The Personality Module

How Can Personality Affect Agents In a Turn-Based Strategy Game?

THESIS

By Arild Johan Jensen and Håvard Nes

Department of Information Science and Media Studies

University of Bergen

Spring 2008

Acknowledgements

Firstly and mostly we must thank Aleksander Krzywinski and Arne S. Helgesen for their excellent work on the Caeneus Agent and StateCraft and Weiqin Chen for her supervision and guidance.

Secondly, we would like to thank all the testers for spending so much of their time with our game and in discussions with us. In addition to Arne, the testers were Ronny Simon Mo, Jarl Helge Utvik, Mannaf Haque, Espen Halvorsen and Dag Christian Weløy. Tomas H. Ekeli, who gave us the idea to use StateCraft as test environment also deserves our gratitude.

Last, but not least, Arild Johan would like to thank his significant other, Hanne Beth Borge, for her support and patience.

Table of Contents

List of figures

List of tables

List of code excerpts

Preface

To prove our hypotheses that personality can be perceived in an agent's logical actions as well as from social engagement, and to observe how different personalities can affect an agent's decision-making, we needed a test environment where an agent's decisions and actions could be manipulated to simulate different personality features. It would also be preferable if these decisions and actions could be experienced by humans. Games have always been a popular testing ground for artificial intelligence. Both of us have played computer games since childhood, so naturally, we felt the attractiveness of game programming. We also judged that getting test subjects to a game would be far easier than to any other application. Upon deciding this, we had to consider the type of game we wanted to work with. Action, arcade, adventure, sports games and the likes were quickly excluded, as agents in that kind of games usually are in the form of an avatar, which would interfere with the tester's perception of the agent's actions. That basically left us with strategy games. A game with controlled from a "gods-eye-view", where a player or agent controls units or pieces on a game board, was considered the optimal choice for our purposes. We therefore decided on using a strategy game, whether it be turn-based or real-time, as test environment for personality in an agent.

After landing on a game type, or rather, two game types, we found ourselves with four options, which are listed and discussed below.

1. To create our own game from scratch.

   While this option clearly would present us with the largest programming task, it would also leave us free to shape our test environment to our needs. We could create the game type we wanted, the agent we wanted and the player interface we wanted. We imagined a simple, turn-based game, where the agent would perform tasks clearly influenced by its personality and easily interpreted by the player, through a simplified user interface.

2. To develop a game using a third-party game engine.

Finding an open-source game engine on the Internet is no difficult task. Whether you want to develop a first-person shooter, a real-time strategy, a turn-based strategy game or almost any other game, a quick search in an open-source community, like SourceForge [1] would provide you with suggestions to game engines possibly suiting your needs. Choosing this option would present us with the game mechanics, allowing us to concentrate on creating the game world, our agent and the user interface, however, it would require us to learn this engine and, perhaps, the programming language in which it is developed.

3. To develop an agent to an existing game.

   Many open-source games are developed purely for multiplayer purposes, thus omitting artificial intelligence completely (e.g. in first-person shooters) or, in strategy games, only including artificial intelligence on unit level. In comparison with the options above, choosing to develop an agent for an existing, complete and playable game would mean considerably less "outside the project"-programming, allowing us to fully concentrate on the agent. Even so, we would be forced to spend time learning the code and possibly creating an API for our agent. Much likely, we would also have to create an agent much more complex than needed for our purposes.

4. To manipulate an existing agent in an existing game.

   This last option would limit our programming efforts to impose personality on an existing agent. This would imply that we would be able to focus all our programming on the personality bit. On the other side, we would have read a lot of code to understand both the game and the agent, and, as in the option above, be restricted to the game's limitations and forced to follow its rules.

We quickly disregarded the first two options, considering them to be too time-consuming. That left option three and four. We found them similar enough to include both categories in our search. Thus, the search for the game which we were to base the whole project on began.

In consent with our conclusions above, we identified several requirements to the game we were searching for.

The game…:

- had to be open-source
- had to be written in an object-oriented programming language, preferably Java
- had to be a strategy game

Finding suitable games turned out to be a far more time-consuming task than anticipated. After trying and rejecting tens and tens of game, we were left with really just one promising candidate, Dark Oberon [2], a real-time strategy game experienced by Blizzard's WarCraft, but with graphics created from shots of Plasticine figures. The game is written in C++ and is only released as a multiplayer game, which means the only AI in the game is on unit level. At first glance, it was exactly what we were looking for. We downloaded the source code and sent an email to one of the developer, presenting our thoughts and ideas. The response was positive, we were very welcome to join the project as developers and they liked our plans on creating an AI player for the game. Marian Cerny, one of the developers, warned us, however, in his/her reply to us that they had learned C++ by programming Dark Oberon, the code being thereafter: *"The present code is quite ugly and the design even more."* One of the challenges for us was that the game had no client-server architecture, which meant that there was no interface for us to work against. The project was in the middle of a heavy refactoring task, but it had started too late and the progress was too slow for us to wait for. Combined with our lack of experience in C++, we eventually decided to discard Dark Oberon and continue our search. However, after rejecting the only candidate we had found so far, our hopes for finding a better option were small.

The setback with Dark Oberon had us rethinking our entire plan when in a conversation about our master's projects, Tomas Ekeli, a fellow student, reminded us of Arne S. Helgesen and Aleksander Krzywinski's thesis about the Caeneus Architecture [3], presented to us in a seminar in the course Advanced Artificial Intelligence (INFO 391). For their thesis, Arne and Aleksander had developed a software implantation of the board-game Diplomacy and an agent playing it. Diplomacy was written in Java, had a clear client-server architecture and was well documented through their thesis. In addition, Aleksander was still with the Institute, and when enquired, he offered to help us getting started. StateCraft [4], as they called their implementation, fit perfectly in category four, t*o manipulate an existing agent in an existing game*, in our analysis and it also fit all of our requirements. Diplomacy and StateCraft will be presented in Chapter 3.

Abstract

The computer game industry has grown immensely in the last decade and artificial intelligence is an increasing part of game development. Personality in agents has been studied, but is not widely implemented in computer games. We developed the Personality Module, a component adding personality to agents, and implemented it in StateCraft, a software version of the board-game Diplomacy. This was evaluated both through simulations in all-agent games and through human testing. The results show that agent personality in a turn-based strategy game like StateCraft both affects the agents' performance and the gameplay experience.

1 Introduction

Counting back to the days of Pac-Man and perhaps beyond, computer players have always reacted emotionally to events in computer games. If a computer-controlled opponent crashes into you in a car game, odds are that you feel mad at that opponent. Perhaps you get some sort of satisfaction by retaliating on the player or car, without concerns about the impact it will have on your game. Or if an opponent in a strategy game conquers one of your most important cities or bases, would you not feel the urge to teach this opponent a lesson by counter-attacking, even though that might not be the right thing to do at the moment? Everyone who has ever played a computer game against a computer-controlled opponent has at one point or another been emotionally affected by the opponent's actions. Whether it be arcade, simulation, strategy, or some other genre, one often perceives a computer opponent to have a mind of its own, with feelings and desires. This is, of course, not the case. A computer-controlled opponent is just that; computer-controlled. It is not a living being, it cannot feel or desire, even though some AI researchers are trying to represent belief, desire and intention in agents.

The computer game industry is a multimillion dollar business and creating a computer game is a huge undertaking. Up to several hundred people can be involved in developing a computer game, in various fields like sounds and music, graphics, physics, marketing and many more – and *artificial intelligence (AI)*. AI is becoming a more and more important part of game development and the audience is becoming more and more demanding. The customers want (in most cases) a tough (but not too difficult) and intelligent challenge. It does not help with fine graphics if the computer controlled opponents walk into walls or the other cars in the race drives in the wrong direction.

We argue that if you do not react emotionally on a computer game, you will not enjoy it, or, at least, *if* you react, your game experience will be vastly improved. If you perceive the opponent to have some sort of personality, you are more likely to react emotionally to its actions, potentially enjoying the game more. To prove this, we have created a personality module to the game StateCraft a software version of the board game Diplomacy [5]. This module only affects the opponents'/agents' actual actions in the game, *not* their social interactions with or the appearance to the other players. Further, we wish to prove that the agents' actual actions, *not their social interaction*, also can create a perception of personality. More specifically, we focus on the following two research questions:

1. Can different personality traits affect the agents' performance in a strategy game?

2. Can agent personality improve the game experience of the players?

The thesis is organized as follows. Chapter 2 discusses theories and related research within Artificial Intelligence and computer games. Chapter 3 presents Diplomacy, the Caeneus Architecture and StateCraft. Design and implementation of the personality module and its integration into StateCraft are described in Chapter 4. Chapter 5 evaluates the module and Chapter 6 concludes the thesis and discusses possible further development.

2 Artificial Intelligence and Computer Games

2.1 Artificial Intelligence

Homer's bronze tripod servants in the Illiad [6] are some of the first known artificial beings in literature. Since then, artificial beings and artificial intelligence have been popular elements in entertainment, both in books, films and, as we shall see later, computer games. In Baum's *The Wonderful Wizard of Oz*, we meet Tik-tok, a mechanical man that runs on clockwork springs[7]. It can think, act and speak, but feels no emotions. In science-fiction movies, robots, computers or machines taking over the world is a popular plot. In WarGames, a 1983 American movie, a nuclear weapon control system acts on its own and very nearly launches a full-scale nuclear attack on the Soviet Union. In the well-known Terminator trilogy there is an open war between humans and machines of control of the planet, while in the Matrix trilogy, the machines have already taken over the planet, using humans as an energy source.

Artificial Intelligence is a wide and uncertain term, with many different interpretations. On one side, some argue that anything that reacts on input, like a thermostat responding to temperature changes, can be said to have artificial intelligence, while the other extreme means that for something to be called artificial intelligent, it must be a copy of human intelligence, including emotions, irrationalities and errors. Most people in the AI community place their definitions somewhere in between, as do we. Inspired by other definitions [8], [9], our definition of AI, used throughout this thesis, is "*a computer program (agent) that controls its own state and goal(s) and gathers and uses any information available to try and reach its goal(s)*". This means that when we use the word *agent*, it implies the characteristics above. An agent can act in a purely virtual environment or in a real environment, such as controlling a robot exploring the surface of Mars. The overlying goal is, of course, in all cases predefined by a human programmer. In this section, we will briefly describe the history of AI, from Homer's abovementioned tripods to today's state-of-the-art technology, before we delve deeper into some of the techniques used today. Finally, we discuss the link between AI and other sciences, mainly psychology.

Artificial intelligence or AI “is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence, but AI does not have to confine itself to methods that are biologically observable.” [10]. It is widely believed that the term AI was first coined by John McCarthy at the Dartmouth Conference in 1955 in the famous paper entitled *“A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence”* by John McCarthy, M. L. Minsky, N. Rochester, and C.E. Shannon [11].

The general theory for AI is mainly based on several characteristics found in normal human being such as deduction, reasoning, problem solving, knowledge representation, planning, motion and manipulation, and perception, amongst others. When solving puzzles, playing board games or make logical deductions a normal human being would go through the process of conscious and step-by-step reasoning. These traits were successfully imitated through algorithms developed by early AI researchers. Nonetheless, AI researchers have yet to unravel the methods of replicating human traits in solving problems using unconscious reasoning. In order for the machines to solve problems, it needs extensive knowledge about the world e.g. objects, properties, categories and relations between objects; situations, events, states and time; causes and effects. Most knowledge is difficult to be represented due to three reasons; default reasoning and the qualification problem [12], unconscious knowledge and the breadth of common sense knowledge. In solving planning problems, the intelligent agent must have the ability of setting an objective and achieving it, thus it needs a way to visualize the future; i.e. having a representation of the state of the world and be able to make predictions about how their actions will change it. The agent must also attempt to determine the utility or the ‘value’ of the choices available to it. For robots to be able to manipulate objects and to navigate, it requires intelligence such as machine’s perception which is a machine’s ability to use input from sensors (for example microphones and cameras) to see the world and computer vision which is a machine’s ability to analyze visual input. These abilities can be found in Honda’s ASIMO robot [13], which uses sensors and intelligent algorithms to navigate stairs and avoid obstacles.

When asked if AI is about simulating human intelligence, McCarthy answered that this is sometimes but not always or even usually the case. McCarthy elaborated that “on the one hand, we can learn something about how to make machines solve problems by observing other people or just by observing our own methods. On the other hand, most work in AI involves studying the problems the world presents to intelligence rather than studying people or animals. AI researchers are free to use methods that are not observed in people or that involve much more computing than people can do”. That knowledge and information could be programmed into a computer: was one of two schools of thoughts that came to dominate AI research. The other view was to express intelligence as formal logic, championed by McCarthy. However, in the last decade or so, a radically different approach which uses statistical tools rather than human-like reasoning has contributed to AI’s great success. The concept of “search space”, innovated by the late Drs. Newell and Simon, is a way of thinking about possible actions and reactions, establishing an objective, considering all of the possible actions that could be taken and then evaluating which actions are most likely at each step.

One of the best-known agent architectures is the Procedural Reasoning System. This was developed by Georgeff and Lansky [14], and is an example of a popular paradigm known as the Belief-Desire-Intention (BDI) approach [15]. This architecture views the system as a rational agent having certain mental attitudes.

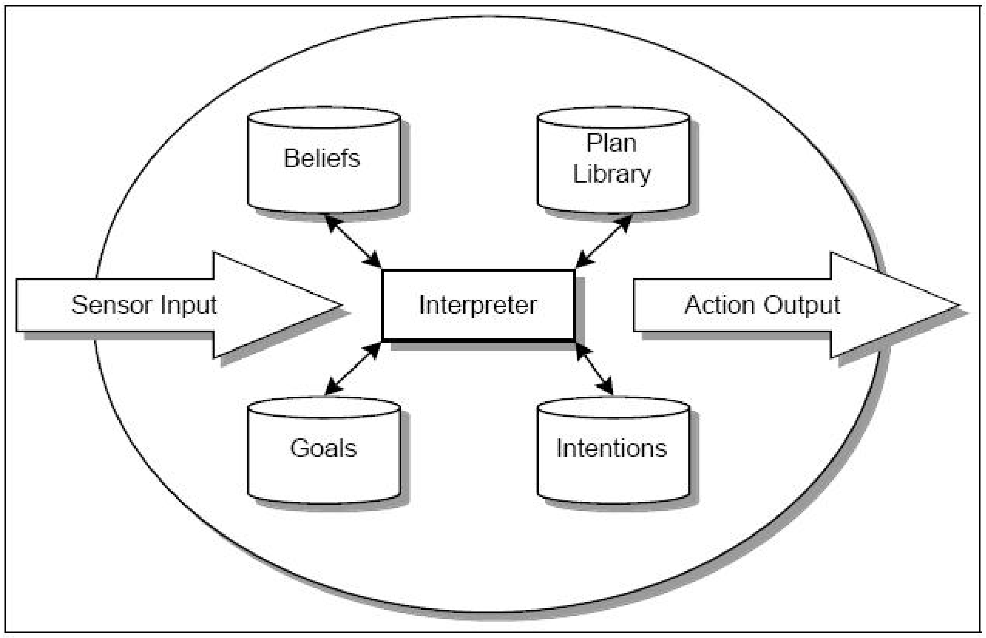


Figure 1: Procedural Reasoning System (PRS), A BDI Agent Architecture

Figure 1 shows the BDI architecture that typically contains four key data structures:

- *Beliefs* represent the informational state of the agent. This means its beliefs about the world (both itself and other agents).
- *Desires* (or goals in this system), represent the motivational state of the agent. They represent objectives or situations the agent would like to achieve or cause.
- *Intentions* represents the deliberative state of the agent. This means what the agent has chosen to do.
- *Plan Library* is a set of plans. These specify courses of action that may be undertaken by an agent in order to achieve its intentions. The plans can also include other plans.

The BDI model is very useful for developing formal models or agents, developing a deep model of agent communication and inferring an agent's internal state from its behaviour.

Another model of personality and emotion which is often used is the Five Factor Model (FFM). This model is based on the Big Five personality traits which are often used in psychology. The Big Five personality traits are five broad factors or dimensions of personality discovered through empirical research [16]. These five factors are: Openness, Conscientiousness, Extraversion, Agreeableness and Neuroticism (OCEAN). The five factors are held to be a complete description of personality. This model is ideally suited to the task of creating concrete models of personality with which to enhance the illusion of believability in computer characters [17].

By adding personality to characters, people often find the generated dialogues entertaining and amusing, even though it might not be the developers intention to make the use of humour.

2.2 Computer Games

Computer games are an intrinsic part of a youngster's lifestyle. With new advancements in technology, these games open a window to a world of fantasy, which enable people to assume various roles and undertake adventures denied to them in real life. However these games have an interesting history, a past which holds the key to its invention and development.

Back in 1952, A.S.Douglas, a Cambridge university student, designed a version of Tic-Tac-Toe as a part of his Ph D degree. The game was set on an EDSAC vacuum - tube computer with a cathode ray display. It was not until 1958 that the first video game was designed by William Higinbotham. "Tennis for Two" was, as the name suggests, set on an amplified tennis court for two players who had to return the ball to one another. In 1962, came 'Space war', the first computer game where the designer Steve Russell and his team used a MIT PDP-1 mainframe to design their game. "Space wars" marked the beginning of an era of computer games. The two-player game comprised space ships and firing photon torpedoes. During the mid-sixties when computers had not yet found their niche and were restricted to research centres, the game was found in every research computer and enjoys popularity to this day. However, Russell did not benefit from the invention since MIT, the institute, which had appointed him, was disappointed that the computer was not used to develop scientific programs.

In 1971, Nolan Bushnell and Ted Dabney created the arcade game “Computer Space”. This was the first arcade game and was based on Steve Russel’s earlier game of “Spacewar!”. The game was unsuccessful due to its difficulty, but was a landmark, being the first mass-produced video game and the first offered for commercial sale. Later, Bushell and Dabney started Atari Computers, and re-released the arcade game “Pong” as a computer game.

During this period, games were usually played by typing commands on the keyboard. But in the 1980s, graphics began taking precedence and games with basic textual commands and graphics, such as “Pool of radiance” and “Bard’s Tale” were designed. In 1983, the arcade game industry experienced a set-back, due to poor quality games. Popular games like E.T and Pac-Man failed miserably. This boosted the popularity of computers and soon enough low cost computers began replacing arcade games.

There were also many high quality graphical interfaces for computers, which could be utilised with the help of the computer mouse. With the popularity of Commodore Amiga computers in 1985, sales saw an upward trend and attempts were made to work on improving its features. Other technological innovations in these years were sound cards, which added to the audio-visual experience of computer games. The 90s, introduced trend setting games like “Wolfenstein 3D” which popularised the first person shooter game genre, as did “Doom”, which set the trend for 3D graphics in games.

The 90s can be termed as the boom time for computer games since the Real Time Strategy (RTS) genre games made an entry with Dune II. The genre was further popularised by Warcraft: Orcs & Humans in 1994 and games like Warcraft II: Tides of Darkness in 1995 gave rise to multiplayer capabilities in the RTS genre.

With broadband connections becoming cheaper in the 1990s, online gaming became very popular. A player usually used a modem or LAN to play RTS games, where the player gets a top down perspective of the battle zone complete with 3D animation. The player can manipulate the game through clicking and dragging the mouse in a real-time environment, as opposed to a turn-based game, where the player has to wait for his/her turn. With Internet becoming more accessible than ever, these games now had their own native support and soon enough players around the globe began interacting with each other.

In 1997, Massively Multiplayer Online Games (MMOG) made an entry with role-playing games like Ultima Online. These MMOG games comprise many genres, including role-playing, flight combat simulation and real time strategy.

Some of the more popular western titles include Blizzard’s The World of Warcraft, Lineage, Lineage II and Runescape. There are many games, which have found their way from movies and literature into the virtual world as well. EA titles such as James Bond series and the more recent Harry Potter series have captured the imagination of gamers just like its acclaimed books and movie adaptations.

Computer games thus shares its history with video games and also with the advancement in computers. With each era, as computers became more superior and grew in capacity, it gave rise to new features. This was features like designing a overall experience of excitement and adventure of computer games.

2.3 AI in games

The first video game with AI, “Tennis For Two” was released in 1958 by Willy Higinbotham. Some years later Steve Russell from MIT released “SpaceWar”, the first game to run on a computer. In 1970, Atari released a home video game system, and the first 3-dimensional game in 1980. This game was used by the US government for training military forces. Later, Nintendo Entertainment System (1984) and PlayStation (1995) were released.

In the broadest sense, most games incorporate some form of AI [18]. The computer allows us to create and interact with completely new virtual worlds that previously existed only in our imagination. The computer is unique in its ability to create an almost infinite variety of challenges at many different time scales. That could be epic quests that take weeks to achieve, or moment to moment actions where we know that if we play just a few more minutes, we will achieve new powers or unlock hidden secrets. The challenges usually arise directly from our interaction with the virtual environment. But it is not just the challenge of a human against the world that engages us. It is also the challenge of competing and cooperating with other intellects.

From the first computer games, the challenge has often been human against human. What is unique to computer games is that we now have the technology to create games where the challenge of playing against other intellects is not restricted to playing against other humans. Computers cannot only create challenging environments, but also create challenging artificially intelligent (AI) characters [19]. AI characters also allow game developers to create games where the challenges and experiences are not just from competition, but also from the social interactions with characters in populated virtual worlds. Artificial intelligence in games is usually used to create the player’s opponents. In recent years there have also been developed AI supporting the players, not only opposing them.

The history of AI in games started in the mid-sixties. Before this, it was only possible to play two-player games against other humans or games where the non-human objects were hard-coded. One example of this kind of game is “Space Invaders”. These aliens that descend on the human player are hard-coded. The aim of this game is to shoot all the aliens before they reach the bottom of the screen. The earliest game with real artificial intelligence released in 1972 by Atari was “Pong”. In this game, the computer tries to block the ball from scoring by hitting it back to the player. This was done by calculating where the ball would cross the line and move the bat to that spot. The player can select between different difficulty settings. If the difficulty level is set low, the computer might move to the wrong spot or not move fast enough to the spot. This was the first computer controlled opponent in arcade games.

For many years the games were simple and were played with a second player instead of computer opponent. The other games Atari released in the seventies, “Pursuit” and “Qwak” were also using a very basic AI. “Pursuit” put the player in the role of a World War I flying ace, where the player has to shoot down enemy planes. In “Qwak” the objective is to play a duck and try to collect all the fruit, killing off the enemies and get the highest score without losing all your lives.

When microprocessors were introduced, this improved the computation and made it possible to improve the AI. During the ‘80s, more complex games were released. One of them was Pac-Man, who created a new genre and appealed to both males and females. The goal of this game is to eat pellets and try to avoid being tagged by the four ghosts (Blinky, Pinky, Inky and Clyde). These ghosts’ goal is to catch Pac-Man by touching him. The AI in the game has given personality to the ghosts:

- Blinky is the one on your tail and can be though of as your shadow
- Pinky is just as fast as Blinky and works with him to ambush and cut you off
- Inky is unpredictable. Sometimes he follows you, and sometimes he goes away from you

- Clyde is very slow. He is always going somewhere on his own. But he sometimes successfully cut you off, but almost never outright chases you

In the 90's first person shooter and role-playing games were also given AI. This gave the games the possibility for evolution and learning. "Creatures" and "Black & White" are two examples of games that were released in the 90's with this kind of AI. In 1997, IBM released a chess-playing computer. On 11th of may 1997 this machine won a six-game match against world champion Garry Kasparov. This victory also surprised many in computer science as well. However, Garry Kasparov accused IBM of cheating and wanted a rematch, but IBM declined and retired Deep Blue.

From the year 2000 and until now, the game AI is developed to use less cheating. The AI has also improved the characters by making them more aware and able to collaborate better.

The different game genres present the player with different environments, different challenges, and different roles for AI to enhance the game experience. The most common roles for AI to play in games are: partners, support characters, enemies, strategic opponents, low-level units, and commentators [20]. Some examples of where AI often is used are:

- Car games: AI is used to keep races close
- Shooter games: AI is used for non-player characters to react on players movement and actions
- Sports games: AI is used for commentators, to help the player and to simulate the opponents
- Real Time Strategy games: AI is used for generals who work on pacing

Until now, the development of good AI for games has not been prioritized [21]. This is because most games have only been able to allocate limited processor resources to the game AI (up to the year 2000, typically about 10% of processor cycles) compared to other aspects of the game. However the speed of graphics cards has been increased much faster than the Central Processing Unit (CPU) last years [22]. More of the graphics processing and game logic moves from CPU to Graphic Processing Unit (GPU), and this frees a lot of resources from the CPU. These CPU resources are then available for the game AI.

Because of these limited CPU resources many AI have used cheating. This is a technique which is very processor efficient and could be successful. However, if the cheating becomes obvious to the player, it will destroy the game playing experience [23]. The cheating most AIs have used includes cases where the AI plays with a different set of rules and is given extra units or resources or additional information about the map or the human players position. This makes the AI predictable and easy to beat once the player has found their weakness and it can also be perceived as unfair by the player.

In most action games, there is a multiplayer mode where each player (human or computer) is fighting for itself. In these cases, the computer enemies should be as challenging as a human without obvious faults or disadvantages. It is easy to make the computer enemies challenging by giving them superhuman reaction times and aiming skills. With more CPU recourses available for AI, cheating can be reduced and the game developers can use other AI techniques instead. This will make a much more believable game AI. In many cases, multiple levels of AI are needed to challenge the range of human players who vary from novice to expert.

In action games, to keep the realism high but still allow the player to win, the AI characters could always give their position away by talking or making noise when they hear or see a glimpse of the player, saying comments such as “What was that?” or “Who goes there?” This alerts the player about there is one or more enemies nearby, and gives the player a chance to get that important first shot. In other situations the enemy characters always miss on the first couple of shots, which further help the human player beat the AI enemy.

Games have been a popular choice for research in AI. One of the reasons is because games can simulate the real world. Many AI techniques have been developed in academic research, but relatively few have been used in development of commercial games. The military have also used AI in training for example pilots. To train four pilots to fly an attack mission can require over 20 planes, air-controllers and many support personnel [20]. By using computer-generated forces, the military can bypass a lot of these costs.

When AI is used to simulate the real world, it has to be a human-level AI. This might also be the case if the aim in a game is to simulate playing against other humans.

Players value a great AI. But it does not always have to be a human-level AI. One might be tempted to think that an AI should always be constructed to model a human player as accurately as possible. Sometimes that is appropriate, but as we review each of the roles an AI character can play, we will see that often human-like behaviour must be sacrificed because human-like behaviour is still beyond the state of the art or because the game is more fun without it. In computer games, the real goal of the AI character often is to enhance gameplay and most of the time that entails being an opponent that fights up until then end and then loses. According to Nareyek, “the goal in game AI is not to compute the most optimal behaviour for winning against the player. Instead, the outcome should be as believable and fun as possible.” [22].

