it took to run it. The simulator is defined as an interface, shown in figure 22. This allows different ways implementing the simulator without having to change how it interacts with the rest of the game.

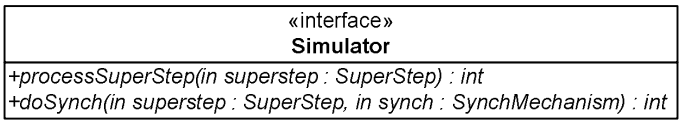

Figure 22: Simulator Interface

Simulator Interface

The simulator interface uses a similar data structure as the other modules for input, shown in figure 23. The difference is that it focuses on BSP super-steps. Each super-step and synchronization must be run individually on the simulator.

The interface has two methods:

- ***processSuperStep(in superstep : SuperStep) : int***
  Method that takes a SuperStep object as input parameter and returns an integer with the time it took to run the super-step.

- ***doSynch(in superstep : SuperStep, in synch : SynchMechanism) : int***
  Method that takes a SuperStep and a SynchMechanism as input parameters and returns an integer with the time it took to do the synchronization.

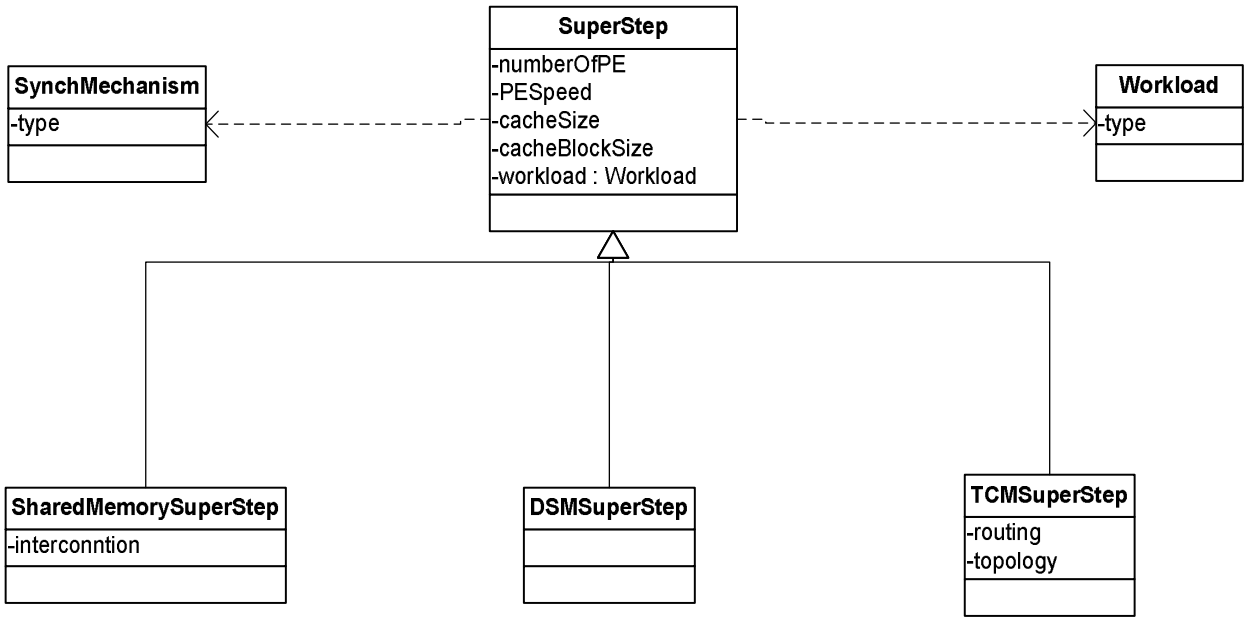

Figure 23: Interface Datastructure

Implementing the Interface

From 4.2.5 I have four approaches for creating a simulation module and I considered each of them for use in the prototype.

Initially I wanted to use a *real simulator* module in the prototype, either by connecting to BSPlab via an interface or creating a simulator myself. Since there is no Java interface for BSPlab and it is outside the scope of this project to create one, the option to connect the game with BSPlab was not possible. Creating a simulator specifically for use in the game is possible but with my limited time and skills in the area of computer architecture simulation it was not a viable option for the prototype.

Instead of using a real simulator I discussed the possibilities of using a combination of *tables* and *formulas* with Lasse Natvig [Natvig 2005]. He offered to help by providing results from simulations and setting up formulas. Unfortunately he was not able to do this due to limited spare time.

The simulator module in the prototype is a based on a *dummy file*. The results from running simulations are only made up numbers with no relevance to a real computer system. For a prototype at this stage the consequences of not having realistic simulation results are not severe. It is still possible to get an impression of what the game-play is like.

5.5 Players

There are two types of players in the game, human and computer controlled opponents, shown in figure 24. The purpose of the player modules is to select an architecture and hardware configuration for playing a match. The human players do this via a graphical user interface while the computer opponents use some type of artificial intelligence. The ComputerManager module treats both types of players in the same way, letting them now when it is their turn to play and waiting for them to finish.

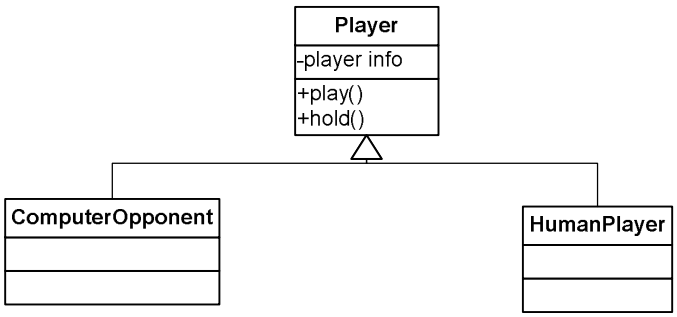

Figure 24: Player Types

As figure 24 shows a Player has two methods.

- ***play()***, tells the player that it his turn to play.
- ***hold()***, tells the player that he should wait until it his turn to play again.

All players have one describing attribute.

- ***Player info***, an object of the Player Info type described 5.3.1.

5.5.1 Human Player

When a human player has his turn to play he uses the User Interface to manage his team. He goes through the phases described in 4.3. The HumanPlayer object provides through the *player info* attribute the User Interface with all the information necessary for presenting the user with information about the league and the game.

5.5.2 Computer Opponent

The purpose of the Computer Opponent is to provide opposition for the player. I have from 4.2.1 that the computer controlled teams should be varied in how difficult they are to beat. To achieve this they must have some kind artificial intelligence that controls how they play and configures their architectures. The difficulty can be set by how good the artificial intelligence is or by outside parameters.

Outside Parameters
There are several restrictions or advantages that can be given to a computer opponent to make them easier or harder to beat, independent of how good their artificial intelligence is.

- *Hardware*, the hardware an opponent can choose influences how difficult he is to beat.
- *Architectures*, the types of architectures an opponent can use. Restricting an opponent to only use a few would make him easier to beat on certain workloads.
- *Penalty*, an opponent can be given a penalty to his simulation results, either in a positive or negative way to adjust difficulty.

