Data Mining and Machine Learning with Computer Game Logs

Stefan Wender
UPI: swen011, ID: 4685895
CompSci 780 Project Report

University of Auckland

Supervisor: Prof. Dr. Ian Watson
Department of Computer Science
University of Auckland, New Zealand

October 2007

Abstract

This report describes the process of analysing a set of server logs which were recorded during games of the team based first-person shooter Team Fortress Classes (TFC). The aim of the analysis is the discovery of patterns in the data set through the application of machine learning algorithms. These patterns could then be used to improve the playing experience for human players or to create artifical players which show the behavior that is given in the patterns and thus appear more human-like.
The report looks at the historical background of machine learning for classic games such as chess and checkers. It then surveys the evolution of computer game AI from its early roots in arcade games in the 1970s to the multi-million dollar productions of the last decade. In the next chapter there is an introduction into computer game genres, goals and limitations of machine learning in those genres as well as an explanation for the principle of online/offline learning to give a better understanding of machine learning in computer games.
Then a short introduction to the game from which the data comes is given to facilitate an understanding of certain aspects of the data set. Afterwards the course of action to prepare the data for the actual analysis is described. This consists of the extraction of the information from HTML files and the cleaning of this information. The database in which this information is stored is described. Then the actual analysis consisting of two main experiments is elaborated. Each of these experiments is subdiveded into the definition of the test data set on which the analysis will be performed, a preliminary analysis which specifies the statistical characteristics of the data set on which the experiment is based and the application of the machine learning algorithms as well as the discussion of the results. After the evaluation a discussion of the outcome and the description of possible future work follows. At the end the conclusion sums up the achievements and results of the project.

Contents

List of Tables

List of Figures

1 Introduction

Computer games are certainly a market of the future and thus also very interesting for research. Technical advances in the hardware market for personal computers are largely driven by the demand of gamers which require ever more graphics power and CPU power (Shao, 2007). Large multinational corporations like Vivendi Universal rely on the success of their gaming division to boost their billion-dollar balance (Times, 2007). And while computer graphics mature more and more towards photo-realism, computer gamers look again for something which distinguishes great games from the mass. Artificial intelligence is supposed to be this next big advancement in computer games. While there has been a significant improvement in the level of intelligence of computer controlled agents, there is still a big gap between the research that is done in this area and techniques which are used in commercial products. This becomes very obvious in the area of AI learning. This subject is one of the most important in computer game AI since learning AI allows not only for actions and events that are predefined by the programmer, but also for actions that are devised by the learning algorithms of the agent. This leads to more adaptable and thus more realistic game play. Significant progress has been made in the area of computer game AI learning in the last decade. But most of the techniques that are used in commercial products are still not genuine learning algorithms, but just algorithms devised to make human players believe that the computer is in fact able to adapt.

One way to apply machine learning to create more realistic gameplay is through the evaluation of games played by human players. The information gained through these games can be used to create computer players that imitate humans and anticipate their behavior. The biggest problem is the acquisition of enough realistic data to extract this behavior. To be able to predict a broad set of behaviors a huge amount of data has to be available that shows realistic games, i.e. not games played in a supervised enviroment. Such a data set hardly ever exists in adequate size and condition, especially in a rather recently discovered research area such as computer games. Therefore we were quite lucky to obtain the large and comprehensive data set on TFC games which is used in this project.

2 Historical Background

This chapter gives a brief overview of the background of machine learning in games. Its first part is a short summary of the developement of AI in classical (board) games which is mostly based on surveys by Fuernkranz and Schaeffer (Schaeffer, 2000)(Fuernkranz, 2001) and summarizes the research done for some of the most influential 'classic' games for AI research. The second part is about the evolution of artificial intelligence in computer games, mostly in commercial games. It is mainly based on (Tozour, 2002) and also briefly touches on the differences between academic research and commercial applications.

2.1 Machine Learning in Games

One of the first to pursue the application of machine learning to game AI was Arthur L. Samuel who in 1947 came up with the idea of creating an a program that could play checkers. The two main papers describing his research, which lasted over the next three decades, are landmark papers for AI in general and for learning AI in particular, since they introduce several techniques and ideas for AI learning which are still in use today in one way or another (Samuel, 1959) (Samuel, 1967). His program was based on the principle of learning the game without previously defined moves. This is in contrast to what became the common approach not only for checkers but also for chess and most other classical games where the computer programs have a certain amount of predefined game states which they search for a solution. In the late 1970s Samuels program was defeated by a checkers program which was developed at the Duke University (Truscott, 1978). This victory led to the assumption that the new program could match just about any human player. The assumption was indirectly proven wrong with the development one of the most advanced checkers programs of the early 90s, *Chinook*, at the University of Alberta. Chinook is based on the very common principles of an opening book, an endgame database, previous knowledge and extensive search through possible moves (Schaeffer et al., 1992). Its evaluation function is not based on learning but has been tuned manually. Chinook won the 'world man-machine championship' in 1994 against the world champion checkers player Marion Tinsley and did not lose any game in the next years before retiring in 1997. In 2007 the developers stated that they computed the weak solution for checkers and Chinook thus cannot lose anymore (Schaeffer, 2007).

The history of computer AI playing checkers shows that even though Samuel started with a machine learning approach, research soon switched to optimizing brute force searching programs which went on to become more effective and eventually unbeatable for human players. Despite this, there are also approaches that use machine learning algorithms to play checkers. Chellapilla and Fogel for example created a checkers program in the late 1990s, which did not use expert domain knowledge but learned playing the game through co-evolution (Chellapilla and Fogel, 1999).

One of the most researched areas in computer AI is the game of chess. Among the first papers to be published on the topic of computer chess was a paper by Shannon (Shannon, 1950) which split algorithms applied to this problem into two types A and B. While both types are basically based on searching for possible moves, Type A algorithms do this by brute-force while Type B includes selective search much like human chess players tend to think. Subsequently Type A algorithms gained popularity due to being easier to implement and to debug. Brute-force programs have managed to beat human grand masters for quite some time now, most well known is the series of duels between then world champion Gary Kasparov and chess programs made by IBMs Deep Blue team (*ChipTest*, *Deep Thought* and the famous *Deep Blue*) (Campbell et al., 2002).

But similar to checkers, considerable research has been done in the area of machine learning for chess. Probably the area most thoroughly studied is the induction of chess, i.e. the classification into won/not won from a given endgame. Quinlan (Quinlan, 1983) for example used the decision tree learning algorithm ID3 to acquire recognition rules. Muggleton (Muggleton, 1990) applied DUCE, a machine learning algorithm that suggests high-level concepts to the user. The suggestions are based on recognized patterns in the rule database and the technique reduces the complexity of the rule base and generates concepts that are meaningful for domain-experts. Another machine learning technique which gained particular interest during the late 1980s is explanation-based learning (Mitchell et al., 1986). This approach was based on the assumption, that the domain theory can be utilized to find an explanation for a certain example which then can be generalized. During the mid-90s a related approach called case-based reasoning (CBR) (Kolodner, 1992) was developed and applied to computer chess. Case-based reasoning is a technique which is quite similar to explanation-based learning and in principle 'Learning by Analogy' (Campbell et al., 2002). In order to solve a problem a CBR system looks for previously encountered similar problems in its case base. It then tries to adjust the solution to these known problems to fit the current one. The problem-solution pair it acquires in this way is then used to extend the existing case base. MAPLE (Spohrer, 1985) is an early system that is based on CBR and learns whenever it makes a fatal mistake, i.e. when it reaches a point where all moves lead to a loss. CASTLE (Krulwich, 1993) is a modular (threat detection module and counterplanning module) system that also learns

through mistakes. Whenever one of its components fails, it performs a self-diagnosis to find out which module failed and then uses explanation-based learning to extend its case base by the missing case. Both CASTLE and MAPLE rely heavily on case-based planning (Hammond, 1989) which itself is based on explanation-based learning.

Another possibility to apply machine learning to computer chess programs is the design of the evaluation function, i.e. the method that evaluates how good a certain move or a certain sequence of moves is. The tuning of this function has actually become the most promising direction for the application of machine learning to computer chess (Campbell et al., 2002). Tunstall-Pedoe (Tunstall-Pedoe, 1991) used a genetic algorithm (GA) to optimize the evaluation function. The fitness of a certain parameter was determined by comparing the result of the function with the moves of a grandmaster. Van Tiggelen came to the conclusion that genetic algorithms are too inefficient for usage in a middle-game application and therefore used an artificial neural network (ANN or NN) instead (van Tiggelen H. J. and van den Herik, 1991). According to the authors this resulted in a more efficient and at the same time more accurate program. Schmidt was not satisfied with the results he got when he used a neural network and used temporal difference learning (TD learning) instead (Schmidt, 1994). The idea of temporal difference learning had already been developed by Samuel for his checkers player (Samuel, 1959) but not really been in use again until Tesauro achieved amazing results with this in his backgammon program (Tesauro, 1992). In chess it is quite hard to evaluate effects of decisions made in the middle-game since the game is only decided at the end. TD learning addresses this problem by trying to minimize the differences between successive position evaluations. This means for example that if the program discovers after a series of evaluations that the assumed outcome is wrong, a previous assumption must have been wrong and the weights of the function have to be changed accordingly.

While checkers and chess have received tremendous attention and research, there are lots of other games for which computer players have been researched using plenty of different techniques. The aforementioned Tesauro created a backgammon player *TD-Gammon* (Tesauro, 1992) capable of beating human players. TD-Gammon is based on neural networks which are trained using TD learning. *Bill* is an Othello program written by Kai-Fu Lee and Sanjoy Mahajan that was among the best during the early 1990s. Besides using deep search and extensive domain knowledge it uses Bayesian learning in its evaluation function (Lee and Mahajan, 1990). The game of poker offers several attributes that make it very interesting for AI research: it offers incomplete knowledge due to the hidden cards, it is played by several agents and it uses concepts such as agent modeling and deception. Therefore a lot of interesting research in the area of machine learning for poker has been produced. *Loki*, a program developed at the University of Alberta, uses explicit learning by observing its opponents and constructing models of these opponents. It then adapts to the play of these opponents (Scha-

effer and et al., 1999). Korb et al. produced a poker program that is based on Bayesian networks (Korb and et al., 1999). The poker program of Dahl (2001) uses reinforcement learning (Russell and Norvig, 2003) to play the poker variation Texas Hold'em. Rubin and Watson (2007) use case-based reasoning for their poker program *CASPER* which plays Texas Hold'em evenly against strong, adaptive competition. Other games that are popular with AI researchers include Go (Mueller, 2000), which has an even larger search space than chess, Scrabble (Sheppard, 2002) and Nine-Men-Morris (Gasser, 1996).

2.2 The Evolution of Computer Game AI

Since the mid 1970s, computer games have been developed that allow a single player to compete with the program itself. Seminal games like *Pac-Man*, *Space Invaders* or *Donkey Kong* used, mostly due to restrictions of available resources, very simple techniques such as finite-state machines, decision trees and production rule systems together with some random decision-making to add less predictable behavior. But while processing power increased in the following decade and games grew ever more complex and better-looking, the AI techniques remained by and large the same. Only in the 1990s more complex techniques were used. One reason for this was the success of strategy games such as MicroProse's *Civilization* or Blizzard's *WarCraft II* , since these games require AI as part of the central playing experience. Besides that strategy games require a range of different AI techniques for unit-level AI as well as for overall strategic and tactical AI. First Person Shooters (FPS) are another game genre which led to the introduction of more complex AI into commercial games. While there have been games which belong to this genre since the early 1990s, they mostly tried to challenge the player by the sheer number of opponents or the amount of firepower he was facing. Significant progress was made in Valve's *Half-Life* which was praised for its tactical AI and EpicGames' *Unreal: Tournament* which included bots that showed tactical behavior and scalability. One computer games genre which is practically based on AI is that of sim games/artificial life (A-Life) games.