2.3.1 AI in Strategy Games

Strategy games usually cast a player in charge of military units, although the larger part also includes non-combat units like resource gatherers, controlled from a “gods-eye-view”. The player usually controls a multitude of units, their types depending on the world and time the game is set in. The most well known turn-based strategy games are the Civilization series. Real-time strategy (RTS) games include re-enactments of different types of battles: historical (Close Combat, Age of Empires), alternative realities (Command and Conquer), fictional future (StarCraft), and mythical (Warcraft, Myth).

A good strategy for how the player defeats the opponents is a very central aspect of an AI in a strategy game. However, it is also very important to be able to realize when the strategy does not work out against the opponents and to be able to change the strategy in a situation like this. The AI in turn-based strategy games should be able to predict a competitor’s high-level strategic choices, for instance, when he will go to war.

In turn-based strategy games two or more opponent are controlling their own empire or civilization. Controlling these civilizations includes issuing commands to units, allocating resources and constructing new units. In turn-based strategy games like the Civilization series or Call To Power II hundreds of different kinds of units, technologies, buildings, resources and terrain are important parts of the game. The AI has to reason with these objects while planning the strategies for the AI players. AI in turn-based strategy games is famous for cheating to provide an adequate level. This means for instance that if a city is attacked the AI can use defenders that are not constructed according to the rules of the game. This ends up working acceptably in these games since the AI includes ways to make sure the player does not feel that the AI has an advantage.

There are, of course, a large number of strategies an agent (or a human player) can use. In RTS games, there are three main strategies that are most common. The first strategy is fast attack, where one tries to beat the opponents very early in the game by rushing out primitive and cheap units. The second is to focus research on combat technology to get better units before the opponents. The third is to focus on the economy early in the game, to later on be able to out-build the opponents.

In most strategy games, the players are faced with problems of resource allocation, scheduling production, and organizing defences and attacks. AI is used in two roles: to control the detailed behaviour of individual units (RTS) and as a strategic opponent that plays against the human [20] (RTS and turn-based). The AI requirements of the individual units differ from the enemies of action and role-playing games. Units must often navigate through complex outdoor environments on their own as well as follow orders generated by the human or strategy level AI. Units can be ordered to move in a variety of formations, attack enemy units and buildings, defend friendly units or buildings, and perform a number of special actions.

Strategy games almost always include non-military units that are used to gather resources, build new buildings, and repair damaged buildings. The resource allocation and unit control tasks cannot be controlled independently. The type of resources gathered and how they are spent must be guided by the needs of the strategic opponent's unit control strategy. Similarly, the unit control strategy must take into account the available resources to avoid selecting a strategy that requires unavailable resources.

In the past, most game developers resorted to static, pre-defined strategies and cheating as mentioned above, to minimize the demands on the strategic opponent. The artificial intelligence for the strategic opponents generally took the form of a single hard-coded strategy created by the game designer when the game level was created. For RTS games this would be a single attack plan, usually a mass frontal assault, executed periodically (for example every 10 minutes) with no regard for the human player's behaviour. If the periodic attacks threatened to wipe out the human player the strategic opponent in some games would reduce the attacking force accordingly.

To implement these pre-defined strategies the strategic opponents usually cheated in a number of ways. One form of cheating allowed the creation of extra units or resources when having no money or factories. Another form of cheating gave the strategic opponent complete knowledge of the map and the locations of the player's forces. This freed the strategic opponent from exploring the map and scouting the enemy to obtain this information. In most cases, players eventually detected that the strategic opponents were cheating, which destroyed the illusion that they were matching opponents with a military genius.

In many strategy games, the AI simply relies on the build order. Because of this, the computer-controlled opponents often are very predictable. These never learn new build orders and are often easily beaten by a human player once the player has found some weakness in the AI's build orders.

2.3.2 Personality in Games

Personalities in agents are often constructed by varying the mapping between emotions and reactions [24]. Each emotion can therefore be associated with a class of reactions typical for each personality, for example, aggressive behaviour in orders to realize aggressive personality. When the agents have different personalities, they also have different plans for achieving their goals. Personality modules contain values, heuristics, social traits and mannerisms which create a more or less believable personality for the agent.

There are several different ways to use personality in games and computer programs. Nass et al. tried in an experiment to see how psychology influences the experienced personality [25]. They divided the agents into dominant and submissive behaviour. The agents expressed themselves differently, but the behaviour was the same. The dominant agent used strong language, while the submissive agent used a weaker language. But the players believed that the agents had the same behaviour as they expressed. An agent with a dominant expression was experienced as playing with a dominant behaviour, while the one with submissive expression as playing with submissive behaviour.

A game like Max Payne has a cinematic story that revolves around a main character and many different opponents. The personality is so strong in the game that the gamer feels the power of the moment in the game. The audio, video, graphics and interaction between players make the situation of the agent more realistic and believable. Another such popular game, Hitman, has a very strong character personality that is an emotionless killing machine targeting specific targets. These character agents show realistic emotions, situation based reactions, language and tone of voice etc, to make the gamer believe in the game setting. Building this factor into the game is extremely important as this is what the gamer looks for in every game.

Creating a main character is one thing, creating a non-playing character is something completely different. Today's computer games have such a high level of detail, high end graphics and realistic environments and characters that gamers can be led to believe that games are set within realistic settings while in the game. But according to Trinity College Dublin, the realistic illusion of the gamers is most often led into disappointment as soon as the gamer begins to interact with a computer controlled non-playing character either though conversation or attitude [26]. Although the non-players look real and act real, due to their lack of controlling intelligence these characters lack the reality when the player to player interaction takes place. With the use of artificial intelligence and applying artificial neural networks in these characters, the TCD Game AI Project at the Trinity College captured and added personalities, moods and relationships [26].

In Age of Empires III each opponent has a distinct personality, with particular strengths and weaknesses patterned on the real persons they are based on (e.g. the English are controlled by Elizabeth I and the French by Napoleon) [27]. Each civilization also has some unique units (ships and soldiers) with more-or-less historically accurate names. The AI in Age of Empires III is known as very challenging and impressive. By adding personalities to the agents, the developers have used the possibility to provide a rather unique gameplay experience. This makes the gameplay very good and entertaining.

One game that allows a gamer to mould the character and its personality according to the gamer is Sims. More popular in the feminine part of computer game players and the best-selling PC game in history, Sims allows the gamer to live a complete life as the gamer wants it to be led. A gamer has many different virtual people to control and leads their lives from waking up, eating, changing, going to work, etc. The upcoming Sims 3 from Electronic Arts is promises to be bigger and better than its previous versions [28]. An online social networking website that works just like a game and does what Sims did even more realistically is ‘The Second Life”. It is a virtual world where real people can live a complete second life. Users can buy, trade, talk and do many activities virtually. The users (called residents) can buy real clothes from Channel, Shoes from Nike and pay through Citibank’s virtual MasterCard to enhance the personality of their character. Although it does not fit the actual definition of a game, it can be called almost a game where people pay actual money to live a virtual life [29].

Anno1701, a German 3D building simulation, is another example of a game which using AI with personality. Each AI has a profile which is different from the other AI’s. The aggressive AI will always build city walls, the cunning AI acquire an audacity value, and the architect of beauty has to decorate his settlement with numerous ornamental objects.

The gamer want realistic graphics, agents, environments and the only solution lies with developing better artificial intelligence into the game characters that do exactly what the user would imagine in real life.

3 Diplomacy and StateCraft

3.1 Diplomacy

Diplomacy is a seven-player turn-based strategy game derived from the great efforts of the major European influences during World War I. The nations players can control in this game are: England, France, Germany, Russia, Italy, Austria-Hungary, and Turkey.

The game board is a map of Europe with the borders of 1914. It contains seventy-three bordering regions and every player begins with three units (Russia with four), fleets or armies, in their native lands. Thirty-five of the seventy provinces on the board are described as “supply countries”, they contain supply centres. The objective of the game is to manage eighteen of the thirty-five supply centres. To manage this, the player has to eliminate as many opponents as possible from the map. This is done by building armies and fleets to use in the battle against the opponents.

Some actions are very complex and not easy to observe. Actions like this can involve up to 10 units. An experienced player will easier understand which actions is relevant and not. At the same time an inexperienced player can also very easy understand which actions are legal and illegal.

To win the game, the player has to use diplomacy and make deals with other players. These important deals are negotiated in sessions before each period of the game. A common strategy is to break alliances at the most advantageous moment to hit the opponent by surprise.

After the negotiation is finished, the resolving of orders is done. Fleets are permitted to move across bodies of water and between coastline regions, at the same time as armed forces can shift onto any neighbouring region. These units have equivalent power in the game.

3.1.1 History

Diplomacy was invented by Allan B. Calhamer in 1954 while he studied European history, political geography and law at Harvard [30]. He originally printed 500 copies. After he sold all in six months, he licensed the game to a publisher. The game was first published in Northern America, but lately it has been published in seven different languages.

Figure 1 An edition of the Diplomacy board game

In 1960 some players began to join play-by-mail communities. Play-by-mail means that each player sends their moves to a central agent called game master. This made it easier to organize the game, since it originally was made for seven players. It is created rules for playing with less than seven players. But by play-by-mail, the players could play with the original rules. Playing the game by mail, however, didn't become a success. Postage was expensive and service was slow. Cheap long distance telephone, next-day mail and faxes were years away.

Most of the early pioneers of postal Diplomacy were drawn from the science fiction community. The sci-fi hobby got tremendous boost in popularity with the Star Trek's premier in 1965, but the postal Diplomacy still grew very little in its first two years. A listing of all known postal Diplomacy players in May 1965 showed only 83 participants.

In 1966, Diplomacy got a boost in popularity. The idea of Diplomacy by mail reached the general wargaming community. This popularity also increased the number of players and publishers, and also contributed concepts such as ratings, conventions and rules to a fledgling hobby.

The original rule book from 1961 had many areas that required interpretation, which was left to the individual gamemaster or publisher to handle. This started to become a serious topic of discussion and concern at this time. Many prominent players and publishers had well-known rules interpretations names after them (Koning Rule, Brannan Rule, Chalker Rule, von Metzke Rule, Miller Rule and several others). This testing of the game and 1961 Rulebook many considering as a major contribution the postal hobby made to the game itself. In 1971 the marker of the game, Games Research Inc. (GRI), incorporated all of the rules into a revised 1971 Rulebook.

Participation peaked again with no new influx of participants. But when GRI included a flyer in the box in 1970, advertising the play of Diplomacy by mail and giving addresses to contact for more information the influx increased again. This simple act launched expansion of the hobby during the 1970's. Some consider this as the "Golden Years" of the hobby.

An unsuccessful attempt was made in 1971 to start a hobby-wide organization (The Diplomacy Association, or TDA), but this effort was quickly rent apart by a bitter dispute amongst its members. This dispute was about issues such as: should the hobby have an organized structure? Should an organization be voluntary or mandatory? Would leadership be democratic or custodial? Later, The International Diplomacy Association

(IDA) was formed in 1972 and operated electing officers, co-ordinating hobby services and collecting dues.

During the 1980’s the hobby grew and a new technology (computers) was on the horizon. In February 1983 Russell Sipe started playing the game by e-mail. By 1990 computers were generally available to anyone, and the game became more popular again. Now the players had the possibility to play the game by electronic mail, orders via fax, next day mail and phone. It became a real pleasure to the participants to play a multi-player game with others across a state, a continent or an ocean. Now many players have this as an organized hobby through the mails.

3.1.2 Rules

There have been several different variants and versions of the rules of Diplomacy. But the major changes have involved giving the countries a wider range of strategic choices and adjusting the map to make the countries nearly equal [31]. The latest is 4th edition from 2000 [32].

Diplomacy starts in the beginning of the 20th century. Europe was at this time a complicated cauldron. The game is played on a map which represents Europe. As mentioned, the player controls one of the seven “Great Powers of Europe” (also referred to as country and nation).

The board is divided into provinces on sea and land. Each province can only be occupied by one unit at time. The player disposes two different types of units, fleets and armies. Russia starts with four provinces with supply-centres, and the other six countries starts with three. At the start some of the supply-centres are unoccupied.

Each year is divided into spring and fall rounds, and starts with spring 1901. In each round every player can negotiate and make deals with the other players. These deals could for example be to support each others units or attack one of the other players. The players can choose whether they will follow the deals or drop them.

By the end of each fall round, every occupied supply-centre is controlled by the one who occupies it, and centres that are not occupied are still controlled by the player who occupied it at the end of last fall turn. When the fall turn is finished, each player counts how many units and supply-centres it controls. Then each player can build one new unit for each supply-centre more than units it controls. The players can only build new units in their original supply-centres.

If one of the players controls eighteen supply-centres by the end of fall turn, this player has won the game. All units have equal strength. Each unit has three different states:

- Move: The unit moves to another province
- Support: The unit adds extra strength to another units move
- Hold: Unit will maintain it's position

In addition to these three states, fleets have a state called convoy. When fleets have this state, it holds its position and an army can move from a coastal province to another coastal province. These coastal provinces have to be the neighbours to the province that the fleet occupies. The fleets can only move to provinces on sea or provinces with a coastline.

When a unit supports another unit, it transmits strength to the unit who is supported. But if the supporting unit is under attack, it can't give the support. If two units compete for the same province, it is the one who is supported which occupies it. If none of the units are supported, they have equal strength and both will be standing. Only one unit can occupy a province at a given time.

3.2 StateCraft

In their thesis, Krzywinski and Helgesen present the Caeneus architecture, an agent architecture developed to handle both a social and a game environment [3]. To test this architecture, they developed a software version of the board game Diplomacy, called StateCraft, and implemented the architecture in an agent controlling a player / nation in the game.

Section 3.3.1 will briefly describe the architecture developed by Krzywinski and Helgesen for this project, the Caeneus Architecture. For a full description of the Caeneus Architecture and the original StateCraft, see [3]. In section 3.3.2 we will outline the technical solution of StateCraft and the game mechanics.

3.2.1 The Caeneus Architecture

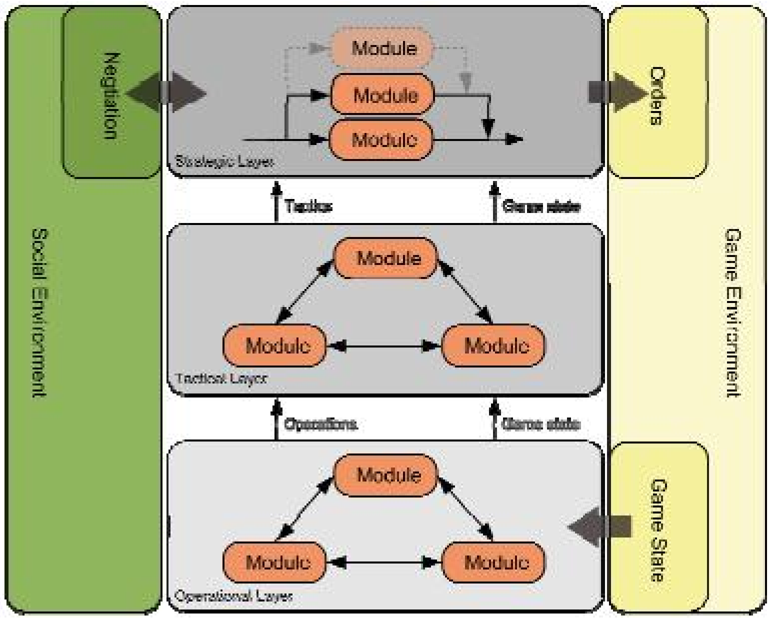


Figure 2 The Caeneus Architecture [3]

StateCraft was developed as a test bed for the Caeneus Architecture (Figure 2). “The Caeneus architecture is a vertically layered one-pass agent architecture developed to meet the challenges in social board games” [3].Three-layered or three-level architectures are an often used approach when designing software agents. “Shakey the Robot”, developed from 1966 to 1972, the first robot to be able to reason about its own actions [33], was designed using a three-level architecture [34]. Typically, the lowest level contains basic methods, like reactive actions and handling of sensory input, while the next two usually are application specific. In the Caeneus Architecture, the three levels are named, respectively, *the operational layer*, *the tactical layer* and *the strategic layer* and are the equivalents to the three main tasks for a player when playing Diplomacy, *monitoring the game board*, *planning moves*, and *engaging in diplomatic negotiations*.

The operational layer is a reactive layer and is triggered at the start of each round, when the game server distributes the new game state and discovers all possible and legal moves for each unit, separately.

The tactical layer combines operations for each unit into a set of operations, a *tactic*. Each tactic contains thus one operation for each of the agent’s units. Each tactic has two values, *potential value* and *factual value*. While potential value represents the value of a tactic regardless of the other players’ moves, the factual value represents the tactics tactical value combined with its chance for success. If a tactic has a very high potential value, but is considered impossible to achieve, its factual will be much lower, while certainty for success will give similar values for potential and factual value.

The strategic layer is by far the most complex of the layers in the Caeneus Architecture. It is built using the Subsumption Architecture [35]. The Subsumption Architecture is a vertically layered agent architecture where each layer or module represents behaviours, where, typically, the lowest is the most primitive and each module added increases the sophistication level. Each module depends only on the modules below, thus, the system will function with only one module. New modules can suppress input or inhibit output of the lower modules, making it easy to add new modules, as we will prove in chapter five. The Subsumption model in the strategic layer implemented in StateCraft consists of four modules, *Choose Tactic, AnswerSupportRequest, SupportSuggester and Relationship.*

The Choose Tactic module receives a list of tactics, a TacticList, from the tactical layer, sorts the list based on each tactic's factual value, creates a subset equalling the agent's number of units and randomly chooses one tactic from that subset to post to the game adjudicator as the agent's actions for each round.

The AnswerSupportRequest module handles requests for support from other players. It receives the game state from the game server and suppresses the TacticList from ChooseTactic's input line and, when receiving a request, considers it according to an internal algorithm and sends a RequestAcceptanceMessage containing its answer back to the requesting player. If a request is accepted, it is stored in the TacticList, which in turn is piped back to the ChooseTactic module. When accepting a request for support, the agent will always honour the agreement, but, as Krzywinski and Helgesen states, the algorithm is too restrictive; in practice no support requests are ever accepted.

The SupportSuggester module is responsible for suggesting support requests to other players. As the AnswerSupportRequest module, this module suppresses the TacticList sent to the ChooseTactic module and receives the game state. In short, it sorts the TacticList according to factual value, selects the first 15% tactics, selects the operations needing support and sorts these by the difference between factual and potential value, thus asking for support for operations with a high potential value. At fixed intervals, the module will send request for one of these operations. If it receives a positive reply, it stores the operation in a list and, when next it suppresses a TacticList, the operations the agent has received support for, get their factual value increased.

The Relationship module handles the agent's relationships with the other players. The agent's relationship with another player can have three states, *Friend*, *Neutral*, and *War*. Each relationship starts as neutral and changes according to that player's actions towards the agent. The Friend state changes to Neutral if the player breaks a deal and to war if the player attacks the agent or moves into a province controlled by the agent. The Neutral state changes to Friend if the player honours an agreement and to War if the player breaks a deal or attacks the agent. The War state changes to Neutral if the player honours an agreement with the agent. The module does not keep track of previous events or states, which means that the agent can have the relation Friend with another player throughout the game, but change to War in the last round if that player moves a unit into a province the agent controls.

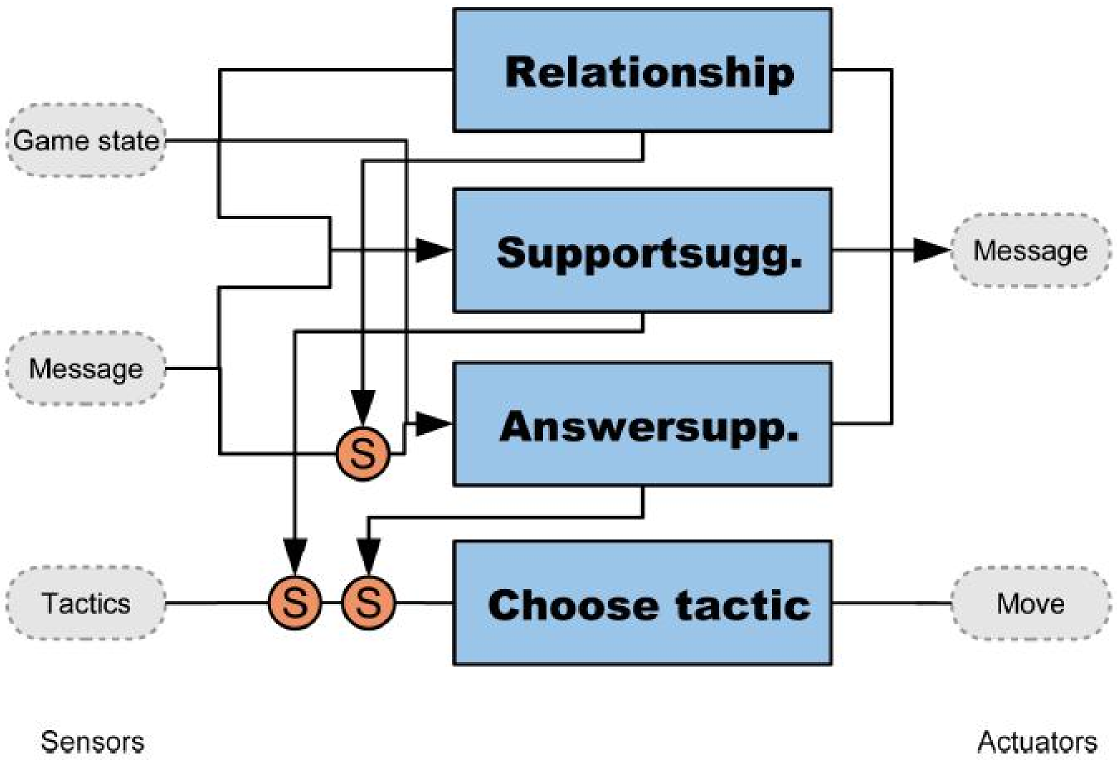


Figure 3 The Subsumption Modules as it was implemented in StateCraft

The implemented version of the strategic layer varies slightly from the designed version. The description above is of the implemented version.

3.2.2 Technical Solution

The game consists of a database for storing persistent data, a game server, a game client and an agent, and game data (map data and graphics files) meant to be located at a web server. The database is a MySQL database and the map data is stored in Extensible Markup Language (XML) files, while the game is written in Java. The client and the agent connects to the game server through the same interface; a socket to the server URL. Server and clients communicate through messages sent over this socket. By making the same set of commands available for the agent and the client, the agent and client get the same possibilities and constraints, thus creating an environment suitable for testing agent architectures.

The main game class, Diplomacy, implements the interface Server and maintains an instance of the class NetServer, which manages incoming connections. The class SocketConnection handles all the network communication between the clients and the server. The clients connect to the server via a URL reference. Thus, the game server must run on a web server (e.g. Apache HTTP Server or Microsoft Internet Information Service). The server sends all output to and reads all input from the console window. Clients (agents or players) log on to the server with a username and a password, both the name of the nation controlled. Agents must also enter game number, while players select a game from a list. Both clients have a graphical user interface (GUI), agents for observation only. The user client class, DiplomacyApplet, is, as the class name suggests, a Java Applet, while the agent class, CaeneusAgent, is a standard Java application.

The persistent data is stored in a MySQL database, which means that the computer running the game server must have the MySQL server installed. Both clients and game server are set up to download map data and graphics files from a web server. This is handled through different Net***Creator (e.g. NetGraphicsCreator) classes.

As the game was meant to be, and tested as, a multiplayer game, all rounds have a set length, or deadline, and when that deadline is reached, the game adjudicator resolves orders given by the clients, and the game server updates the game state accordingly and distributes the new game state to all clients.

The source code for StateCraft was downloaded from the Open Source software development web site SourceForge.net [1]. Server URL and game data locations were hard-coded, so some adaptations hade to be made to the code to get a running application. The project has yet to reach a stable release.

3.2.3 StateCraft – A Walkthrough

This section will briefly describe how the game StateCraft is executed and played. For a more thorough description, see the README in Appendix B (the README is written for our manipulated version of StateCraft, but the basics of the gameplay are the same). We will assume that a game server and a web server hosting the data and graphics files are provided. We will also assume that a development environment is set up to run the user client and at least one agent. The game can be played with one to seven clients, where the clients can be human or agent controlled. The game starts when the game server is started, so one should try to start all clients simultaneously, to derive any player of the advantage of starting ahead.

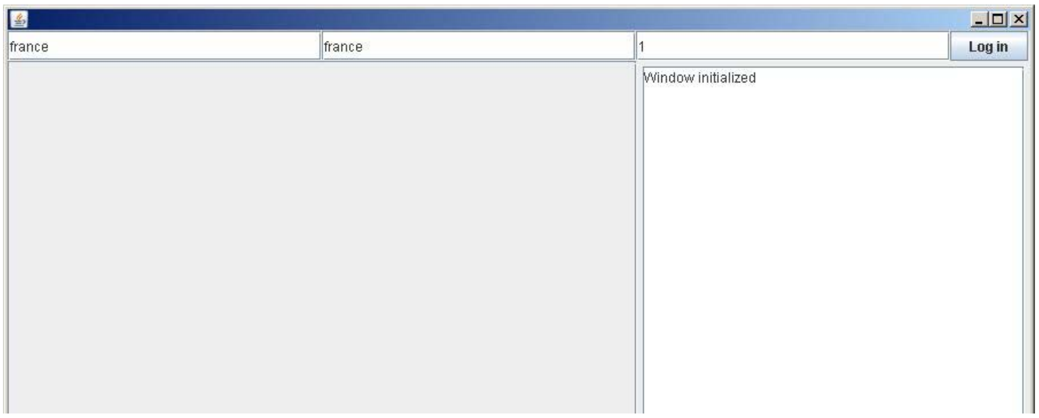


Figure 4 The StateCraft Agent's login window

Figure 4 shows the agent’s login window. The three fields represent username, password and game number. Username and password are always the nation’s name in lowercase. The game number is needed as the game server can potentially run several games at a time. In this example, the game server runs only one game, game number 1.

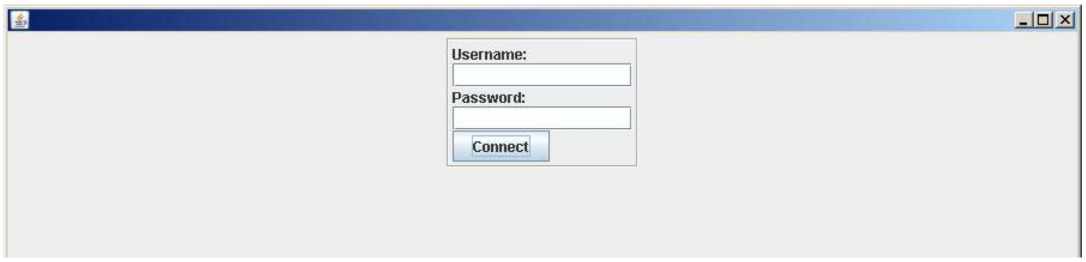


Figure 5 The user client’s login window

The user client’s login window is shown in Figure 5. The user must provide a username and password, as for the agent consisting of the controlled nation’s name in lowercase. The user then presses the connect button to connect to the game server.