Artificial Intelligence
The artificial intelligence of a computer opponent is the way he makes choices and manages his team. In the prototype I use predefined workload which makes it easier for a computer opponent to know what the best architectural configuration is. A best possible configuration for each workload can be determined and used as a basis for the way the computer opponents plays. The difficulty is then set by how far from that best configuration the opponent makes his choice. An easy to beat opponent would always be very far from the best configuration while a hard opponent could be pretty close or always correct if he should be impossible to beat.

Another approach is to just let the opponent make totally random choices within the outside parameters. These two ways of creating a computer opponent are just one of many ways of doing it and it is not within the scope of this project to develop a realistic computer opponent.

5.6 ComputerManager

This module is responsible for controlling the flow of the game, keeping track of the state of the game, organizing the league and updating the data-structure. Figure 25 shows the module and the modules it uses.

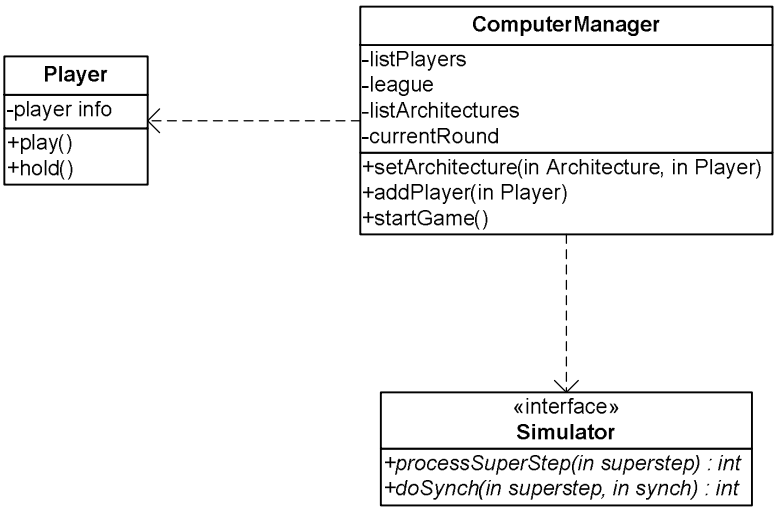

Figure 25: ComputerManager and depending modules

As figure 25 shows ComputerManager has the following attributes.

- ***listPlayers***, a list of all the players taking part in the game.
- ***league***, the datastructure for the league described in 5.3.1.
- ***listArchitectures***, a list of the architecture configuration for each player.
- ***currentRound***, the current round to be played.

As figure 25 shows ComputerManager has the following methods.

- ***setArchitecture(in Architecture, in Player)***, sets the Architecture for the given Player in listArchitectures.
- ***addPlayer(in Player)***, adds a Player to listPlayers.
- ***startGame()***, starts the game.

The flow of game is shown in figure 26 as a state diagram. Before the game-play starts there are two states that makes the necessary preparations for playing, adding players and setting up the game. When the teams are ready the game enters a cycle of managing teams and playing matches before finishing.

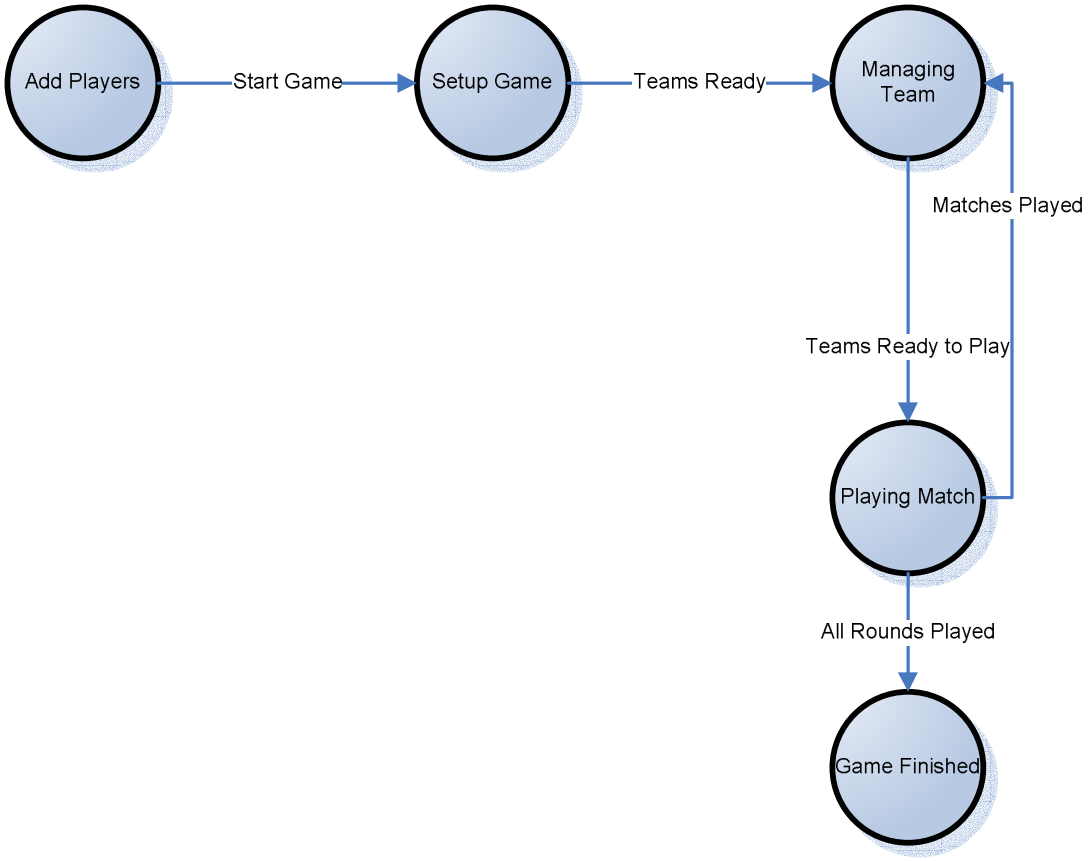

Figure 26: ComputerManager State Diagram

5.6.1 Add Players State

In this state the participating players are added to the game with the *addPlayer()* method and when all the players have been added the *startGame()* method is called to end the *Add Players* state and proceed to the *Setup Game* state. Figure 27 shows the sequence of method calls.