Maxis' *SimCity* was one of the first games in this genre. Especially noteworthy for the complexity of its agents is *The Sims* which uses fuzzy state machines and A-Life technologies. The games of the *Creatures* series are basically A-Life simulators which make use of a whole range of AI techniques to simulate the evolution of the 'Norns' that populate the game. More recently Lionhead Studio's *Black&White* games are built around a complex reinforcement learning approach that allows the player to train a 'creature'. These games show one of the most advanced game AIs to date.

While these games do use advanced AI techniques, the most common technique still remains the simpler and thus easier to create and debug rule-based systems. This gap between

academic research and the computer gaming industry has not gone unnoticed. Laird states in his paper (Laird and van Lent, 2001) that while there is significant room for academic research in computer games, the computer games industry tends to go into its own direction. He then concludes that it is up to academic researchers to close this gap by using computer games as test beds to develop AI methodologies which then can be used in commercial games. According to Nareyek, the academic community has so far failed to achieve this (Nareyek, 2004) (Nareyek, 2007). He states that apart from the usage of a few common techniques, in particulary the A* algorithm for path finding, usually no academic research is used in commercial products. He also claims that common academic research goes into a direction that is all but uninteresting to the computer games industry.

On the other hand there have been quite a few attempts to bridge this gap between academia and the industry. Champandard (2003) uses the open-source framework FEAR (Flexible Embodied Animat Rchitecture) together with the commercial First Person Shooter *Quake 2* to evaluate several AI techniques proposed by the academic community. He succeeds at implementing these various techniques for different tasks of a non-player character (NPC), a so-called bot. Gold (2005) does research using a commercial game engine as test bed to develop a prototype game. He reaches the conclusion that it is indeed possible to mix commercial game development and research. Miikkulainen et al. (2006) argue that the AI techniques which are usually applied in modern games are in fact not appropriate at all for the purposes they are used for. On the other hand machine learning techniques which are not used at all in commercial products such as neuroevolution (neural networks in combination with genetic algorithms) are particularly well suited for computer games.

2.3 Log Evaluation in Computer Game Research

Data mining is used in many commercial applications and research about data mining techniques is quite common. Despite this the usage of data mining techniques to search for patterns in computer game logs is very rare. This is amazing since it allows for a number of possible benefits even for commercial games as mentioned in (Kennerly, 2007). The benefits that can be gained through the discovery of patterns in computer game usage data include the discovery of possible game improvements which in turn can result in increased player satisfaction. Patterns can also show ways to generate more challenging computer players by generating AI that behaves more like humans. This is for example the motivation behind the RoboCup Coach Competition (Kuhlmann et al., 2006). For this competition coach programs are created which control their own football team and create a model of the opposing team through extracting behavior patterns from performance logs of the opponents. The extracted information can then be used against the opponent. The success of a coach program in this competition is determined by its ability to extract such patterns from the performance logs of the opposing team. In (Tveit and Tveit, 2002) the authors suggest to adapt the concept of web usage mining to the new domain of mining game data logs. The proposed concept is mainly aimed at mining data from Massivly Multiplyer Online Games (MMOGs) but could be adapted to accommodate for other types of games as well. The University of Alberta has a project which tries to make use of logs from the FPS game *Counter-Strike* which is based on *Half-Life*. Their aim is to create better AI using these logs. The recordings were generated by their own program during the observation of matches of very good human players against other human players (CSAI, 2007).

3 Aims, Areas of Application and Limitations of Machine Learning in Computer Games

This chapter gives a short overview over the basic aims and limitations of computer AI learning.

3.1 Aims of Computer Game AI Learning

This subsection is concerned with the aims of computer game AI learning, i.e. which benefits result from applying machine learning techniques to computer games in the first place. It is largely based on (Spronck, 2005). Adaptive AI makes the game 'self-corrective', allowing it to rectify mistakes that were made during programming. This can be both offline, i.e. during the creation of the game or online while playing the game. This can not only solve mistakes which have been made involuntarily by the creators, but can also be used to cope with so called 'exploits'. Exploits are not actual mistakes in the program but merely unforeseen behavior by the player which leads to a significant advantage. If the AI is adaptive it can develop a new behavior to counter the strategy of the player. This can only happen online. Another positive aspect of using machine learning AI is the possibility to generate programs with automatically scaled degrees of difficulty. The AI can adjust itself to the capabilities of the player, thus enhancing the overall playing experience. The playing experience is of course also greatly improved through adaptive AI because the player perceives non-static behavior as much more realistic.

3.2 Possible Areas of Application and Limitations

This subsection is concerned with the types of games that learning AI can be used for. The subchapter also describes constraints that limit the usage of learning AI.

Laird and van Lent describe in their landmark paper about the usage of computer games as AI research application area (Laird and van Lent, 2001) different genres of computer games in which adaptive AI can be used (Figure 3.2). They also describe the roles which AI can play in these different types of games as well as characterizing limitations to the usage of AI in computer games. Kirby (Kirby, 2003) also addresses these limitations but considers them

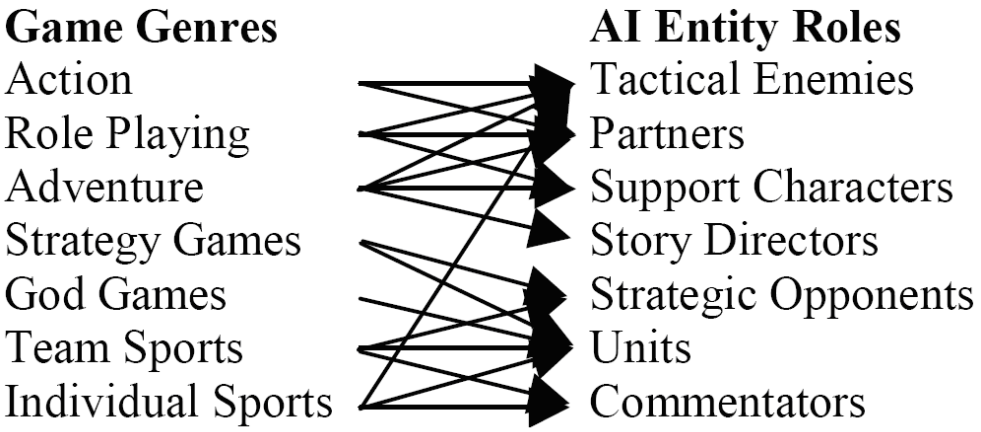

Figure 3.1: AI roles in different game genres (Laird and van Lent, 2001)

regarding machine learning algorithms. He describes four basic criteria, mostly regarding resource demands by the learning AI, which have to be fulfilled to make the application of machine learning algorithms to a particular area useful.

1. The AI must be able to read the input to be learned through usage of a reasonable amount of resources.

2. The AI must be able to store learned knowledge through usage of a reasonable amount of resources.

3. The AI must be able to use the learned knowledge through usage of a reasonable amount of resources.

4. The application of machine learning at this particular point results in an improvement for the game playing experience.

Kirby then goes on to describe certain methods to assure these conditions are met. To assure that the amount of resources used to read the input is reasonable, an appropriate knowledge representation has to be chosen, possibly through preprocessing input signals. To gain the correct knowledge from a certain input, certain problems have to be taken into account, such as associations between past and present inputs or differences in the level of difficulty of input data (Kirby, 2003).

4 The Data

This chapter covers topics related to the data which is used in the experiments. These topics include a description of *Team Fortress Classic*, the game from which the log files are recorded with explanations what the attributes of the data set actually mean. It furthermore contains a description of how the data was extracted from the web files it was stored in as well as a depiction of the database it was moved into.

4.1 Team Fortess Classic (TFC)

This section contains a short description of the game, its classes and weapons as well as important game mechanics and is mostly based on (Half-Life, 2007) and (Wikipedia, 2007).

4.1.1 The Game

TFC is a team-based online multiplayer first-person shooter in which teams of players compete in different scenarios against other teams. It was first developed as a standalone game but later converted to be released as a modification (mod) of the game *Half-Life*. A mod is an extension to an already existing game which uses the underlying architecture of that game. The mod can also use parts of other resources from that game such as textures, sounds or scripts but can also be a completely new game.

4.1.2 Gameplay

TFC offers a range of different scenarios in which teams can compete against each other. The data that is used for the experiments only comes from *Capture-The-Flag* (CTF) scenarios. In these scenarios the players are split into two equal teams. Both teams have a base in which a flag is stored at a certain place. The aim for both teams is to capture the flag of the opposing team and return it to their own flag. This can be done by picking it up through simply running over the place where the enemy flag is displayed and dropping it of at your own flag through simply running across that location. A capture is awarded with 10 points and is on most maps the only way of gaining points in the game. If the flag carrier is killed while on the way to his own flag, he drops the flag. The flag can then be picked up by one of the killed player's team members who can then try to capture it or the flag can automatically

be returned to its original base by an enemy through just walking across it. If a player is killed they are resurrected ('respawn') in their own base after a very short while, usually a few seconds.
Since the enemy flag can only be captured by a team if their own flag is at their base, it is important for a team to not only focus on taking the opposing team's flag but also on defending their own flag. Therefore an effective mix of attacking and defending classes has to be used by the teams. Every game lasts for a predefined amount of time. The team which scores the most points during that time wins the game.

4.1.3 Classes

At the beginning of every game each player has to pick a class, i.e. the type of his avatar. A change of class later in the game is possible but has the same effect as being killed, i.e. a short waiting time before being resurrected in the base. TFC offers nine different classes with different equipment, weapons, armor, speed, and special abilities. These differences account for the fact that no class is stronger than all the other classes but every class has its areas of application. The best team is thus defined by a multifaceted combination of classes.

Figure 4.1: The different classes in TFC: From top left: Demoman, Engineer, HWGuy, Medic and Pyro From bottom left: Scout, Sniper, Soldier and Spy

Demoman

The Demoman has average speed, health and armor and is best at close quarters combat. He has special explosive weapons which can deal a great amount of damage over a short period of time and thus are valuable for attacking as well as for defending. On a few maps a Demoman is required to access certain areas of the map through opening the entrance by explosives. The Demoman's main weakness is against long range weapons.

Engineer

The Engineer is able to build a *sentry gun*, a stationary gun tower which usually forms the backbone of the defense of a team. He can upgrade the gun twice and thus improve firepower and speed. The Engineer is also able to build ammunition dispensers as well as teleporters which allow team members to be teleported from a defined entry to a defined exit. Apart from that the Engineer is mainly used as a supporter.

Heavy Weapons Guy

The Heavy Weapons Guy (HWGuy or HWG) is extremely well armed and has the best armor and health in the game. His drawback is that he is very slow and thus not very flexible. His main machine gun takes a short time to become active but is very efficient once it starts firing.

Medic

The Medic is able to heal team members' hit points as well as infections and is also able to infect enemy players. The Medic is the second fastest class in the game, very mobile and serves well as offense. He is also able to do *conc* jumps, i.e. use his own grenades to propel him through the air enabling very long jumps.