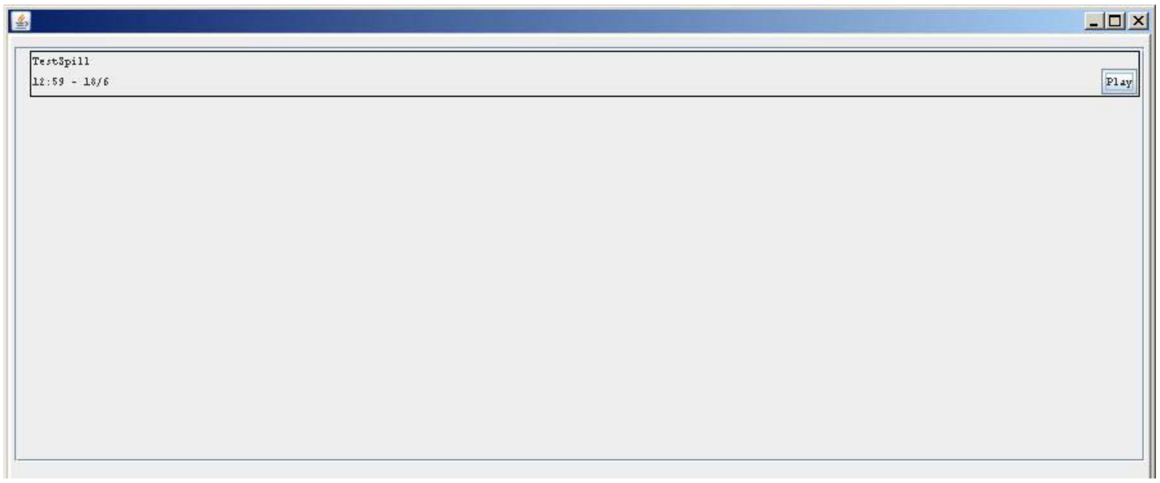
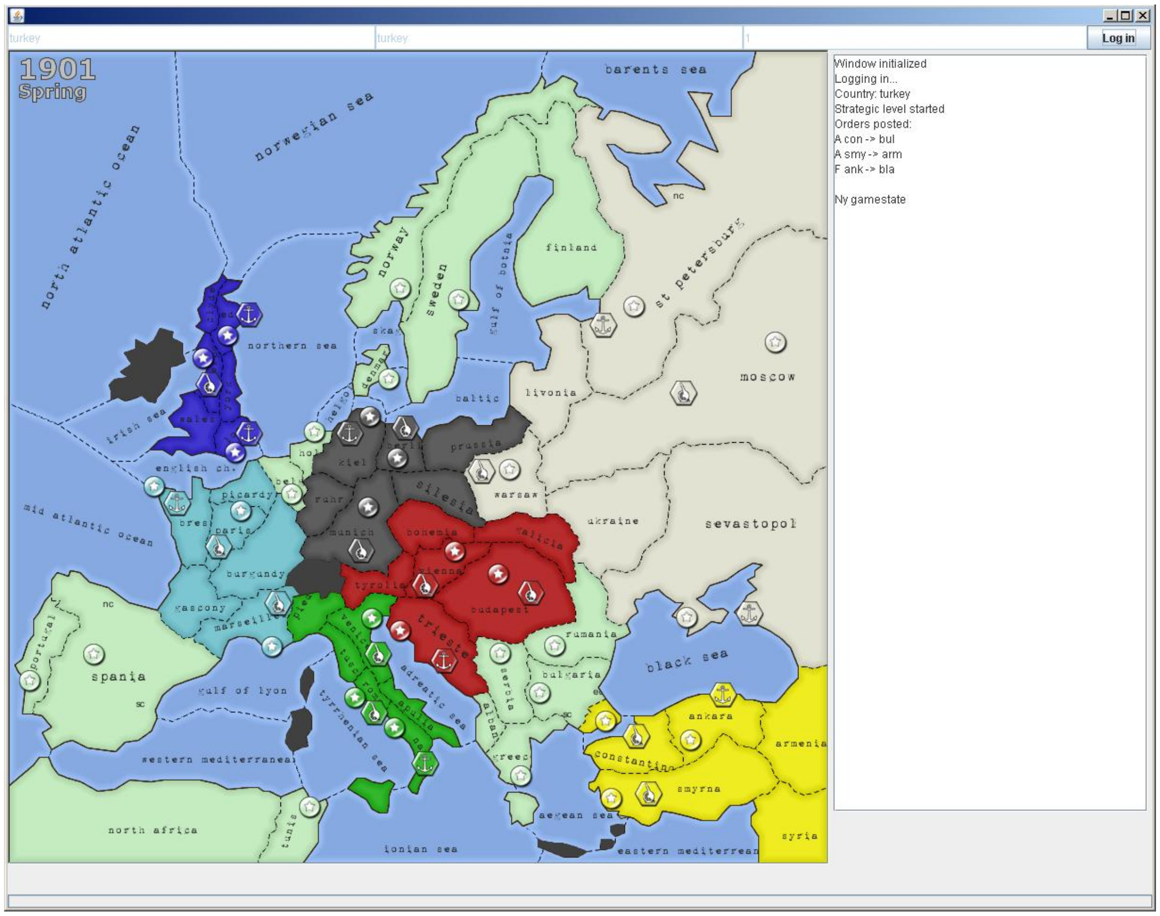

TestSpill
12:59 - 18/6

Figure 6 The user client listing available games on server

When connected to the game server, the user client displays a list of available games, in this example one game, TestSpill. The name and creation date of the game is follows by a Play button to start the game.



```
Window initialized
Logging in...
Country: turkey
Strategic level started
Orders posted:
A con -> bul
A smy -> arm
F ank -> bla

Ny gamestate
```

Figure 7 The StateCraft Agent interface.

When all clients are ready to join the game, the user(s) press the Log in button in the agent interface and the Play button in the user interface to start playing the game. Figure 7 shows the agent interface. The agent is playing Turkey and the board is in its starting position. In the upper right corner, the agent's orders for this round are printed. This interface is for observation only, it cannot manipulate the game events in any way.

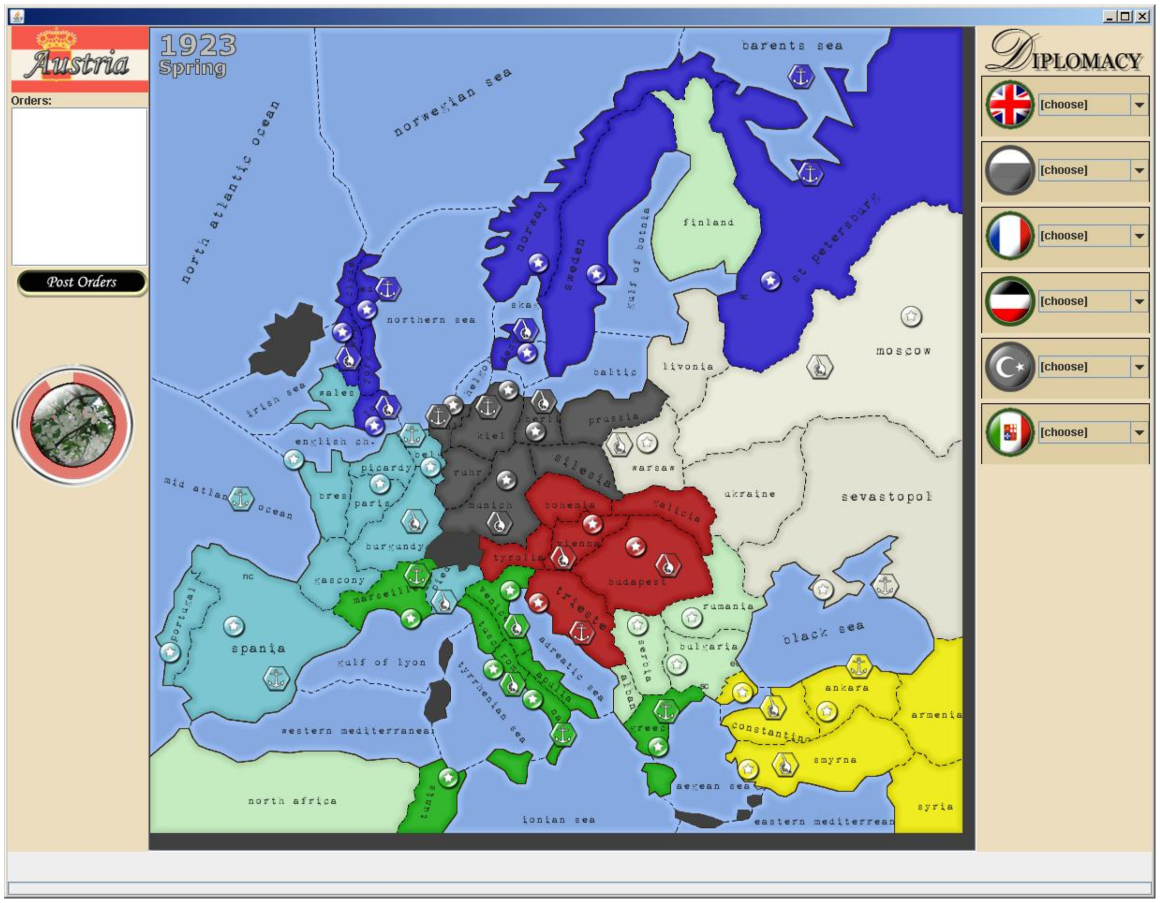


Figure 8 The user client's game interface

In the upper left corner in the user client game interface (Figure 8), the flag of the nation you are controlling is shown. Below this flag is a field where you can view and send your orders to the game server. The circle to the left is the timer, when the circle is closed, the game moves to the next round. The diplomacy panel is displayed in the right section of the window. From this, the player can send diplomatic proposals to the other nations. The disabled (greyed out) buttons indicate that no player or agent controls that nation.

One year in the game consists of five rounds, one round for each of the seasons and a build round after the winter round. The first and third round, spring and fall, are the actions round, in these rounds, the players can issue commands to their units and engage in diplomatic discussions. An order to a unit can contain one of the following actions: move, hold or support. A move order will transport a piece from one province to an adjacent province. A hold order will keep the piece in it current location. A support order will aid another unit moving into a region or defending its current region of occupancy. Orders are issued by clicking a unit and selecting an action from the pop-up menu displayed (Figure 9).

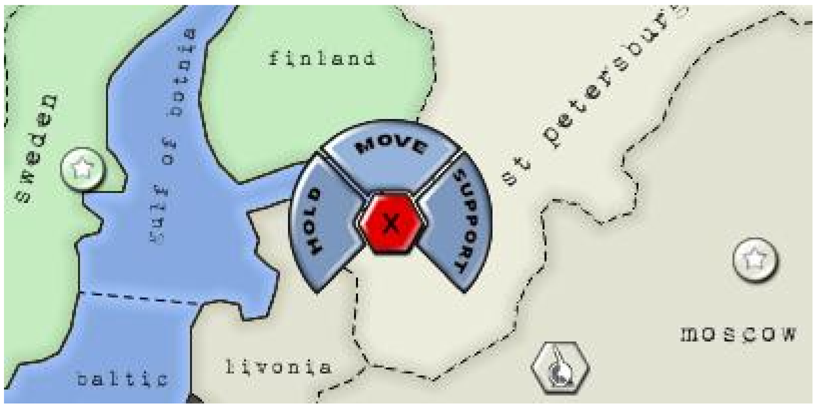


Figure 9 The orders pop-up menu

After issuing an order to a unit, a symbol will illustrate this order, with. e.g. an arrow for a move order (Figure 10). When all orders are given, the player presses the Post Orders button in the upper left corner of the window (Figure 11).

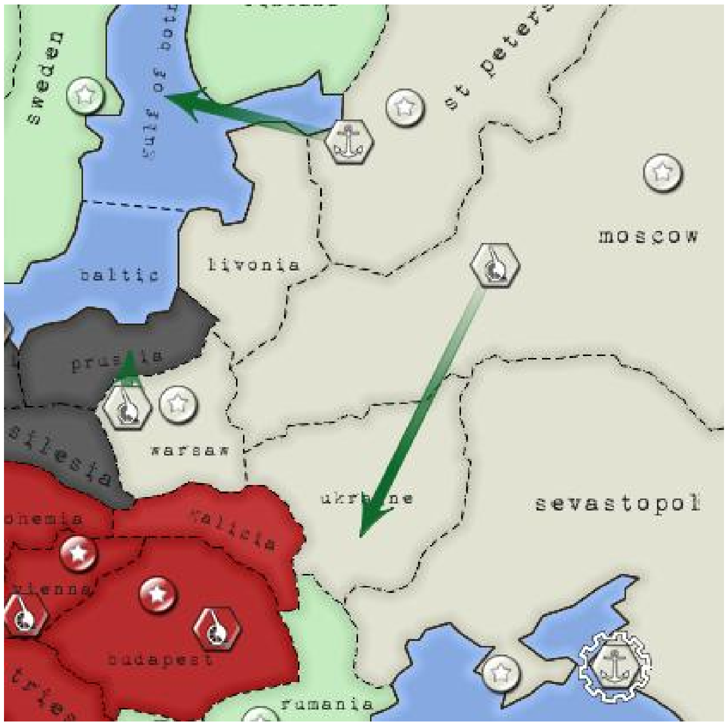
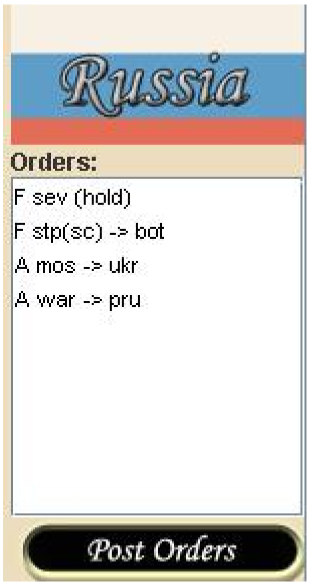


Figure 10 Orders given

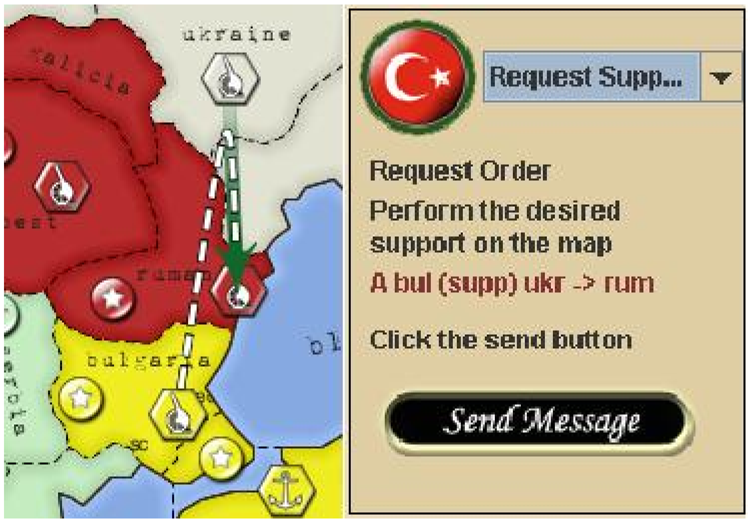


```
F sev (hold)
F stp(sc) -> bot
A mos -> ukr
A war -> pru
```

Figure 11 The orders pane

These orders must be posted before the round is over. Orders can be cancelled at any time by issuing and posting new orders. The orders are not registered with the game server before pressing the Post Orders button.

The diplomatic options are limited to requesting support from other nations. To request a support, the Request Support option is selected from the drop-down menu beside each opponent's flag in the diplomatic section. The player then issues the support command it wants the other nation to perform, in the same way as issuing orders to its own units and presses the Send Message button (Figure 12).



Request Order
Perform the desired support on the map
A bul (supp) ukr -> rum
Click the send button

Figure 12 Asking for support

When a diplomatic message is received, either a request or an answer to a request, the dark green circle around the nation's flag turns bright green. Pressing the flag displays the message. As for the orders, diplomatic requests or answers must be sent before the round is over.

The second and fourth round of a year, summer and winter, are no-action rounds. In these rounds, the result of the previous rounds' orders is displayed, with green arrows for successful actions and grey arrows for unsuccessful actions. Actions can be unsuccessful due to illegal orders or failed moves / attacks.

In the build round, the players can build new units. Units can only be built in native supply centres, that is, centres that nation started the game with. A player can have one unit for each controlled supply centre. If the number of units exceeds the number of supply centres, the player must disband the exceeding number of units. If no units are selected for disbanding, the game server will disband the corresponding number of units automatically.

The game is in theory over when one nation controls eighteen or more supply centres after a fall round. However, the game server provides no mechanics for handling this; it will continue the game until manually terminated.

4 The Personality Module

4.1 Personality Attributes

4.1.1 Zamora's Personality Attributes

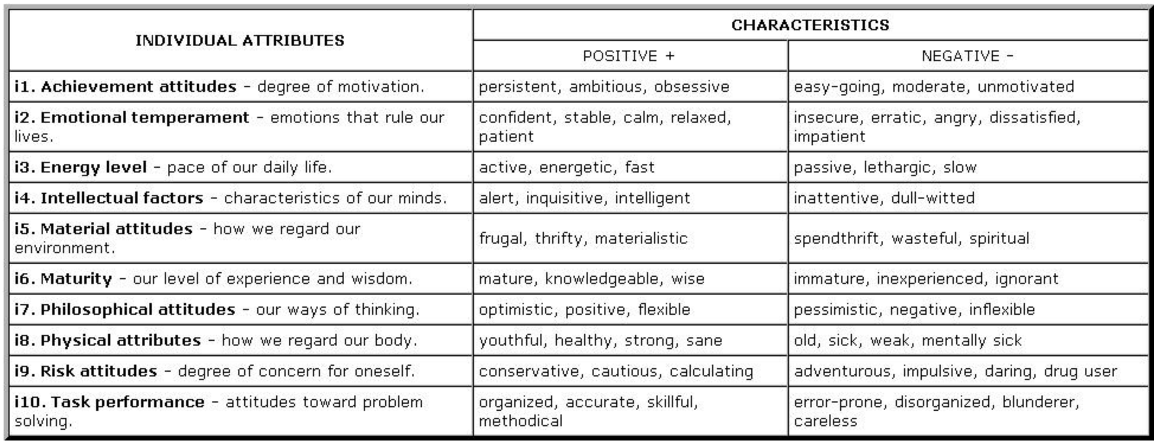
Zamora [36] defines the term personality as "the totality of character attributes and behavioural traits of a person" , while Ryckman [37] defines personality as "a dynamic and organized set of characteristics possessed by a person that uniquely influences his or her cognitions, motivations, and behaviours in various situations".

Psychologist Raymond B. Cattell (1905-1998) defined personality as "*that which permits a prediction of what a person will do in a given situation*" [36]. In computer games, however, agents – without personality – tend to be easily interpreted by humans, thus easily defeated. We believe that by simulating personality in agents, both the gameplay experience and the challenge will increase.

In StateCraft, the agent has to both operate individually and to cooperate with other agents in the game. To give this agent personality, we need to use attributes that affect its individual decisions and its relations to other agents. Zamora divides the character attributes into *individual* and *social attributes* (see table 1 and 2). Individual attributes are set parts of our personality that we are born with and that do not change much during a lifetime. Social attributes are personality traits that we develop during our upbringing. By reaching adulthood, they are firmly set in our personality and are not easily altered. Although Zamora uses these attributes as the foundation of his *Personality Compatibility Analysis*, we believe that they can be used also to individually describe a person's personality. The compatibility issue will not be discussed in this paper.

The table of individual attributes is divided into three columns, the first naming and briefly explaining the attributes, the second and third describing the positive and negative effects of having more or less of each attribute. Note that positive and negative does not mean the same as high and low. Having a positive degree of *Achievement attitudes* would mean to have a higher degree of that attribute, while having a positive degree of *Emotional temperament* would mean to have a lower degree of that attribute. Although there are extremities, most human beings will place themselves somewhere around the middle of the axis.

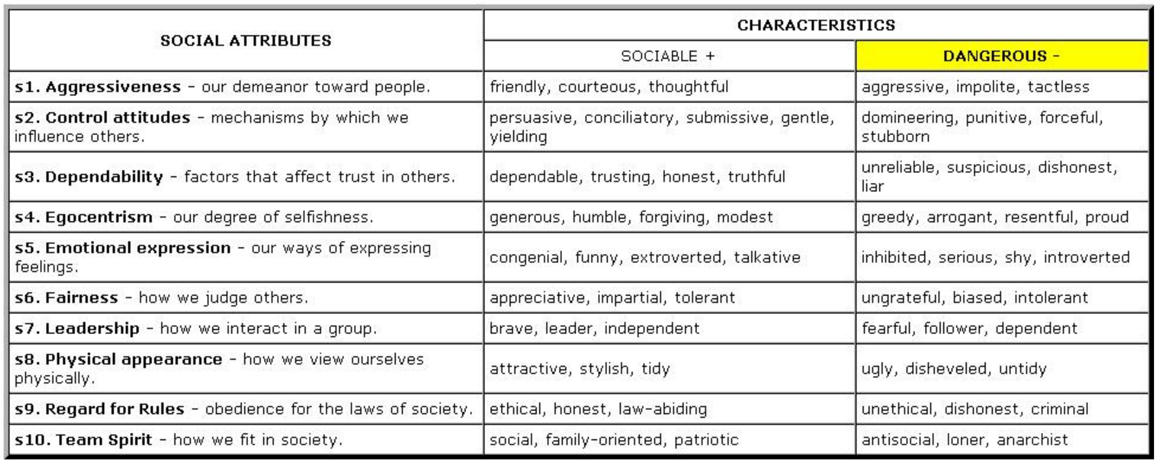
Table 1 Zamora's individual attributes



INDIVIDUAL ATTRIBUTES	CHARACTERISTICS	
	POSITIVE +	NEGATIVE -
i1. Achievement attitudes - degree of motivation.	persistent, ambitious, obsessive	easy-going, moderate, unmotivated
i2. Emotional temperament - emotions that rule our lives.	confident, stable, calm, relaxed, patient	insecure, erratic, angry, dissatisfied, impatient
i3. Energy level - pace of our daily life.	active, energetic, fast	passive, lethargic, slow
i4. Intellectual factors - characteristics of our minds.	alert, inquisitive, intelligent	inattentive, dull-witted
i5. Material attitudes - how we regard our environment.	frugal, thrifty, materialistic	spendthrift, wasteful, spiritual
i6. Maturity - our level of experience and wisdom.	mature, knowledgeable, wise	immature, inexperienced, ignorant
i7. Philosophical attitudes - our ways of thinking.	optimistic, positive, flexible	pessimistic, negative, inflexible
i8. Physical attributes - how we regard our body.	youthful, healthy, strong, sane	old, sick, weak, mentally sick
i9. Risk attitudes - degree of concern for oneself.	conservative, cautious, calculating	adventurous, impulsive, daring, drug user
i10. Task performance - attitudes toward problem solving.	organized, accurate, skillful, methodical	error-prone, disorganized, blunderer, careless

The table of social attributes has the same structure, but instead of dividing the characteristics into positive and negative, they are divided into *sociable* and *dangerous*. As opposed to the individual attributes, here a personality with attributes in the latter category is regarded as unfavourable.

Table 2 Zamora's social attributes



SOCIAL ATTRIBUTES	CHARACTERISTICS	
	SOCIABLE +	DANGEROUS -
s1. Aggressiveness - our demeanor toward people.	friendly, courteous, thoughtful	aggressive, impolite, tactless
s2. Control attitudes - mechanisms by which we influence others.	persuasive, conciliatory, submissive, gentle, yielding	domineering, punitive, forceful, stubborn
s3. Dependability - factors that affect trust in others.	dependable, trusting, honest, truthful	unreliable, suspicious, dishonest, liar
s4. Egocentrism - our degree of selfishness.	generous, humble, forgiving, modest	greedy, arrogant, resentful, proud
s5. Emotional expression - our ways of expressing feelings.	congenial, funny, extroverted, talkative	inhibited, serious, shy, introverted
s6. Fairness - how we judge others.	appreciative, impartial, tolerant	ungrateful, biased, intolerant
s7. Leadership - how we interact in a group.	brave, leader, independent	fearful, follower, dependent
s8. Physical appearance - how we view ourselves physically.	attractive, stylish, tidy	ugly, disheveled, untidy
s9. Regard for Rules - obedience for the laws of society.	ethical, honest, law-abiding	unethical, dishonest, criminal
s10. Team Spirit - how we fit in society.	social, family-oriented, patriotic	antisocial, loner, anarchist

4.1.2 Choosing Attributes for the Agent

Most of the attributes Zamora defines are irrelevant to an agent in StateCraft. For example, our agent does not have any physical appearance and thus no consciousness of its physical appearance, nor has it philosophical abilities or the ability to mature. In addition, giving an agent 20 different personality attributes would render it impossible to distinguish the different attributes in a game like StateCraft. We therefore chose to concentrate on two individual attributes, *emotional temperament* and *risk attitudes* and two social attributes, *aggressiveness* and *regard for rules*. However, to make the attributes intuitive and easy to understand, we let our attributes vary from the table. As explained in section 4.1.1, a *high* degree of an attribute does not necessarily mean the same as a *positive* degree. Thus, if changed to Boolean values, *true* and *positive* would not be equal for all the attributes in Zamora's table. We therefore assume that if an attribute is set to true, this equals a high degree of that attribute, regardless of positivity or negativity. The chosen attributes are explained below.

- **Emotional temperament**

  A person with a high degree of emotional temperament is more likely to react emotionally to events in the world, whereas a low degree renders it more likely for a person to react logically and reasonably.

  In StateCraft, this attribute will give the agent the ability to have "feelings", that is, to react emotionally on events in the game. If this attribute has a high value, the agent will focus only on the gain an action *may* give, if successful. It does not, for example, consider other players' possible actions or the strategic disadvantage of leaving a province undefended.

- **Risk attitudes**

  A person with a high value on this attribute takes chances in his or her life. It can be financial chances to get rich quickly, it can be smoking when you know the dangers, or it can be base chuting for the fun of it. A person with a low value, on the other hand, likes to play it safe, likes being certain of the outcome of his or her actions.

A high degree of risk attitudes makes an agent in StateCraft more daring, favouring attack over defence. If false, the agent tends to be a bit more careful, thus often seeing defence as the better option.

- **Aggressiveness**

  Aggressive persons tend to be just that, aggressive. They interact with other people in an aggressive manner, often being impolite or even rude.

  As the social section of StateCraft is rather limited and agents and players have no possibility of putting emotions in their communication, this attribute affects agents in StateCraft in a slightly different manner. An aggressive agent lets happiness or anger influence its actions. For practical purposes, we assume an agent is always angry with its enemies and happy with its friends. An aggressive agent will value attacks on enemies more, while a non-aggressive agent will have a lower threshold for attacking friends.

- **Regard for rules**

  Having regard for rules, means that a person abides the rules and laws in his or her society. A person with no regard for rules is very likely defined by its community as a criminal.

