REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

Rational Dialog in Interactive Games

Maria Arinbjarnar

Master of Science
June 2007

Reykjavík University - School of Computer Science

M.Sc. Thesis

REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

Rational Dialog in Interactive Games

by

Maria Arinbjarnar

Thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Master of Science

June 2007

The undersigned hereby certify that they recommend to the School of Computer Science at Reykjavík University for acceptance this thesis entitled **Rational Dialog in Interactive Games** submitted by Maria Arinbjarnar in partial fulfillment of the requirements for the degree of **Master of Science**.

Date

Dr. Luca Aceto, supervisor
Professor, Reykjavík University

Dr. Finn Verner Jenssen
Professor, Aalborg University

Dr. Hannes Högni Vilhjálmsson
Lektor, Reykjavík University

Rational Dialog in Interactive Games

by

Maria Arinbjarnar

June 2007

Abstract

"I am fairly sure of this that none ever willingly errs".

Socrates

The motivation for this research is an increased call for highly interactive games in virtual realities with intelligent Non-Player Characters (NPCs). The NPCs currently implemented in computer games are rarely autonomous, although some have an emotional drive and a set of goals to chase. Their actual interactions are usually pre-set or very limited. Additionally the games themselves have pre-set narratives that result in games that the average player does not care to play numerous times, simply because the game is always the same.
The question addressed in this thesis is whether an NPC will interact with a player and other NPCs in a rational and goal driven way when given a past life and a decision mechanism based on a causal network like a Bayesian network. Will the NPC adopt a strategy that will maximize its pay-offs?
To answer this question I built an engine that creates NPCs that have past lives, a knowledge base and tools to find a sentence to speak in a rational dialog. The knowledge base and past lives of the NPCs are created from plots that a Dynamic Plot Generating Engine (DPGE) creates. The DPGE creates continuously new plots for murder mystery games that are logically consistent. The decision mechanisms of the NPCs are modelled using Multi-Agent Influence Diagrams (MAIDs), a mathematical method for multi-agent decision-making in competitive situations.
The engine created clearly indicates a basis to create NPCs that can participate in a rational dialog by calculating optimal sentences on the fly. The time complexity is linear in respect to number of sentences and more than half of the sentences are calculated in less than 1 minute. With some standard optimizations these results can be greatly improved.

Vitræn samræða í gagnvirkum leikjum

eftir

Maria Arinbjarnar

Júní 2007

Útdráttur

"Ég er nokkuð viss um það að engin gerir nokkurntímann viljandi mistök".

Socrates

Kveikjan að þessari rannsókn er aukin eftirspurn eftir gagnvirkum leikjum í sýndar-veruleika (e. virtual reality) með vitrænum (e. rational) tölvu-stýrðum leikmönnum (e. Non-Player Characters (NPCs)). Þeir NPCs sem eru í tölvuleikjum í dag eru í fæstum tilfellum alveg sjálfvirkir, Þó hafa sumir sjálfvirka tilfinningasvörun og eru gjarnan markmiðs drifnir. Samt er megnið af samskiptum þeirra við umhverfið og aðra spilara forskriftað og frekar takmarkað. Auk þess eru leikirnir sjálfir með forskriftaðann söguþráð sem verður til þess að fáir vilja endurtaka leikinn vegna þess að sagan er alltaf sú sama.
Hér er þeirri spurningu varpað fram hvort NPCs geti átt vitræn og markmiðs drifin samskipti við aðra leikmenn ef hver NPC fær ævisögu og algrím til ákvarðanatöku sem er byggt á ákvarðanatöku neti eins og *Bayesian networks*. Til að svara þeirri spurningu þá byggði ég vél sem býr til slíka NPCs. Þekkingargrunnurinn og ævisaga leikmannanna er búin til með *Dynamic Plot Generating Engine* (DPGE), sem býr til nýjar lógískar flækjur fyrir morðgátu leiki. Ákvarðanatöku algrím leikmannanna notast við *Multi-Agent Influence Diagrams* (MAIDs), sem er stærðfræðileg aðferð, sem notast við *Bayesian networks*, fyrir tölvu verur (e. agents) til að taka ákvarðanir í samskiptum við aðrar verur þar sem ríkir samkeppni.
Vélin gefur skýrt til kynna að það er grundvöllur til að skapa NPCs sem geta tekið þátt í samræðum með vitrænum hætti. Vöxtur MAIDs er línulegur miðað við fjölda setninga. Meira en helmingur setninga var ákvarðaður á innan við mínútu, þann tíma er hægt að flýta töluvert með nokkrum vel þekktum bestunar aðferðum.