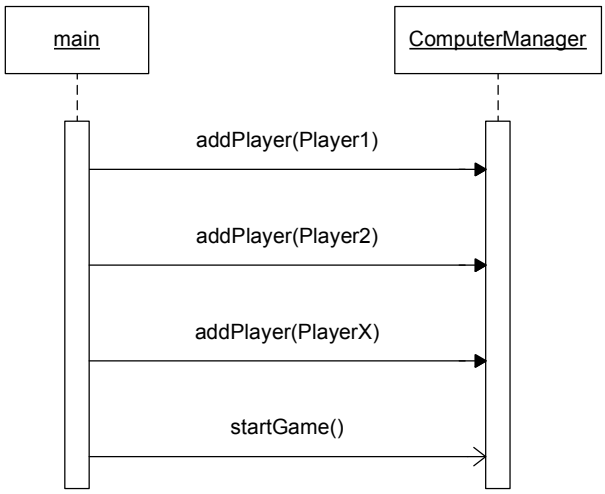

Figure 27: Sequence of method calls in Add Players

5.6.2 Setup Game State

In this state the league is set up with rounds and matches according to the structure described in 5.3.1. If all players should play a match each turn there must be 4, 8, 16 or 32 players in the league. The league structure is then assigned to the *league* attribute in ComputerManager and the *league* attribute in the Player Info object for each player. When the league, rounds and matches are set up the module proceeds to the *Managing Team* state

5.6.3 Managing Team State

In this state the list of players*, listPlayers* is traversed and each player is in turn told to play. The sequence of method calls is shown in figure 28. After all the players have set their architecture and been told to hold, the modules proceeds to the *Playing Match* state

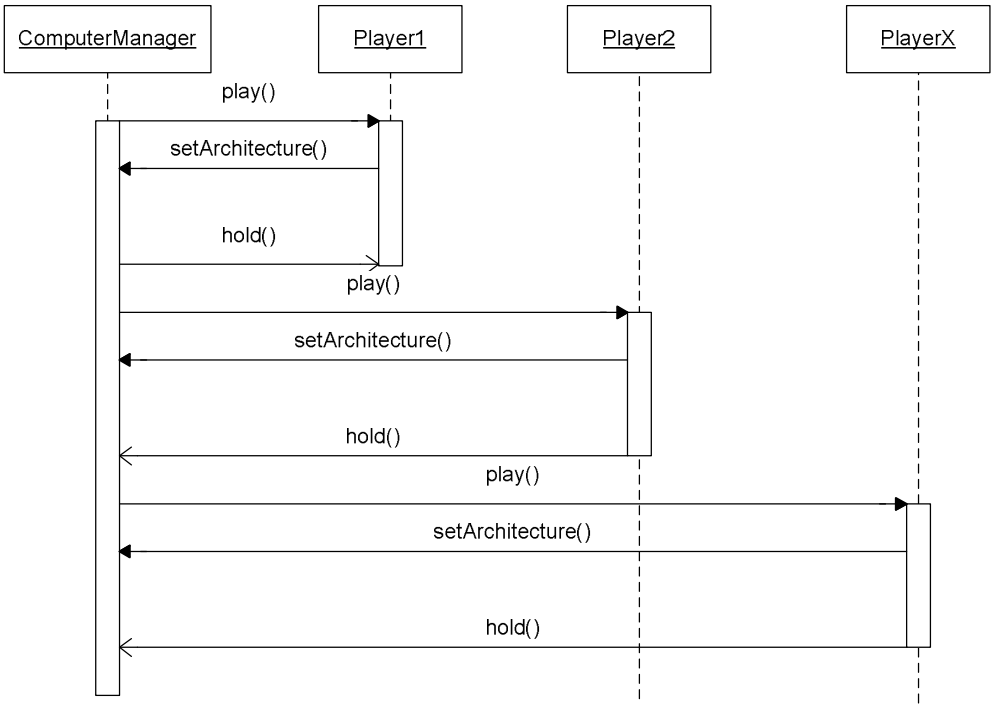

Figure 28: Sequence of method calls for Managing Team

5.6.4 Playing Match State

In this state the matches in the current round are run on the simulator and the results are stored in the corresponding Match Result from 5.3.1. The sequence of method calls shown in figure 29 is done for each super-step of a workload in a match. When all the matches in a round has been played the module either proceeds to the *Managing Team* state if there are any rounds left or to the *Game Finished* state if all rounds are played.

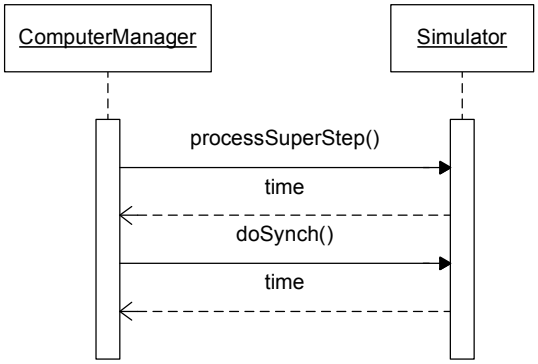

Figure 29: Sequence of method calls for simulation

5.6.5 Game Finished State

This is the end state of the game and module just waits for the program to shut down. The players are still able to review their team configuration and results.

5.7 User Interface

The purpose of the *User Interface* module is to provide the human player in the game with a graphical user interface where he goes through the phases described in 4.3. The two main phases where the player interacts with the game are:

- Manage Team
- Review Results

In the Manage Team phase the player configures the architecture according to the data structure described in 5.3.2. In the prototype there is no financial aspect in the game and thus the player has access to all hardware and architecture options from the beginning. In the Review Results phase the player is shown an overview of the information in the league data structure described in 5.3.1. All the information that needs to be presented to the user can be accessed through the Player Info object.

The graphical user interface implemented in the prototype is purely for testing purposes and I will not go into the details of how it was developed, but it was made using Jasc Paint Shop Pro [Corel 2005] and standard graphical libraries in Java. If someone is going to develop the game further I recommend using it purely as an example of how it could be done and design a new and better graphical user interface.

6. Prototype Documentation

Chapter 5 covered mostly the high level design of the prototype and I will in this chapter cover some of the details of the actual implementation in Java. I will not go into the details of the actual Java code, but rather give an overview of how the game is played and how to modify the prototype. There has not been any time for evaluation of the prototype or testing, but I give my own evaluation of what works and what doesn't work, based on my experience from implementing it.

The full Java code for the prototype is provided as appendix A. The code documentation is written as Javadoc [Sun: Javadoc 2005] and can be found in appendix B. How to run the prototype is explained in appendix C.

6.1 Playing the Game

To give an impression of how the prototype works without running it I have put together a selection of screenshots from it and a description of how things work.

Starting a game

The first screen the player meets is shown in figure 30. It has two buttons, New Game and Load Game. Pressing the New Game button starts a new game. The Load Game button is not implemented in the prototype but is intended to load a saved game state from a file.

Figure 30: Starting a Game

Creating a Player

After selecting New Game the player proceeds to the Create New Player window shown in figure 31, where he has to enter his name and the name he wants on his team. Pressing the Continue button starts the game.

Figure 31: Creating a Player

Managing the team

The main window in the prototype is the Manage Team window. Here the player can access information about the league and his team and manage hardware and architecture. As figure 32 shows there are several buttons but only some of them are working in the prototype. Pressing the Play Match button lets the game know he is ready to play a match.