  In Diplomacy, the players (or agents) have no real possibility of breaking the game rules, as they are defined by the game and absolute. The diplomacy section, however, is without rules. Any agreement between two (or more) players is only as binding as each player chooses it to be. An agent with a high regard for rules will honour its agreements, while an agent with low regard for rules, tends to put little weight on diplomatic agreements, breaking them as often as not, whatever gains its strategic goals more.

We can see that while the first three attributes affect the agent’s gameplay, *Regard for rules* will affect the agent’s social behaviour.

4.2 Implementation of the Personality Module

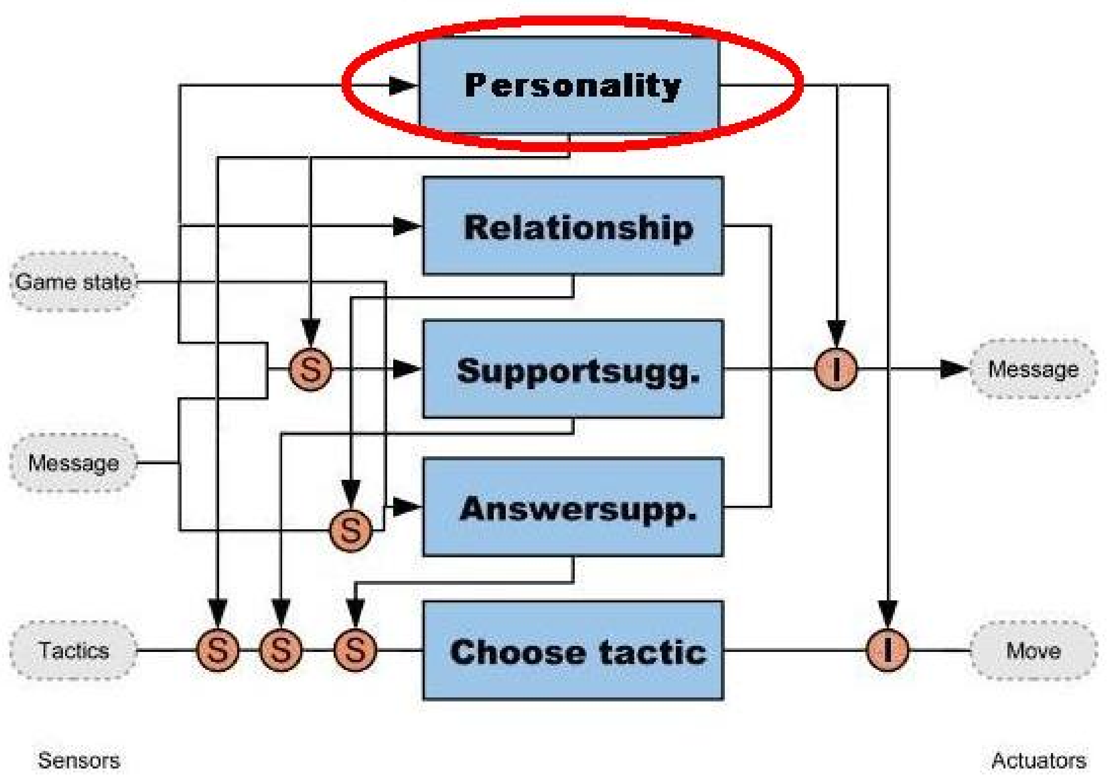
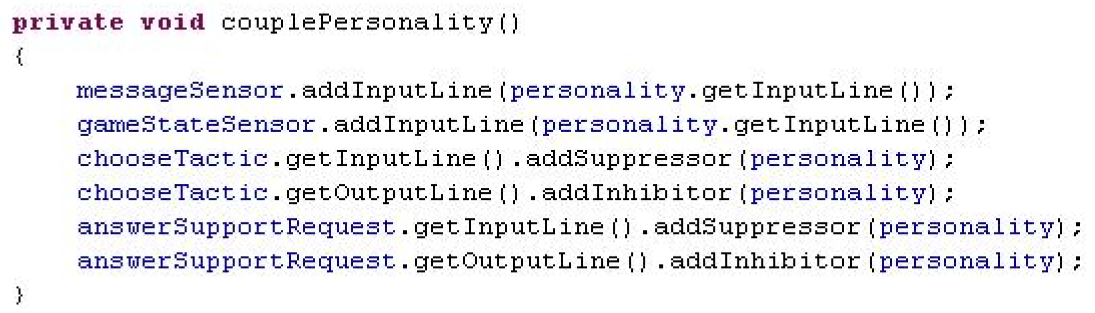


Figure 13 The Subsumption architecture for the strategic layer, with the personality module on top

This section describes the implantation of the Personality Module, how the module is connected to the existing code, how it is structured and how its classes and methods are constructed. Due to earlier experience, Eclipse [38] was chosen as our development environment. As described in section 3.2.1, the strategic layer of the Caeneus Architecture is designed using a subsumption model. We prove that this architecture is easy to expand by adding our Personality module as a new layer in this model. As can be seen in Figure 13, the Personality module is placed on top of the Relationship module. It receives the game state from the game server, and suppresses the TacticList sent to the ChooseTactic module and messages sent to the AnswerSupport module, in addition to inhibit messages sent from the AnswerSupport module and TacticLists sent from the ChooseTactic module. This was done by adding an instance of the Personality class to the Strategic class, the main class of the Strategic layer, and coupling it to the necessary suppressors and inhibitors, as can be seen in Code Excerpt 1.



```
private void couplePersonality()
{
    messageSensor.addInputLine(personality.getInputLine());
    gameStateSensor.addInputLine(personality.getInputLine());
    chooseTactic.getInputLine().addSuppressor(personality);
    chooseTactic.getOutputLine().addInhibitor(personality);
    answerSupportRequest.getInputLine().addSuppressor(personality);
    answerSupportRequest.getOutputLine().addInhibitor(personality);
}
```

Code Excerpt 1 Strategic.couplePersonality()

The couplePersonality() method connects the Personality module to the input lines for messages and game states and to suppressors and inhibitors as described above. This gives the module control over all data sent to or from the ChooseTactic and AnswerSupport modules, without these modules ever knowing it – in concurrence with the designs of the Subsumption architecture.

4.2.1 The Implemented Personality Attributes

To ease the implementation and make the game intuitive and easy to configure, we made some adjustments to the personality attributes. The implemented attributes are listed below, describing how they affect the gameplay and the game mechanics.

Ideally, all attributes should be ranged values, e.g. one to ten. For evaluation purposes, however, all implemented attributes are Boolean; they can be either *true* or *false*. The agent can be launched with any combination of these four attributes and each attribute affects the decisions made by the underlying modules independently.

- **Emotional temperament**

  An emotional agent will act on emotions. In our personality module, an agent with emotional temperament will see only the potential values of operations, disregarding all dangers and larger tactic considerations. We believe this will make the agent appear unstable, not particularly tactical at times.

  - **Affects:** TacticList
  - **True:** The agent disregards factual value and uses potential value for all tactics
  - **False:** No changes

- **Risk attitudes**

  An agent concentrating on defending its provinces and only attacking when the risks seem minimal, would be perceived as an agent playing safe, while an agent attacking when it can, trying to seize opportunities while, perhaps, leaving provinces unguarded, would feel more risk taking. This attribute manipulates the agent's valuing of hold operations.

  - **Affects:** TacticList
  - **True:** All tactics get a decrease in factual value of 10% for each hold operation.
  - **False:** All tactics get an increase in factual value of 10% for each hold operation.

- **Aggressiveness**

  The aggressiveness attribute affects how the agent considers its friends and enemies. An aggressive agent will act on emotions, favouring attacks on enemies, while a non-aggressive agent will consider its actions more coldly, valuing relations with other players less. The intention is to make the player more aware of the relations to aggressive agents.

  - **Affects:** TacticList
  - **True:** All tactics get an increase in factual value of 10% for each move into an *enemy* province.

    - **False:** All tactics get an increase in factual value of 10% for each move into a *friendly* province.

- **Contempt for rules**

  As mentioned in 3.3.3, the diplomacy section of StateCraft is rather limited. An agent will never initiate diplomatic talks, only respond to requests from the player and almost always refuse; it was rather difficult to give the agent regard for rules. We therefore changed this attribute to *Contempt for rules*. In their description of the implementation of the Caeneus Architecture, Krzywinski and Helgesen states *"the agent never lies directly - though we can imagine other modules, which through suppressing and inhibiting, might achieve lies. For instance, a random chance that the agent lies about a support could be implemented to achieve a deceiving behaviour."* An agent with contempt for rules will accept diplomatic requests more often than not, but it will still refuse to act on them, thus achieving a deceiving behaviour.

    - **Affects:** RequestAcceptanceMessage
    - **True:** 75 % of all requests for support are accepted, regardless of whether the agent intends to honour the agreement or not.
      - False: No changes.

4.2.2 Classes and Methods

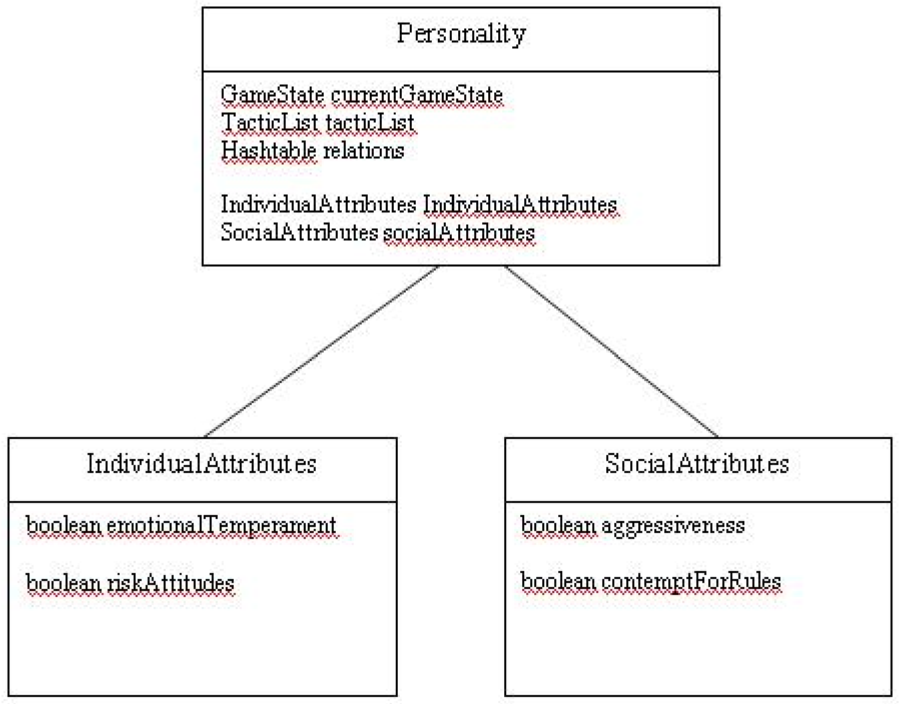


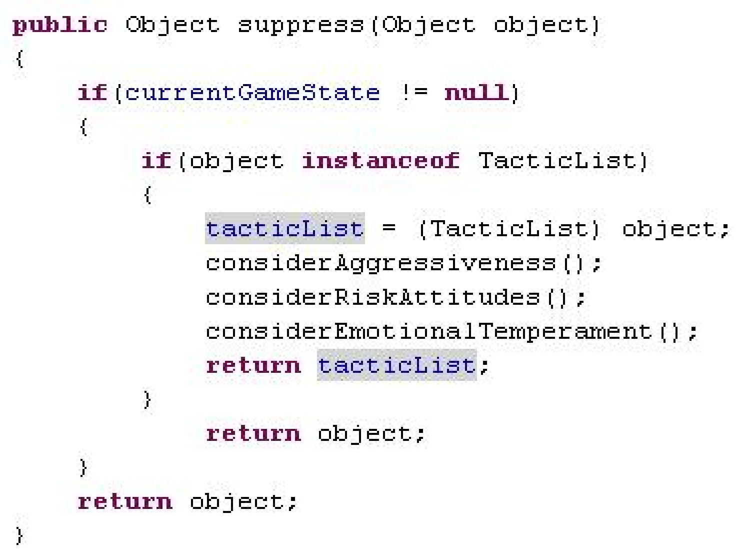
Figure 14 The Personality Module's main classes

The Personality Module is applied to the strategic layer of the Caeneus Agent, on top of the subsumption model in this layer, as can be seen in Figure 13. The Personality Module has three main components; the SocialAttributes class, the IndividualAttributes class and the Personality class (Figure 14)**.**

SocialAttributes: this class holds the social attributes and setters and getters to these. The attributes have false as the default value.

IndividualAttributes: this class holds the individual attributes and the setters and getters to these. The attributes have false as the default value.

Personality: This is the main class in the Personality Module. It holds instances of SocialAttributes and IndividualAttributes. The Personality class contains four *consider* methods, one for each personality attribute, in addition to methods for receiving, suppressing and inhibiting objects from the input and output lines. Each of the consider methods manipulates data according to the description of the attributes in the section above. When input or output from other modules is suppressed or inhibited, the appropriate method is called before the data is piped back to the input or output lines.



```
public Object suppress(Object object)
{
     if(currentGameState != null)
     {
          if(object instanceof TacticList)
          {
               tacticList = (TacticList) object;
               considerAggressiveness();
               considerRiskAttitudes();
               considerEmotionalTemperament();
               return tacticList;
          }
               return object;
     }
     return object;
}
```

Code Excerpt 2 The suppress method in the Personality class

As shown in Code Excerpt 2, the methods affecting the TacticList (considerAggressiveness, considerRiskAttitudes and considerEmotionalTemperament) are triggered when a TacticList is suppressed from the ChooseTactic module’s input line. Each of these methods modifies the TacticList before piping it back to the input line, where it moves on to the ChooseTactic module. The ChooseTactic module, knowing nothing of the recent manipulation of the TacticList, then sorts this according to its internal algorithm, chooses a tactic and sends it to its output line. The considerContemptForRules() method is triggered when the module inhibits a diplomatic response message from the AnswerSupportRequest module. The inhibitor for the ChooseTactic module’s output line was initially implemented to make the module able to manipulate the TacticList transmitted from the ChooseTactic module. We found it more reasonable, however, to do all TacticList manipulations before it reaches the ChooseTactic module, so this inhibitor was not used in the final version of the Personality module.

The *consider* methods are described in detail in the following:

considerEmotionalTemperament() - If the EmotionalTemperament attribute is set to true, this method sets the factual value of a tactic equal to the potential value. This means that when a tactic is chosen, the choice is solely based on the value that move has for the agent, disregarding factors like probability of success, relations to other nations and danger for other provinces or units. If this attribute is set to false, no changes are made to the TacticList.

considerRiskAttitudes() - This method also manipulates the TacticList coming from the operational layer of the Caeneus Architecture. It browses through all operations of all tactics in the TacticList and modifies the factual value of all hold operations. If the RiskAttitude attribute is set to true, it will *decrease* the factual value of all tactics containing hold operations with 10% for each operation. If the attribute is set to false, it will *increase* the same values with 10%. This implies that if the attribute is set to true, the agent is more likely to take risks, thus valuing defensive operations (hold operations) less. If the attribute is set to false, the agent favours the safe, thus valuing defensive operations more.

considerAggressiveness() – As the two methods above, this method manipulates the TacticList. If the Aggressiveness attribute is set to true, it will increase the factual value of all tactics containing moves against *enemy nations* with 10% for each move. If the attribute is set to false, it will increase the factual value of all tactics containing moves against *friendly nations* with 10% for each move. An aggressive agent lets its emotions affect its decisions, while a non-aggressive agent coldly considers its options, disregarding relations. This gives the effect that an aggressive agent is more likely to attack its enemies, while a non-aggressive agent is more likely to disregard relationships and see only the value of the move itself.

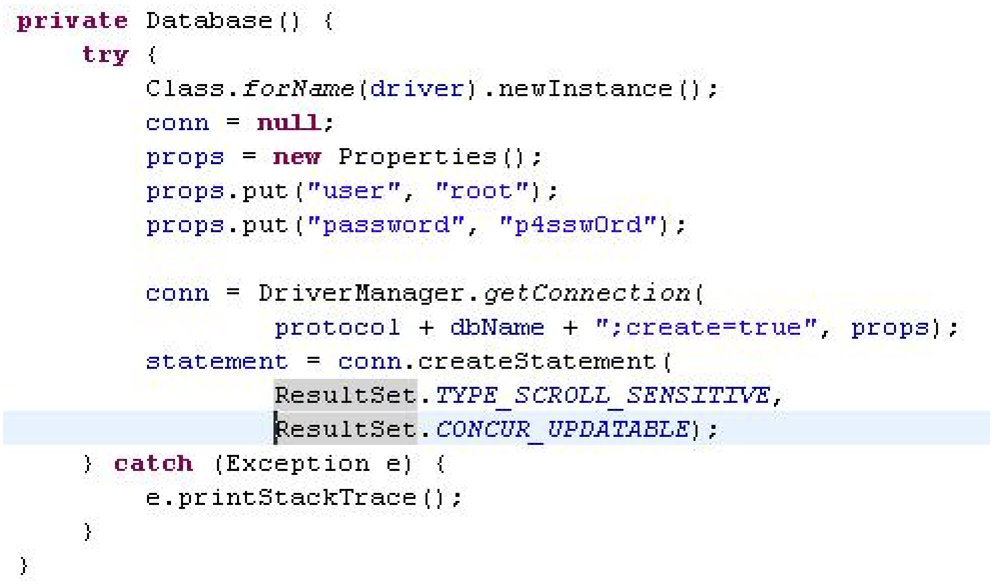
considerContemptForRules() – If ContemptForRules is set to true, this method simply accepts 75% of all requests, regardless of how the underlying layers consider these requests. If it is set to false, no changes occur. It is important to notice that this does not affect the agent's decision on whether to act on this request or not. This is to illustrate that this attribute makes an agent hard to trust, as it will break an agreement more often than not.

4.3 Modifications Made to StateCraft

StateCraft was originally designed as a network game, where clients, both human client and agents, connected to a game server through a network. In addition, the game was designed to download all game data and graphics files from a web server. This makes the threshold for playing the game very high; one need a game server, a game data and graphics server and you need to connect one client for each country in the game. Also, executables existed for neither clients nor server, so one would need a Java compiler to run each of the components. In order to make the game easily accessible for anyone who wanted to play it, whether to test it for us or to play it for fun, we needed to make it able to run as a stand-alone application with as few requirements to the platform as possible, and to make it easily available for potential players. To transform StateCraft to a stand-alone application, we needed to change the database used by the game from MySQL to an embedded database, make the game data available locally and to instruct the clients to connect to a local game server on the same computer as the clients. The availability issue was solved by making the game available for download from a web server. The version available for download will later be referred to as *the distributed version*.

4.3.1 From MySQL to Apache Derby

The first step was to remove the dependency of a database server installed on the game server machine. The solution we came up with was to replace the existing SQL database with Apache Derby database [39], an Apache DB Subproject [40]. Apache Derby is a small, open-source, purely Java relational database that provides an embedded JDBC driver which allows it to be embedded in Java applications. The database is relatively small, only about two megabytes, so it is well suited for easy distribution and uses the Structured Query Language (SQL) as query language. The class Database in package server.core handles all database communication. This class has been altered to connect to and communicate with an embedded Apache Derby database (Code Excerpt 3).

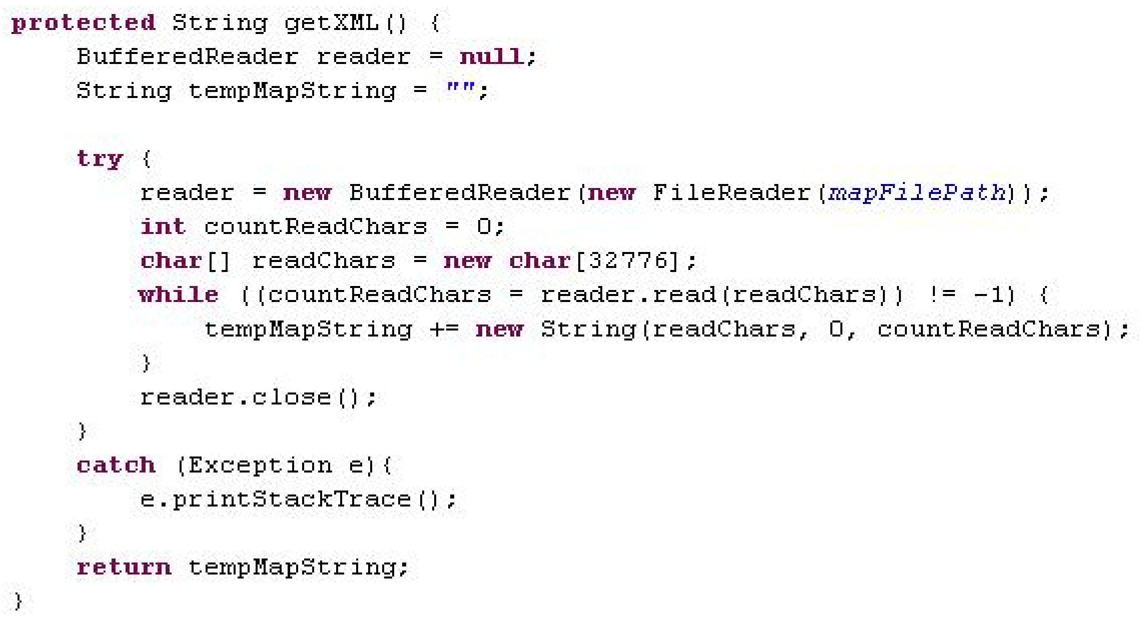


```
private Database() {
    try {
        Class.forName(driver).newInstance();
        conn = null;
        props = new Properties();
        props.put("user", "root");
        props.put("password", "p4ssw0rd");

        conn = DriverManager.getConnection(
                protocol + dbName + ";create=true", props);
        statement = conn.createStatement(
                ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Code Excerpt 3 The Database class constructor

4.3.2 Location of Game Data

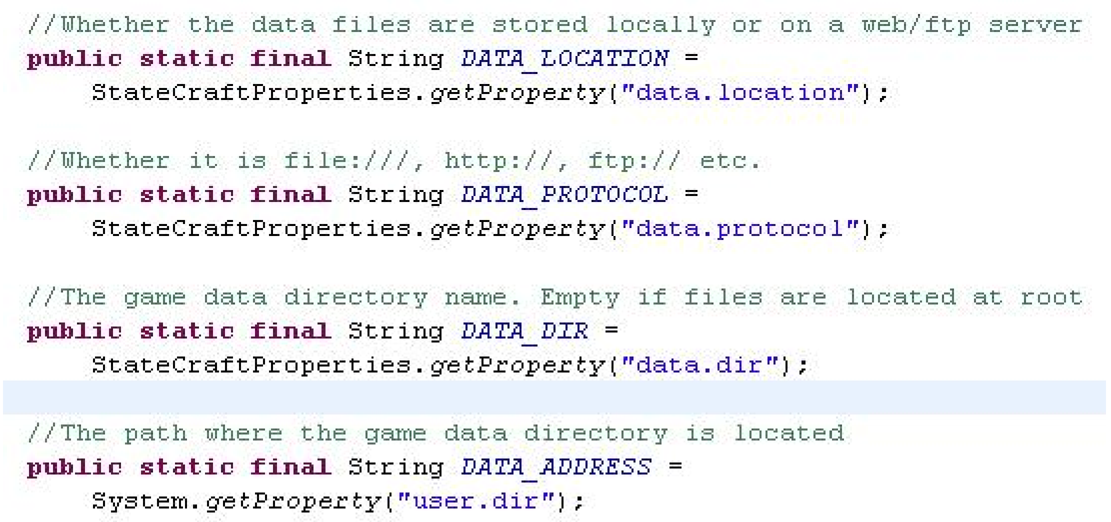
The game data consists of graphics files in Portable Network Graphics (PNG) format and XML documents containing map data. The game was originally designed to access these files from a web server. Net***Creator classes were made to connect to the server through a socket and fetch the XML files, while the graphics files are fetched individually from the classes needing them. The Net***Creator contains only methods for loading the files from a location; the generation of the data is inherited from abstract, generic classes using an XML parser. This made it easy to create Local***Creator classes to load the XML files from a relative, local directory, **<game directory>\GameData**, and inherit the handling of the files from the abovementioned abstract classes. Three such classes were created, all inheriting from abstract superclasses. *LocalStateCreator*, inheriting from the *StateCreator*, loads the game data file **new_game.xml** and creates from this the starting game state. Code Excerpt 4 shows an example of the Local****Creator classes used to access game data and graphics files locally. *LocalGraphicsCreator*, inheriting from *GraphicsCreator*, loads the graphics file **map_graphics.xml** and creates the game map, while *LocalMapCreator*, inheriting from *MapCreator*, loads the map data file **map_data.xml**, which contains the metadata for the map; like which country does a province originally belong to, which provinces contains a supply centre and which province has a path to where.



```
protected String getXML() {
    BufferedReader reader = null;
    String tempMapString = "";

    try {
        reader = new BufferedReader(new FileReader(mapFilePath));
        int countReadChars = 0;
        char[] readChars = new char[32776];
        while ((countReadChars = reader.read(readChars)) != -1) {
            tempMapString += new String(readChars, 0, countReadChars);
        }
        reader.close();
    }
    catch (Exception e){
        e.printStackTrace();
    }
    return tempMapString;
}
```

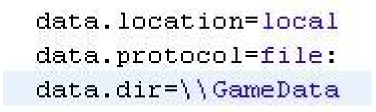
Code Excerpt 4 LocalMapCreator.getXML()

The PNG graphics files, located in the same directory as the XML files, contain graphics for elements such as flags, units, supply centres, menus and timer. The location of these files was held by constants in the utility class HTTPToolkit. We modified this class to get the constants from a properties file, **statecraft.properties**, and created a class, *StateCraftProperties*, for extracting them from the properties file. The environment specific constants are fetched from the system. This solution makes it easy to configure where to access the files, whether located at a local address, an http address or an ftp address. Code Excerpt 5 and Code Excerpt 6 show how the game configuration works.