To Tómas V. Albertsson,
. . . gave me all the time that I needed for the thesis work, taking care of our children and more than his share of household chores and tolerated a frequently to stressed wife.

Acknowledgements

Thanks to:
Professor Luca Aceto at Reykjavik University, gave me his backing to tackle this unprecedented work at the Computer Science Department of Reykjavik University even though there was no expert, in either of the main fields used in the research, Bayesian nets and game theory, available at the university. Luca Aceto's input and reviews have been an invaluable aid in the research and writing of the thesis.

Professor Finn Verner Jensen at Aalborg University, and the Machine Intelligence group in Aalborg University, Denmark. I visited their department twice during the research work and their expert input and criticism is invaluable.

Assistant Professor Hannes Högni Vilhjálmsson at Reykjavik University, for good discussions and references to related work.

Publications

Some of the material presented in this thesis has appeared in other publications:

(Arinbjarnar, 2007) Rational Dialog in Interactive Games. In *Proceedings of AAAI Fall Symposium on Intelligent Narrative Technologies*. Westin Arlington Gateway, Arlington, Virginia.

(Arinbjarnar, 2008) Dynamic Plot Generation Engine. In *Proceedings of the Workshop on Integrating Technologies for Interactive Stories*. Playa del Carmen Mexico.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

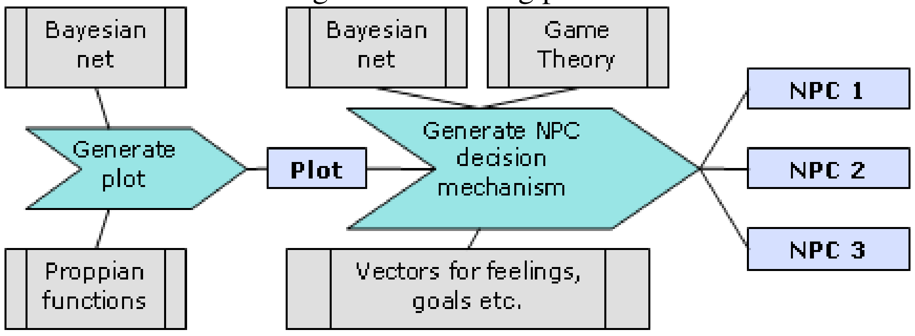
The motivation for the research described in this thesis is the increased demand for highly interactive games. Moreover, with a growing game industry, there is a continuous call for games that are intellectually challenging. I firmly believe that computer games that include dynamically changing plots and rationally interactive Non-Player Characters (NPCs) will open up a completely new genre of games. The techniques adopted in constructing such games may well be usable in interactive teaching programs and behavioral analysis.

By dynamically changing plots I refer to patterns of events or main stories in narrative or drama based games that change in response to actions by NPCs or players. A *rationally interactive NPC* is an NPC that interacts with both players and other NPCs in a rational way. Rationality is defined as the choice of actions which best satisfy a person's objectives. These objectives are desires that motivate the individual (Heap, Hollis, Lyons, Sugden, & Weale, 1992). Rationality is explicitly defined and discussed in this thesis both from the mathematical viewpoint of game theory and from a philosophical viewpoint where aid is sought from the ancient Greek philosophers.

The problem domain of creating rationally interactive NPCs is vast and thus I emphasize on creating a rational dialog in the sense that an NPC decides on what to say in a dialog by finding a rational sentence to speak. The specific problem investigated in the thesis is whether an NPC will interact with a player and other NPCs in a rational and goal driven way when given a past life and a decision support mechanism based on a causal network like a Bayesian network (Jensen, 2001). Will the NPC adopt a strategy that will maximize its payoff? The success of the research should be measured by the terms defined by game theory (Heap et al., 1992) and by the NPC's determination in maximizing his independent payoff, e.g. his goals.