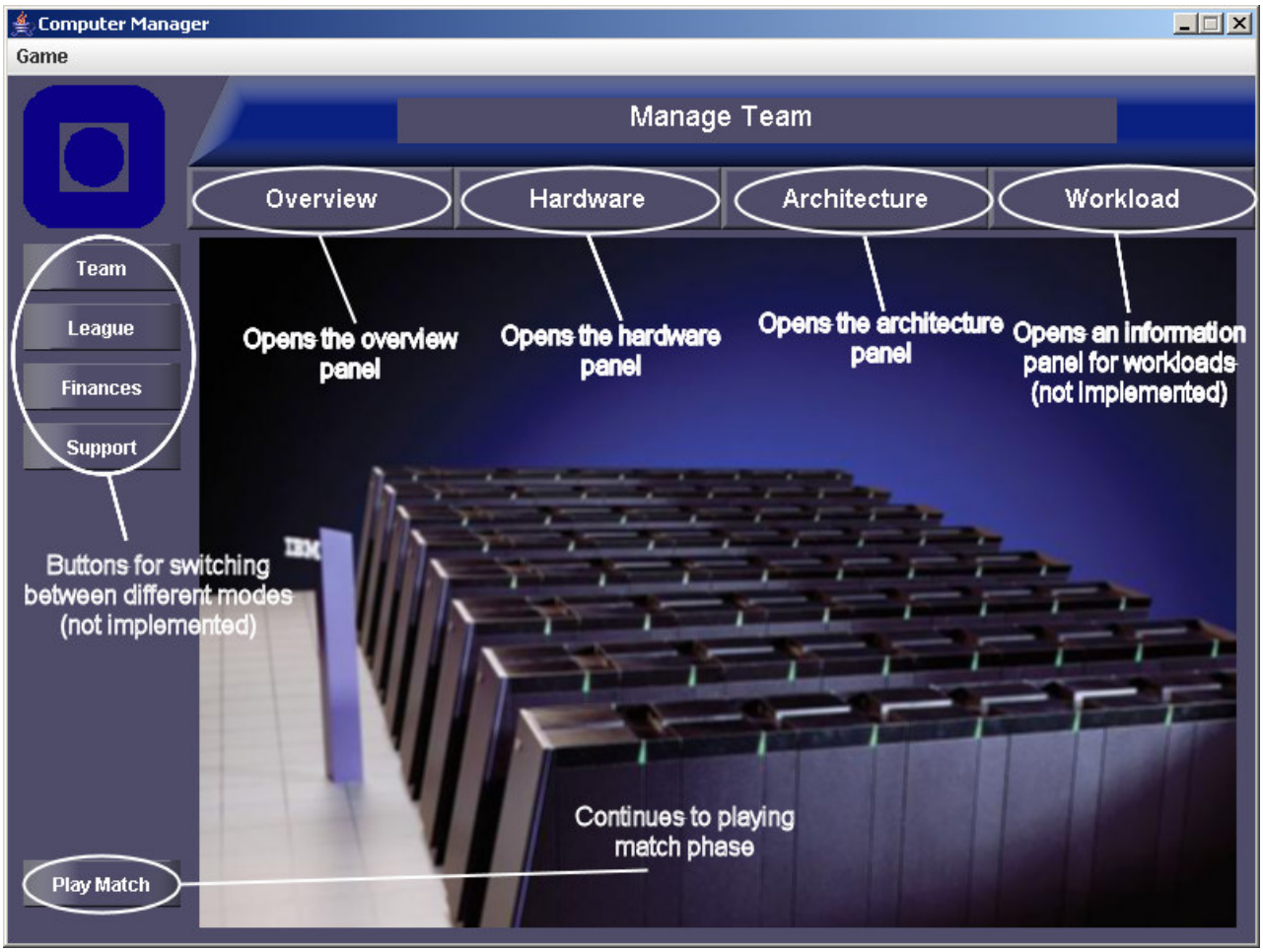

Figure 32: Manage Team

Pressing the Overview button opens the display of information shown in figure 33. Here the player gets an overview of his team, architecture and the league table.

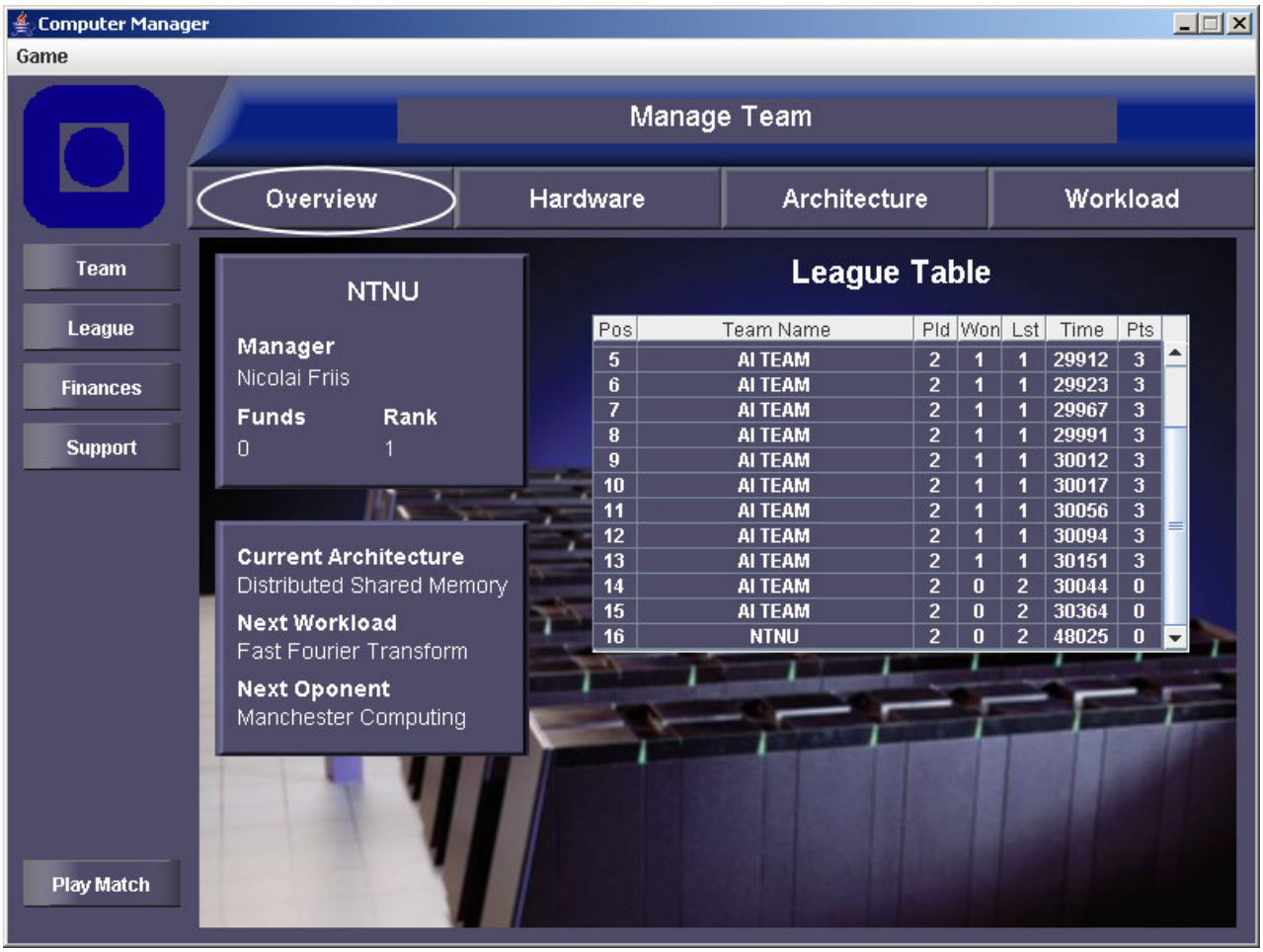

Figure 33: Overview of Team

Pressing the Hardware button opens the interface for selecting and setting parameters of processing elements and memory, shown in figure 34. The Undo button at the bottom restores the configuration to the last saved state and the Save button saves the currently selected configuration.