```
//Whether the data files are stored locally or on a web/ftp server
public static final String DATA_LOCATION =
    StateCraftProperties.getProperty("data.location");

//Whether it is file:///, http://, ftp:// etc.
public static final String DATA_PROTOCOL =
    StateCraftProperties.getProperty("data.protocol");

//The game data directory name. Empty if files are located at root
public static final String DATA_DIR =
    StateCraftProperties.getProperty("data.dir");

//The path where the game data directory is located
public static final String DATA_ADDRESS =
    System.getProperty("user.dir");
```

Code Excerpt 5 Constants from HTTPToolkit.java



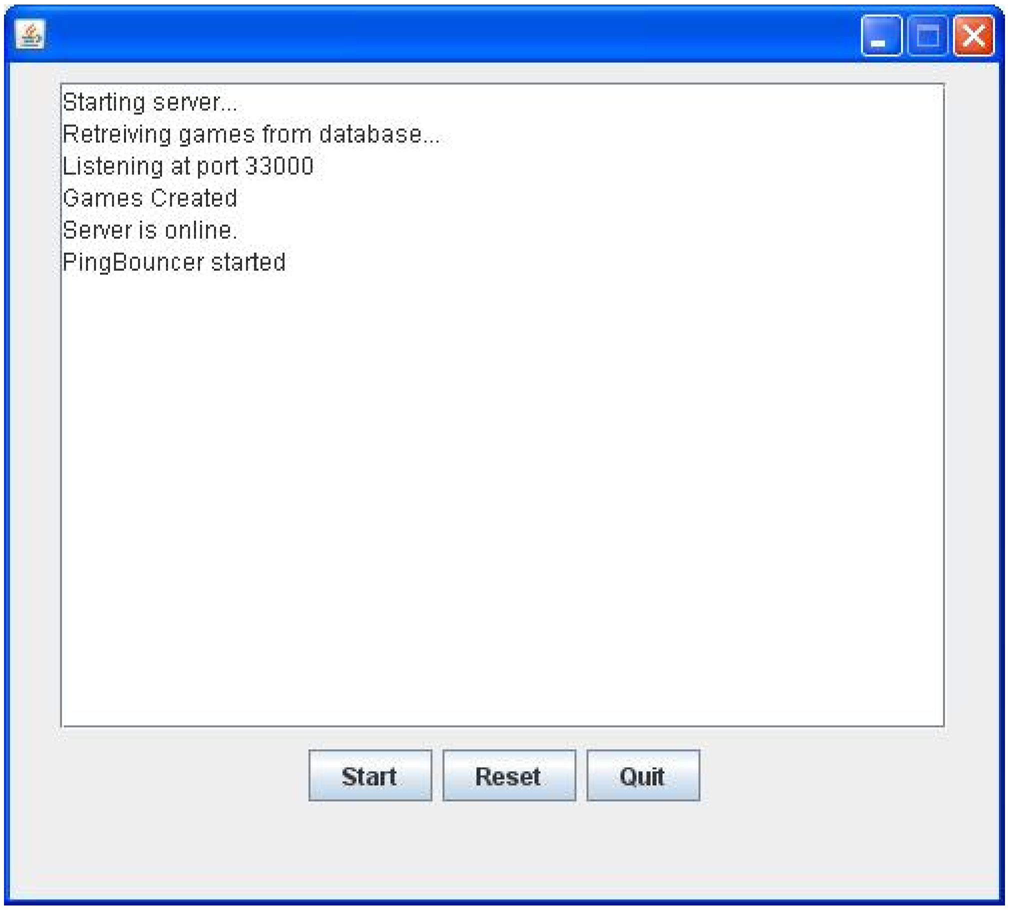
```
data.location=local
data.protocol=file:
data.dir=\\GameData
```

Code Excerpt 6 Extraction from statecraft.properties

4.3.3 Game Connection

We have made no changes in the actual architecture; the game still consists of one server and one to seven clients, all connected through a socket and communicating through messages sent over this socket. However, as with the graphics files location described in section 4.3.2, we have put the connection info, like connection URL, in a property file. This way, it is easy to change between local connection and connection to a remote server. A property loader class sets the appropriate variables according to the properties in the property file. The distributed version, however, does not use the property file; it is prefixed on a local connection.

All output from server has been routed to an output window instead of sending to console, as the game is started without a console window. This output window also contains a start / pause button, a button for resetting the game data (start new game) and a quit button. The start / pause button was used when running the game with agents only. In the distributed version this button is hidden, as the player controls the progress of the game. The buttons represent all possible input a player might need to give the server. If the quit button is pressed, the server sends a quit message to all clients (thus making them close) and then shuts down. Pressing the reset button resets all game data in the database to its original state.

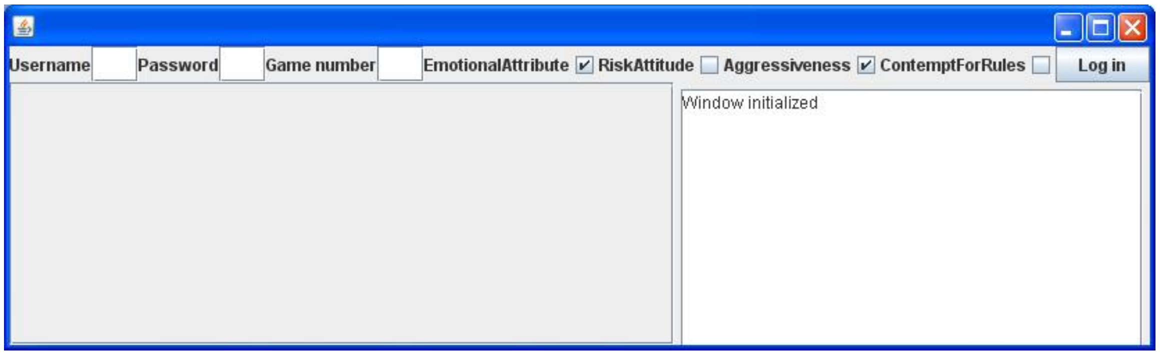


```
Starting server...
Retreiving games from database...
Listening at port 33000
Games Created
Server is online.
PingBouncer started
```

Figure 15 The server output window

4.3.4 Graphical User Interface

The agent class (CaeneusAgent) has been given a new constructor which takes a username and a password (both equals the nation's name in lowercase, e.g. "france"), game number (in the distributed version, only one game, game number 1, exists) and the Boolean value of the four personality attributes and then connects the agent without creating the GUI. This is to prevent the game from showing GUIs for the agents when run as a stand-alone application, so that the player sees only the client GUI. If no arguments are given, the agent will start with the login window and will show the agent interface when connected. If the server is not available, the agent keeps trying to connect. The reason for this is to give slower machines time to start the server. If, however, the server fails to start, the agent will loop until terminated by the user. Attributes can be configured as arguments or using the agent user interface (see Figure 16). Checking a checkbox sets the corresponding attribute to true while leaving it clear sets it to false.



Window initialized

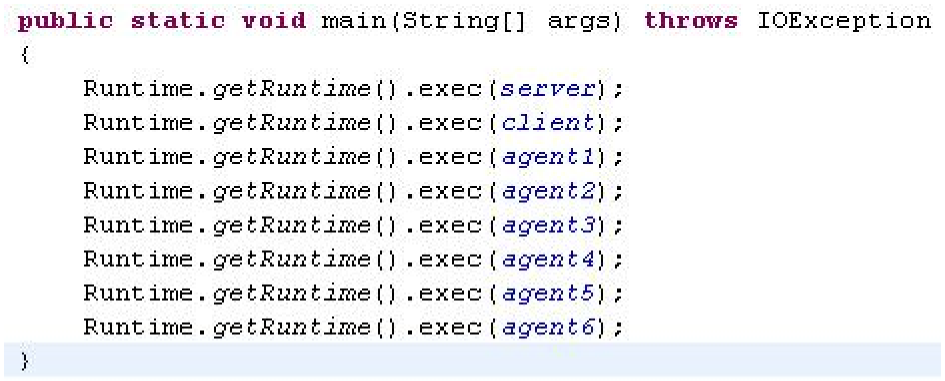
Figure 16 The agent's login window

The original user client class, DiplomacyApplet, was, as the name suggests, an applet, which was unsuitable for our needs, so we created a new user client class, UserInterface. UserInterface uses most the same elements as DiplomacyApplet, but the login screen has been removed. In the distributed version of the game, the player can only play one nation, Russia. Thus, when started, the client connects to the game server automatically, using "russia" as username and password, hiding the login screen. When connected, a list of the games available is shown, together with a play button. In the distributed version, one and only one game is available, as mentioned above. To play, the user presses the play button. This closes the window and opens the game interface, which is identical with the original game interface, except for two buttons; a quit button and a next round button. The quit button, if pressed, sends a quit message to the server (which, in turn, sends a quit message to all clients) and then closes the client. The next round button is what drives the game forward. To give the player the time he needs and wants and due to there being one and only one player in each game, we removed the deadline for each round and replaced it with a next round message sent from client to server. When receiving this message, the server moves the game to the next round.

As described in section 3.1.2, when a nation controls eighteen supply centres at the end of the fall turn, the game is over and that nation is the victor. The original StateCraft contained no mechanics for handling this, the game just kept moving on. To inform a player of a game won or lost, we implemented a simple method, checkForVictory() in the game server. This method checks whether any nation possesses eighteen or more supply centres and returns that nation if true or nothing if false. If a nation is returned, the game server sends a message to all players, which displays a dialog box showing this nation's name and an explanation text.

4.3.5 Game Execution and Distribution

To make the game as easy as possible to run, we wanted to create one single executable file that starts and sets up all parts of the game. For testing purposes, we also wanted to force the user to play one nation and one nation only. The latter has been achieved by skipping the client login screen, as mentioned above. To bundle the game into one executable file, we first needed to create a launcher class to launch the game server and all the clients. The *GameLauncher* class was created for this purpose and the game server, the client and the agent were exported to separate Java ARchive (JAR) files. The *GameLauncher* class uses the static *Runtime* instance to execute these JAR files. The JAR file containing the agent is executed six times, creating one instance for each agent.

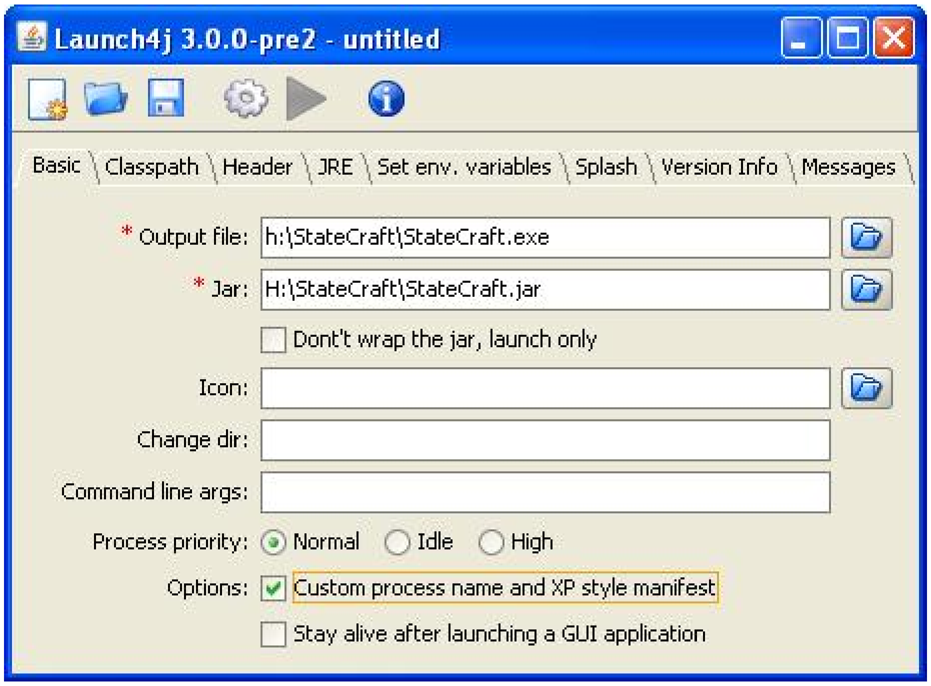