There are three main problems that the above-mentioned question raises and all of them will be tackled in the thesis. First it is necessary to create the past life of the NPCs. Secondly each NPC needs a causally driven decision mechanism that contains the NPC's knowledge base characteristics and information that he gathers along the way. The third and final problem is to make the NPC capable of finding equilibrium as defined by game theory so that he can be said to maximize his payoff when talking to another NPC. My solution to the problem stated above takes the form of a prototype computer engine whose general structure is described in Figure 1.1.

Figure 1.1: The big picture



The game setting that I have chosen for this research is a murder mystery game. At each new game, initialized by the player, there is a completely new narrative plot and set of characters. The initial plot is created with the Dynamic Plot Generating Engine (DPGE) (Arinbjarnar, 2008) that creates new mystery plots on demand using a Bayesian net and Proppian functions and can be seen to the far left in Figure 1.1. Bayesian nets are causally connected belief networks that use the rule of Bayes to determine the probability of some event or property given the conditional probability that other properties hold. Proppian functions refer to functions that are adapted to creating murder mystery plots from the morphology that the Russian structuralist Vladimir Propp constructed (Propp, 1968). Each resulting plot is consistent and coherent, meaning that the murderer has a credible motive and had the opportunity and means to commit the murder. Moreover the murderer can be discovered by pure deduction based on clues found in the game. The DPGE is the result of my BSc project but the version I employ in this work has been extended with motives and relations between characters of the plot.

The plot generated by the DPGE is used to create the initial setup of the NPCs. At the start of each game there is some initial configuration that the game world is in. This means that each NPC has some life history and is connected to the murder victim and/or other NPCs in the game and that the plot is “fully developed" up to this point in time. Note that this is only at the beginning of the game, when the NPCs start to interact with the world then the stasis is lifted and the plot develops. This initial plot is used to generate the decision mechanism of each NPC using Bayesian nets and game theory; see Figure 1.1. Each NPC is given a knowledge base that in respect to the configuration of the world and the NPC's role within the game. As an example a suspect knows everything about himself but has for the most part only approximate knowledge of the crime and other NPCs. The murderer on the other hand has in addition to a complete knowledge of his own person also a fairly complete knowledge of the crime. So each NPC gets a blueprint from the newly generated plot in the form of a Bayesian net that is identical to the one that generated the plot. The knowledge that he should have in respect to his role is instantiated. Moreover he is equipped with tools to be able to interact rationally with other NPCs or the player. The NPCs are also equipped with a Bayesian net to estimate how other NPCs see the world. Finally the NPC has tools to update his knowledge base with information from other NPCs.

As the game progresses the NPCs will interact with each other or the player by exchanging information with them. At each turn the currently active NPC needs to find a rational thing to say: that is, he wants to speak a sentence that maximizes his payoffs. In order to do so he needs to find an equilibrium by the means of game theory. This the NPC accomplishes in three steps.

- First, the NPC generates possible sentences and possible replies.
- Next, he calculates the optimal reply for each of the sentences.
- Finally, he calculates the optimal sentence given the optimal replies.

The method described above is developed using Multi Agent Influence Diagrams (MAIDs) (Koller & Milch, 2003). MAIDs are extended Bayesian nets specifically designed to find Nash equilibria for incomplete data, which is exactly the problem that the NPCs face.

The result is a rationally interactive NPC that is able to receive information from other NPC, evaluate it and, if he finds it sound, add it to his knowledge base. Thus the plot develops by means of interactions between the characters very much as it would in real life. The NPCs have no distinction between another NPC and a player, so if a player takes the part of a suspect or murderer defined by the plot engine then the NPCs will interact with the player just as any of the NPCs.

There where challenging technical problems that needed to be addressed in achieving rational dialog, the most prominent problems where the following two. First was the construction of a knowledge base for the NPCs. This problem was twofold the knowledge base needs to be sufficiently abstract to handle any type of knowledge and it may not become to complex so that it will take to much time to compute it. The technical problems of complexity where solved by splitting and aggregating nodes as is discussed in Section 7.1 and by abstracting and then aggregating the decision variables to the net using relational aggregation as is discussed in Section 7.2. The knowledge base was divided into specific sections called subnets that use sufficient abstraction to be able to deal with any type of knowledge and handle causal connections between the abstract knowledge objects, (see Section 9).