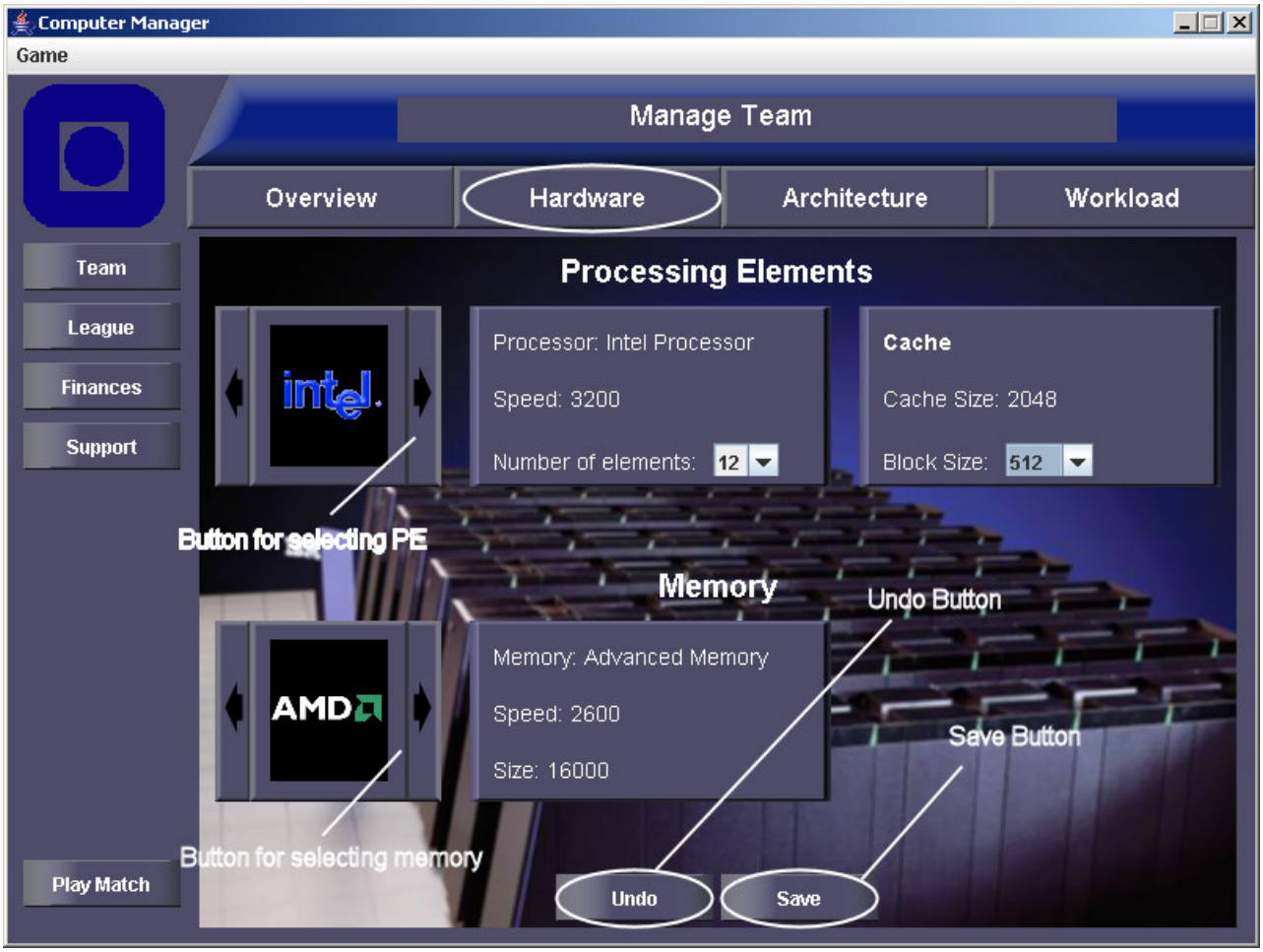

Figure 34: Hardware selection

Pressing the Architecture button opens the architecture configuration panel shown in figure 35. Here the player can choose type of architecture and set parameters for synchronization and interconnection.