Pyro

The Pyro is by design a class with medium speed, armor and health which uses its weapons to set enemy players on fire. This causes damage over time and distracts since the flames are visible on the inflamed player's screen. Since the Pyro is seen as one of the weakest classes in the game it is hardly ever used, especially in more competitive games or league games.

Scout

Scouts are the fastest class in the game and thus the foremost class that is used for capturing the flag. They have few hit points and weak armament and thus usually prefer running away to fighting. Scouts, just as Medics, possess the ability to use their grenades to *conc* jump.

Sniper

Snipers employ a long range rifle as their primary weapon. This rifle can only be used when they are standing completely still and will show a red laser dot while aiming. This dot can of course give away the Sniper's position but the rifle also does extremely high damage. Another downside of the rifle is that it takes a few seconds to aim. The Sniper moves very slow and is thus not very useful for capturing flags or changing the position quickly.

Soldier

The Soldier has not special skills but combines high mobility, good armor and good health with excellent armament. Its most powerful weapons are the rocket launcher and the nail grenade which can be used to deal damage to a large area.

Spy

The Spy possesses the unique ability to disguise himself as a member of the enemy team. This allows him to get behind the enemy lines and steal the flag or gather intelligence. However if he attacks or picks up the enemy flag he is uncovered. Scouts and Spys also possess the ability to uncover enemy Spys by simply touching them. The Spy has only average speed, low health and his main attack weapon is his knife which allows him to nearly instantly kill an enemy player.

4.1.4 AI in Team Fortress Classic

As it is one of the possible applications described in 1 to produce better AI for the computer players in TFC, the existing bots in TFC are of interest. While Half-Life was originally not designed to contain separate bots, dedicated players soon created a development kit which allowed programmers to plug custom computer players into the main application and into modifications. This development kit was used to create a number of bots of varying quality for TFC. Due to the complex nature of TFC gameplay, i.e. bots had to be able to act in a team and pursue predefined goals apart from the usual "'defeat all enemy players"', the development of these bots proved to be harder than the development of bots for simple Deathmatch modifications. One of the most sophisticated bots is FoxBot (FoxBot, 2007).

This bot uses predefined waypoints to find its way around the maps and follows certain goals according to the current state of the game and its class.

4.2 Generating the Data Set

This section describes the origin of the data and how the data set is retrieved from its original online source. It also describes how the data is stored and the basic cleaning that is done to ensure a consistent data set.

4.2.1 Original Log Files

TFC is an online game which runs on a server to which the players of both teams connect. This server offers the possibility to record a game and store a certain set of information about the game in a log file. Valve defines a log standard which has to be adhered to by all mod developers (Valve, 2007).

4.2.2 The Blarghalizer

The Blarghalizer (blarghalizer.org, 2007) is a website which provides the service of analyzing the log files of TFC matches. It offers a form through which people can upload a TFC log file. After uploading the log file, scripts on the website parse the log file, analyze the data contained in it, process the information and generate a number of HTML files that display the information. Figure 4.2.2 shows the central HTML page of a TFC match on *www.blarghalizer.org*. The website also offers the option of downloading all HTML files which belong to one log file in one compressed file.
Since its launch in early 2006 information about more than 10000 TFC games has been uploaded and stored on this webpage.

4.2.3 Downloading Log Files

Since the webpage unfortunately deletes the original log files after it extracted all necessary information to generate the HTML files, the only data sources are these HTML files. Because they contain a lot of information more than once and generally store the information in a way that is not easily machine-readable, the data has to be extracted before it can be anaylized for patterns. The first step is downloading these files. Since the HTML files of the more than ten thousand log files which form the test base have a size of approximately ten gigabytes, the compressed files are downloaded which are only 10% of that size. This is done by using a PHP script to subsequently open all pages which contained the links to the games, extracting all names from the HTML source code and storing these names in a MySQL database.

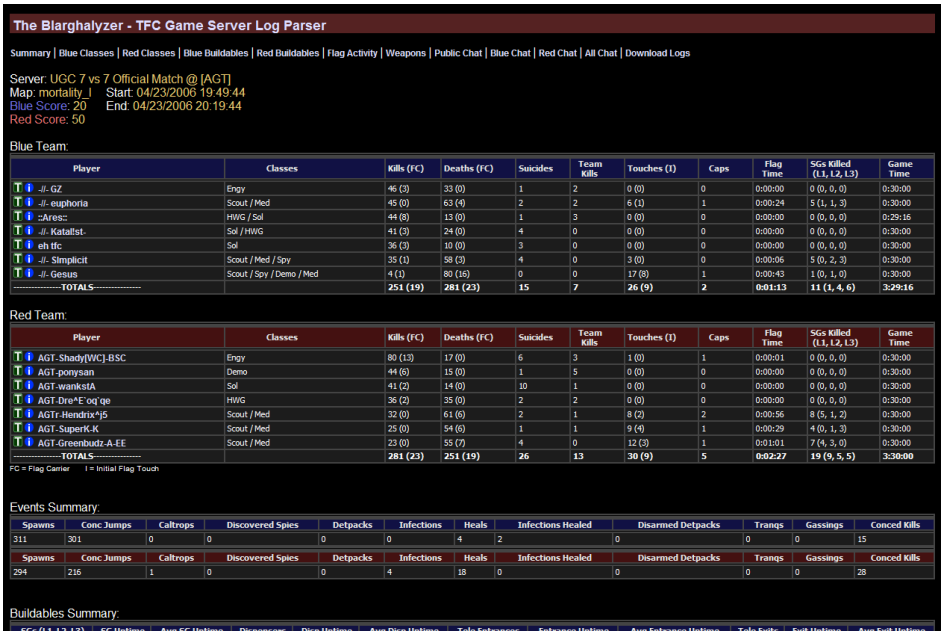

Figure 4.2: Top of the main page for a game description

The entries are then used to feed an XML-based download manager which downloads the compressed files for the single games.

4.2.4 Extracting Data from the HTML Files

After downloading all compressed files the actual test data is extracted from the source code of the HTML files. This was done by using a number of PHP scripts which utilized regular expressions and string analyzing methods to extract the desired information from the source code of the HTML files. The information is then stored in a MySQL database. Part of such a PHP script can be seen in Appendix B. Basically one script is used to extract information from one type of HTML file. The different types of HTML files which are analyzed are listed below.

- Main game page
- Class listings for both the red and the blue team
- Buildables for both the red and the blue team (Sentry Guns, Armor Dispensers etc.)
- Flag activities
- Pages for the single players

4.2.5 The Database

The choice of MySQL as the preferred method of storing the data does not seem the obvious first choice since only very few machine learning techniques can directly operate on relational databases. Instead the more common option would be a simple text file or a CSV file which can be fed directly into a multitude of machine learning algorithms. Since it is stored in a database the data therefore has to be extracted to generate the data sets which can be analysed by common machine learning algorithms.
The reasons for this kind of storage are several advantages of a relational database over a simple text file for this specific task.

1. A relational database offers compression and allows a better usage of disk space by getting rid of redundancies in the data, especially if the data is normalized as in this case.

2. It is easier to extract different sets of data for different tasks from a relational database.

3. A relational database allows faster access and thus faster retrieval and changes of the stored data.

4. Since working on such a big amount of data always carries the risk of an error half way through a routine, it is a huge advantage if the database accounts for safe transactions.

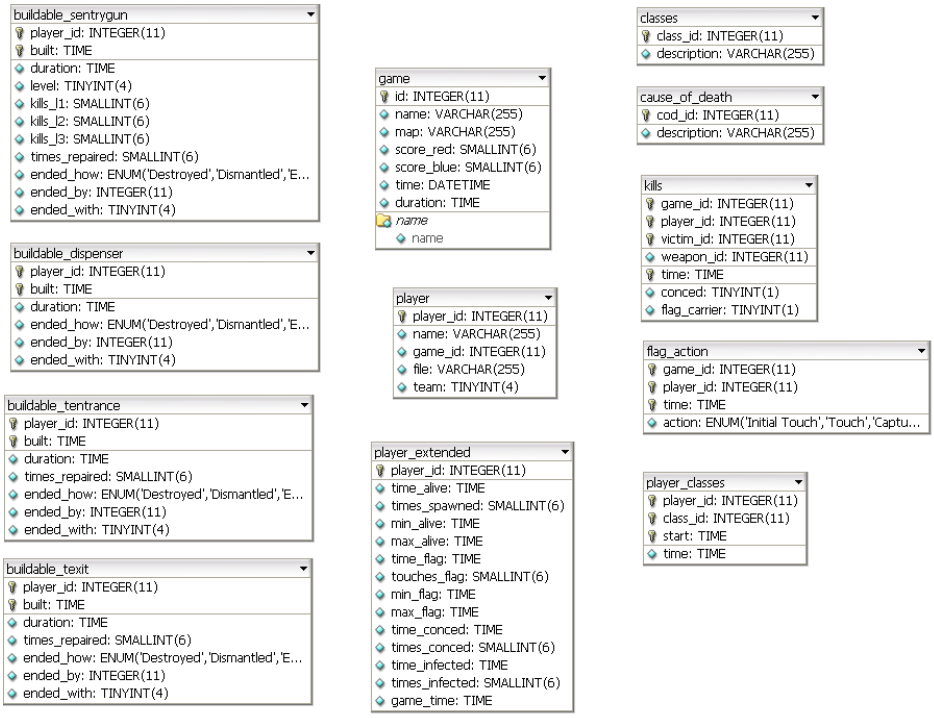

Figure 4.3: Database Structure

The structure of the database that is used to store the information from the TFC log files can be seen in figure 4.3.

4.2.6 Data in the Database

This section shortly explains the data which is extracted from the HTML files. Since not all of this data is used in every part of the analysis, the sections describing the different methods of analysis mention separately which part of the data set is used in the particular section.

Game

The information on a game consists of the name of this game, the map on which it was played, the starting time as well as its duration.

Player

The information on single players is stored in the three tables *player*, *player_extended* and *player_classes*. In *player* the player's name, the team for which he played as well as a reference to the game in which he played is stored. *player_extended* contains information on the time he was in the game, the minimum time and maximum time as well as the aggregate time

he was alive, the number of times he was resurrected ('respawned'), the number of times he carried the flag and the amount of time he carried it, aggregate as well as minimum and maximum. The table also contains information on how often he was infected or infected someone himself and the number of times he was hit by a concussion grenade or hit someone with it. *player_classes* contains the information on what classes a player picked, when he picked the class (relative to the starting time of the game) and for how long he played that class. This table cross-references the classes table where the names for the nine different classes are stored together with a unique id.

Buildables

The set of tables containing information about the structures an Engineer can build consists of *buildable_dispenser* for armor dispensers, *buildable_sentrygun* for sentry guns, *buildable_tentrance* for entrances of teleporters and *buildable_texit* for exits of teleporters. These tables contain a cross-reference to the entry of the player that built them, the number of times they were repaired, the time they were built as well as the duration they were operational, the way in which they were disposed of and, if they were destroyed, a cross-reference to the entry of the player who did it as well as the weapon with which it was done. In addition for sentry guns it is stored which level of upgrade they reached and the number of eliminated players for each level.

Kills

The *kills* table contains for each kill the player who did it, the victim, the cause of death, the time when it took place, if the victim was under concussion and whether it was carrying the flag when it was killed.