Secondly there was the problem of determining abstract types that where needed to find an equilibrium by the terms of game theory as is described in Section 6.3. A naive approach resulted in an exponential number of types. The solution to this was the use of MAIDs and the construction of two decision setups for each NPC, one for the optimal reply that he expects in a dialog and an other for the optimal sentence that he should say in respect to the reply that he expects. This is explicitly described in Sections 8, 9.4 and 10.4.

This project is only a proof of concept and the current implementation is small and very limited. The NPCs have a fairly small knowledge base of just over 3000 nodes in their Bayesian nets. The algorithm for finding equilibrium lacks many possible optimizations that could speed it up considerably. Still equilibrium is reached on average in just under three minutes and sometimes in just 11 seconds. Additionally the complexity of this approach is linear in respect to number of sentences.

In the process of this research it has become my firm opinion that rationality and game theory are some of the necessary keys to create an intelligent machine. In particular that it is essential to have an intelligent agent rational. Such a notion is not new, it appears in Plato's dialogs as arguments from Socrates that no one will willingly choose an erroneous action, meaning that no one would choose an action that he himself beliefs will do him greater harm than good.

Section 2 succinctly states the necessary preliminary information needed to read this thesis. I strongly recommend that the avid reader scan that section before continuing. Section 3 describes related work. Section 4 discusses with examples the philosophical aspect of rationality. In Section 5 I introduce the Russian structuralist Propp and how his work relates to the thesis and the project. Sections 6, 7, 8 are technical details that only those who want the full picture should read. In Section 6 I give as detailed description as necessary of game theory to fully grasp the techniques from game theory that are being used. In Section 7 I describe the basic technicalities of the Bayesian net and define some extra algorithms that are necessary to efficiently build the NPCs decision mechanism. In Section 8 I describe the MAIDs and how they are used to find Nash Equilibrium. In Section 9 I describe in detail the knowledge base and the general make up of the NPC. In Section 10

I describe a complete run of the engine from creating a plot and until an NPC has spoken. Finally in Section 12 I summarize and discuss the results and point out future work to be done.

Chapter 2

Preliminaries

In this project two strong fields are combined namely game theory and belief networks, to solve problems in interactive computer games. Work of many prominent scientists in computer science, mathematics, structuralism and even philosophy are used to construct a sufficiently sound approach for solving the complex problem at hand. This results in the need to choose one terminology to use consistently throughout the project. I have after some consideration decided on the terminology used by Fudenberg and Tirol in their book (Fudenberg & Tirole, 1991). The reason is that it is a very well defined terminology that has been used throughout the game theory society for a number of years. It is well developed, succinct and unambiguous. To describe those elements that are used in belief networks, and that the game theory terminology does not cover, I use the terminology of Koller and Milch in their article (Koller & Milch, 2003). This is also because this is the terminology used to define the MAIDs that I rely on to calculate Nash equilibrium in this project.

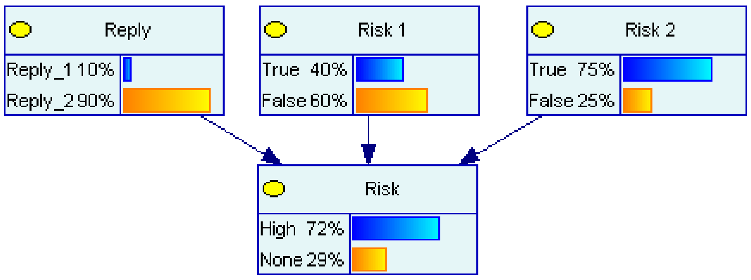
Henceforth in this project I will use the word *player* as the active NPC that needs to make a decision. This is in accordance with the use of the word player in game theory. This explicitly means that player is never referring to a human being unless specifically indicated.

Opponent is the player that the current player is talking to and can be a human being or another NPC. No distinction will be made between whether the opponent is a human or the computer unless it has some significance. The player is not aware of a distinction between a human and a computer opponent and reacts in the same way to both.

A player is denoted as $i \in \mathcal{I}$ and his opponent is denoted as $-i \in \mathcal{I}$, where $i \neq -i$. This is described in detail in Section 6.