```
public static void main(String[] args) throws IOException
{
     Runtime.getRuntime().exec(server);
     Runtime.getRuntime().exec(client);
     Runtime.getRuntime().exec(agent1);
     Runtime.getRuntime().exec(agent2);
     Runtime.getRuntime().exec(agent3);
     Runtime.getRuntime().exec(agent4);
     Runtime.getRuntime().exec(agent5);
     Runtime.getRuntime().exec(agent6);
}
```

Code Excerpt 7 GameLauncher.main()

The *Runtime's* execute method, *exec(String[])*, takes one argument, a String array containing the command to be executed. The commands have the form **java –jar <filename> <argument 1,argument N>**. The code in Code Excerpt 7 is from the distributed version. The version used for simulations would have the client replaced by a seventh agent. This last agent was started without arguments, to provide us with the possibility of observing the game. The GameLauncher class was in turn exported to a JAR file.

As people without a software development background are not necessarily familiar with JAR files, the threshold for testing the game would be lower if we distributed it as an executable (EXE) file. As opposed to other programming languages, like C++ and VB.NET, Java offers no possibilities to compile the code into an EXE file, so third-party software was needed. Launch4j Executable Wrapper [41] is an Open Source application for wrapping JAR files into executable files for Windows, see Figure 17. It is lightweight and easy to use, thus well suited for our purposes.


Launch4j 3.0.0-pre2 - untitled

Basic | Classpath | Header | JRE | Set env. variables | Splash | Version Info | Messages

* Output file: h:\StateCraft\StateCraft.exe

* Jar: H:\StateCraft\StateCraft.jar

☐ Dont't wrap the jar, launch only

Icon:

Change dir:

Command line args:

Process priority: [✓ Normal] Idle High

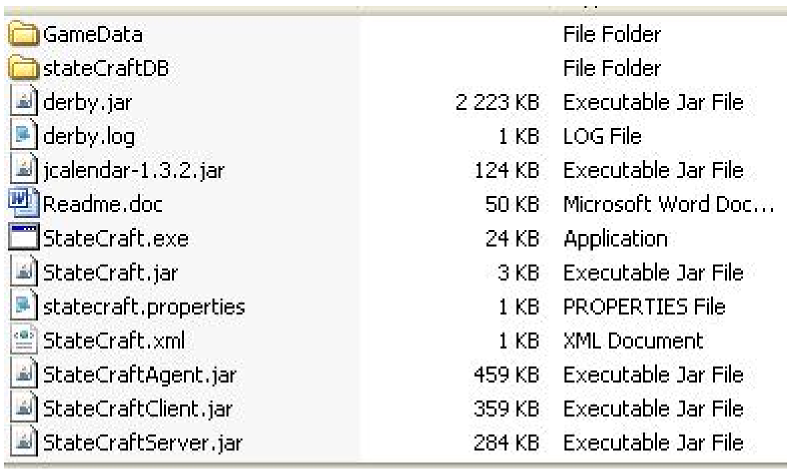
Options: ☑ Custom process name and XP style manifest

☐ Stay alive after launching a GUI application

Figure 17 The Launch4J interface

Of course, by doing this, all platform-independency is removed; the EXE file is only executable in a Microsoft Windows environment. Users with another platform would have to run the JAR file directly. As Windows is by far the most used platform, this was not considered as an issue. The executable checks if the needed Java Runtime Environment (JRE) is installed, if not, it provides the user with the URL for downloading the JRE. If the needed JRE is present, the application is executed.

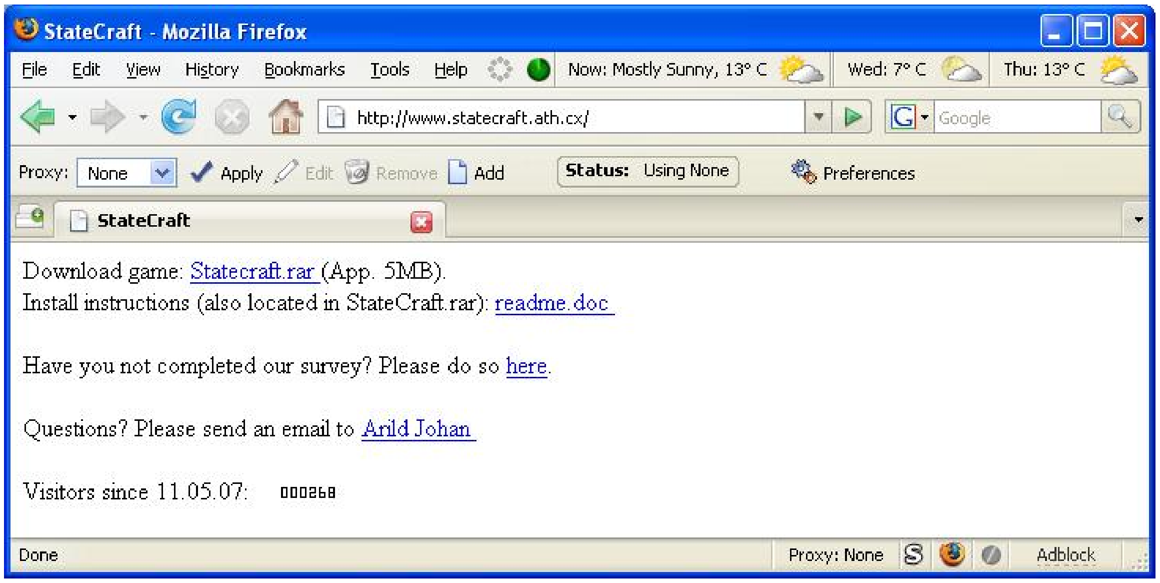
Table 3 The game files



GameData		File Folder
stateCraftDB		File Folder
derby.jar	2 223 KB	Executable Jar File
derby.log	1 KB	LOG File
jcalendar-1.3.2.jar	124 KB	Executable Jar File
Readme.doc	50 KB	Microsoft Word Doc...
StateCraft.exe	24 KB	Application
StateCraft.jar	3 KB	Executable Jar File
statecraft.properties	1 KB	PROPERTIES File
StateCraft.xml	1 KB	XML Document
StateCraftAgent.jar	459 KB	Executable Jar File
StateCraftClient.jar	359 KB	Executable Jar File
StateCraftServer.jar	284 KB	Executable Jar File

This left us with the files and directories listed in
Table 3. These elements were compressed to an archive file using WinRar [42].

To distribute the game, we set up a web server using the Apache HTTP Server 2.2.3 [43] and created a simple web site. We registered the free domain http://www.statecraft.ath.cx at DynDNS [44], a website providing a service which enables mapping to a dynamic IP address. This page (Figure 18) also contained a link to the evaluation questionnaire and to the readme file, which also was included in the archive file, and a simple counter to track the number of visitors.


StateCraft - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Now: Mostly Sunny, 13° C
Wed: 7° C
Thu: 13° C
http://www.statecraft.ath.cx/
Google
Proxy: None Apply Edit Remove Add
Status: Using None
Preferences
StateCraft
Done
Proxy: None
Adblock

Download game: Statecraft.rar (App. 5MB).
Install instructions (also located in StateCraft.rar): readme.doc

Have you not completed our survey? Please do so here.

Questions? Please send an email to Arild Johan

Visitors since 11.05.07: 000268

Figure 18 The StateCraft download web site

5 Evaluation

In developing the personality module, we plan to shed some light on two research questions: firstly, whether different personality traits could affect the agents' performance in a strategy game, and secondly, and most importantly, whether giving agents personality can improve the game experience. The evaluation is thus divided into two sections. In the first part, section 5.1, we will describe how we tested the agents' performance by running a series of simulations with and without the personality module. The second part, section 5.2, describes how we tested the game with human participants and how they experienced the game.

Krzywinski and Helgesen [3] tested their architecture through test plays of the game with six human players and one agent and asked their participants to guess which nation was controlled by the agent. We have implemented four personality attributes and want to see if different combinations of these make the agents play differently and affect the game experience for one player. In addition, we wanted to see if the personality attributes affect the agents' performance. Thus, we needed to make the game run with six agents and one player, as well as with seven agents and an observer GUI. A lab test with the technical solution of the original StateCraft would require one game server for each player and one machine required for each game server. As each of our test subjects would play as the single human against six agents, eight computers would be needed for each test! We therefore decided that the best way for us to get the data material we wanted, was to make the game run as a stand-alone application and make it available for download on a web server, as described in section 4.3. This way, it could be easily obtained and run by anyone who wanted to test the game (or simply play it for fun), regardless of geographic locations and computer environment. The test with only agents was handled by ourselves and run from the development environment.

5.1 Simulations

To evaluate the differences in gameplay performance, we ran two series of 50 simulated games with only agent-controlled players. In Test Series 1, the Personality module was disabled, in Test Series 2, it was enabled. Table 4 shows the configuration of the agents in test series 2.

Table 4 The Personality attributes for Test Series 2

Country	EmotionalTemp	RiskAttitudes	Aggressiveness	ContemptForR
England	X	X	X	X
France				
Italy	X	X		
Germany			X	X
Austria-Hungary	X		X	
Turkey		X		X
Russia	X			X

No country ever achieved the 18 centres required for a standard Diplomacy win in either test series. Since no country was able to win the game, the country with the most supply centres by 1930 is considered the winner.

Table 5 The rankings from Test Series 1

Place	Country	Number of supply centres at end of simulation
1	France	5.86
2	Austria-Hungary	5.3
3	England	5.24
4	Italy	4.8
5	Russia	4.44
6	Germany	4.18
7	Turkey	4.08

Table 6 The rankings from Test Series 2

Place	Country	Number of supply centres at end of simulation
1	France	6,3
2	Germany	5,54
3	England	5
4	Italy	4,8
5	Turkey	4,5
6	Austria-Hungary	4,44
7	Russia	4,36

As Table 5 and Table 6 show, France was the best nation in both test series. It averaged 5.86 centres in Test Series 1 and 6.3 in Test Series 2. France and Turkey (4.08/4.5) were the only two countries to perform better in Test Series 2. Out of the 50 games in Test Series 2, France only once ended the game with less than 2 supply centres. In this game, it had 0 centres, while Italy had 8 supply centres and with that won the game.

Out of the seven countries, France is in the best position geographically. To the West it has the Atlantic Ocean. In StateCraft, England, Germany and Italy lie along its other borders. Other countries will have to pass through one of these countries to reach France. The Risk Attitudes attribute is France’s best protection. In Test Series 2, we saw that countries were more conservative in their actions. This means France was less likely to be attacked. France’s geographical position also protects the country when the Aggressive AI is active. Most countries will have to cross either the ocean or France’s border countries. The border countries are less likely to attack France since they have to worry about their neighbours as well.

Like France, Turkey also has fewer countries touching its borders. It is flanked by Russia, Austria-Hungary and Italy. The Black and Mediterranean Seas make the country more accessible by water than France. Unlike France, Turkey experienced 4 games where it ended with less than 2 supply centres in Test Series 2. The Risk Attitudes AI seems to be Turkey's weakness. One of its starting supply centres, Constantinople, is easily accessible by its neighbours. Turkey will lose Constantinople if it plays it safe. On the other hand, the Aggressive AI and an Army supported by a Fleet unit usually helps Turkey take at least 2 supply centres in Austria-Hungary.

Austria-Hungary and England had the largest decline in supply centre average from Test Series 1 to 2, but are also the only two countries never to end the game with less than 2 supply centres in Test Series 2. Austria-Hungary's average improved by 0.86, going from 4.44 to 5.30 supply centres. This jump exceeds France's increase of 0.44. During Test Series 1, Austria-Hungary had 3 games of more than 8 centres (10, 10 and 11) and 3 games with less than 2 centres (0, 0 and 1), while England had 2 games with more than 8 supply centres (9 and 9) and none with less than 2.

Austria-Hungary's three supply centres, Vienna, Budapest and Trieste, are almost completely surrounded. Turkey's Constantinople, Russia's Sevastopol and Warsaw, Italy's Venice and Germany's Munich are all within two moves of an Austria-Hungary supply centre. Only to the south-east is there a buffer of four neutral regions. Austria-Hungary is in a sink or swim position. It's closeness so many supply centres makes it easy either to advance into many countries or be invaded on multiple fronts. In games where Austria-Hungary had more than 8 supply centres, Italy either had 0 or 1 centre remaining. It is likely that Austria-Hungary acquired Venice early in the game and then captured Naples or Rome. Italy had 5-7 supply centres when Austria-Hungary had 0 or 1.

In Test Series 1, more countries ended the game with less than 2 Supply Centres than in Test Series 2. In Test Series 1, this occurs 21 times: Germany 7 times, Italy 5, Russia and Austria-Hungary 3, France 2 and Turkey 1. It only happened 13 in Test Series 2. The

Risk Attitude attribute set to false encourages countries to defend their supply centres. With Germany and Russia being two of the three countries that most often ended the game with less than 2 supply centres, it seems that this carefulness not always is a good strategy.

Table 7 The different coalitions when Germany ended the game with less than 2 supply centres

	England	France	Italy	Austria-Hungary	Turkey	Russia
1	4	8	3	8	4	7
2	8	5	6	7	3	4
3	9	9	2	4	6	4
4	7	6	7	6	4	3
5	6	7	5	0	7	8
6	8	7	6	2	5	5
7	5	10	4	4	4	6

Geographically, Germany's supply centres are well protected against one invading army. Thus, the 7 times that Germany ended the game with less than 2 supply centres, it was attacked by at least two countries (Table 7). Austria-Hungary (8), Russia (7) and France (8); France (9) and England (9); England (8), Italy (6) and Austria-Hungary (7); England (6), France (7), Turkey (7) and Russia (8) and France (10) and Russia (6) are some of the coalitions that attacked Germany.

England also benefited from its geographical location. Its three supply centres are protected by ocean regions. England was the only country not to end a game with less than 2 supply centres. Between both test series, it only ended 3 games with 2 supply centres. In Test Series 2, England ended 2 games with more than 8 supply centres. These results indicate that England has the greatest stability over time when compared with the other nations.

France is the most successful expansionist, but regardless of whom the winner was, England always maintained at least 2 supply centres. In Test Series 1, France won 7 games with more than 8 supply centres (11, 10, 10, 9, 9, 9, and 9). It was almost as successful in the tests with a standard AI by winning 5 games with more than 8 supply centres (10, 10, 9, 9 and 9).

StateCraft is a very challenging game for a realistic AI, as it has to cope with both the game world and the social world. However, due to the limitations in the diplomacy section of StateCraft, we can assume that diplomatic actions played an insignificant role in the simulations.

A comparison of the results from the two simulation series shows noticeable differences. Larger testing series (200 or more games each) should bear similar results. The AI with Personality series showed a more balanced European continent. The AI without Personality game series had more ending with countries having either more than 8 or less than 2 supply centres. Personality AI games ending with neutral supply centres only happened 4 times (5 times without personality).

5.2 Human Feedback

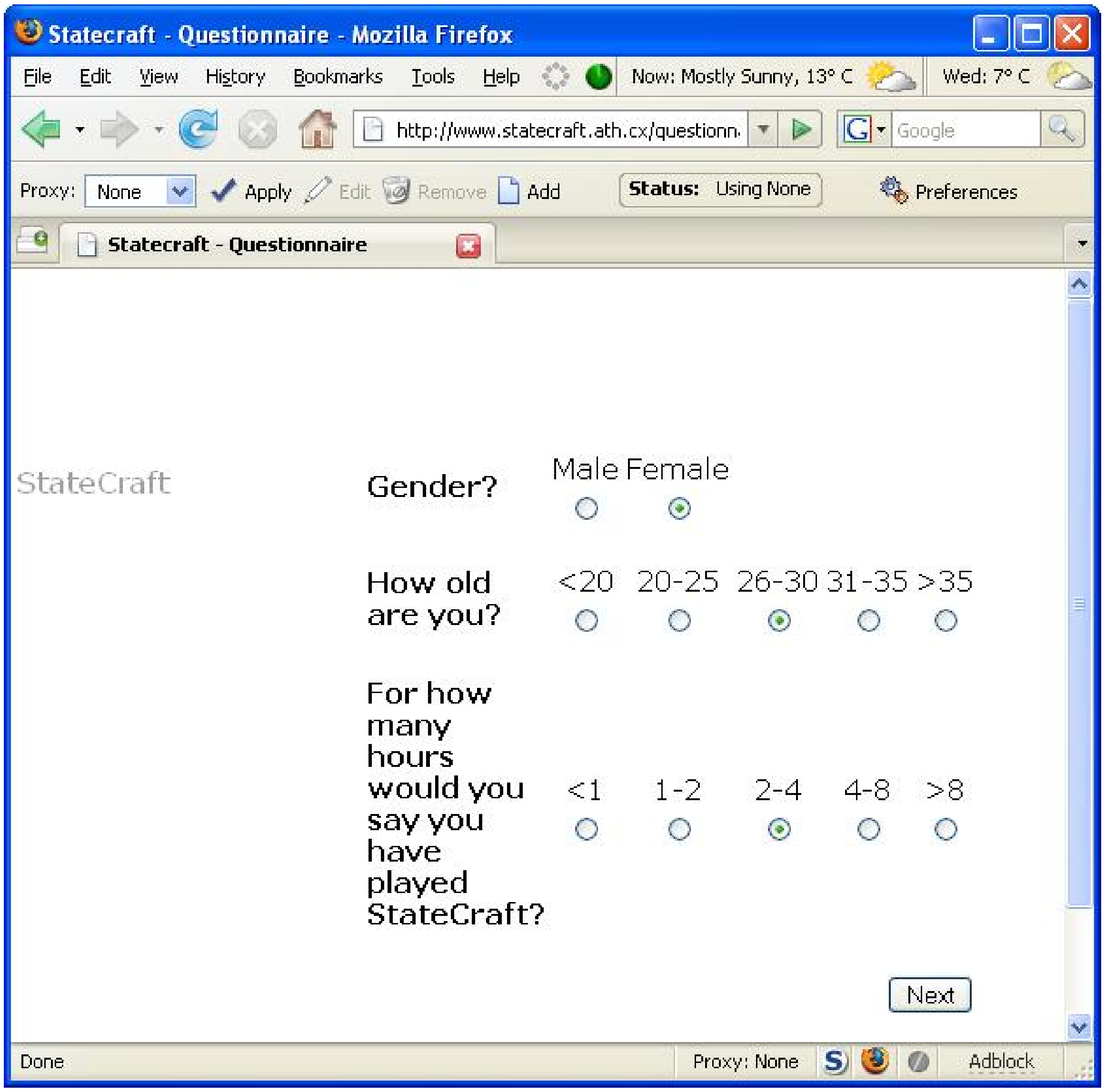
In this section, we will discuss the results from the human players testing StateCraft and how they relate to the implemented personality attributes. The section consists of three sub-sections. In the first, we will describe the test subjects as a group and how data were collected. Next, we give a statistical overview of the test results, while the third compares the results with the personality attributes.

5.2.1 The Testers and the Questionnaire

To comprise all types of computer game players, we initially set no demands for those who wanted to test StateCraft. All that was needed to test the game was an available computer and, preferably, an Internet connection to download the game. To use other platforms than Microsoft Windows, however, familiarity with command line execution of Java archive (JAR) files was needed. As reactions from testers began to reach us, it soon became clear that due to the game’s rather complex rules, some gaming background was needed to master and enjoy the game. Familiarity with strategy games in general and

Diplomacy in particular, seemed to be a great advantage, as the eight pages long README distributed with the game (see Appendix B), was reported too extensive to read. Despite these apparent obstructions, people played the game and even enjoyed it.

As described in chapter 4.3.5, the game was made available from a web site. At this web site, we also placed a link to a feedback form, a questionnaire, where the testers could evaluate the game and the experience, in addition to providing us with some basic information about themselves and their gaming background.


Statecraft - Questionnaire - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Now: Mostly Sunny, 13° C
Wed: 7° C
http://www.statecraft.ath.cx/questionn
Google
Proxy: None Apply Edit Remove Add
Status: Using None
Preferences
Statecraft - Questionnaire
Done
Proxy: None
Adblock

StateCraft

Gender? Male Female [✓ Female]

How old are you? <20 20-25 [✓ 26-30] 31-35 >35

For how many hours would you say you have played StateCraft? <1 1-2 [✓ 2-4] 4-8 >8

Next

Figure 19 The first page of the questionnaire.

The questionnaire was written in Active Server Pages (ASP) and the results were stored in a Microsoft Access database. The first part, shown in Figure 19, and the second, contain questions regarding the tester’s person and background information, the third enquires about the levels of entertainment and difficulty StateCraft presents, while the last part asks specific questions about the agents, their gameplay and their personality. The last page provides a comment field and a field where the tester can enter his or her email address if s/he wants to play additional versions of the game. For the full questionnaire, see Appendix E.

In addition to the data collected from the questionnaire, we also had conversations with some of the test subjects during the testing, conversations that provided us with valuable information not possible to retrieve from the questionnaire. Especially Ronny, an experienced board-game and Diplomacy player, sent us frequent and long emails with observations from the game and revealed an insight in and knowledge about Diplomacy far superseding our own.

5.2.2 The Test Results

We got a total of six feedbacks from testers, all male, their age divided evenly between 21-25 and 26-30 (Table 8). All the testers had some gaming background, although of varying degree (Table 9), but only two had ever played Diplomacy (Table 10). Most of the testers had never heard of the game. All the testers without any Diplomacy experience reported the game to be somewhat difficult, one tester reported to only win the game once, while the two with previous knowledge of the game found StateCraft easy to win. The testers novice at Diplomacy perceived no difference between the gameplay of the different opponents and perceived few or none distinct personality traits, while the experienced noticed clear distinctions between the different agents’ gameplay. When sorting on age or general gaming experience, no clear differences between the feedbacks were found. When sorting on time spent with the game, one tester, a tester with previous knowledge, stood out, with a reported time consummation of eight hours or more (the questionnaire's maximum choice). The other testers reported to have spent between one and four hours with the game.

Table 8 Testers by age

Age	21-25	26-30
Testers	2	4

Table 9 Testers by game experience

Gaming experience	1 (least)	2	3	4	5 (most)
Testers	0	1	0	3	2

Table 10 Testers by Diplomacy experience

Diplomacy experience	1 (least)	2	3	4	5 (most)
Testers	4	0	0	1	1

For a full report of the test results, see Appendix F.

5.3.3 Perceived Personalities vs. Implemented Personalities

Table 11 The personality attributes in the distributed game

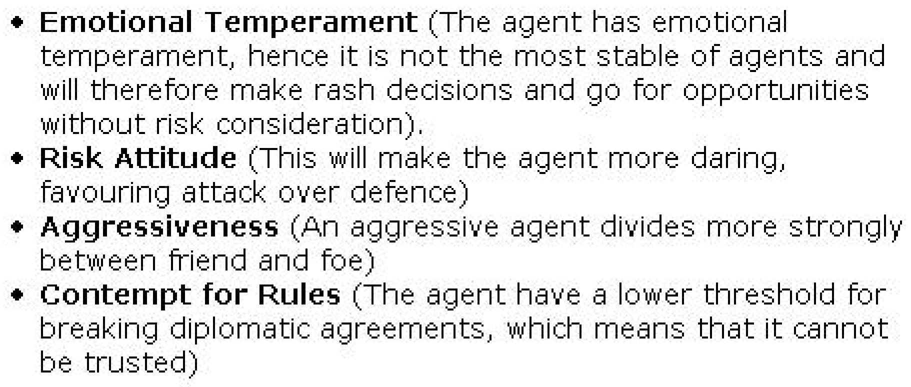
Country	**EmotionalTemp**	**RiskAttitudes**	**Aggressiveness**	**ContemptForR**
England	X	X	X	X
France				
Italy	X	X		
Germany			X	X
Austria-Hungary	X		X	

Turkey		X		X

Table 11 shows the personality attributes configured in the game distributed from our website. As described in section 4.2.1, the attributes *Risk Attitude* and *Aggressiveness* had impact on the agent's decision-making both when true and false, while *Emotional Attitude* and *Contempt For Rules* only affected the agent when set to true. We therefore assume that if an agent is described as *passive* or *inactive* in any way, the tester means *not Aggressiveness*. Similarly, *defensive* or *protective* will be interpreted as *not Risk Attitude*. *Emotional Attitude* will only be regarded if described as true. Feedback from the testers and our own experience showed that the diplomacy section of the game was rather limited, that the agents never took the initiative to diplomatic talks and never followed up on a diplomatic agreement, regardless of personality attributes. Still, an agent with *Contempt For Rules* set to true will accept 75 % of all support requests, and by never honouring these agreements, it may create an illusion of deviousness, albeit a deviousness that will not trick a player for long.

The questionnaire contained two sections regarding the personality attributes. First, the testers were asked to describe how they perceived the agents' behaviour in their own words. The feedback from the testers here varied a great deal. England was by one described as "*Not very aggressive. Was more into holding than conquering.*" while another felt that this agent was "*quite opportunistic*". France was perceived both to be an "*aggressive agent that will attack often and also takes unguarded territory*" and "*to be very passive*". All the agents were described as both passive and aggressive by different testers. No other descriptions were used. One of the experienced players found France, Italy, Turkey and Austria-Hungary passive, Germany aggressive and described England as "*to play ok most of the time with good moves and mostly ok support moves*" and by that hit close to the actual aggressiveness for most of the agents. All the other testers came out at least 50 % wrong. However, one must keep in mind that the testers at this point knew nothing of our personality attributes, and based on the testers' comments, their interpretation of *aggressiveness* and *passiveness* is closer to the true and false values of *Risk Attitudes*.

The next part described the personality attributes and contained checkboxes where the testers could check the attributes they felt matched each of the agents' gameplay.



- **Emotional Temperament** (The agent has emotional temperament, hence it is not the most stable of agents and will therefore make rash decisions and go for opportunities without risk consideration).
- **Risk Attitude** (This will make the agent more daring, favouring attack over defence)
- **Aggressiveness** (An aggressive agent divides more strongly between friend and foe)
- **Contempt for Rules** (The agent have a lower threshold for breaking diplomatic agreements, which means that it cannot be trusted)

Figure 20 The questionnaire's description of the personality attributes

The testers were not told that *not checking* a box means that they felt that the agent *did not* match this attribute, so here, unchecked boxes are not regarded as the opposite of a checked box. Again, we saw great variation in the testers' perception of the agents' personalities. As we can see in Table 12, when added together only four attributes were never checked. This gives a hit ratio of 47.8 %.

Table 12 The sum of all the testers (the Os mark matches with the implemented attributes)

Country	**EmotionalTemp**	**RiskAttitudes**	**Aggressiveness**	**ContemptForR**
England	O	O	O	O
France	X	X	X	X
Italy		O	X	O
Germany	X	X	O	X
Austria-Hungary	O	X	X	X
Turkey	X	O	O	O

As in the previous section, there was a noticeable variation between the experienced testers and the rest when it came to the three attributes concerning the actual game play, that is, all the attributes but *Contempt For Rules*.

Table 13 Attributes perceived by the experienced testers

Country	**EmotionalTemp**	**RiskAttitudes**	**Aggressiveness**	**ContemptForR**
England	O		O	
France	X			

Italy		O	X	
Germany	X	X	O	
Austria-Hungary	O		O	
Turkey		O	X	

Table 14 Attributes perceived by testers without Diplomacy experience

Country	**EmotionalTemp**	**RiskAttitudes**	**Aggressiveness**	**ContemptForR**
England		O	O	O
France		X	X	X
Italy			X	O
Germany	X			X
Austria-Hungary		X	O	X
Turkey	X	O	X	O

As we can see from Table 13 and Table 14, while the experienced tester hit five attributes wrong and had a hit ratio of 58 %, the rest of the testers in sum had a hit ratio of 41.1 %. This solidifies the impression from the previous section of the questionnaire that a player who knows the game has a much better possibility of recognising personal traits in his or her opponents.

5.3.4 Manual Feedback

As from the questionnaire, the manual feedback from the testers varied largely. While Ronny, an experienced computer game, board game and Diplomacy player, understood the game interface instantly and won all games from the start, Espen, new to Diplomacy and not very experienced in strategy games, had a hard time figuring out the game controls and rules and tried a long time before he managed to win a game. Always winning a game without problems eventually leads to a deterioration of the game experience. Ronny commented “You only lose if you screw up when playing against the AI“ and “The kind of risks that you can take against the AI players when they are all AIs are things that you are unable to do when you are playing opponents who are able to plan turns ahead and can read strategic type decisions based on your current board positions.“

Due to this, levels of difficulty were the most important thing Ronny felt the game lacked, in addition to a better diplomacy interface and an AI planning ahead.

6 Conclusion and Future Work

The goal of this thesis was to answer the following two research questions:

1. Can different personality traits affect the agents' performance in a strategy game?
2. Can agent personality improve the game experience of the players?

This has been done by completing the following tasks:

- Studying relevant literature and work
- Implementing the Personality module and integrating it in the Caeneus Architecture
- Transforming StateCraft into a stand-alone application
- Evaluating the Personality module

6.1 Relevant Literature and Work

In chapter two, we have presented relevant work and literature. We have briefly discussed computer games and AI. Some research has been done personality AI, but not much has been implemented in games. Some exceptions, like Age of Empires III, have been discussed. Techniques used to implement personality AI in games have not been discussed in this thesis. The focus in this thesis has been to see how such personalities can affect agents in a turn-based strategy game. As a part of this, StateCraft with the Personality module has been compared with the original StateCraft in a series of simulations.

6.2 Implementing the Personality Module

To answer our research questions, we needed a test environment, a game in which personality attributes could be applied. We had several options for finding a suitable game; we could develop a game from scratch, develop a game to an existing game engine, develop an AI component for an existing game or modify an existing agent. We agreed on developing the Personality Module for the Caeneus Agent and using StateCraft as test environment.

The Personality module provides the possibility of configuring four different personality attributes for the Caeneus Agent. Three of these attributes affect the agent’s actions on the game board, while the fourth affects the agent’s diplomatic decisions. The personality attributes used were chosen from Zamora’s individual and social attributes [36].

Possible improvements to the Personality module can be the possibility to grade the personality attribute, give them values e.g. from 0-10 and to add more attributes. Feedback from the testers showed that the limited options in the diplomacy section of the game made it impossible to evaluate the agent’s diplomatic actions and that agents never proposed deals or followed up on agreements.

6.3 Modifying StateCraft

A computer game should be easily accessible and easy to run. StateCraft, only accessible for source code download and dependent on a game server, a web server and several client machines, was neither. We transformed StateCraft into a stand-alone application to run with one user client and six agents and made it available for download from a web server.

As this application essentially starts eight separate Java applications, it is a resource demanding application testers experienced to be at times very slow. This could be improved by removing the original server-client architecture and wrapping it into one single Java application. However, this would also deprive the game of the multiplayer possibility.

6.4 Evaluating the Personality Module

The evaluation of the Personality module consisted of two parts, each corresponding to one of the research questions. To investigate whether the personality attributes affected the agents' performance, we ran a series of simulations with and without the Personality module and recorded the results. The game inevitably crashed after some time and the agents seemed to be very defensive after reaching a certain number of supply centres, thus no agent ever achieved to win the game. Due to this, we ran the simulations for fifteen game years before terminating them and recording the results, that is, the number of supply centres each agent controlled at the point of termination.

Test data from the testers was retrieved using an online questionnaire and stored in a database. The questionnaire included questions about the testers' age and gaming background and specific questions about StateCraft and the agents' gameplay. Among the testers, only two persons had ever played Diplomacy in any form before testing StateCraft. The retrieved data shows that all testers perceived personality differences in the agents, but that testers with Diplomacy experience hit much closer to the actual configured attributes.

6.5 Conclusion

The results from the simulations show that there were perceivable differences in the agents' performance with and without the Personality module. This indicates that the Personality module affects the agents' gameplay performance.

The evaluation of the data retrieved from the questionnaire shows that some experience with a strategy game like StateCraft is needed to perceive personality in agents based solely on their gameplay. Playing the original StateCraft was not a part of this test, so it is not possible to say whether the Personality module *improved* the gameplay experience, but as all testers perceived a notion of personality in the agents, we can assume that the

Personality attribute *affected* the gameplay experience. It also seems that a certain degree of knowledge of a strategy game is required to correctly perceive different personalities in agents.

7 References

1. SourceForge.net, *SourceForge.net*. 2008.
2. Cerny, M. and P. Knut, *Dark Oberon*. 2006.
3. Krzywinski, A.a.H., Arne S., *The Caeneus Architecture - An Agent Architecture for Social Board Games*, in *Department of Information Science and Media Studies*. 2006, University of Bergen: Bergen. p. 142.
4. Krzywinski, A. and A.S. Helgesen, *StateCraft*. 2006.
5. Wikipedia, *Diplomacy the Game*. 2008, Wikipedia.
6. Homer, *The Illiad*. Ca. 800 B.C.
7. Wikipedia, *Tik-Tok*. 2007.
8. Russel, S. and P. Norvig, *Artificial Intelligence: A Modern Approach*. 2003.
9. Luger, G.F., *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. 2005.
10. McCarthy, J., *What is Artificial Intelligence*. 2007.
11. McCarthy, J., et al., *A proposal for the Dartmouth summer research project on Artificial Intelligence*. 1955.
12. McCarthy, J. and P.J. Hayes, *Some Philosophical Problems from the Standpoint of Artificial Intelligence*. 1969.
13. Honda, *ASIMO*. 2008, Honda Motors co.
14. Georgeff, M.P. and A.L. Lansky, *Reactive Reasoning and Planning*. 1987.
15. Bratman, M.E., D.J. Israel, and M.E. Pollack, *Plans and resource-bounded practical reasoning*. 1988.
16. Goldberg, L.R., *The structure of phenotypic personality traits*. 1993.
17. André, E., et al., *Exploiting Models of Personality and Emotions to Control the Behavior of Animated Interactive Agents*. 2000.
18. Bourg, D.M. and G. Seemann, *AI for Game Developers*. 2004: O'Reilly Media.
19. Woodcook, S., *Game AI: The State of the Industry*. 2000.
20. Laird, J.E. and M.v. Lent, *Human-Level AI's Killer Application: Interactive Computer Games*. 2001.
21. Charles, D., *Enhancing Gameplay: Challenges for Artificial Intelligence in Digital Games*. 2003.
22. Nareyek, A., *AI in Computer Games*. 2004.
23. Fairclough, C., et al., *Research Directions for AI in Computer Games*. 2001.
24. Rizzo, P., et al., *Personality-Driven Social Behaviors in Believable Agents*. 1997.
25. Nass, C., et al. *Can Computer Personalities Be Human Personalities?* 1995.
26. Namee, B.M. and P. Cunningham. *Enhancing Non Player Characters in Computer Games using Pshychological Models*. 2003 [cited; Available from: http://www.ercim.org/publication/Ercim_News/enw53/cunningham.html.
27. Microsoft, *Age of Empires III*. 2005.
28. inc, E.A. *The Sims 3*. 2008 [cited; Available from: http://thesims3.ea.com/.
29. Labs, L. *Second Life*. 2008 [cited; Available from: http://secondlife.com/.
30. Wikipedia, *Allan B. Calhamer*. 2006.
31. Calhamer, A., *The Invention of Diplomacy*. 1974.
32. Calhamer, A.B., *The Rules of Diplomacy*. 2000.
33. Wikipedia, *Shakey the Robot*. 2007, Wikipedia.org.

34. Nilsson, N.J., *Artificial Intelligence: A New Synthesis*. 1 ed. 1998, San Francisco, California: Morgan Kaufman. 513.
35. Brooks, R.A., *A Robust Layered Control System For A Mobile Robot.* 1986.
36. Zamora, A., *Scientific Psychic*. 2004.
37. Ryckman, R.M., *Theories of Personality (with InfoTrac)*. 8 ed. 2003: Wadsworth Publishing. 720.
38. Foundation, E., *Eclipse*. 2007.
39. Apache, *Apache Derby database*. 2007.
40. Apache, *Apache DB.* 2007.
41. Kowal, G., *Launch4j Executable Wrapper*. 2006.
42. RARLAB, *WinRar*. 2007.
43. Apache, *Apache HTTP Server 2.2.3*. 2007.
44. DynDNS.com. *DynDNS*. 2007 [cited; Available from: http://www.dyndns.com/.

Appendix A

This appendix contains excerpts from the game data files map_data.xml and map_graphics.xml and the entire file new_game.xml.

1. map_data.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<map version="1.0">
 <province centre="yes" id="smy" name="smyrna" country="neutral" home="turkey"
type="coast">
         <force type="default">
           <path province="syr" type="both"/>
           <path province="arm" type="army"/>
           <path province="ank" type="army"/>
           <path province="con" type="both"/>
           <path province="aeg" type="fleet"/>
           <path province="eas" type="fleet"/>
         </force>
 </province>
 <province centre="yes" id="ank" name="ankara" country="neutral" home="turkey"
type="coast">
         <force type="default">
           <path province="arm" type="both"/>
           <path province="bla" type="fleet"/>
           <path province="con" type="both"/>
           <path province="smy" type="army"/>
         </force>
 </province>
 <province centre="yes" id="con" name="constantinople" country="neutral"
home="turkey" type="coast">
         <force type="default">
           <path province="smy" type="both"/>
           <path province="ank" type="both"/>
           <path province="bla" type="fleet"/>
           <path province="bul" type="army"/>
           <path province="bul" type="fleet" coast="ec"/>
           <path province="bul" type="fleet" coast="sc"/>
           <path province="aeg" type="fleet"/>
           <path province="bul" type="both"/>
         </force>
 </province>
 <province centre="no" id="syr" name="syria" country="neutral" type="coast">
         <force type="default">
           <path province="smy" type="both"/>
```

```
             <path province="arm" type="army"/>
             <path province="eas" type="fleet"/>
           </force>
 </province>
 <province centre="no" id="arm" name="armenia" country="neutral" type="coast">
           <force type="default">
             <path province="sev" type="both"/>
             <path province="bla" type="fleet"/>
             <path province="ank" type="both"/>
             <path province="smy" type="army"/>
             <path province="syr" type="army"/>
           </force>
 </province>
.
.
.
</map>
```

2. map_graphics.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<map version="1.0">
 <province id="smy">
          <area type="land">
            <line x1="789" y1="794" x2="799" y2="788" province="syr"/>
            <line x1="799" y1="788" x2="805" y2="788" province="syr"/>
            <line x1="805" y1="788" x2="813" y2="780" province="syr"/>
            <line x1="813" y1="780" x2="817" y2="774" province="syr"/>
            <line x1="817" y1="774" x2="811" y2="766" province="arm"/>
            <line x1="811" y1="766" x2="809" y2="760" province="arm"/>
            <line x1="809" y1="760" x2="807" y2="754" province="arm"/>
            <line x1="807" y1="754" x2="799" y2="746" province="arm"/>
            <line x1="799" y1="746" x2="791" y2="740" province="arm"/>
            <line x1="791" y1="740" x2="783" y2="738" province="ank"/>
            <line x1="783" y1="738" x2="773" y2="736" province="ank"/>
            <line x1="773" y1="736" x2="767" y2="730" province="ank"/>
            <line x1="767" y1="730" x2="763" y2="726" province="ank"/>
            <line x1="763" y1="726" x2="757" y2="724" province="ank"/>
            <line x1="757" y1="724" x2="751" y2="730" province="ank"/>
            <line x1="751" y1="730" x2="747" y2="738" province="ank"/>
            <line x1="747" y1="738" x2="741" y2="742" province="ank"/>
            <line x1="741" y1="742" x2="737" y2="746" province="ank"/>
            <line x1="737" y1="746" x2="729" y2="752" province="ank"/>
            <line x1="729" y1="752" x2="721" y2="756" province="ank"/>
            <line x1="721" y1="756" x2="715" y2="758" province="ank"/>
            <line x1="715" y1="758" x2="705" y2="762" province="con"/>
            <line x1="705" y1="762" x2="697" y2="764" province="con"/>
            <line x1="697" y1="764" x2="691" y2="764" province="con"/>
            <line x1="691" y1="764" x2="685" y2="762" province="con"/>
            <line x1="685" y1="762" x2="681" y2="762" province="con"/>
            <line x1="681" y1="762" x2="677" y2="766" province="con"/>
            <line x1="677" y1="766" x2="669" y2="766" province="con"/>
            <line x1="669" y1="766" x2="663" y2="766" province="con"/>
            <line x1="663" y1="766" x2="657" y2="766" province="con"/>
            <line x1="657" y1="766" x2="649" y2="766" province="con"/>
            <line x1="649" y1="766" x2="639" y2="762" province="con"/>
            <line x1="639" y1="762" x2="631" y2="762" province="con"/>
            <line x1="631" y1="762" x2="627" y2="762" province="con"/>
            <line x1="627" y1="762" x2="621" y2="774" province="aeg"/>
            <line x1="621" y1="774" x2="615" y2="776" province="aeg"/>
            <line x1="615" y1="776" x2="615" y2="782" province="aeg"/>
            <line x1="615" y1="782" x2="621" y2="786" province="aeg"/>
            <line x1="621" y1="786" x2="625" y2="780" province="aeg"/>
            <line x1="625" y1="780" x2="627" y2="786" province="aeg"/>
            <line x1="627" y1="786" x2="627" y2="792" province=""/>
            <line x1="627" y1="792" x2="629" y2="798" province=""/>
            <line x1="629" y1="798" x2="635" y2="806" province="aeg"/>
            <line x1="635" y1="806" x2="647" y2="806" province="aeg"/>
            <line x1="647" y1="806" x2="645" y2="810" province="aeg"/>
            <line x1="645" y1="810" x2="649" y2="814" province=""/>
            <line x1="649" y1="814" x2="657" y2="812" province=""/>
            <line x1="657" y1="812" x2="663" y2="814" province="eas"/>
            <line x1="663" y1="814" x2="671" y2="820" province="eas"/>
            <line x1="671" y1="820" x2="679" y2="822" province="eas"/>
            <line x1="679" y1="822" x2="689" y2="816" province="eas"/>
            <line x1="689" y1="816" x2="693" y2="806" province="eas"/>
            <line x1="693" y1="806" x2="705" y2="808" province="eas"/>
            <line x1="705" y1="808" x2="721" y2="812" province="eas"/>
            <line x1="721" y1="812" x2="735" y2="816" province="eas"/>
            <line x1="735" y1="816" x2="751" y2="816" province="eas"/>
            <line x1="751" y1="816" x2="759" y2="806" province="eas"/>
            <line x1="759" y1="806" x2="765" y2="800" province="eas"/>
            <line x1="765" y1="800" x2="775" y2="800" province="eas"/>
            <line x1="775" y1="800" x2="783" y2="798" province="eas"/>
            <line x1="783" y1="798" x2="789" y2="794" province="eas"/>
        </area>
        <centre x="641" y="795"/>
        <force type="default" x="678" y="788"/>
        <name x="708" y="786" label="name" angle="0"/>
    </province>
    <province id="ank">
        <area type="land">
            <line x1="791" y1="740" x2="801" y2="728" province="arm"/>
            <line x1="801" y1="728" x2="797" y2="714" province="arm"/>
            <line x1="797" y1="714" x2="797" y2="706" province="arm"/>
            <line x1="797" y1="706" x2="803" y2="702" province="arm"/>
            <line x1="803" y1="702" x2="797" y2="690" province="arm"/>
            <line x1="797" y1="690" x2="789" y2="688" province="bla"/>
            <line x1="789" y1="688" x2="781" y2="688" province="bla"/>
            <line x1="781" y1="688" x2="775" y2="682" province="bla"/>
            <line x1="775" y1="682" x2="763" y2="682" province="bla"/>
            <line x1="763" y1="682" x2="747" y2="682" province="bla"/>
            <line x1="747" y1="682" x2="733" y2="682" province="bla"/>
            <line x1="733" y1="682" x2="719" y2="686" province="bla"/>
            <line x1="719" y1="686" x2="695" y2="698" province="bla"/>
            <line x1="695" y1="698" x2="701" y2="710" province="con"/>
            <line x1="701" y1="710" x2="699" y2="714" province="con"/>
            <line x1="699" y1="714" x2="699" y2="720" province="con"/>
            <line x1="699" y1="720" x2="705" y2="728" province="con"/>
            <line x1="705" y1="728" x2="701" y2="732" province="con"/>
            <line x1="701" y1="732" x2="709" y2="740" province="con"/>
            <line x1="709" y1="740" x2="715" y2="746" province="con"/>
            <line x1="715" y1="746" x2="715" y2="758" province="con"/>
            <line x1="715" y1="758" x2="721" y2="756" province="smy"/>
            <line x1="721" y1="756" x2="729" y2="752" province="smy"/>
            <line x1="729" y1="752" x2="737" y2="746" province="smy"/>
            <line x1="737" y1="746" x2="741" y2="742" province="smy"/>
            <line x1="741" y1="742" x2="747" y2="738" province="smy"/>
            <line x1="747" y1="738" x2="751" y2="730" province="smy"/>
            <line x1="751" y1="730" x2="757" y2="724" province="smy"/>
            <line x1="757" y1="724" x2="763" y2="726" province="smy"/>
            <line x1="763" y1="726" x2="767" y2="730" province="smy"/>
            <line x1="767" y1="730" x2="773" y2="736" province="smy"/>
            <line x1="773" y1="736" x2="783" y2="738" province="smy"/>
            <line x1="783" y1="738" x2="791" y2="740" province="smy"/>
          </area>
          <centre x="725" y="733"/>
          <force type="default" x="757" y="683"/>
          <name x="721" y="709" label="name" angle="0"/>
 </province>
</map>
```

3. new_game.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<state season="0" year="1901">
	<country name="england">
		<province id="edi"/>
		<province id="cly"/>
		<province id="lvp"/>
		<province id="yor"/>
		<province id="wal"/>
		<province id="lon"/>
		<army id="lvp" moves="lvp"/>
		<fleet id="edi" moves="edi"/>
		<fleet id="lon" moves="lon"/>
	</country>
	<country name="france">
		<province id="bre"/>
		<province id="pic"/>
		<province id="gas"/>
		<province id="par"/>
		<province id="bur"/>
		<province id="mar"/>
		<army id="mar" moves="mar"/>
		<army id="par" moves="par"/>
		<fleet id="bre" moves="bre"/>
	</country>
	<country name="germany">
		<province id="kie"/>
		<province id="ber"/>
		<province id="pru"/>
		<province id="sil"/>
		<province id="mun"/>
		<province id="ruh"/>
		<army id="ber" moves="ber"/>
		<army id="mun" moves="mun"/>
		<fleet id="kie" moves="kie"/>
	</country>
	<country name="italy">
		<province id="pie"/>
		<province id="ven"/>
		<province id="tus"/>
		<province id="rom"/>
		<province id="apu"/>
		<province id="nap"/>
		<army id="ven" moves="ven"/>
		<army id="rom" moves="rom"/>
```

```
        <fleet id="nap" moves="nap"/>
    </country>
    <country name="austria">
        <province id="boh"/>
        <province id="tyr"/>
        <province id="gal"/>
        <province id="vie"/>
        <province id="bud"/>
        <province id="tri"/>
        <army id="vie" moves="vie"/>
        <army id="bud" moves="bud"/>
        <fleet id="tri" moves="tri"/>
    </country>
    <country name="turkey">
        <province id="con"/>
        <province id="ank"/>
        <province id="arm"/>
        <province id="syr"/>
        <province id="smy"/>
        <army id="con" moves="con"/>
        <army id="smy" moves="smy"/>
        <fleet id="ank" moves="ank"/>
    </country>
    <country name="russia">
        <province id="stp"/>
        <province id="war"/>
        <province id="lvn"/>
        <province id="mos"/>
        <province id="sev"/>
        <province id="ukr"/>
        <army id="war" moves="war"/>
        <army id="mos" moves="mos"/>
        <fleet id="sev" moves="sev"/>
        <fleet id="stp(sc)" moves="stp(sc)"/>
    </country>
</state>
```