Flag Activities

The table *flag_actions* contains information which relates to the flags. It contains a cross-reference to the player who performed the action, the time when the action took place as well as the kind of action that took place. Possible actions are Initial Touch, Touch (pick up a flag that a teammate dropped), Captureand Coast-To-Coast capture.

4.2.7 Basic Data Cleaning

This section describes the basic actions which were taken to verify that the data set only contains valid entries. Throughout the analysis it was verfied that all data contained only belongs to games which are comparable to each other and entries which did not meet this criterium were deleted.

Game Deletion

In order to work on a valid data set the database is cleared of games which do not meet some standards which are set to acquire meaningful results. Games that have a duration of less than 25 minutes are deleted to be able to compare the games with more or less equal preferences among each other. Furthermore games with more players in one team than in the other team are deleted, since these games obviously are biased towards one team or another. Some recordings were faulty and did not record data for one or more players. Since this biases the data for the whole game, those games were deleted as well.

Kill Deletion

A surprisingly large number of kills was identified to have taken place at time 0:00:00, i.e. just when the game starts. Another surprising characteristic of this behavior was the fact that just about all of these kills were suicides, i.e. the players killed themselves. Furthermore their classes were recorded incorrectly, i.e. the weapon they killed themselves with belongs to a class that is different to the one that is recorded for them at that time.
The explanation for this behavior is that the game only starts after a short period described as "'pre-game"'. In this time the players, especially more advanced players which have gone through the initial phase very often, play around with classes they usually wouldn't take and quite often get themselves killed. Since kills don't affect the score in a game and the game has not started yet, this has no affect on the outcome of a game.
To remove the effect of these "'fun"'-kills, all kills which happen in the first three seconds, i.e. every kill that happened before game start and in the first three seconds, are removed from the database. This effectively removes about 48000 kills or approximately one percent of all recorded kills from the database.

4.2.8 Statistical Characteristics of the Test Data

The original data set consisted of information from 10663 game logs. Through the cleaning process described in section 4.2.7, i.e. limiting the data to games which are longer than 25 minutes and have an equal number of players in each team as well as only complete recordings for each player the number of game logs making up the data set is culled down to 6099. This constitutes effectively the pruning of 42% of the available games but is a step that has to be taken in order to gain valid results. The resulting data set has the follwing statistical characteristic (Table 4.1).

Number of recorded games:	6099
Number of recorded players:	90524
Average players per team:	7.36
Number of recorded kills:	4,352,279
Average kills (including own team) per player per game:	48.01
Number of recorded class choices:	157,767
Average number of different classes per player:	1.74
Number of recorded flag captures:	84,469
Included Coast-To-Coast-Captures:	15,087 (17,86% of all Captures)
Average number of captures per game:	13.73
Number of Touches:	597,821
Average number of Touches per Capture:	7.08
Average recorded game time:	29 Minutes 17 Seconds

Table 4.1: Data Set Statistics

4.2.9 Problems with the Size of the Data Set

The size of the data set seems sufficient to be able to find valid patterns. Even after cleaning the data set from invalid data more than 6000 games remain. This should still be enough to retrieve representative patterns. However the maps the games were played on pose a serious problem.

On the one hand there are more than 2000 games which have an *(unknown)* as their map meaning they could have been played on any map and thus cannot be used in any analysis which takes maps into account. On the other hand, the remaining 4000 games were played on 162 different maps. While there are some preferred maps as stated in section 5.1, the distribution is not as much focussed as would have been desirable. Even the ten most played maps have only between 130 and 200 games each 5.1. This would leave only this many instances which can be compared to each other. One could now argue that the map is only important in very few occasions and usualy can be ignored. However the analysis in section 5.1.1 shows that this is not true. Therefore the actual data set which an analysis is performed on is only a very small subset of the actual test data.

5 Data Analysis

This chapter covers two approaches to using machine learning algorithms on the data set in order to find patterns. This is done by applying classification algorithms which try to predict the outcome of a game based on previously learned hypotheses.
For each apprach first the anticipated pattern is described more closely as well as the subset of the data in which this pattern is supposed to exist. Then the statistical analysis which is the reason for this assumption is shown. Afterwards different machine learning algorithms are used to detect the existence or absence of the pattern in the data set. The results of the machine learning algorithms are discussed subsequently.
A limitation that applies to both approaches is the size of the data sets to which the algorithms are applied. Since the expected outcome should be win or loss, draws are ignored. This reduces the data set which are used for training and test purposes by about 6% on average. Another limitation is the dependancy of patterns on the map 5.1.1. This reduces the size of the data set which can be used to train an algorithm even for the most played maps to only 5% of the whole data set. The resulting number of data points for the ten most popular maps can be seen in table 5.1.

Map	**Number of games played on this map**
shutdown2	160
schtop	154
fry_baked	127
openfire_lowgrens	116
hellion	105
siege	103
ss_nyx_ectfc	96
monkey_l	92
mortality_l	90
2mesa3	87

Table 5.1: Number of data points for the ten most popular maps

5.1 Game Result Prediction Based on Team Composition

Since TFC is a team-based game where the strategy relies heavily on which classes make up a team this section is concerned with effects of the team composition. One assumption is that the usual composition of a team is based on the map the game is played on. This is due to the different structures of the maps where the different classes can use their strengths better on one map than on the other. A Sniper for instance usually only performs well on maps with good hiding places and vast open spaces. On the other hand on a map with lots of enclosed spaces a Soldier would outperform a Sniper easily. This should lead to the average team composition being different from map to map.

5.1.1 Average Class Usage per Map

To validate this assumption the average proportion of the total game time for each class is computed for the ten maps that were played the most in the data set. The usage of these maps guarantees that enough samples exist to draw valid conclusions. Since a lot of games have been played on these maps this also means that they are fairly balanced and give a good overview of average playing style. Furthermore this automatically excludes avoids so-called *funmaps*. Funmaps are maps that are not played in a normal competition mode but reward and facilitate usually non-normal behavior. One example of a funmap would be a map which is simply made up of two walls on each side of the map with a large open space in between. This structure makes any other class than a Sniper all but useless.

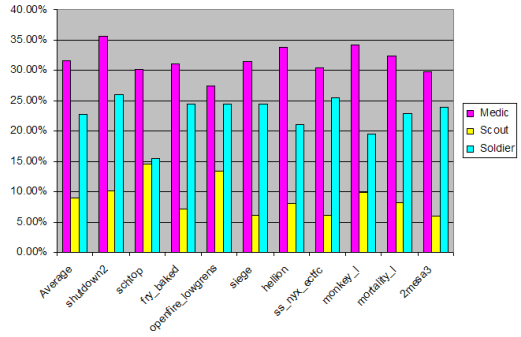

Figure 5.1: Playtime for Medic, Scout and Soldier

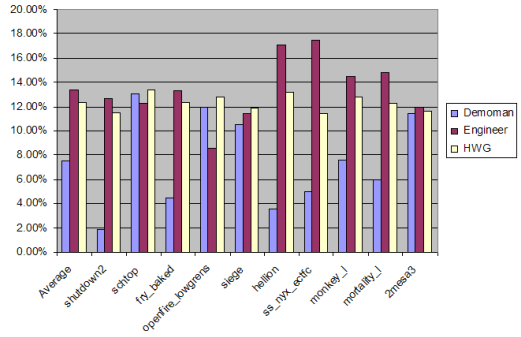

Figure 5.2: Playtime for Demoman, HWGuy and Engineer

Figure 5.1 to Figure 5.3 show the percentage of time for which each of the different classes is played on the different maps onb average. Figure 5.4 shows the distribution of the play time among all classes per map. The average already shows that in general the different classes are played for different amounts of time; the classes of Pyro and Sniper for instance are hardly ever played at all. Certain classes like Soldier and HWGuy are played for about

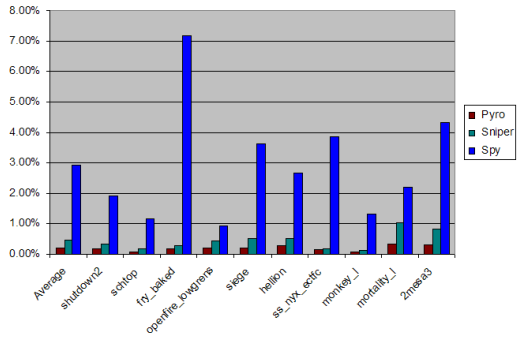

Figure 5.3: Playtime for Spy, Sniper and Pyro

the same amount of time, i.e. their average, regardless of the map. Others like Medic and Engineer show variations in play time which can be up to 50% between two maps. A third kind to which the classes of Demoman and Spy belong shows huge variations in play time of several 100% between maps. These graphs show that there is a connection between the map which is played and the average team composition. This leads to the conclusion that if there is a preferred team composition for a certain map, this is probably because this composition leads to better results. Therefore it should be possible to extract rules which show what class combination exactly leads to the best result on a certain map.

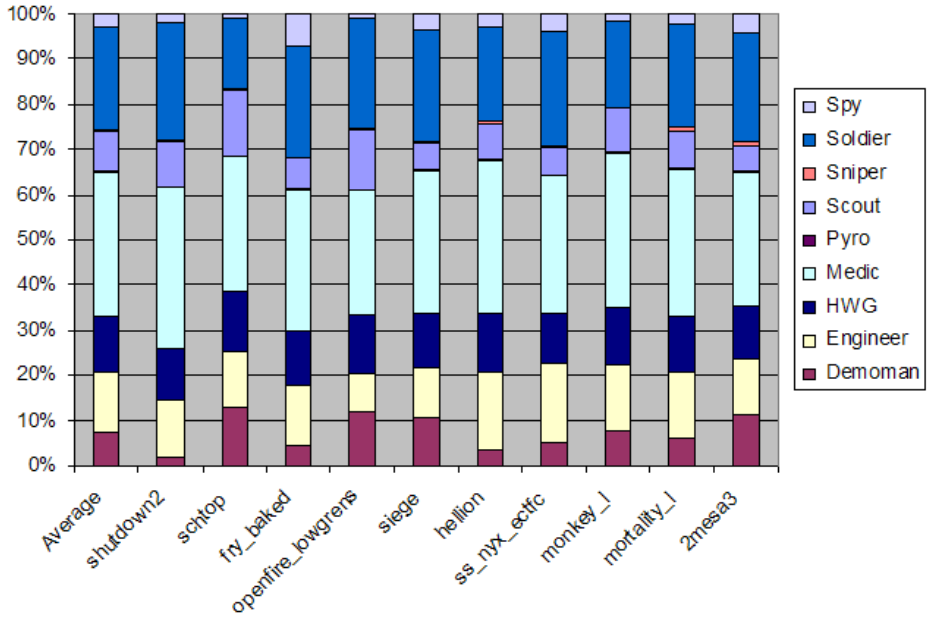

Figure 5.4: Distribution of Playtime per Class for the ten most popular maps

To be able to apply machine learning algorithms successfully to the data set to learn the most successful class combination for a map, the team composition obviously has to differ between the winning team and the losing team. The diagrams in Figure 5.5 to Figure 5.8 show the differences in playtime per class for the four maps that were played the most.