Graphical display issues. It is necessary to remark on the visual representation of the Bayesian net. I use the very good graphical tool Genie (DSL, 2007a) to visually represent the Bayesian nets. In Genie the values in the variables are rounded to two digits for better viewing and transparency. This can cause some apparent discrepancy as can be seen in the $Risk$ variable in Figure 2.1.

Figure 2.1: Example of value discrepancy



Intuitively the values $High$ = 72% and $None$ = 29% should add up to exactly 100% and not 101%. The actual values in this example are $High$ = 0.715 and $None$ = 0.285 which indeed do ad up to 1. This will not be explained again in this project. Instead I will rely on the reader's deductive intelligence, when evaluating graphical examples, to realize this basic rounding of values.

Code practices. When displaying algorithms or code I try to simplify the presentation for the sake of clarity. The prototype engine that I have developed is coded in Java (Sun, 2007) and the code is thus just as illegible as most things written in such an overly verbose programming language. When presenting the main algorithms I use Python syntax (Python, 2007) in order to simplify the display. I also use the algorithm2e Latex package which means that the Python syntax is not meticulously followed.

Chapter 3

Related work

There are numerous experiments dealing with automatic plot generation. Many are trying to construct a story or narration of some sort. These are sometimes called interactive storytelling engines. Storytelling engines usually have two main goals; first to create an interactive narrative, a preset narrative that can be influenced by the player. This means that some major narrative goals or events need to be accomplished but that the player can influence the narration in some small ways. Moreover that the characters should be dynamically responsive to the player's character and actions. Secondly to have the interface of the storytelling engine sufficiently simple so that non-technical people can use them to author new stories. There are a number of examples of this technique and here I will name some of them.

A recent one is Thespian (Si, Marsella, & Pynadath, 2005) which emphasizes on creating autonomous agents that are both responsive to the user and consistent with their own internal motivations and goals. Moreover that the characters and the story line can be easily authored by people lacking programming skills.

Another recent example is the Open ended Proppian Interactive Adaptive Tale Engine (Fairclough & Cunningham, 2004). The engine creates interactive narrations and is in a large part based on Propp's Morphology (Propp, 1968). The most interesting part of this engine is a gossip system that connects the NPCs together and spreads news and opinions about the current player character in respect to his actions.

The Story Engine (Schneider, Braun, & Habinger, 2003) is also based on Propp's Morphology. This is basically an engine to assist in writing multi linear stories for Role-Play Games. It uses the Propp's Morphology to avoid deadlocks and to assist in creating a literary sound narration.

Finally a system that is worth mentioning is the Character Focused Narrative Generation (Riedl & Young, n.d.). There the authors have a set of actor-agents that each play out a character. The actors plot a few moves onwards with assistance of a director agent and a blackboard. Blackboard is a message board to post messages between agents. The movements of the actors are causally dependent on each other and this restricts the plot development and creates seemingly unnecessary complications.

The problem with such forced narration, as has just been described here, is that the chance of a dead end or some other major logical conflict is very high and in some cases practically unavoidable. For example let's imagine a wizard having a vital key to enter some important place. Frequently a game would have a wizard give a quest like killing a beast and retrieving a vial of its blood. As discussed by Lindley and Eladhari (Lindley & Eladhari, 2002) many games have the problem that if you happen to kill the beast before you meet the wizard for the first time, then he cannot give you his quest. The problem here is with perspectives and backwards causality. The player is supposed to have met the wizard *before* killing the beast and that if he failed to meet the wizard prior to killing the beast that then he can't get the key. This is equivalent to having a constraint that a student must registering at the university before taking a course in Calculus. That if the student first takes a course in calculus and then registers at a university that then he can not graduate because he is not allowed to retake the course or use the scores from the previous course. When put into perspective it becomes obvious that this is backwards logic. If the player kills the beast then the beast is dead and if he meets the wizard afterwards and can give him the vial of blood as proof, then he should be rewarded, just as it does not matter when and where you learn calculus. If you pass the tests and have mastered the subjects required at a specific university then you should be able to graduate from that university. This problem of forced narration is widely recognized and some solutions have been offered. Many involve a graph of some sort as the following work shows.