Appendix B – The README

Readme.doc

Table of contents:

1. Introduction

In our master's thesis, we have modified a software version of the game Diplomacy. This version was made by Aleksander Krzywinski and Arne S. Helgesen for their master's thesis. We have modified it to suit our needs. The original game can be found at http://sf.net/projects/StateCraft. We would like to thank Aleksander and Arne for providing us with this game and Aleksander for all the help he has given us in the previous months.

We hope that however reads this document would like to try the game, play it for ten minutes, ten hours or ten days and answer a few questions afterwards. The questions will be available in a document at our website, http://www.statecraft.ath.cx in a few days. Thank you very much for your help!

If you have any questions, comments or anything else, do not hesitate to send us an email on arild.jensen@student.uib.no or h.nes@student.uib.no.

2. Installation

Unrar the archive and run StateCraft.exe. JRE 1.6.0 is needed. If not detected by the game, Sun download page will open. When you run StateCraft.exe, two windows will open; the game server window and the user login window.

- *The game server window* shows output from server and contains reset and quit buttons. Reset button is for resetting the game state to its original state. After pressing the reset button, the game needs to be restarted.

- *The user login window* is where you choose a game to play – for now, there will be only one. When connected to server, the window will show a list of the one game, with a Play button. When this button is pressed, the window will disappear and the *user game interface* will open. This may take a while. When the user game interface appears: start playing!

3. Gameplay / Rules

Diplomacy is a social and strategic board game created in 1954. The complete rule set can be found at http://www.diplomacy-archive.com/resources/rulebooks/2000AH4th.pdf or somewhere else on the net. We will describe in short what you need to play StateCraft.

The game consists of different types of regions, two types of units and seven battling nations.

- Nations / Region owners:

- England (Blue)
- France (Turquoise)
- Germany (Dark Grey)
- Austria (Red)
- Italy (Green)
- Russia (Grey)
- Turkey (Yellow)
- Neutral (Light Green)
- Sea (No owner, Light Blue)
- Unavailable in game (Black)

- Region types:

- Sea
- Coastline
- Inland

- Unit types:

- Army
- Fleet

Half of the land regions contain supply centres, a circle with a star. Each country starts with three supply centres and three units, except from Russia, who starts with four of each. A nation can only supply one unit (army or fleet) for each supply centre it controls and only original supply centres can build units (e.g. England can only build units in the supply centres in Edinburgh, Liverpool and London). To control a supply centre (or a region without), a nation must hold it through a winter round. To win the game, one nation must hold 18 of these centres through a winter round.

The game starts in 1901 and each year consists of five rounds, Spring, Summer, Autumn, Winter and Build. Each nation places its unit action orders in the spring and autumn rounds, and its build and disband unit orders in the build round. The summer and winter rounds show the results of last rounds orders, that is, which are valid orders and which are not.

In this version of the game, you can only play Russia and only against six computer opponents. You start with four supply centres, two fleets and two armies, in spring 1901, a move round. To move a unit, press the left mouse button on the unit to open orders menu. Orders given are showed on the map with arrows for moves, a broken white line for support, a wavy white line for convoys or a white "fortress" for hold orders. *Be aware of the fact that the validity of the orders given is first controlled after the round, which means that only valid orders will be registered and invalid orders cannot be undone.* All orders given will show in the orders window in the upper left corner. An order for a unit will cancel all previous orders for the same unit. When you have issued orders for all your units, press the Post orders button. If you issue new orders after pressing the Post orders button, press it again. That will cancel the previous orders posted.

As mentioned above, only valid orders will be effectuated by the game adjudicator. *The game interface does not validate orders given in any way.* In fact, you can issue orders for other nations' units, although these will not be registered in the orders window. You can also order a move from one corner of the map to another and you can order support of non-existing moves or convoy of non-existing armies. In the following, you will be given a brief explanation of which orders are valid orders.

- Fleet: A fleet can move from one sea region to another, between a sea region and a coastline region or between to adjacent coastline regions. It can not cross a sea region directly (as in move directly from an English province to Norway). A fleet can also convoy armies across the sea region in which it in a round resides (as in convoy an army from an English province to Norway).

- Army: An army can move between all adjacent land provinces. If convoyed by a fleet, it can move across sea regions. It is possible to be convoyed over several sea regions in one turn, but you will need one fleet for each sea region.

- Both: Both unit types can hold, that is, not move. This order will be automatically given if no other order is present. Both units can support both holds and moves.

When you have posted your orders, press the Next round button. The game server will now put all orders into effect simultaneously, according to the following rules:

- Only one unit can occupy a region at a given time.

- All units have the same strength. If two (or more) units have orders to move into the same region, both orders are cancelled by the adjudicator.

- A defending force will always fend off an attack of the same strength.

- To increase the strength of a defending unit (hold order) or an attack (move), the unit must get support from other units. A unit can only support a hold in or an attack into an adjacent region.

After resolving the rules, the game moves to Summer and all the ordered actions for all nations are showed on the map, valid moves with green arrows, invalid with grey. Beaten units are shown with a disband symbol, which means that the unit will be disbanded. When pressing Next Round, the valid orders are effectuated and the game moves to Autumn, where you again can place orders for your units. Winter comes after Autumn and has the same role as Summer. The last round in each year is the Build round. In this round you can build or disband units. A nation can only support one unit for each supply centre, but may have fewer. Units can only be built in native supply centres, that is, centres that a nation controls when the game starts. Units are built by pressing the left mouse button on a supply centre and selecting the appropriate unit to build. If you lose one or more supply centres, you may find yourself with more units than supply centres. You will then have to disband units to even out the number between units and supply centres. If you do not choose units to disband, the computer will do it for you. Disband a unit by pressing your left mouse button on the unit and selecting the appropriate menu option.

4. Diplomacy

The diplomacy part of the game is what gave it its name and is an important aspect of the game. In this version, however, the diplomacy part is reduced to one option: asking for support. The only way to win a battle is to have greater strength. Thus you will often find yourself in need of support from another nation to conquer a region – or defend one. This can be achieved by requesting support. Press the drag-down menu under a nation’s flag, choose Request support, perform the support move you want the computer do order and press the Send request button. When a nation answers (or when the computer sends a request to you), the circle around that nation’s flag will turn bright green. Pressing the flag will show you the message. Remember that no agreements are binding, you can accept or refuse whatever you want, and there is no direct relationship between the diplomatic part and the board part of the game, which means that even if another nation accepts your request, you have no guarantee that it will honour the agreement.

5. Hints, Tricks and HOWTOs

- When selecting support for a move, select support from the unit’s menu, and press on the region from which the move originates, and then press the destination of the move. Do not press on the unit getting the support, this will open that unit’s menu and just confuse everyone.

- When supporting a hold, select from the unit’s menu, and then press once in the holding unit’s region.

- When convoying, select the source and target region, not the unit being convoyed.

- Some regions (Spain, Bulgaria and St. Petersburg) have two coastlines, NC and SC. A fleet cannot move directly between these coastlines or to the sea region outside the coastline it is on.

- When building a fleet in a region with multiple coastlines, you can choose which coastline to place in on.

- Remember that the computer can lie, cheat, fake and backstab – but so can you!

6. Troubleshooting

If (trouble) {game.restart();}

7. FAQ

7.1 About the game

I found a bug, what do I do with it?

Try not to let it bug you.

When I press Play, the window disappears and nothing happens. Why?

Wait. Something will happen. Soon.

Can I play another country?

Yes, but not in this version we have bundled for you. To play other countries and/or multiplayer, you have to change the settings in the properties-file and start each jar separately and with appropriate arguments. Ask us if you want to, but after the testing period☺ You can also checkout the original code, without our modifications, from the repository at http://sf.net/projects/StateCraft. I believe it is not a stable build.

Can I play multiplayer?

See the above answer.

I made a mistake and would like to load the game. How do I do that?

You cannot. The game saves itself automatically after each round, thus meaning that you cannot quit the game to undo a round.

How do I know if I can build units and how many or if I have to disband some?

If you cannot build units, the build units menu will not appear. If you can build one, you can order the build of ten units, but of course, you will get only one. No info is provided if you need to disband a unit. Some counting may be needed.

What is that circle to the left of the map for?

That is a timer for multiplayer games. Do not mind the time indicator, but please enjoy the pictures.

Why do not all windows close when I press quit?

That will be a bug. Close separately. You may need to close a java.exe process in Task Manager (for Windows). Sorry about that.

7.2 Other questions

Who *are* you?

We are two master students doing our master's thesis about artificial intelligence in computer games.

Did you two create this software version of the game?

No. Alexander Krzywinski and Arne S. Helgesen created this game for their master's thesis handed in February 2006. The original game is available at http://sf.net/projects/StateCraft.

Did you two do anything besides distributing the game and writing this document?

Yes. We have implemented a module that affects the agents' gameplay and modified the game so that in addition to run as a server-client application, it can run as a stand-alone application.

What questions will the feedback document contain?

Wait and see, we do not want to make you biased when playing the game.

I just want to play this extremely fun game, not answer your boring questions! May I?

Of course, but please do not. We need your help.

Appendix C – The UserInterface class

```
package client.applet;

<imports>

public class UserInterface extends JFrame implements SystemMessageListener, Client
{
  private static final long serialVersionUID = 7270614693079877666L;

  private LoginPanel loginPanel;
  private GameBrowserPanel serverPanel;
  private GameFrame gameFrame;
  private DiplomacyClient client;
  public static OutputPanel outputPanel;

  public UserInterface()
  {
    client = new DiplomacyClient();

    loginPanel = new LoginPanel(this);
    loginPanel.setVisible(true);

    outputPanel = new OutputPanel(20, 40);
    outputPanel.setVisible(true);
    outputPanel.setEditable(false);

    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(loginPanel, BorderLayout.NORTH);
    getContentPane().add(outputPanel, BorderLayout.CENTER);
    addWindowListener(new DiplomacyWindowListener());

    setSize(900, 900);
    setVisible(true);
    setDefaultCloseOperation(UserInterface.DO_NOTHING_ON_CLOSE);
  }

  public UserInterface(String nick, String password)
  {
    client = new DiplomacyClient();

    loginPanel = new LoginPanel(this);
    loginPanel.setVisible(false);

    outputPanel = new OutputPanel(20, 40);
```

```
        outputPanel.setVisible(true);
        outputPanel.setEditable(false);

        getContentPane().setLayout(new BorderLayout());
        getContentPane().add(loginPanel, BorderLayout.NORTH);
        getContentPane().add(outputPanel, BorderLayout.CENTER);
        addWindowListener(new DiplomacyWindowListener());

        setSize(900, 900);
        setVisible(true);
        setDefaultCloseOperation(UserInterface.DO_NOTHING_ON_CLOSE);
        connect(nick, password);
    }

    public void connect(String nick, String password)
    {
        client.setSystemMessageListener(this);
        outputPanel.println("Connecting...");
        client.connect(nick, password);
    }

    public void connected(ConnectedMessage message)
    {
        if (message.isLoggedIn()){
            setPlayerID(message.getReceiver());
            getContentPane().add(outputPanel, BorderLayout.SOUTH);
            getContentPane().remove(loginPanel);
            serverPanel = new GameBrowserPanel(this);
            getGameList();
            getMap();
            getContentPane().add(serverPanel);
            validate();
            outputPanel.println("Connected!");
        }
        else {
            JOptionPane.showMessageDialog(this, "Username/password incorrect or User
already online", "Connection failed", JOptionPane.ERROR_MESSAGE);
        }
    }

    public void startGame(StartPlayingMessage message) {
        outputPanel.println("Starting game...");
        gameFrame = new GameFrame(message.getCountry());
        client.setGameMessageListener(gameFrame);
        client.setDiplomaticMessageListener(gameFrame);
        gameFrame.requestInitalData();
        this.getContentPane().removeAll();
        this.validate();
        setVisible(false);
    }

    /**
     * This method updates the panel in the server window with a list of
     * games that are playing on the server.
     *
     * @param gameListMessage a GameListMessage with the games
     */
    public void updateGames(GameListMessage gameListMessage) {
        serverPanel.updateList(gameListMessage.getGames());
    }

    /**
     * Asks the server for a list of games currently playing
     */
    public void getGameList() {
        client.getGameList();
    }

    public void getMap() {
        client.getMap();
    }

    /**
     * Tells the server that this player wants to join the specific game
     * @param game_id
     */
    public void join(int game_id) {
        client.join(game_id);
    }

    /**
     * @return
     */
    public int getPlayerID() {
        return client.getPlayerID();
    }

    public void setPlayerID(int playerID) {
        client.setPlayerID(playerID);
    }

    /**
```

```
 * Tells the server that this player wants to play the specified game
 *
 * @param game_id
 */
public void play(int game_id) {
  client.play(game_id);
}

/* (non-Javadoc)
 * @see shared.network.listeners.SystemMessageListener#receive(
    shared.network.messages.SystemMessage)
 */
public void receive(SystemMessage message) {
  if (message instanceof ConnectedMessage){
    connected((ConnectedMessage) message);
  }
  if (message instanceof GameListMessage) {
    GameListMessage gameListMessage = (GameListMessage) message;
    if(gameListMessage.getGames().size() == 0)
    {
      UserInterface.outputPanel.println(
      "Game list contains no games. Requesting new list...");
      getGameList();
    }
    else
    {
      updateGames((GameListMessage) message);
    }
  }
  if (message instanceof StartPlayingMessage) {
    startGame((StartPlayingMessage) message);
  }
  if (message instanceof MapMessage) {
    Map.setInstance(((MapMessage)message).getMap());
  }
  if (message instanceof QuitGameMessage){
    System.exit(0);
  }
}

public void sendQuitMessage()
{
  ClientPostOffice.getInstance().sendMessage(
            MessageFactory.createQuitGameMessage());
}
```

```
  public static void main(String[] args) {
    if(args.length == 0)
    {
      UserInterface ui = new UserInterface();
    }
    else
    {
      try
      {
        String nick = args[0];
        String password = args[1];
        UserInterface ui = new UserInterface(nick, password);
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
  }
}
```

Appendix D – The Personality class

```
package client.ai.strategic.personality;

<imports>


public class Personality extends Module implements Inhibitor, Suppressor
{
  public static boolean tmpEmotionalTemperament = false;
  public static boolean tmpRiskAttitudes = false;
  public static boolean tmpAggressiveness = false;
  public static boolean tmpContemptForRules = false;

  private static final int ACCEPT_MESSAGE_PERCENTAGE = 75;

  private Strategic strategic;
  private GameState currentGameState;
  private IndividualAttributes individualAttributes;
  private SocialAttributes socialAttributes;
  private TacticList tacticList;
  private Hashtable relations = new Hashtable();

  public Personality(Strategic strategic)
  {
    this.strategic = strategic;
    individualAttributes = new IndividualAttributes(
      tmpEmotionalTemperament, tmpRiskAttitudes);
    socialAttributes = new SocialAttributes(tmpAggressiveness, tmpContemptForRules);
  }

  public void setIndividualAttributes(
      boolean emotionalTemperament, boolean riskAttitudes)
  {
    individualAttributes.setEmotionalTemperament(emotionalTemperament);
    individualAttributes.setRiskAttitudes(riskAttitudes);
  }

  public void setSocialAttributes(boolean agressiveness, boolean contemptForRules)
  {
    socialAttributes.setAgressiveness(agressiveness);
    socialAttributes.setContemptForRules(contemptForRules);
  }

  @Override
```

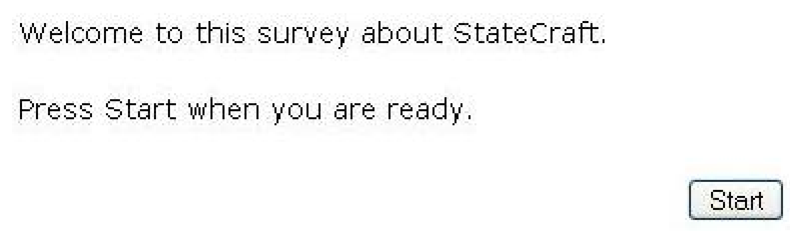
```
public void receive(Object object)
{
  if(object instanceof GameState)
  {
    currentGameState = (GameState) object;
  }
}

public Object suppress(Object object)
{
  if(currentGameState != null)
  {
    if(object instanceof TacticList)
    {
      tacticList = (TacticList) object;
      considerAggressiveness();
      considerRiskAttitudes();
      considerEmotionalTemperament();
      return tacticList;
    }
      return object;
  }
  return object;
}

public Object inhibit(Object object)
{
  if(object instanceof Tactic)
  {
    return object;
  }
  if(object instanceof java.util.Hashtable)
  {
    relations = (Hashtable) object;
  }
  if(object instanceof RequestAcceptanceMessage)
  {
    return considerContemptForRules((RequestAcceptanceMessage) object);
  }

  return null;
}

/**
 * This method checks whether the agent has EmotionalTemperament true or
 * false and modifies its actions accordingly.
 *
 * If the agent has EmotionalTemperament = true, it disregards the factual value
 * of a tactic and only considers the potential value.
*
 * If the agent has RiskAttitudes = false, no changes to normal gameplay will occur.
 *
 * @param void
 * @return void
 */
private void considerEmotionalTemperament()
{
  for(int i = 0; i < tacticList.size(); i++)
  {
    if(individualAttributes.getEmotionalTemperament())
    {
      tacticList.tacticAt(i).changeFactualValue(
                tacticList.tacticAt(i).getPotentialValue());
    }
  }
}

/**
 * This method checks whether the agent has RiskAttitudes true or false
 * and modifies its actions accordingly.
 *
 * If the agent has RiskAttitudes = true, it will decrease the factual value of
 * all tactics containing hold operations with 10% for each operation.
 *
 * If the agent has RiskAttitudes = false, it will increase the factual value of
 * all tactics containing hold operations nations with 10%for each operation.
 *
 * It does this by browsing through all operations. If the operation is a hold
 * and RiskAttitudes = true, the factual value of the move is decreased by
 * 10%. If the operation is a hold and RiskAttitudes = false, the factual value
 * of the move is increased by 10%.
 *
 * @param void
 * @return void
 */
private void considerRiskAttitudes()
{
  Order order;

  for(int i = 0; i < tacticList.size(); i++)
  {
    for(int j = 0; j < tacticList.tacticAt(i).getNumUnits(); j++)
```

```
        {
          order = tacticList.tacticAt(i).getOperations().operationAt(j).getOrder();

          if(order instanceof HoldOrder)
          {
            if(individualAttributes.getRiskAttitudes())
            {
              int factualValue = tacticList.tacticAt(i).getFactualValue();
              int newFactualValue = 0;
              if(factualValue > 0)
              {
                newFactualValue = (int) (factualValue * 3);
              }
              else
              {
                newFactualValue = (int) (factualValue * 0.9);
              }
              tacticList.tacticAt(i).changeFactualValue(newFactualValue - factualValue);
            }
            else if(!individualAttributes.getRiskAttitudes())
            {
              int factualValue = tacticList.tacticAt(i).getFactualValue();
              int newFactualValue = 0;
              if(factualValue > 0)
              {
                newFactualValue = (int) (factualValue * 0.9);
              }
              else
              {
                newFactualValue = (int) (factualValue * 1.1);
              }
              tacticList.tacticAt(i).changeFactualValue(newFactualValue - factualValue);
            }
          }
        }
      }
    }

    /**
     * This method checks whether the agent has Aggressiveness true or false
     * and modifies its actions accordingly.
     *
     * If the agent has Aggressiveness = true, it will increase the factual value of
     * all tactics containing moves against enemy nations  with 10% for each move.
     * If the agent has Aggressiveness = false, it will decrease the factual value of
     * all tactics containing moves against friendly nations with 10% for each move.
```

```
 * It does this by browsing through all operations. If the operation is a move,
 * it checks the agent's relationship with the owner of the target province.
 *
 * @param void
 * @return void
 */
private void considerAggressiveness()
{
  Country opponent;
  Order order;

  for(int i = 0; i < tacticList.size(); i++)
  {
    for(int j = 0; j < tacticList.tacticAt(i).getNumUnits(); j++)
    {
      order = tacticList.tacticAt(i).getOperations().operationAt(j).getOrder();

      if(order instanceof MoveOrder)
      {
        //Get owner of target province
        opponent =
                      currentGameState.getProvinceState((
                      order.getTarget().getProvince())).getOwner();
        //Check if owner is self or null
        if(opponent != null && opponent != strategic.getCountry())
        {
          //If the agent has the attribute aggressiveness
          if(socialAttributes.getAgressiveness())
          {
            if(relations.get(opponent).toString().contains("War"))

            {
              int factualValue = tacticList.tacticAt(i).getFactualValue();
              int newFactualValue = 0;
              if(factualValue > 0)
              {
                newFactualValue = (int) (factualValue * 1.1);
              }
              else
              {
                newFactualValue = (int) (factualValue * 0.9);
              }
              tacticList.tacticAt(i).changeFactualValue(
                        newFactualValue - factualValue);
            }
          }
          else if(!socialAttributes.getAgressiveness())
          {
            if(relations.get(opponent).toString().contains("Cooperate"))
            {
              int factualValue = tacticList.tacticAt(i).getFactualValue();
              int newFactualValue = 0;
              if(factualValue > 0)
              {
                newFactualValue = (int) (factualValue * 0.9);
              }
              else
              {
                newFactualValue = (int) (factualValue * 1.1);
              }
              tacticList.tacticAt(i).changeFactualValue(
                      newFactualValue - factualValue);
            }
          }
        }
      }
    }
  }
}

/**
 * @author Onkel Arild
 *
 * This method checks whether contemptForRules is true.
 * If true, then it returns a new message, which has a
 * 75% chance of being accepted.
 *
 * If false, it returns param.
 *
 * @return RequestAcceptanceMessage
 * @param RequestAcceptanceMessage
 */
private Object considerContemptForRules(RequestAcceptanceMessage message)
{
  if(socialAttributes.getContemptForRules())
  {
    int random = (int) (Math.random() * 100);

    if(random > ACCEPT_MESSAGE_PERCENTAGE)
    {
      RequestAcceptanceMessage answer = new RequestAcceptanceMessage(
              message, false);
            answer.setReceiver(message.getReceiver());
            System.out.println("Message rejected.\n");
            return answer;
          }
          else
          {
            RequestAcceptanceMessage answer = new RequestAcceptanceMessage(
                    message, true);
            answer.setReceiver(message.getReceiver());
            System.out.println("Message accepted.\n");
            return answer;
          }
        }
        return message;
      }
    }
```

Appendix E – The Questionnaire

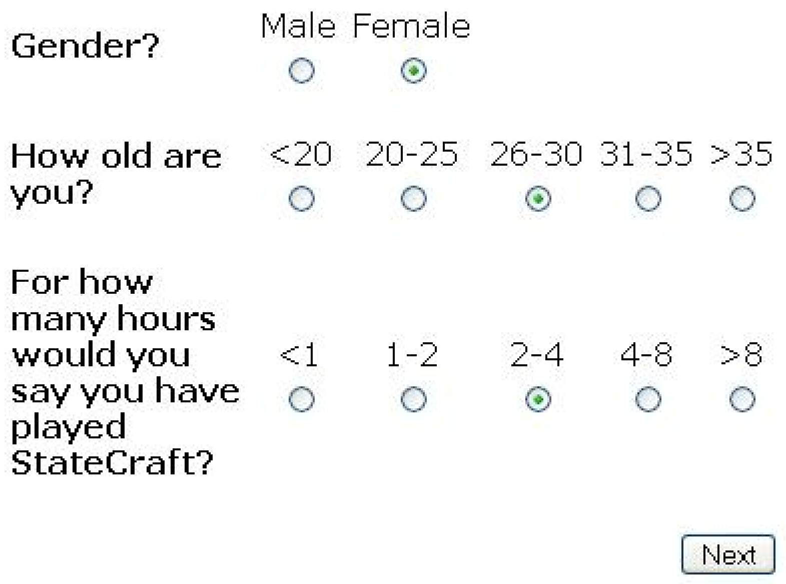
Welcome page



Welcome to this survey about StateCraft.