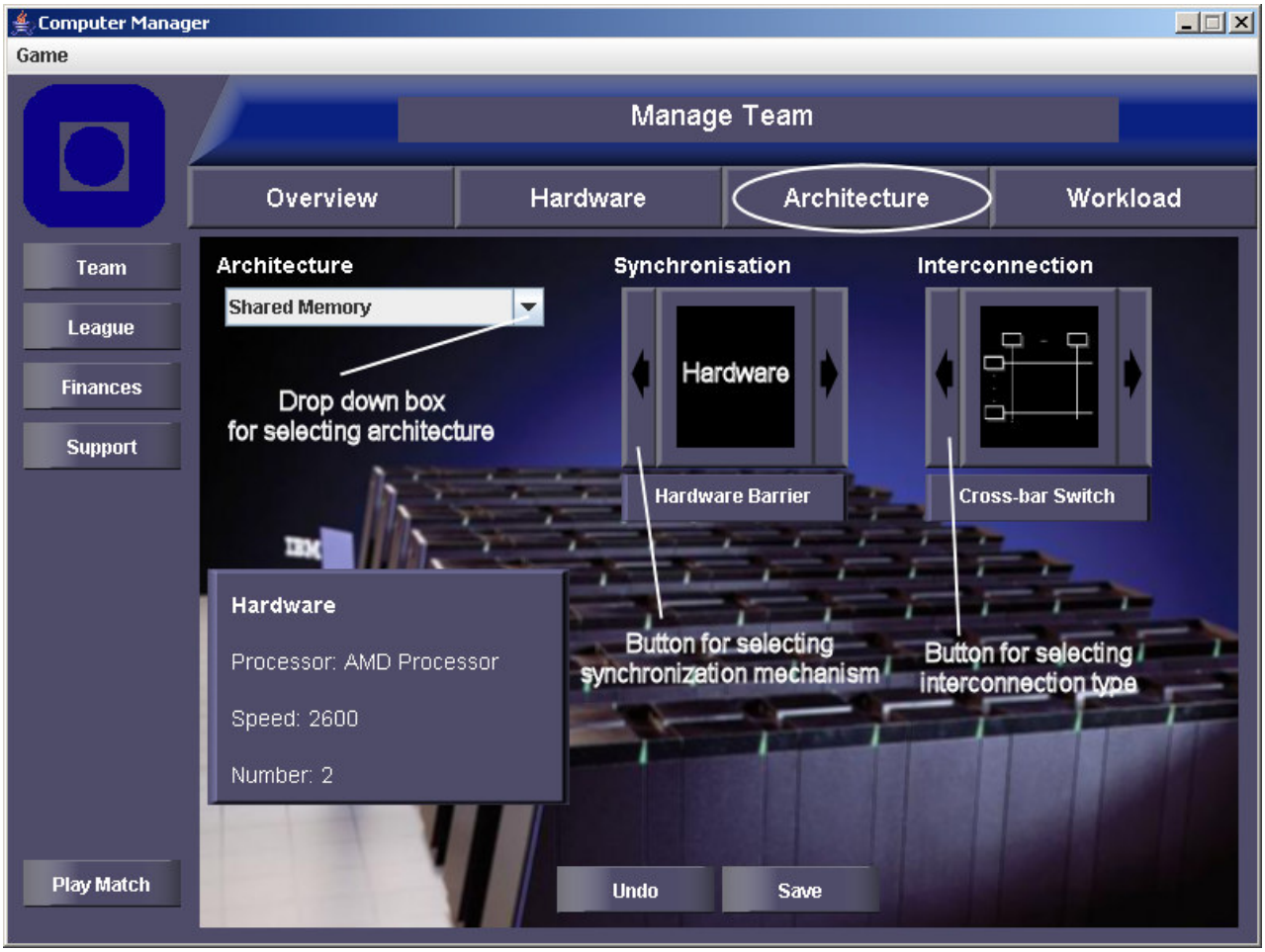

Figure 35: Architecture configuration

Results from Matches

After pressing the Play Match button the game plays the matches of the current round and presents the results to player, shown in figure 36.

Figure 36: Match Results

6.2 Modifying the Prototype

There are some parts of the prototype that can be modified without making direct changes to the Java source code. They are the text used on components like buttons, icons and other graphic images and descriptions and names of hardware and architecture components.

My recommendation for further work and modification of the prototype is covered in chapter 7 and I will not go into the details of how to make them.

6.2.1 Text

The language used in the implementation of the prototype is English. To make changing the language easier, all the text used for labels, buttons and menus are stored in one file, instead of being coded into each module. This file can either be modified or replaced by a new one with a different language.

The file to modify or replace is named *StringResources* and is located in the *gui.resources* package, see appendix A. It is written in Java and figure 37 shows the format of the text stored in it. It consists of a two dimensional array of strings, where the first string is the

identifier and the second one is the text to be displayed. After the file has been modified it must be compiled using a Java compiler.

```
static final Object[][] contents = {
        {"title", "Computer Manager"},

        {"Overview", "Overview"},
        {"Hardware", "Hardware"},
        {"Architecture", "Architecture"},
        {"Workload", "Workload"},
        {"Team", "Team"},
        {"League", "League"},
        {"Finances", "Finances"},
        {"Support", "Support"}

    };
```

Figure 37: Format of Language File

6.2.2 Icons and Images

All icons and images used in the prototype are stored in the *gui.images* package, see appendix A. They are stored as Portable Network Graphics [PNG 2005] files and can either be replaced or modified by a graphics editor. One important thing to notice when replacing files is that if they don't have the same size it might result in some unpredictable effects on the looks of the user interface. Table 7 shows a list of the files in the package with a description of what they are for.

Table 7: Image Files

2dmesh.png	Icon for 2D mesh topology
2dtorus.png	Icon for 2D torus topology
3dhypercube.png	Icon 3D hypercube topology
3dmesh.png	Icon for 3D mesh topology
background.png	The background image
backgroundsmall.png	Smaller background image used for the start new game window
blockpanel.png	Background of a block panel
bluebar.png	The blue top bar in the manage team window
bsp.png	Icon for BSP synchronization
crossbarswitch.png	Icon for cross-bar switch interconnection
empty.png	Default icon used when there is no other available
flatarrowleft.png	Icon for left selection arrow, not pressed
flatarrowleftsel.png	Icon for left selection arrow, pressed
flatarrowright.png	Icon for right selection arrow, not pressed
flatarrowrightsel.png	Icon for right selection arrow, pressed
hardware.png	Icon for hardware barrier

	synchronization
menubut.png	Icon for the overview, hardware, architecture and workload buttons, not pressed
menubutsel.png	Icon for the overview, hardware, architecture and workload buttons, pressed
mulitport.png	Icon for multi port interconnection
multibus.png	Icon for multi bus interconnection
multistagenet.png	Icon for multistage network interconnection
ntnulogo.png	The NTNU logo used in the top left corner
selpanel.png	Image used as background for button panels
sidebutton.png	Icon used on several types of buttons like the undo and saved buttons, not pressed
sidebuttonsel.png	Icon used on several types of buttons like the undo and saved buttons, pressed
sidepanel.png	Image used as background for button panels
singlebus.png	Icon for single bus interconnection
software.png	Icon used for software barrier synchronization
syslocalbus.png	Icon used for system and local bus interconnection

6.2.3 Components

All hardware and architecture components used in the prototype have an associated file with a textual description of the component and parameters.

Information Sheet

Each component has an information sheet that describes the component. The format of the information sheet file is a standard text file with information stored on a line by line basis, shown in figure 38. All the information sheet files are stored in *the gui.resources* package, see appendix A.

1. **#*Info*#**, defines that it is an information sheet file.
2. **#*Title*#**, defines that the next line is the title or name of the component.
3. The string for the title.
4. **#*Text*#**, defines that the next line is the description of the component.
5. The textual description of the component.
6. **#*Icon*#**, defines that the next line is the icon representing the component.
7. A string with the path to the icon for the component.
8. **#*Link*#**, defines that the next lines are links to related information, in the example in figure 38 there are no links.
9. **#*End*#**, defines that this is the end of the file.

```
1  #Info#
2  #Title#
3  Multiple Buses
4  #Text#
5  Several buses can be introduced to decrease
   contention. This enables several PEs to
   access several memory modules simultaneously.
   Many buses increase the wiring density,
   propagation delay and cost, so there is an
   upper limit on the usefulness of the
   extension.
6  #Icon#
7  gui/images/multibus.png
8  #Links#
9  #End#
```

Figure 38: Information sheet format

Processing Elements

All types of processing elements have their own configuration file in the *gui.resources.pe* package, see appendix A. The format is similar to the information sheet with a standard text file and information stored on lines, shown in figure 39.