The DPGE (Dynamic Plot Generating Engine) that I use to create a murder mystery plot and past life for characters differs from the above solution in one significant way. The plot that results from the DPGE is used to decide the past, that is, what has happened int the world and for the characters up to the point in time that the game starts. The DPGE is not intended to form future actions.

Verbrugge (Verbrugge, 2002) proposes a narrative flow graph that is based on a Petri net (Reisig, 1988) that enforces constraints that he claims are necessary for a narrative game. The petri nets are used to ensure that a player can not advance in the game unless he has fulfilled certain goals from an authored narration. Of course he has a perfectly valid point if the aim is to enforce a predetermined narration but to enforce a predetermined narration is not appropriate in an open-ended dynamic game world.

Lindley and Eladhari (Lindley & Eladhari, 2002) also discuss this problem of forced narration in some detail. They propose a couple of solutions. The former is to apply Boyce-Codd Normal Form to a causal net (Lindley & Eladhari, 2002) and the latter is to apply object orientation and loose coupling. Their ideas are very sound but as with (Verbrugge, 2002) they still enforce a preset narration.

The following experiments do not attempt to force a specific storyline onto the game or make actions of characters in the game causally dependent on each other.

In the non-linear interactive storytelling game engine (NOLIST) (Bangsø, Jensen, Jensen, Andersen, & Kocka, 2004) they utilize a Bayesian network to determine the culprit of a murder mystery; the Bayesian network is dynamically changing in response to actions and observations made by the player. It is not preset but rather it uses the player's moves and the logical inference of the net to determine the culprit. For example if the player finds a body and a gun lying beside the body then the probability that the murder victim was shot with the gun increases. The proposers of NOLIST do this to be able to construct an engine that will create a dynamic storyline with respect to player actions and choices. The plot and the culprit are not known by the game engine in the beginning but are determined in the course of the game. Thus NOLIST creates the past in reaction to player interaction. Although NOLIST is highly adaptive to player interaction it might actually be too reactive. Players are likely to play games in similar manners each time and are thus not good at surprising themselves.

This use of Bayesian nets to form a murder mystery plot resembles the DPGE to some extent. Except that NOLIST does not fix the mystery plot at the start but continuously develops it through game play. The DPGE on the other hand fixes the initial plot at the start and then expects characters in the game to use it as background for future actions.

The Virtual Story Teller (Theune, Rensen, Akker, Heylen, & Nijholt, 2004; Theune, Faas, Nijholt, & Heylen, 2002) deploys a director agent and character agents. The director agent's job is to direct the character agents towards meaningful goals to enforce an emerging storyline, the character agents are very independent and their actions are controlled by their feelings and by the priority of their goals, both of which are continuously changing in light of actions and input from their surroundings. The director agent can forbid or accept an action chosen by a character but he cannot command the character, which essentially keeps the character actions coherent and sensible for that character. The characters are given individual characteristics that have a direct influence on their feelings, reactions and response to input from other characters and the environment. They are also given goals, primary and secondary, that are the driving force of their actions. Thus a storyline emerges by the NPCs reactions to in-game events as they attempt to fulfill their goals. This has not been implemented in an actual game and it appears that the player would be hard pressed to be at the right place at the right time to interact intelligently with an NPC and participate in the storyline.

An interesting approach of mapping existing theories of human behaviorism into computational models uses C Language Integrated Production System (CLIPS) to build a rule based system to manage agent emotions, drives and relationships with other agents (Chaplin & Rhalibi, 2004). CLIPS provides a complete environment for the construction of rule and or object based expert systems. That system uses the Iterated Prisoners Dilemma (Heap et al., 1992) to give the agents a chance to defect and for other agents to show vengefulness if they find out about the defection. The authors implement the Iterated Prisoners Dilemma through a set of norms. They apparently manage to connect a rule based system with emotional drive based agents so that the agents are able to sustain themselves and to coexist to some degree. The results put forth appear to be from a very limited scenarios where the expected behaviors have been isolated.

Another approach compares adjustable rule sets and Neural Nets to model human behavior (Tolk, 2002). The proposers of this approach discover that even with adjustable rule sets, the rules still need to be situationally adequate, that is they need to be closely modeled for the actual scenario at hand and that Neural Networks will give a better performance. On the other hand it is not always clear what the Neural Nets learn, e.g. how they come to determine the expected reaction to stimuli, while the rule based systems are much more transparent.