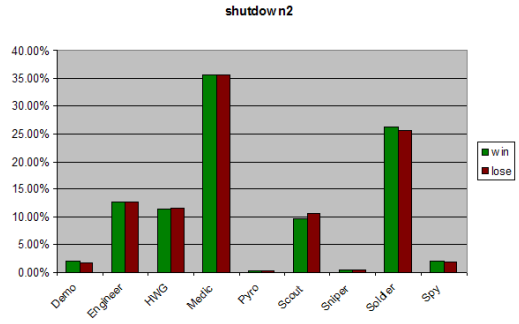

Figure 5.5: Playtime on *shutdown2*

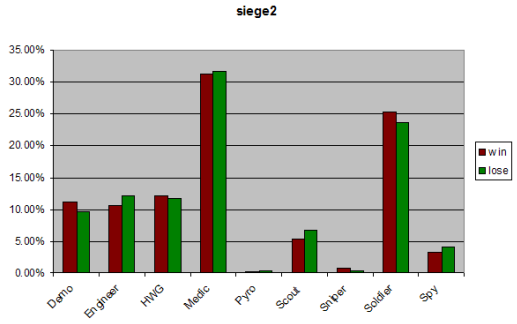

Figure 5.6: Playtime on *siege2*

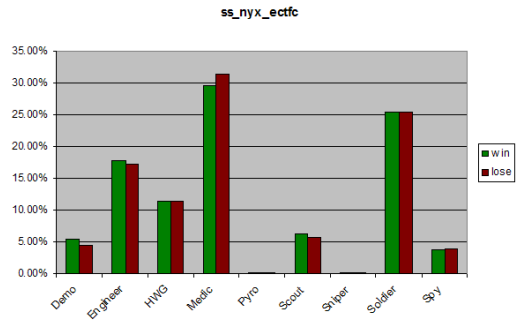

Figure 5.7: Playtime on *ss_nyx_ectfc*

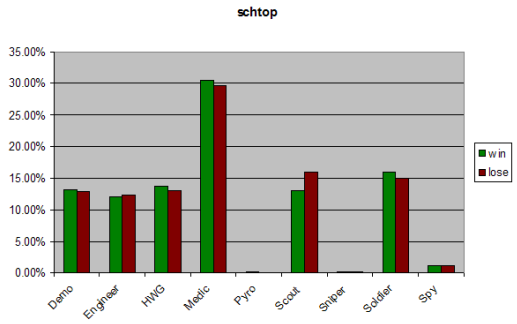

Figure 5.8: Playtime on *schtop*

The diagrams show a startling absence of differences in team composition between the winning teams and the losing teams. Visible differences for the classes Sniper and Pyro bear only minor significance because of the negligible overall amount of time those two classes are played on average. Differences for the five classes with the highest amount of time played on average (Demoman, Engineer, HWGuy, Medic and Soldier) are never more than one percent of the overall amount of time per game. The only classes which show slightly promising tendencies in their amount of usage are Scout and Spy. The observed behavior leads to the assumption that applying machine learning algorithms might not lead to the desired identification of winning teams through team composition. This in turn would mean that it is not possible to specify the ideal team composition through machine learning.

Result for Classification According to Class Compositions

	J48	Naive Bayes	Nearest-Neighbour	Neural Network
shutdown2	59.94%	52.51%	59.54%	62.15%
schtop	65.53%	53.60%	64.47%	61.67%
fry_baked	63.36%	55.25%	60.82%	64.24%
openfire_lowgrens	67.18%	57.25%	62.34%	65.11%
siege	63.15%	51.63%	60.66%	60.2%
hellion	57.59%	62.18%	55.04%	61.11%
ss_nyx_ectfc	53.29%	47.10%	50.46%	55.21%
monkey_l	69.11%	53.56%	66.67%	68.89%
mortality_l	60.14%	51.31%	65.40%	60.18%
2mesa3	50.89%	58.22%	53.33%	50.67%

Table 5.2: Correctly classified instances based on team compositions

	J48	Naive Bayes	Nearest-Neighbour	Neural Network
shutdown2	0.43	0.47	0.40	0.39
schtop	0.40	0.47	0.36	0.40
fry_baked	0.38	0.44	0.39	0.37
openfire_lowgrens	0.37	0.43	0.38	0.36
siege	0.38	0.48	0.39	0.41
hellion	0.45	0.40	0.45	0.39
ss_nyx_ectfc	0.48	0.52	0.50	0.44
monkey_l	0.33	0.47	0.33	0.33
mortality_l	0.41	0.49	0.35	0.40
2mesa3	0.49	0.42	0.47	0.50

Table 5.3: Mean absolut error for classification based on team compositions

The following tests with machine learning algorithms confirmed this assumption. Regardless of the applied algorithm and map it was not possible to achieve a correct classification above 67% for an input set for any map. The average was actually a lot lower with the data sets for some maps being classified correctly in only about 50% of all cases, i.e. a random pick would have the same result.

Figure 5.9 shows an example decision tree from Weka which was generated for data on the map *schtop*. The algorithm which was applied is J48 which created unpruned C4.5 decision trees. As the figure shows, the only three classes that are used for classification are Scout, Soldier and Pyro. Pyro is a class which is hardly ever played, has an average of less than 1% playtime on that map and therfore has a huge variance in playtime. This means that the class is determined according to just two of the nine attributes (classes) while the other

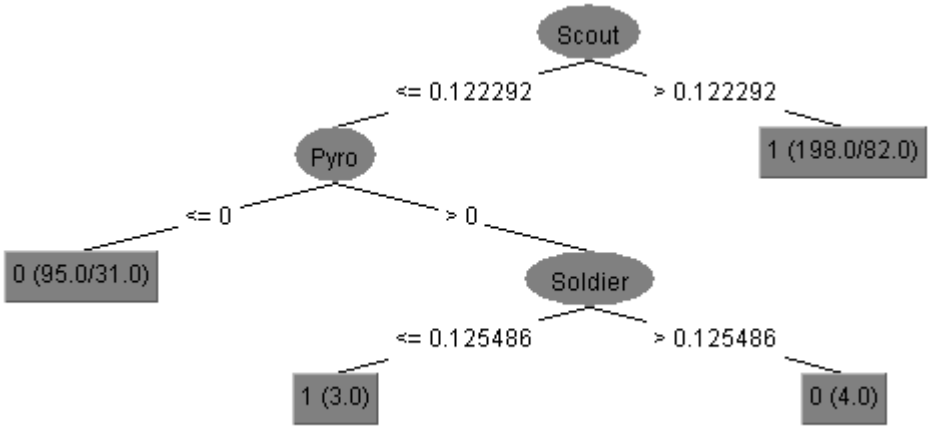

Figure 5.9: C4 Decision Tree for Win/Loss Classification

seven classes are ignored. Also the differences in these two classes are remarkably small on average.

5.1.2 Results

The reasons for this behavior lie in the used data set. The initial culling reduces the data set to data from games which took longer than 25 minutes and have an even number of players. The average time played is 29 minutes and 17 seconds and thus even higher. If we take into account the fact that the number of games above 31 minutes is less than 1% of all games, we can assume that the desired duration of most games is 30 minutes. This duration is the standard duration for games played in official TFC leagues (TFLeague, 2007). Since almost every league game is played with an equal number of players on both teams while games which are played by random players often have dissimilar numbers of players for both sides because players can leave and join at will, this further increases the amount of official league games in the data set. This is actually a good thing in the first place since these games tend to be played by organized teams which consist of advanced players. Since one of the possible uses for the application of machine learning algorithms to game logs was the creation of more able and more realistic computer agents, the evaluation of games from advanced players is desirable. Since the teams show all more or less the same behavior according to their composition depending on the map, we can assume that they are all more or less on the same skill level. The conclusion that this skill level is high comes from the observation that even though the data set is quite big and thus would have to have several games of teams which have even better class combinations than the average team, this is not the case. All teams seem to be making essentially the same decisions when it comes to class selection.
The downside of this data is that advanced players often adapt to the most successful style

of playing. The distinction between one team of advanced players and another team of advanced players is far more difficult than the distinction between an advanced team and a mediocre team. This becomes obvious in the experiment described above. The upside though is, that since we are apparently dealing with advanced players which all make the same choices when it comes to class selection for a certain map, their choices are probably the most successful strategy right now. Since one aim of these experiments was to find the most successful combination of classes, this aim has still been achieved even though not through the successful application of machine learning algorithms.

5.2 Predicting Game Outcome Based On Class Performance

This chapter evaluates the effects of variations of the performance of certain classes on the outcome of a game. The definition of a player's performance is obviously quite subjective and even with a complete visual recording of a players performance in a game the opinions about it would differ. The available data is far from being a complete representation of the player's actions and only covers key data of his performance. On the one hand this is an advantage since it already reduces possible input data for machine learning algorithms. On the other hand the performance has to be determined through very limited data. Based on the given data the definition of performance for the different classes is therefore defined according to their primary purpose in the game. This purpose is drawn from statistics across the data set. The characteristics according to which the purpose of a certain class can be defined are limited by the given attributes in the data set. The main attributes are *kills* and *flag activities*. The performance for a player is defined by the difference between the players performance data and the average performance data of the winning team on a certain map.

5.2.1 Kill Distribution

Figure 5.10 shows the average distribution of kills according to classes. The diagram contains both the values for winning as well as for losing teams.
It is also important to notice that several of the classes have damage-over-time weapons (e.g. *Flamethrower* and *Incendary Gun* for the Pyro or *Infection* for the Medic) and throwable explosives like *Grenades* can explode after a class change and would thus be counted as a kill for the wrong class. An exemplary test for the class specific weapons of Pyro and Medic showed that this error accounts for less than 1% of the kills of a class and thus is negligibly small.
It would be a problem though if this error would be evenly spread across the other classes since classes with few kills would receive disproportional many wrongly assigned kills in relation to their own kills. But since a class only gets assigned a kill if it is picked during

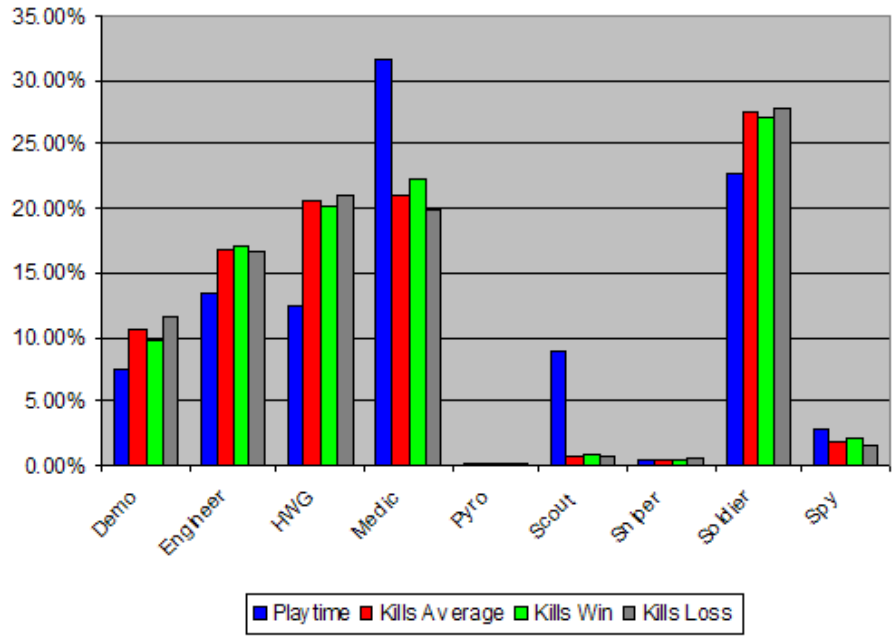

Figure 5.10: Distribution of Kills per Class for the ten most popular maps

the game right before receiving a kill which was initiated by the previously chosen class, the amount of wrongly associated kills a class receives is proportional to the time a class is played. Through the ratio between time played and kills, both in relation to the accumulated values, it is possible to define the average expected performance of a class in relation to kills.
The classes are split into three different major categories according to their expected performance in relation to the number of their kills.