1. ***#Speed#***, defines that the next line is the speed of the processor.
2. The speed of the processor.
3. ***#CostBuy#***, defines that the next line is the cost of buying this processor.
4. The cost of buying the processor.
5. ***#CostRent#***, defines that the next line is the cost of renting this processor.
6. The cost of renting this processor.
7. ***#CostCache#*** , defines that the next line is the cost of purchasing cache for this processor.
8. The cost of purchasing cache for this processor.
9. ***#Name#***, defines that the next line is the name of the processor.
10. The name of processor.
11. ***#Description#***, defines that the next line is a short description of the processor.
12. A short description of the processor.
13. ***#Infosheet#***, defines that the next line is the path to the information sheet for this component.
14. The path to the information sheet for this component.

```
1  #Speed#
2  3200
3  #CostBuy#
4  1400
5  #CostRent#
6  500
7  #CostCache#
8  5
9  #Name#
10 Intel Processor
11 #Description#
12 Intel Processor
13 #Infosheet#
14 gui/resources/pe/intelinfo.txt
```

Figure 39: Processing Elements File

The cost of buying and renting is included in the format of the file but the financial aspect of the game is not implemented in the prototype, so these fields and values are not being used.

Memory

All types of memory elements have their own configuration file in the *gui.resources.memory* package, see appendix A. The format is similar to the information sheet with a standard text file and information stored on lines, shown in figure 40.

1. ***#Speed#***, defines that the next line is the speed of the memory.
2. The speed of the memory.
3. ***#Size#***, defines that the next line is the size of the memory.
4. The size of the memory.
5. ***#CostBuy#***, defines that the next line is the cost of buying the memory.
6. The cost of buying the memory.
7. ***#CostRent#***, defines that the next line is the cost of renting the memory.
8. The cost of renting the memory.
9. ***#Name#***, defines that the next line is the name of the memory.
10. The name of the memory.
11. ***#Description#***, defines that the next line is a short description.
12. A short description.
13. ***#Icon#***, defines that the next line is the icon representing the memory.
14. Path to the icon for the memory.
15. ***#Infosheet#***, defines that the next line is the information sheet for the component.
16. Path to the information sheet for the component.

```
1  #Speed#
2  2600
3  #Size#
4  16000
5  #CostBuy#
6  2200
7  #CostRent#
8  1000
9  #Name#
10 Advanced Memory
11 #Description#
12 Advanced Memory
13 #Icon#
14 gui/resources/pe/amd.png
15 #Infosheet#
16 gui/resources/pe/amdinfo.txt
```

Figure 40: Memory Component File

The cost of buying and renting is included in the format of the file but the financial aspect of the game is not implemented in the prototype, so these fields and values are not being used.

6.3 Evaluation

It was not a part of the project to evaluate and test the prototype, but I will give some comments based on my own experiences from developing and implementing it. I will focus on the prototype itself and not the general conceptual ideas of the game. Hopefully my evaluation and comments might be of some use if someone is going to continue working on the simulation game.

I think the overall design of the prototype works well. It is divided into clearly defined modules that can be implemented separately and replaced or modified without affecting the others.

The graphical user interface I designed is purely for prototyping purposes and I do not recommend spending any time on developing it further. It serves as an example of how it can be done, but has too many bugs and design flaws. Creating a visual representation of match being played is one thing I think could work very well. It would make it more like a game and less like a simulator.

The implementation of the game logic in the prototype has certain limitations, like only being able to handle 4, 8, 16 or 32 players. It is also not possible to make modifications to the data structure in the game without having to change large parts of the game logic.

For evaluation purposes it might be worth considering implementing the financial aspect of the game. The prototype is designed so that it should be possible to add it without having to rewrite the entire game.

7. Further Work

One of the goals I defined for this project was to create a list of suggestions for further work with developing a game for use in learning computer architecture. I divide my suggestions into two types; the first is concerned with the general development of suitable game concepts for the area and the second focuses on the development of the Computer Manager idea and continuing on what I started with the prototype.

7.1 Developing New Concepts

The Computer Manager idea is not necessarily the best concept for a computer architecture learning game. There are other possibilities. I described two other ideas in chapter 3.2; Computer Tycoon and Computer City.

I have the following suggestions for further work on developing game concepts.

- Develop the Computer Tycoon and Computer City ideas further, with a prototype and a proper evaluation of both of them.

- Do a thorough evaluation of the use of simulation games for learning computer architecture, based on the experiences from the developed prototypes.

- Develop new conceptual ideas both for simulation games and other types of games based on the experiences from this project and other prototypes.

7.2 Further Development of the Prototype

The prototype is far from a complete game but it shows how some of the ideas in the Computer Manager concept works. Several parts of the game needs further work both with the conceptual design and on the implementation side. There is also a need to do a proper evaluation of the Computer Manager prototype and the design and idea behind it.

I have the following suggestions for areas of the Computer Manager game that needs further work.

- **Artificial Intelligence and Computer Opponents**
  The computer opponents in the game must have a realistic behavior to make the game challenging and fun to play. They can't just be simple algorithms that use the same configuration over and over again. Developing artificial intelligence that provides a challenge even to the most experienced computer architects is a task that requires a lot of work.

- **Workloads**
  In the prototype the workloads are predefined types of applications. It might be possible to develop a system for workloads that both provide variation and predictability for the player. Variation in the form that the player never has to play the exact same match twice and predictability so that he is able to do an analysis on how to best run workload. To achieve predictability a workload should not only be described by text but also by a set of parameters.

- **Content**
  The biggest challenge I faced in the development of the prototype was the lack of specified content, what the players should learn from the game. Computer architecture is a huge area and I think if the game is going work as a learning game someone with experience in teaching and computer architecture should specify exactly what the learning content of the game should be.

- **Content Management**
  A system for managing and modifying the content of the game would make it possible to use the game for other purposes than it was developed. It would also give the game a longer life cycle, if teachers could add and modify the content without having to rely on programmers.

- **Resources Management**
  The resource management aspect of the game is not implemented in the prototype but I think it is a very important part of the game. It adds an extra element to the game that standard simulators don't provide. The idea behind is not perfect either and should be developed further.

- **User interface**
  A good user interface is important, especially in game that is supposed to be fun. The user interface in the prototype is simple and gets the job done, but it is difficult to modify and develop further.

- **Multiplayer**
  An alternative to using computer opponents is a multiplayer mode, where only human players compete against each other. This is an idea that I think should be considered and developed further before focusing on computer opponents.

- **Simulator**
  The simulation part of the game needs a lot of work. As it is now it doesn't qualify to be called a simulation. One possible solution is to develop a way of connecting the game with BSPlab, but I think a simulator module designed specifically for the game could be better in many ways. It wouldn't have the same requirements for accuracy and would be easier to modify to fit new types of content.

8. Conclusion

This project started out with little more than some ideas of how a computer game could be made around simulation of computer architectures. My job was to take these ideas and develop them into something concrete; specific game concepts, a game design and a prototype. As the project evolved I discovered that developing a simulation game for use in learning required knowledge and skills from many different areas. I found that I had to narrow the focus down to only covering small parts of each area.

There is a wide range of different types of games and possibilities for making computer based learning games for computer architecture and one could spend lots of time and resources on researching and developing the ideas. There is however also a possibility that computer game based learning is not really suited for this purpose and this is something that always should be considered.