Davis and Lewis integrate an emotional model into two computational architectures (Davis & Lewis, 2003). First in a four layer computational architecture emotions are accepted to be appraisal states with descriptive or valancing and causal or arousal processes. This concept is used to provide a control or regulatory framework to model the different forms of “emotion inducing" events. They base this on the assumption that all agent events and actions, internal and external, can be described in terms of this four layer computational architecture of emotion. The second computational architecture proposed by Davis and Lewis is the Children Reasoning about Intentions, Beliefs and Behavior computer model (CRIBB) which is based upon a general sketch for belief-desire reasoning in children. It simulates the knowledge and inference processes of a competent child solving false-belief tasks. This is an ongoing research.

An approach that vaguely resembles the intended decision mechanism of this proposal is from (Silverman, Bharathy, O'Brien, & Cornwell, 2006). They use three sets of trees called short term Goals, Standards for behavior of self and others, and long term Preferences, or GSP trees. These trees are specific to each character motive modeled, e.g. they are specific for sex, culture and other character specific attributes. The trees have probabilities to determine the next action based on the previous action. For example, a Somalian civilian female that is in a state of "wanting to belong" then she is 70% likely to *obey orders* and 30% likely to try to *protect her friends*. These trees are clearly very speciffic.

Finally B.J. Rhodes proposed an interesting approach using heuristic planning techniques (Rhodes, 1996) in hybrid networks. Rhodes focuses on developing characters that use planning techniques to realize their goals. Each characters has multiple means to accomplish his goals. The character should also have multiple and sometimes conflicting goals to achieve in a dynamically changing world. The project was fairly successful and promising but needs more work.

Chapter 4

Rationality and Philosophy

In the process of this research on rationality in computer played characters I have had the fortune to discuss my ideas with very intelligent and proficient researchers. They have for the most part acted very rationally. Still there are times when they have not appeared to be acting in a rational way. Perhaps this is because they where tired or had other more important things on their minds or perhaps when I was evaluating their behavior I was seeing the situation from a different perspective.

When evaluating my own behavior in hindsight I see far to many instances where I myself acted irrationally and I often wander why I do not learn from it and take just a few more moments to think before acting. It would greatly reduce my irrational behavior.

Notice that this is always an evaluation made from a different perspective than the actual person has when deciding how to act. I also see and know that when I have acted irrationally I did indeed find it to be very rational at the moment of acting, albeit not necessarily very rational just a heartbeat later. I have never experienced a moment where I have decided to do something irrational without some kind of rationality behind the behavior. Even when deciding to act very silly or in a manner that will clearly be damaging to me or something that I care about.

Surely such a statement that I do not remember ever having intentionally acted irrationally is a futile argument and has no bearing in a scientific research. Surely it is evident that people are very irrational and frequently base their decisions on feelings and emotions with little or no thought of the consequences. We can take many examples of this. Parents loosing their temper at their children over trivialities. Gamblers loosing all their possessions over night in a game of cards. People correcting the clerk in a supermarket when he gives them to much change.

It can really be argued that if people would act rationally that then most everyone would fare better than they do. We would make much fewer mistakes. We would state our needs and opinions at appropriate times and in an appropriate manner. Feelings and emotions would not cloud our judgment and so we would never reveal a secret by accident, be offensive, rude or shy away from an important discussion because of fear.

I disagree! I firmly belief that it is exactly the case that we always act rationally and that this is exactly the reason for the apparently irrational behaviors that we constantly exhibit.

Here follows a formal argument based on Platos dialogs (Plato, 1967) where Socrates discusses virtue, justice and knowledge.

4.1 Socrates and rationality

Socrates and Protagoras agree that knowledge is a noble thing that is able to govern man. That when he has learned what is good an what is bad that then his intelligence is a sufficient aid for man to act rationally and choose good over bad. At first sight this appears to be a very naive perspective but their argument is sound and is as follows.

First it is necessary to determine what is meant by good and bad and how we evaluate anything as being either good or bad or some degrees thereof.