1. **Main fighting classes**
   These are classes which are responsible for a disproportional large amount of kills. Their performance is therefore outlined very well by the number of kills they achieve. Classes of this category are HWGuy , Soldier , Sniper and Demoman as well as Engineer to a lesser degree.

2. **Semi-fighting classes**
   These are classes which account for a reasonable amount of kills compared to their play time, but their strengths lie elsewhere. Therefore their performance can only be graded to a certain degree by looking at their kills. Classes of this category are Medic , Spy and Pyro .

3. **Supporting classes**
   These classes are responsible for no significant proportion of the overall kills. Their main

purpose obviously lies elsewhere and their performance can therefore not be evaluated by looking at kills. The only class in this category is Scout.

A kill always means a temporary weakening of the enemies' defense or offense. Therefore it should be possible to deduct parts of the outcome of a game from the number and distribution of kills for the two teams. Since the kills don't influence the score directly, the absolute numbers instead of the relative percentages can be used in the process. To achieve comparable results this has to be done in relation to the number of players in one team since more players usually means more kills as can be seen in Figure 5.11.

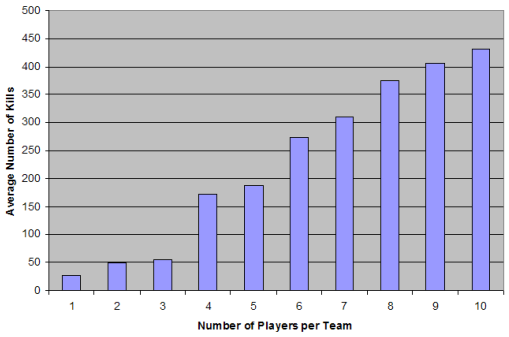

Figure 5.11: Average Number of Kills by Number of Players in a Team

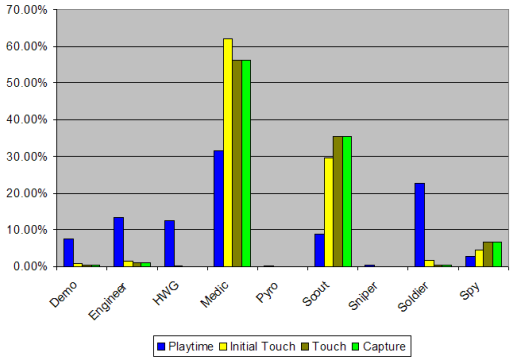

Figure 5.12: Average Flag Activities on the ten most popular maps

5.2.2 Flag Activities

Figure 5.12 shows the average distribution of flag activities (Initial Touches, Touches and Captures) among the different classes. As with kills, this diagram shows interesting statistics for some classes.

While there are slight differences between the three recorded flag activities for one class, the general tendency is always the same for all three different actions, i.e. if a class has 10% of all Captures it is has also about 10% of all Touches as well as about 10% of all Initial Touches. From here on the flag activities will therefore be described separately unless mentioned otherwise.
The classes can be split into two categories according to their expected performance in relation to flag activities.

1. **Main flag classes**
   These classes account for a disproportional big amount of flag activities, therefore their performance can be rated based on this amount of flag activities. The classes that

belong into this category sorted by the markedness of their relation to flag activities are Scout, Medic and Spy.

2. **Minor flag classes**
   All other classes fall into this category. Most of these classes have only a few flag activities which depend less on the class and more on the individual game situation, i.e. picking up a flag that was dropped by a team member. HWGuy and Sniper have nearly no flag activities at all, probably due to their slow movement speed.

Certain maps favor certain classes regarding flag activities. For instance the map *openfire_lowgrens* shows an unusually high percentage of class actions for the minor flag classes Soldier, HWGuy and Demoman. This is due to the fact that this map has large open ranges where the speed advantage of the main flag classes does not come into play as often as on other maps. Instead classes with higher hitpoints have a better chance. Since the Captures determine the outcome of a game, the absolute numbers can not be used in this evaluation. They are directly linked to the attribute they should predict. However the distribution should show if the classes for which it is on average easier to capture a flag have performed well in a match compared to the other classes on the team.

5.2.3 Problems with this Approach

This evaluation of the players' performances according to the criteria described has a number of problems beside the fact that it is a very strong simplification of the term "'performance"'. It is basically only applicable to classes whose main purpose lies in one of the evaluated areas. Classes which also have more versatile possibilities like the Medic which heals and disinfects team members are not evaluated objectively. The class which suffers most from this is probably Spy . While Spies have on average a decent number of kills and flag activities, their abilities reach far beyond that. One of the primary tasks of a Spy is, as the name suggests, the gathering of information in enemy territory. If a Spy would focus all his effort on gathering information instead of capturing flags and killing enemies this might help the team more than an average Spy but would be penalized by this performance evaluation. For that reason only the performance of classes for which the main purpose lies in one or more areas for which data is available will be used to determine the outcome of a game. Another problem is that this evaluation does not do justice to more sophisticated player behavior. It does not indicate if a player spent the whole game playing poorly or if he was standing in the base defending the flag and thus helping the team on a grander scale. But since this is true for both teams such performances should balance each other out.

5.2.4 Classes and Attributes Contributing to their Performance

Demoman Performance is based solely on the kills.

Engineer Performance is based solely on the kills. This is not absolutely accurate since the Engineer is usually defined by the built machines. Since these are not part of the evaluation, it can only be evaluated by the effectivenes of these machines, particularly the sentry gun.

HWGuy Performance is based solely on the kills.

Medic Performance is based on kills and on flag activities. Even though Medics have other abilities they play a major part in flag activities and also have a reasonable amount of kills and therefore can be evaluated based on this data.

Pyro Not evaluated. Pyros only make up a very small percentage of the overall classes; therefore they would be too heavily influenced by single performances.

Scout Performance is based solely on the flag activities. Scouts are the primary class for capturing flags and are the only class that has nearly no kills whatsoever.

Sniper Performance is based solely on the kills. Even though there are only very few Snipers, nearly every single one of them has no flag captures whatsoever. Therefore it is safe to assume that Snipers only ever influence the game through kills.

Soldier Performance is based solely on the kills.

Spy Not evaluated. Despite Spies having decent kills and flag activities, the benefits of two main purposes of Spies, gathering intelligence and destroying sentry guns, cannot be evaluated effectively.

5.2.5 Performance Calculation

To implement the approach described above, several defaults have to be specified. The base values according to which the performance of a class on a certain map is evaluated are the average values of all winning teams on that map. The averages of those values accross the ten most popular maps for kills and flag activities can be seen in Figure 5.13. The values for the single maps can be seen in appendix A The attribute according to which the performance of a class is calculated is the percentage of the total playtime of a game this class is played. This percentage determines the expected values for kills and flag activities. However a team

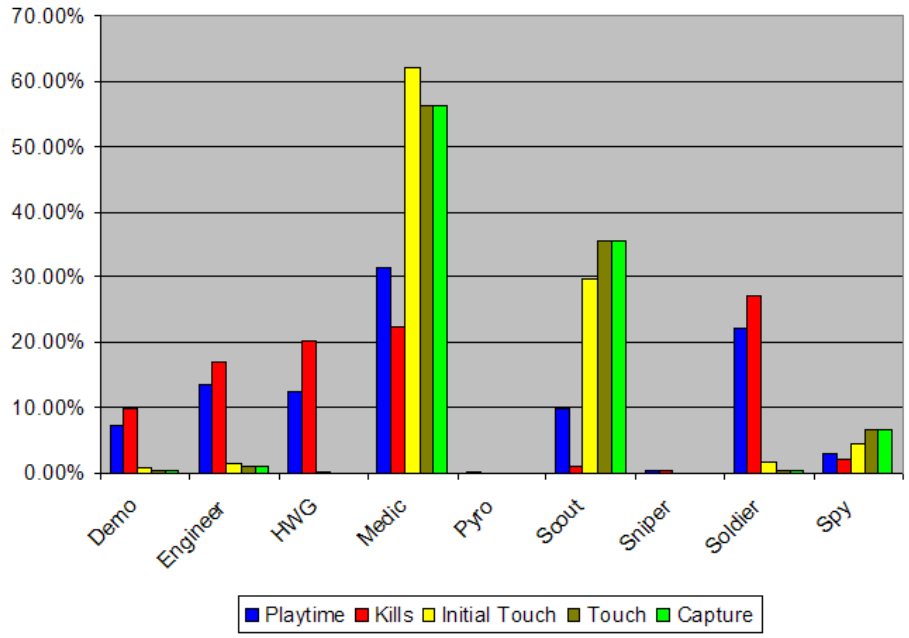

Figure 5.13: Average kills and flag activities for the ten most popular maps

usually never plays each class for exactly the same amount of time as the average distribution in figure 5.13 specifies. Therefore the expected values have to be adjusted according to the playtime in the actual game.

5.2.6 Implementing Performance Evaluation According to Kills

The algorithm which is used to calculate the performance values according to kills consists of a series of separate steps.

1. Generate a diagram using the average percentage of kills and the average percentage of playtime for the map of the current instance. These values are computed according to the data from all winning teams that played on this map. This step results in the expected kills/playtime ratio.

2. Read the playtime distribution per classes for both teams for the game that should be evaluated from the database.

3. Calculate the expected kill distribution according to the playtime distribution. This happens by simple multiplication with the previously computed average kills/playtime ratio.

4. Normalize the resulting distribution to 100%. This is necessary since the distribution will not be exactly 100% if the playtime distribution is not precisely like the average distribution.

5. Multiply the result with the expected (average) number of kills for this map for this number of players. The result is a diagram which contains for each class the number of kills it is expected to achieve in this game.

6. Divide the **actual** kills each class achieved through the expected number. The ensuing number (converges towards 0 for actual kills converging towards 0) denotes the performance of this class in this game with respect to kills.

5.2.7 Implementing Performance Evaluation Based on Flag Activities

This process follows closely the procedure described above in section 5.2.6. Since the flag activities directly influence the score (on most maps the score is determined only through Captures) the absolute values for those attributes cannot be used. Therefore only the relative percentages for the flag activities are applied which means that step 5 from the procedure is omitted.

5.2.8 Results

This section describes the results of the application of different machine learning algorithms to classify the outcome of a game. The algorithms that were applied are J48 (pruned C4.5 decison trees), a Naive Bayes classifier, an instance-based learner using a nearest neighbour classifier and a 2-layer neural network that uses backpropagation to train. The algorithms were executed as well as evaluated in the Weka Explorer environment. The experiment was conducted using 10-fold cross validation. Since both kills and flag activities depend to a certain degree on the map which they are played on (see appendix A) the data sets which are evaluated are the games played on the four most popular maps.