The results of the project are a classification of types of simulation games that can be used for creating a game of this type. I have developed conceptual ideas for some of them and designed a game called Computer Manager. The prototype is an implementation of some of the ideas and shows one way making the game. It will be up to others to decide if the project is worth developing further, but my own opinion is that a simulation game can be a valuable tool for learning about computer architecture.

9. References

Ahdell R Andresen G. (2001). "*Games and simulations in workplace eLearning*" Master Thesis, NTNU: Department of Industrial Economics and Technology Management

[Brackeen 2004] Brackeen, David (2003). *"Developing Games in Java"* New Riders

[Corel 2005] Corel (2005). Jasc Paint Shop Pro, Website: http://www.corel.com

[Culler, Singh 2003] Culler, David E., Singh Jaswinder Pal (1999). *"Parallel Computer Architecture, A Hardware/Software Approach"* Morgan Kaufman Publishers

[Desmo-J 2005] Desmo-J: A Framework for Discrete-Event Modelling and Simulation (2005). University of Hambur, Department of Computer Science, Website: http://asi-www.informatik.uni-hamburg.de/desmoj/

[Djupdal, Natvig 2004] Djupdal Asbjørn and Natvig Lasse (2004). "*Age of Computers II – An Improved System for Game Based Teaching"* Norsk Informatikkonferanse (NIKK) 2004

[Dybdahl, Uthus 1997] Dybdahl, Haakon, Uthus Ivan (1997). *"Simulation of the BSP model on different computer architectures"* NTNU Trondheim

[EA 2005] Electronic Arts (2005). Sim City, Website: http://www.simcity.com

[eclipse.org 2005] Eclipse.org (2005). Website: http://www.eclipse.org/

[Eidos 2005] Eidos (2005). Championship Manager 5, Website: http://www.championshipmanager.co.uk/

[Fishwick 1994] Fishwick Paul A. (1994). *"Simulation Model Design and Execution"* Prentice Hall

[Fishwick 1995] Fishwick Paul A. (1995). *"Computer Simulation: The Art and Science of Digital World Construction"* Internet Article: http://www.cis.ufl.edu/~fishwick/introsim/paper.html

[Friis, Tollefsrud 2004] Friis, Nicolai, Tollefsrud, John Ola (2004). "*Computer Game Based Learning, Studies and Analysis"* NTNU Trondheim.

[Hennessy, Patterson 2003] Hennessy, John L, Patterson, David A. (2003). *"Computer Architecture, A Quantitative Approach"* Morgan Kaufman Publishers

[Lucas Art 2005] Lucas Art (2005). X-Wing Alliance, Website: http://www.lucasarts.com/products/alliance/default.htm

[Macromedia 2005] Macromedia (2005). Flash Platform, Website: http://www.macromedia.com/

[MathWorks 2005] The MathWorks (2005). Website: http://www.mathworks.com/

[Microsoft 2005]). Microsoft (2005). Microsoft Flight Simulator, Website: http://www.microsoft.com/games/flightsimulator/

Misra, Jayadev (1986). *"Distributed discrete-event simulation"* ACM Computing Surveys, Volume 18, Issue 1.

Nareyek, Alexander (2004). *"AI in Computer Games"* Queue, Volume 1, Issue 10.

[Natvig 2005] Dr.Ing. Lasse Natvig, Professor in computer architecture at Group for computer Architecture and Design, Department of Computer and Information Science.

[Natvig, Line 2004] Natvig Lasse and Line Steinar (2004). "*Age of Computers – Game-Based Teaching of Computer Fundamentals"* ITiCSE '04 June 28-30, 2004, Leeds, UK.

Perry, Deborah (2003). "What Makes Learning Fun" Internet article: http://www.selindaresearch.com/learning.htm

[Pizza 2005] Pizza Tycoon (2005). Website: http://www.pizzatycoon.org/

[PNG 2005] Portabel Network Graphics (2005). Website: http://www.libpng.org/pub/png/

[Prensky 2001] Prensky, Marc (2001). "*Digital game based learning*" New York McGrawHill

[Simics 2002] Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B. (2002). "*Simics: A full system simulation platform"* Computer, Volume 35, Issue 2.

[Sirer, Walsh 2004] Sirer, Emin Gün, Walsh, Kevin (2004). *"Staged simulation: A general technique for improving simulation scale and performance"* ACM Transactions on Modeling and Computer Simulation, Volume 14, Issue 2.

[Sony 2005] Sony Computer Entertainment (2005). Gran Turismo 4: The Real Driving Simulator, Website: http://www.us.playstation.com/Content/OGS/SCUS-97328/Site/

[Sun 2005] Sun Microsystems (2005). Java 2 Platform, Standard Edition, Website http://java.sun.com/

[Sun: Javadoc 2005] Sun Microsystems (2005). Javadoc Tool, Website: http://java.sun.com/j2se/javadoc/

Supnik, Bob (2004). *"Simulators: Virtual Machines of the Past (and the Future)"* Queue, Volume 2, Issue 5.

[SweLL 2005] The Swedish Learning Lab (2005). Website: http://www.swedishlearninglab.org

[Take 2 2005] Take 2 Interactive Software (2005). Railroad Tycoon 3, Website: http://www.railroadtycoon3.com/

[Ubi 2005] Ubi Soft Entertainment (2005). Panzer General, Website: http://panzergeneral3.com/

[Valiant 1990] Valiant Leslie (1990). *"A Bridging Model for Parallel Computation",* CACM Volume 33, Issue 8.

Appendix A: Java source code

A zip file containing the source code, run able version of the prototype and javadoc has been delivered to Lasse Natvig and can be obtained by contacting him.

The directory *sourcecode/ComputerManager* contains the Java code files for the project and configuration files for components.

The source code is divided into the following packages:

- *architecture* – simulation and architecture configuration files
- *datastructure* – the implemented data structure
- *defaultsimulator* – a simple implementation of a simulator module
- *gamecore* – the main game modules
- *gui* – the graphical user interface
    - *gui.createplayer* – the components used for creating a new player
    - *gui.images* – the images used by the interface
    - *gui.initscreen* – the components used in the start up screen
    - *gui.manageaction* – actions used by the manage team interface
    - *gui.manageteam* – the components used for managing the team
    - *gui.playmatch* – the components used for playing a match and displaying results
    - *gui.resources* – text resources used by the interface

Appendix B: Code Documentation

A zip file containing the source code, run able version of the prototype and javadoc has been delivered to Lasse Natvig and can be obtained by contacting him.

The directory *javadoc* the generated javadoc files. They can be opened with the *javadoc/index.html* file.

Appendix C: Running the Prototype

A zip file containing the source code, run able version of the prototype and javadoc has been delivered to Lasse Natvig and can be obtained by contacting him.

The directory *compiled/ComputerManager* contains a run able version of the prototype. To run the prototype start the file *compiled/ComputerManager/start.bat.* It requires that the Java run time environment is installed on the computer.