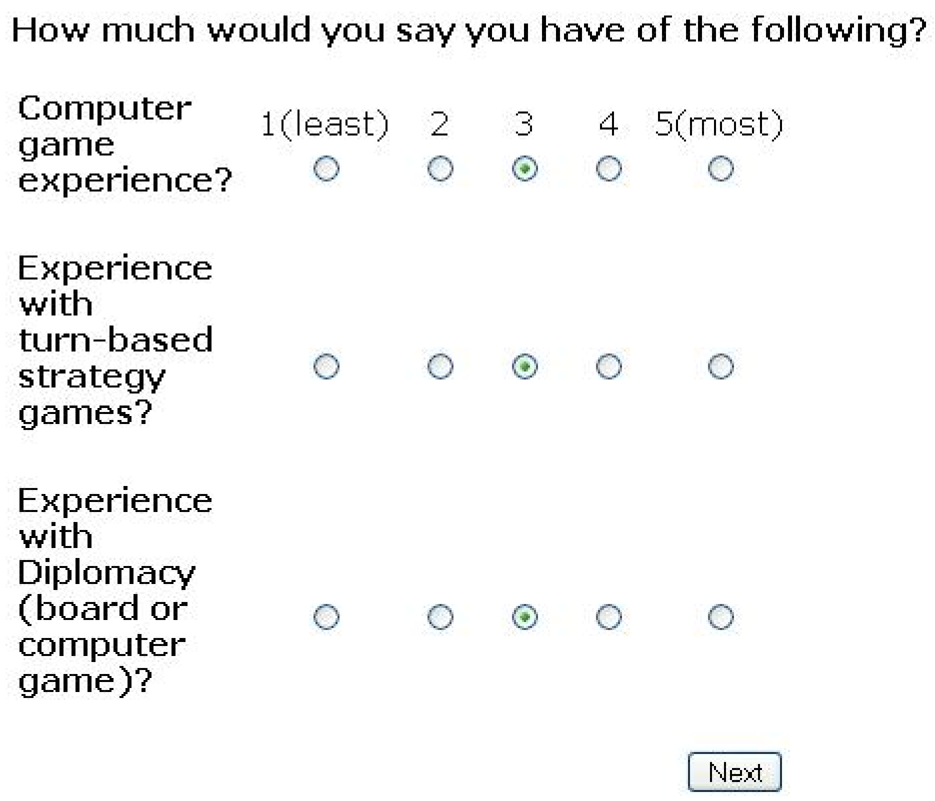
Press Start when you are ready.

Section one



Gender?	Male	[✓ Female]			
How old are you?	<20	20-25	[✓ 26-30]	31-35	>35
For how many hours would you say you have played StateCraft?	<1	1-2	[✓ 2-4]	4-8	>8

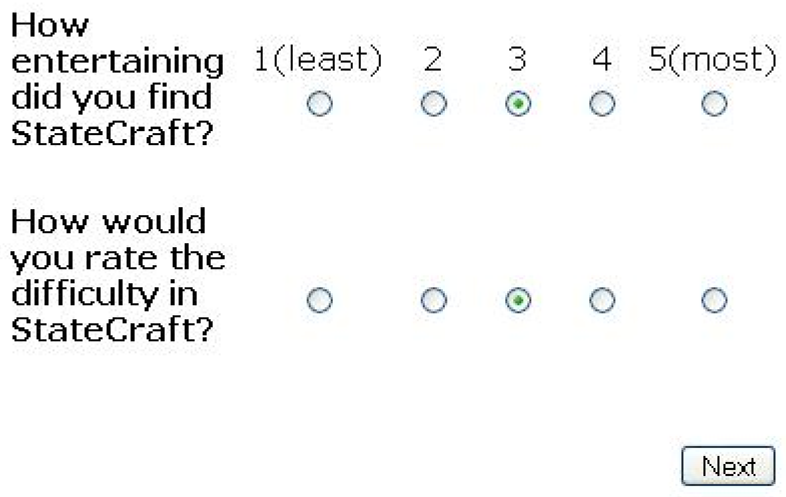
Section two



How much would you say you have of the following?

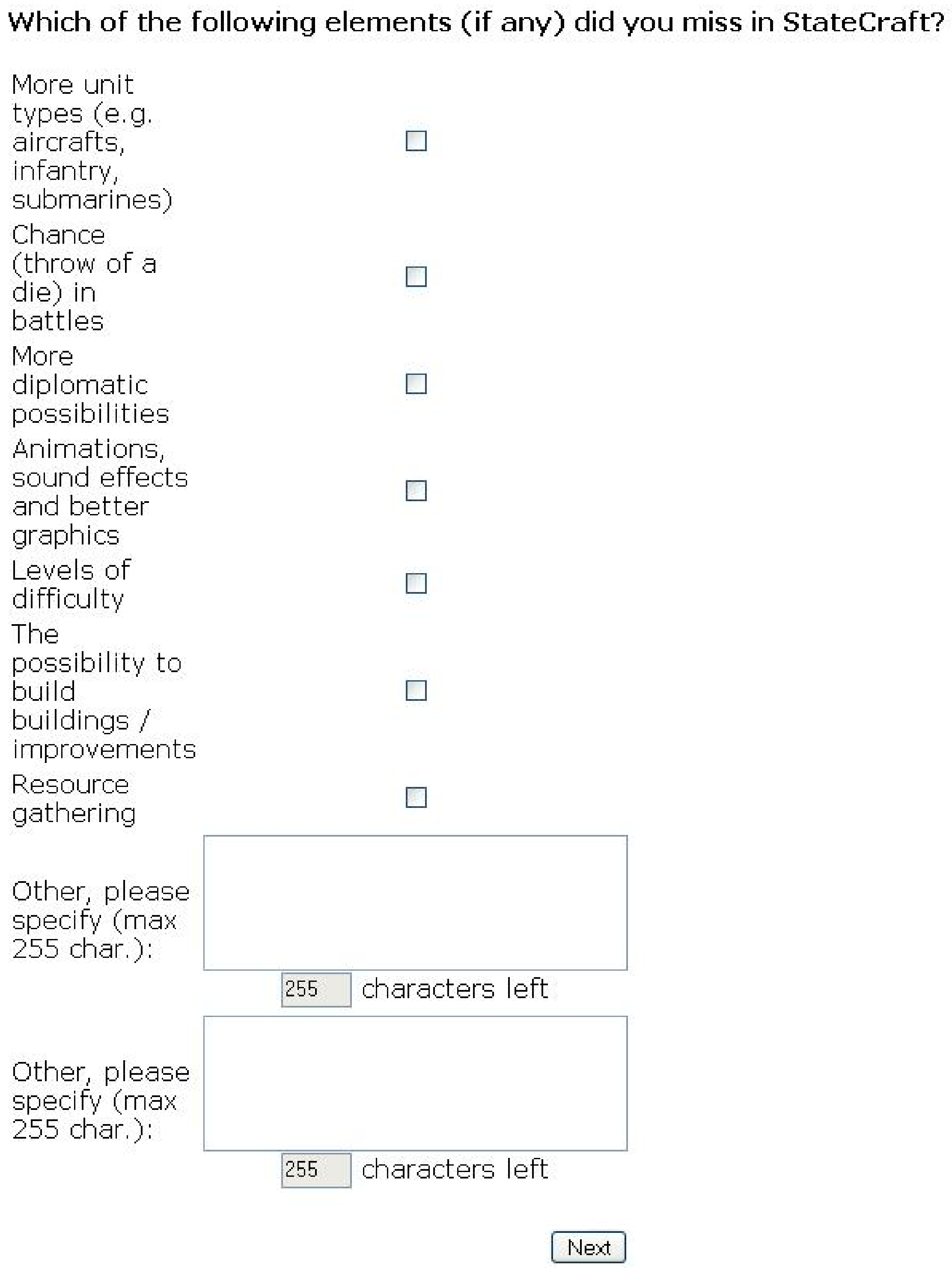
	1(least)	2	3	4	5(most)
Computer game experience?	○	○	[✓ ●]	○	○
Experience with turn-based strategy games?	○	○	[✓ ●]	○	○
Experience with Diplomacy (board or computer game)?	○	○	[✓ ●]	○	○

Section three



	1(least)	2	3	4	5(most)
How entertaining did you find StateCraft?	○	○	[✓ ●]	○	○
How would you rate the difficulty in StateCraft?	○	○	[✓ ●]	○	○

Section four

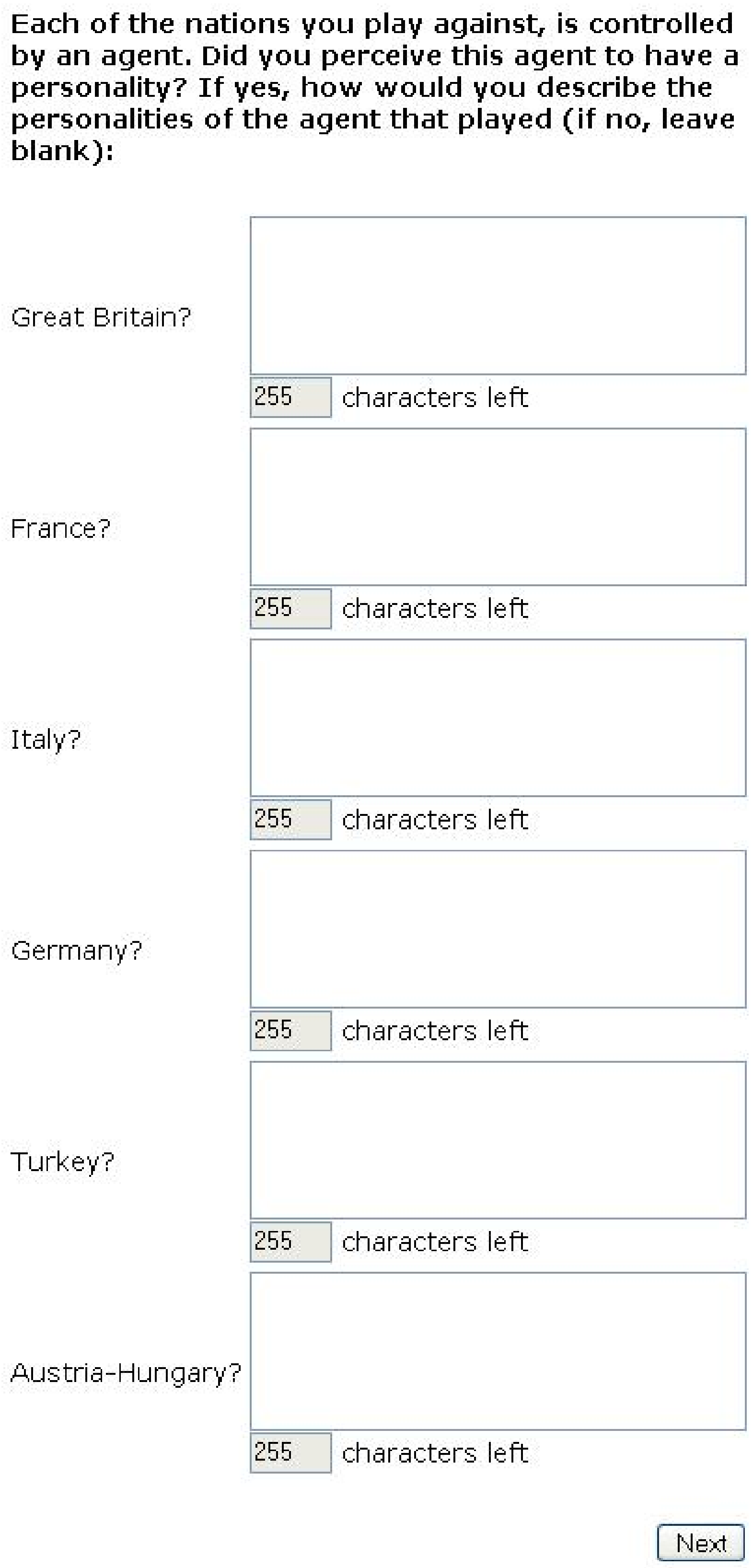


Which of the following elements (if any) did you miss in StateCraft?

More unit types (e.g. aircrafts, infantry, submarines)	☐
Chance (throw of a die) in battles	☐
More diplomatic possibilities	☐
Animations, sound effects and better graphics	☐
Levels of difficulty	☐
The possibility to build buildings / improvements	☐
Resource gathering	☐
Other, please specify (max 255 char.):	255 characters left
Other, please specify (max 255 char.):	255 characters left

Next

Section five



Each of the nations you play against, is controlled by an agent. Did you perceive this agent to have a personality? If yes, how would you describe the personalities of the agent that played (if no, leave blank):

Great Britain?

255 characters left

France?

255 characters left

Italy?

255 characters left

Germany?

255 characters left

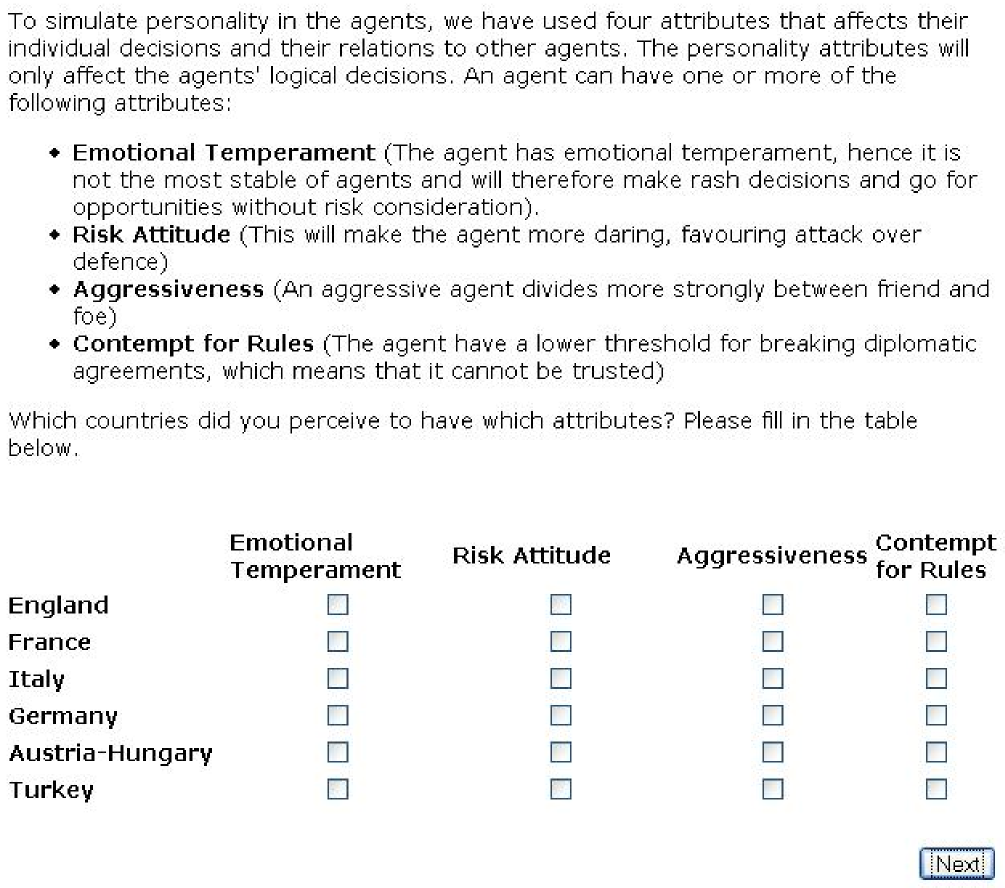
Turkey?

255 characters left

Austria-Hungary?

255 characters left

Section six



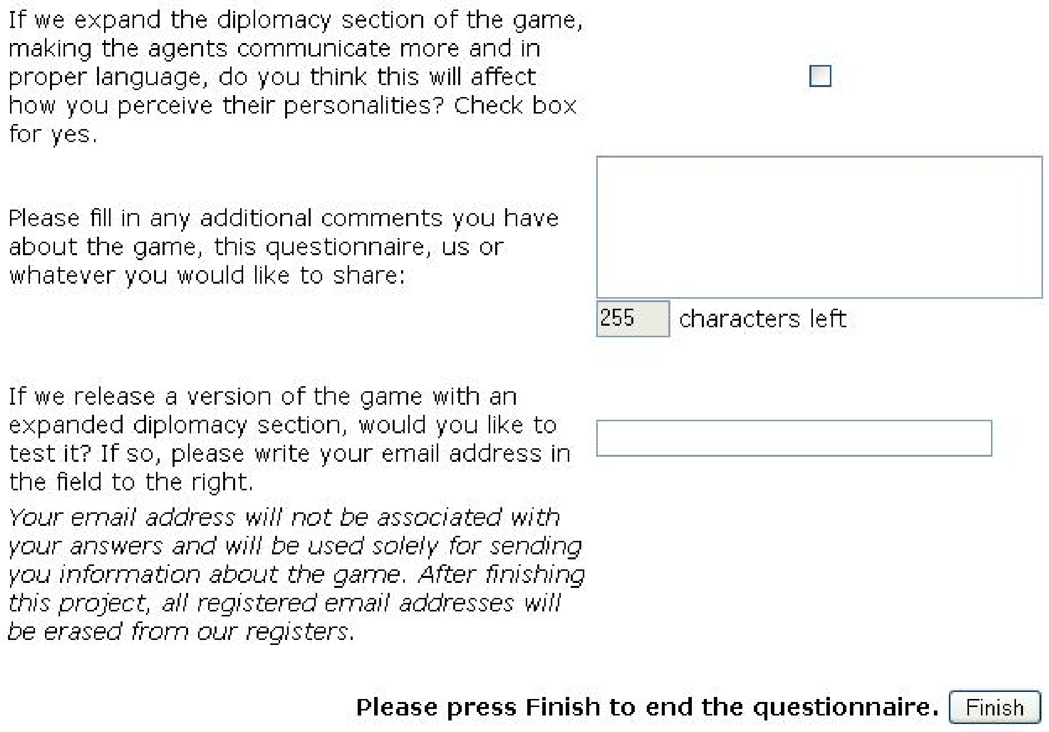
To simulate personality in the agents, we have used four attributes that affects their individual decisions and their relations to other agents. The personality attributes will only affect the agents' logical decisions. An agent can have one or more of the following attributes:

- **Emotional Temperament** (The agent has emotional temperament, hence it is not the most stable of agents and will therefore make rash decisions and go for opportunities without risk consideration).
- **Risk Attitude** (This will make the agent more daring, favouring attack over defence)
- **Aggressiveness** (An aggressive agent divides more strongly between friend and foe)
- **Contempt for Rules** (The agent have a lower threshold for breaking diplomatic agreements, which means that it cannot be trusted)

Which countries did you perceive to have which attributes? Please fill in the table below.

	Emotional Temperament	Risk Attitude	Aggressiveness	Contempt for Rules
England	☐	☐	☐	☐
France	☐	☐	☐	☐
Italy	☐	☐	☐	☐
Germany	☐	☐	☐	☐
Austria-Hungary	☐	☐	☐	☐
Turkey	☐	☐	☐	☐

Section seven



If we expand the diplomacy section of the game, making the agents communicate more and in proper language, do you think this will affect how you perceive their personalities? Check box for yes. ☐

Please fill in any additional comments you have about the game, this questionnaire, us or whatever you would like to share:

255 characters left

If we release a version of the game with an expanded diplomacy section, would you like to test it? If so, please write your email address in the field to the right.

Your email address will not be associated with your answers and will be used solely for sending you information about the game. After finishing this project, all registered email addresses will be erased from our registers.

Please press Finish to end the questionnaire. Finish

Appendix F – The Test Results

Section one

	Gender	**Age**	**Hours played Statecraft**
Tester 1	Male	20-25	1-2
Tester 2	Male	20-25	2-4
Tester 3	Male	26-30	1-2
Tester 4	Male	26-30	4-8
Tester 5	Male	26-30	>8
Tester 6	Male	26-30	2-4
Median	Male	26-30	1-2 / 2-4

Section two

Values range from 1 (least) to 5 (most)

	Computer game experience	**Turn-based experience**	**Diplomacy experience**
Tester 1	4	3	1
Tester 2	2	1	1
Tester 3	4	3	1
Tester 4	4	4	1
Tester 5	5	5	4
Tester 6	5	4	5
Average	4	3.3	2.2
Median	4	3 / 4	1

Section three

Values range from 1 (least) to 5 (most)

	Entertainment level	**Difficulty level**
Tester 1	3	4
Tester 2	2	3
Tester 3	2	4
Tester 4	2	2
Tester 5	1	1
Tester 6	4	2
Average	2.3	2.7
Median	2	2 / 4

Section four

	More unit types	**Chance in battles**	**More diplomatic possibilities**	**Better sound & graphics**
Tester 1	YES			
Tester 2			YES	
Tester 3	YES		YES	YES
Tester 4	YES	YES		
Tester 5			YES	
Tester 6			YES	

	Difficulty levels	**Building / Improvements**	**Resource gathering**
Tester 1		YES	
Tester 2			
Tester 3	YES		
Tester 4	YES		
Tester 5	YES		
Tester 6	YES		

	Other 1	**Other 2**
Tester 1		
Tester 2		
Tester 3		
Tester 4		
Tester 5	A good working AI - in diplomacy this is the alpha and omega.	
Tester 6		

Section five

	Great Britain	**France**
Tester 1		
Tester 2		
Tester 3	Not very aggressive. Was more into holding than conquering	
Tester 4	Quite opportunistic agent, who ceases the chance to conquer territory that is left unguarded. Attacks occasionally.	Aggressive agent that will attack often and also takes unguarded territory.
Tester 5	GB seems to play ok most of the time with good moves and mostly ok support moves	France has problems with German aggressiveness and seems to be very passive.
Tester 6	Was very careful and did not perform many offensive moves.	

	Italy	**Germany**
Tester 1		
Tester 2		
Tester 3		
Tester 4	Also aggressive but not to the same degree as France.	More of a neutral agent.
Tester 5	Italy never does much. Gets stuck on attacking Trieste due to high potential value, but does not outflank for support. Essentially hamstringing early operations because it's only moving two units	Germany is very aggressive, but faulty support play.
Tester 6	Rather passive	Played rather stupid and did not defend very good

	Turkey	**Austria-Hungary**
Tester 1		
Tester 2	aggressive	a little aggressive
Tester 3	Seemed to me to be the weakest player, but somehow managed to do better than me!!	
Tester 4	A very aggressive agent. Probably the most aggressive agent in the game.	Very defensive agent, that rarely attacks.
Tester 5	Turkey is very passive. Poor decision-making. Lack of good support orders, too many unsupported attack moves, poor fleet move decision-making. Never follows up effectively on conquest.	Austria-Hungary is passive. Unsupported attack moves. Expecting chain moves to be present when opposing forces are obviously moving into same territories. Eschews support moves for chain moves. Lack of tactical movement to get supports.
Tester 6	Aggressive	

Section six

	England			
	Emotional Temperament	**Risk attitude**	**Aggressiveness**	**Contempt for rules**
Tester 1				YES
Tester 2		YES		
Tester 3			YES	
Tester 4		YES	YES	
Tester 5			YES	
Tester 6	YES			

	France			
	Emotional Temperament	**Risk attitude**	**Aggressiveness**	**Contempt for rules**
Tester 1				YES
Tester 2				
Tester 3				
Tester 4		YES	YES	
Tester 5				
Tester 6	YES			

	Italy			
	Emotional Temperament	**Risk attitude**	**Aggressiveness**	**Contempt for rules**
Tester 1				YES
Tester 2				
Tester 3				
Tester 4			YES	YES
Tester 5		YES		
Tester 6		YES	YES	

	Germany			
	Emotional Temperament	**Risk attitude**	**Aggressiveness**	**Contempt for rules**
Tester 1	YES			
Tester 2				
Tester 3				
Tester 4	YES			YES
Tester 5	YES		YES	
Tester 6		YES		

	Austria-Hungary			
	Emotional Temperament	**Risk attitude**	**Aggressiveness**	**Contempt for rules**
Tester 1		YES		
Tester 2			YES	
Tester 3				
Tester 4				YES
Tester 5	YES		YES	
Tester 6	YES			

	Turkey			
	Emotional Temperament	**Risk attitude**	**Aggressiveness**	**Contempt for rules**
Tester 1		YES		
Tester 2			YES	
Tester 3	YES			YES
Tester 4		YES	YES	
Tester 5				
Tester 6		YES	YES	

Section seven

	If we expand the diplomacy section of the game, making the agents communicate more and in proper language, do you think this will affect how you perceive their personalities? Check box for yes.
Tester 1	YES
Tester 2	YES
Tester 3	YES
Tester 4	YES
Tester 5	YES
Tester 6	YES

	Additional comments
Tester 1	
Tester 2	
Tester 3	The game might have benefited from some intuitivism. Og kanskje litt mindre ressurssluking.. Løkke t med mesteroppgaven!
Tester 4	
Tester 5	If agents communicate with proper language, then yes, personalities would be easier to perceive. As it stands the diplomacy interface is poor and communicates little. The all-around poor AI decision-making is bound to mask the personalities.
Tester 6	

The authors:

Arild Johan Jensen, student no 135490
Email: ajoje@broadpark.no
Mobile: 91736278

Håvard Nes, student no 176351
Email: haavard.nes@gmail.com
Mobile: 9751516