Result Classification According to Kill Performance

	shutdown2	*schtop*	*fry_baked*	*openfire_lowgrens*
J48	70.25%	66.71%	71.15%	70.67%
Naive Bayes	59.31%	53.34%	62.56%	51.22%
Nearest-Neighbour	74.63%	69.66%	66.18%	64.28%
Neural Network	76.50%	71.53%	69.10%	69.71%

Table 5.4: Correctly classified instances based on kills performance

	shutdown2	*schtop*	*fry_baked*	*openfire_lowgrens*
J48	0.32	0.35	0.29	0.31
Naive Bayes	0.40	0.46	0.37	0.48
Nearest-Neighbour	0.25	0.30	0.34	0.36
Neural Network	0.25	0.30	0.3	0.32

Table 5.5: Mean absolut error based on kills performance

On the one hand the results shows an obvious improvement to the classification based on team composition. Even though the maximum of 76.5% correctly classified instances for the multilayered perceptron is still far from the desired values, this shows promise in regard to future performance evaluation in combination with flag activities evaluation.
On the other hand an experiment conducted using only the information about which team has a bigger number of absolute kills achieved already about 66% correctly classified results. Therefore it is questionable if the addition of class-wise performance really achieved such a big improvement.

Result Classification According to Flag Activity Performance

	shutdown2	*schtop*	*fry_baked*	*openfire_lowgrens*
J48	58.94%	69.47%	60.14%	66.07%
Naive Bayes	51.06%	49.49%	48.87%	56.39%
Nearest-Neighbour	54.69%	58.88%	60.11%	58.01%
Neural Network	57.44%	54.53%	62.58%	60.56%

Table 5.6: Correctly classified instances based on flag activities performance

	shutdown2	*schtop*	*fry_baked*	*openfire_lowgrens*
J48	0.43	0.38	0.42	0.35
Naive Bayes	0.49	0.51	0.50	0.44
Nearest-Neighbour	0.45	0.41	0.40	0.42
Neural Network	0.45	0.48	0.41	0.43

Table 5.7: Mean absolut error based on flag activities performance

The results from applying the machine learning algorithm performance only based on flag activities are rather disappointing. No algorithm achieves a better result than 66% correctly classified instances regardless of the map with the average being far lower. Despite this it is still possible the classification will improve in combination with the kill performance as an effect of the combined information.

Result Classification According to Weighted Performance

As described in section 5.2.4 the performance of the different classes is defined differently in regards to kills and flag activities. Therefore those two types of performance have different weights depending on the respective class (Table 5.8). Instead of improving the result from

	Demoman	Engineer	HWGuy	Medic	Pyro	Scout	Sniper	Soldier	Spy
Kills	1	1	1	0.5	0	0	1	1	0
Flags	0	0	0	1	0	1	0	0	0

Table 5.8: Weighting of the performance attributes

the simple kills based classification the weighted combination with flag activities results in less instances being correctly classified than even for the simple flag activities based classification. On average the neural network performs best with the C4.5 decision trees being a close second. The nearest-neighbour comes third and the Naive Bayes classifier is by far the worst method of classification.

	shutdown2	*schtop*	*fry_baked*	*openfire_lowgrens*
J48	58.19%	63.09%	59.58%	62.87%
Naive Bayes	49.69%	51.22%	51.39%	49.97%
Nearest-Neighbour	58.56%	66.71%	54.53%	59.04%
Neural Network	60.13%	64.97%	53.59%	60.10%

Table 5.9: Correctly classified instances based on weighted performance

	shutdown2	*schtop*	*fry_baked*	*openfire_lowgrens*
J48	0.45	0.39	0.41	0.39
Naive Bayes	0.49	0.48	0.48	0.50
Nearest-Neighbour	0.41	0.33	0.45	0.41
Neural Network	0.41	0.39	0.46	0.42

Table 5.10: Mean absolut error based on weighted performance

6 Discussion and Future Work

The analysis in chapter 5 does not produce the expected patterns. Instead the algorithms are shown to produce very unsatisfying results in regards to classification. This could be due to the fact that there simnply are no patterns in this data. On the other hand it would also be quite possible that the patterns are just not as obvious as previously assumed. One indication for this is that the data set consists mostly of games by expert players, an assumption that is made due to the results of the first experiment. This assumption has of course to be validated. In order to do this, a sufficiently big data set of games between amateurs and expert players would be required to perform a validating experiment.
Also while the original concept of classifying game results class performance didn't produce convincing results immediately, the approach still remains promising. The performance metric obviously has to be refined to achieve better results. The evaluation based on kills already showed interesting results even though this can to a certain degree be attributed to the fact that the number of kills showed a direct relation to the game outcome. Though the evaluation based on flag activities did not result in improved correct classification it is still assumed that the performance of certain classes like the Scout can be evaluated through it. One way to do this would be through indirect usage of the absolute values for expected flag activities, i.e. using the absolute values to compute a relative performance measure. Furthermore the performance of Engineers can be based on data from the structures they build.
Possible future work also includes the extension of the data set since the website where the data set was recorded is still running and more than 1000 games have been added since the download of the data set for this project. This new data could either be used to extend the exisiting data set or to create a test set which is then applied to check if the retrieved hypotheses still work on the new data.
An interesting possible patterns that could be evaluated in the future is if flag activities in general and captures in specific can be related to kills, i.e. the fighting is heavier if a flag is taken. This would also include determining if a series of deaths for the defending team would automatically indicate a capture. It would also be interesting to look for patterns in the timeline of a game, i.e. if it is the same as in many team sports where players tend to make more mistakes towards the end of a game and how this affects attributes such as kills and captures.

Both experiements in this project have been based on the map the games were played on. As stated in 5 this limits the training set size even for the most played map to less than 5% of the whole data set. It would thus be very interesting to see what the search on the whole data set produces for patterns which are independent from the map.

7 Conclusion

This report described the process of obtaining, cleaning and analysing a set of game logs from the game Team Fortress Classic . The retrieval of the files from the web as well as the extraction of information from these files through PHP scripts was elaborated as well as the cleaning of the data set of redundant or faulty data.
Following this the actual analysis was performed. Machine learning algorithms were applied to determine the existence or absence of patterns in the data set through classification. Two kinds of patterns were researched. First for patterns that indicated a relation between the team composition and the outcome of a game. The application of machine learning algorithms to search for these patterns did not yield results. The percentage of correctly classified instances was never above 70% regardless of the applied algorithm. The search for the second kind of patterns, a relation between the performance of the different classes and the outcome of a game was disappointing as well. The performance based on kills produced an acceptable classifcation of up to 76% correctly classified instances but the performance based on flag activities and on a weighted combination of both was very bad.
Despite this the project resulted in some interesting observations on the nature of the data set. The first experiment was based on the finding that classes have different amounts of playtime throughout the games. The playtime as well as many other attributes were also shown to vary between different maps. The first experiment resulted in the conclusion that the data set consists of games that were played by advanced players. The second experiement was based on the discovery that the different classes have different tasks in the game which is shown very clearly in the data set. This enabled the definition of performance as a measure. Summarized the project resulted in several interesting discoveries in the data set and paved the way for future work on this data set.

A Basic Statistics for the Ten Most Popular Maps

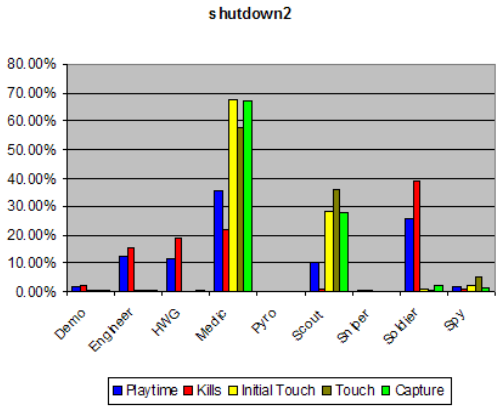

Figure A.1: Average Kills and Flag Activities on *shutdown2*

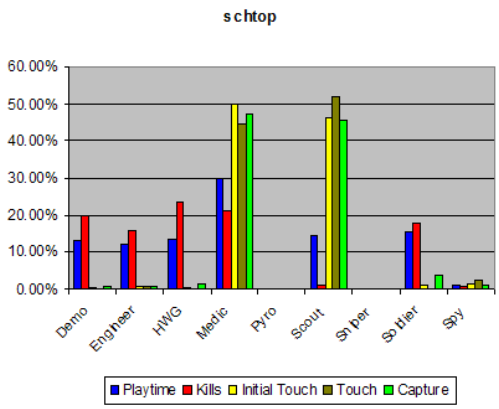

Figure A.2: Average Kills and Flag Activities on *schtop*

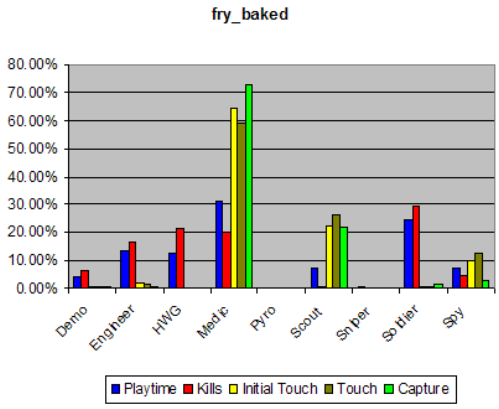

Figure A.3: Average Kills and Flag Activities on *fry_baked*

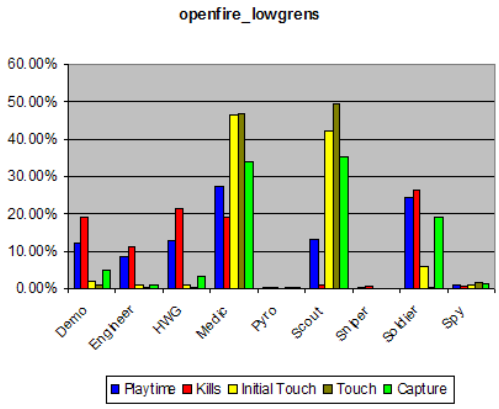

Figure A.4: Average Kills and Flag Activities on *openfire_lowgrens*

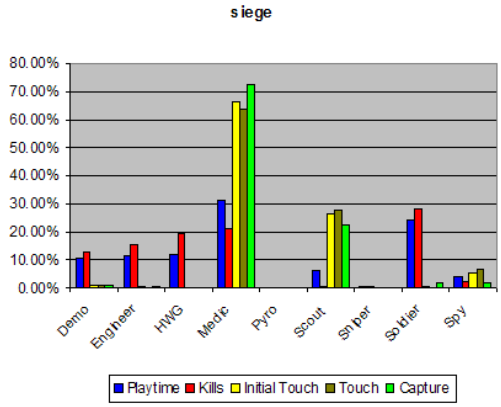

Figure A.5: Average Kills and Flag Activities on *siege*

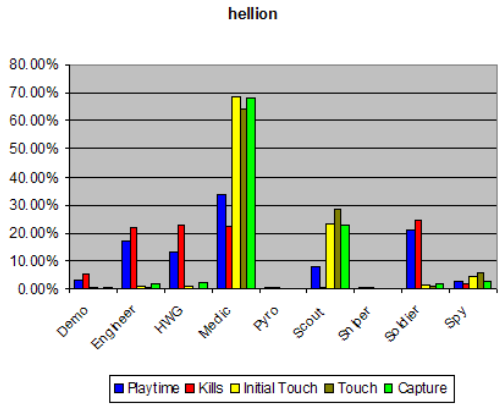

Figure A.6: Average Kills and Flag Activities on *hellion*

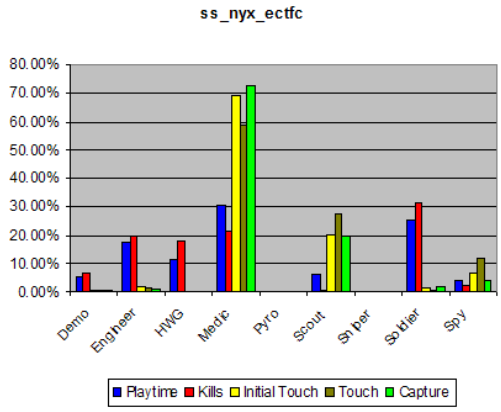

Figure A.7: Average Kills and Flag Activities on *ss_nyx_ectfc*

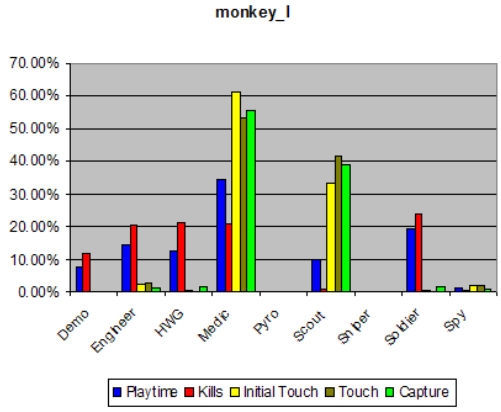

Figure A.8: Average Kills and Flag Activities on *monkey_l*

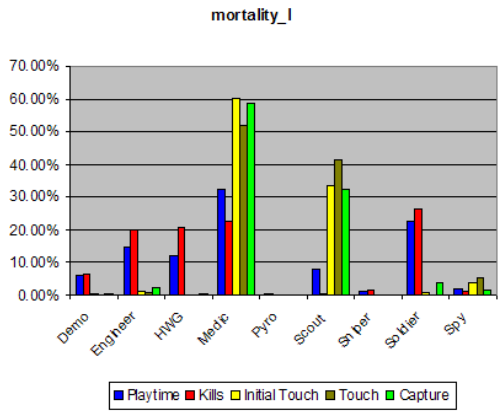

Figure A.9: Average Kills and Flag Activities on *mortality_l*

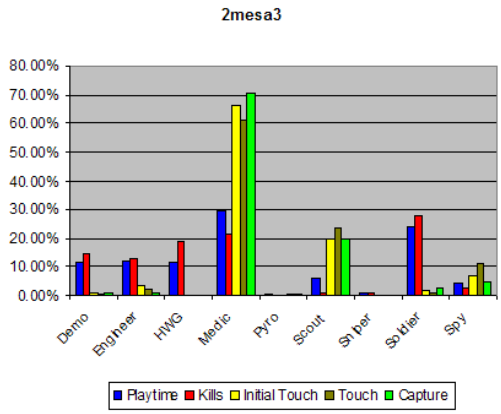

Figure A.10: Average Kills and Flag Activities on *2mesa3*

B PHP Script to Extract Player Information

This script is an extract from the complete script which was used to gain information from a HTML page that conatins data for a single player.

```
//open player file
$fh = fopen("$dataDir/$gameName/".$team_file[$i], 'r');
$contents = fread($fh, filesize("$dataDir/$gameName/".$team_file[$i]));

//get classes played by player with playtime
$subStr = getSubstring($contents, "Classes:<br /><small>",
"</small></td></tr></table>");
preg_match_all("|(<img width=\"48\" height=\"96\" src=\"(.*?)\" title=\"(.*?)
\((0:\d\d:\d\d) - 0:\d\d:\d\d\)\" /><br />(0:\d\d:\d\d))|", $subStr,
$single_classes, PREG_SET_ORDER);

//step through all possible classes
for($x = 0; $x < count($single_classes); $x++) {
$cID = $header_map[$single_classes[$x][3]];
$query = "INSERT INTO player_classes (player_id, class_id, start, time)
VALUES ($playerID, $cID,'".$single_classes[$x][4]."' ,
'".$single_classes[$x][5]."')";
mysql_query($query);
}

//get kills for active player
$subString = getSubstring ($contents, "\nKills:<br />", "width: 100%;\"
/>Deaths");
//get player names
preg_match_all("|.html\">(.*?)</a>|", $subString, $single_kills,
PREG_SET_ORDER);
```

Bibliography

blarghalizer.org (2007). The blarghalizer - tfc server log parser. URL: `http://www.blarghalyzer.org/ParsedLogs.php` [last checked: 17/09/2007].

Campbell, M., Jr., A. H., and Hsu, F.-H. (2002). Deep blue. *Artificial Intelligence*, 134 no. 12:5783.

Champandard, A. (2003). *AI Game Development: Synthetic Creatures with Learning and Reactive Behavior*. New Riders Games.

Chellapilla, K. and Fogel, D. (1999). Evolving neural networks to play checkers without relying on expert knowledge. *IEEE Trans. Neural Networks*, 10(6):1382–1391.

CSAI (2007). The university of alberta counter-strike ai project. URL: `http://ircl.cs.ualberta.ca/games/cs/` [last checked: 26/09/2007].

Dahl, F. A. (2001). A reinforcement learning algorithm applied to simplified two-player texas hold'em poker. In *Proceedings of the 12th European Conference on Machine Learning*. Springer-Verlag.

FoxBot (2007). Foxbot for tfc. URL: `http://foxbot.planetfortress.gamespy.com/down.htm?forum/news.php` [last checked: 17/09/2007].

Fuernkranz, J. (2001). *Machine learning in games: A survey*, page 1159. Nova Biomedical.

Gasser, R. (1996). *Solving Nine Mens Morris*, page 101113. Cambridge University Press, Cambridge, MA.

Gold, A. (2005). Academic ai and video games: A case study of incorporating innovative academic research into a video game prototype. In *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games*. IEEE.

Half-Life, P. (2007). Team fortress classic overview. URL: `http://planethalflife.gamespy.com/View.php?view=TFCGameInfo.Detail&id=11&game=6` [last checked: 17/10/2007].

Hammond, K. (1989). *Case-based Planning: Viewing Planning as a Memory Task*. Academic Press, Boston, MA.

Kennerly, D. E. (2007). Better game design through data mining. URL: `http://www.gamasutra.com/features/20030815/kennerly_01.shtml` [last checked: 26/09/2007].

Kirby, N. (2003). *Getting around the Limits of Machine Learning*, pages 603–611. Charles River Media, Hingham, MA.

Kolodner, J. (1992). *Case-Based Reasoning*. Morgan Kaufmann.

Korb, K. B. and et al., A. N. (1999). Bayesian poker. In *UAI'99 - Proceedings of the 15th International Conference on Uncertainty in Artificial Intelligence*, pages 343–350.

Krulwich, B. L. (1993). *Flexible Learning in a Multi-Component Planning System*. PhD thesis, The institute for the Learning Sciences, Northwestern University, Evanston, IL.

Kuhlmann, G., Knox, W. B., and Stone, P. (2006). Know thine enemy: A champion robocup coach agent. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, page 14631468.

Laird, J. and van Lent, M. (2001). Human-level ai's killer application: Interactive computer games. *AI Magazine*, Summer 2001:1171–1178.

Lee, K.-F. and Mahajan, S. (1990). The development of a world class othello program. *Artificial Intelligence*, 43(1):21–36.

Miikkulainen, R., Bryant, B., Cornelius, R., Karpov, I., Stanley, K., and Yong, C. H. (2006). *Computational intelligence in games*. IEEE Computational Intelligence Society, Piscataway, NJ.

Mitchell, T., Keller, R., and Kedar-Cabelli, S. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1(1):47–80.

Mueller, M. (2000). *Generalized thermography: A new approach to evaluation in computer Go*, page 203219. Universiteit Maastricht, Maastricht.

Muggleton, S. (1990). *Inductive Acquisition of Expert Knowledge*. Turing Institute Press, Addison Wesley.

Nareyek, A. (2004). Computer gamesboon or bane for ai research? *Knstliche Intelligenz*, 18(1):4344.

Nareyek, A. (2007). Game ai is dead. long live game ai! *Intelligent Systems*, 22(1):9–11.

Quinlan, J. R. (1983). *Learning efficient classification procedures*, pages 463–482. Tioga, Palo Alto.

Rubin, J. and Watson, I. (2007). Investigating the effectiveness of applying case-based reasoning to the game of texas hold'em. In *Proc. of the 20th. Florida Artificial Intelligence Research Society Conference (FLAIRS)*. AAAI Press.

Russell, S. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice-Hall.

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211229.

Samuel, A. L. (1967). Some studies in machine learning using the game of checkers.ii - recent progress. *IBM Journal of Research and Development*, 11(6):601617.

Schaeffer, J. (2000). The games computer (and people) play. *Academic Press*, 50:189–266.

Schaeffer, J. (2007). Checkmate for checkers. URL: `http://www.nature.com/news/2007/070716/full/070716-13.html` [last checked: 09/09/2007].

Schaeffer, J., Culberson, J., Treloar, N., Knight, B., Lu, P., and Szafron, D. (1992). A world championship caliber checkers program. *Artificial Intelligence*, 53 no. 2-3:273290.

Schaeffer, J. and et al., D. (1999). Learning to play strong poker. In *Proceedings of the ICML-99 Workshop on Machine Learning in Game Playing*.

Schmidt, M. (1994). Temporal-difference learning and chess. Technical report, University of Aarhus, Aarhus, Denmark.

Shannon, C. E. (1950). Programming a computer for playing chess. *Philosophical Magazine*, 41:265–275.

Shao, J. (2007). Report highlight for dataquest insight: Gaming pc market dynamics and outlook.

Sheppard, B. (2002). World-championship-caliber scrabble. *Artificial Intelligence*, 134:241275.

Spohrer, J. (1985). Learning plans through experience: A first pass in the chess domain. In *Intelligent Robots and Computer Vision, Volume 579 of Proceedings of the SPIE - The International Society of Optical Engineering*, pages 518–527.

Spronck, P. (2005). *Adaptive Game AI*. PhD thesis, Maastricht University, the Netherlands.

Tesauro, G. (1992). Temporal difference learning of backgammon strategy. In *Proceedings of the 9th International Conference on Machine Learning 8*, pages 451–457.

TFLeague (2007). Tfleague - rules. URL: `http://www.tfleague.com/rules.php` [last checked: 17/09/2007].

Times (2007). Online games revenues hit $1bn. URL: `http://business.timesonline.co.uk/tol/business/industry_sectors/technology/article1544234.ece` [last checked: 14/09/2007].

Tozour, P. (2002). *The Evolution of Game AI*, pages 3–15. Charles River Media, Hingham, MA.

Truscott, T. (1978). The duke checkers program.

Tunstall-Pedoe, W. (1991). Genetic algorithms optimizing evaluation functions. *ICCA Journal*, 14(3):119–128.

Tveit, A. and Tveit, G. B. (2002). Game usage mining: Information gathering for knowledge discovery in massive multiplayer games. In *Proceedings of the International Conference on Internet Computing, Session on Web Mining*, pages 636–642. CSREA Press.

Valve (2007). Hl log standard. URL: `http://developer.valvesoftware.com/wiki/HL_Log_Standard` [last checked: 17/09/2007].

van Tiggelen H. J., A. and van den Herik (1991). *ALEXS: An optimization approach for the endgame KNNKP(h)*, pages 161–177. Ellis Horwood, Chichester.

Wikipedia (2007). Team fortress classic. URL: `http://en.wikipedia.org/wiki/Team_Fortress_Classic` [last checked: 17/09/2007].