We consider the following to be good: pleasure, health, fitness, wealth, skill, knowledge, virtue, etc. We consider the following to be bad: pain, disease, illness, obesity, ignorance, poverty, misery, etc. We can define this more explicitly. We consider to be good anything that results in pleasure either immediate or in the end. We consider bad anything that results in pain either immediate or in the end. such as eating good food carries immediate pleasure but can result in obesity and thus pain. Exercise is painful at the start but results in a fitter body and better health, e.g. pleasure. Now that the words good, bad, pleasure and pain are explicitly defined we can use them to capture the essence of the rest of the argument, this will make the argument more concise.

Exercise then is both good and bad. When we decide whether to exercise then we evaluate the remote pleasure with the immediate pain of starting. If the pain of starting is less than the remote pleasure then we will exercise. When deciding weather to eat we also compare the immediate pain or pleasure depending on whether the food has a good or bad taste with the result which can be pleasure such as better health, satisfying hunger or it can be pain as in increasing obesity.

Clearly we choose by evaluating how good and how bad it appears to us at any given time. We consider something to be good if either the immediate or remote pleasure is greater than the immediate or remote pain. Good and bad are not absolute off/on values but a measurement of good and a measurement of bad weighted by their remoteness, such that if the measurement of good is greater then the measurement of bad then it is a positive value and when the measurement of bad is greater then the measurement of good then it is a negative value.

As an example let's consider two individuals John and Jack who are deciding weather to exercise. John likes to exercise he is in shape and when he exercises he feels more alert and at ease. He finds exercising to be refreshing and to reduce stress. He also realizes that as a consequence of exercising he will be in better shape and health. This remote pleasure is an added bonus for John.

Jack on the other hand does not like to exercise. He is out of shape and when he exercises he gets dizzy and tired and most of his muscles complain. He can't stop thinking of all kinds of things that would give him greater immediate pleasure. Jack knows though that if he continuous to exercise then he will gradually get into shape and then the exercise will start to be more pleasurable. Moreover he knows that if he continuous to exercise then he will be in better health and most of his other activities will be more pleasant.

It is evident that John considers exercise to be good and so his rational choice is to exercise. Jack on the other hand has a more challenging problem. If Jack considers the pain of exercising to be greater than the remote pleasure then Jack considers the act of exercising to be bad. This is despite the fact that he realizes the overall benefits of exercising. If Jack in this way considers the act of exercising to be bad then it would be irrational of him to exercise. He has weighted the good and bad and finds exercising to be bad. Now Jack may very well regret his decision later on and he may very well be in considerable need of an exercise because of his bad physical health, but Jack himself needs to realize this at the moment of making his decision for it to be a rational decision for him to exercise.

Moreover there are other things to consider, Jack is hungry and there is a hot dog stand nearby. Jack sees only the pleasure of eating the hot dog and the remote pleasure of satisfying his hunger. The good outweighs the bad and so eating a hot dog is good, it is the rational thing for Jack to do.

Know John looks up from his exercises and sees Jack eating the hot dog. From Johns perspective then Jack is being very irrational. This is because John has difficulties in realizing the values that Jack used in his equation.

From this it follows that we can in fact not expect an optimal outcome based on a rational decision. We can only assume that it is the optimal decision that this individual calculated based on his current data and his skills in calculating a rational decision. For to assume that people will fare better when acting rationally then we also have to assume that they will have sufficient skills to calculate the overall optimal decision each time and to have unlimited access to all relevant data. Moreover they would need to have infinite amount of time to be able to evaluate every possible action in respect to both long term and short term affect.

Another question needs to be raised and answered in this respect, will a man always choose the right thing if he knows all the possible outcomes of an action? My answer to this is that he will choose the right thing if it is the rational thing for him to choose as he sees it. It does not necessarily hold that what is optimal for an individual is also optimal for the whole. It may very well be that there is a conflict of interest between the individual and the community.

let's also address the formerly proposed irrational actions, namely:

- Parents loosing their temper at their children over trivialities.
- Gamblers loosing all their possessions over night in a game of cards.
- People correcting the clerk in a supermarket when he gives them to much change.

If people are so rational then why will not parents see that a child has limited control of its movements and body functions and lacks the ability to realize consequences of its actions. A child is prone to make constant repeated mistakes until it matures sufficiently. The reason is of course that when the parent is interacting with the child then the parent frequently lacks the ability to evaluate the bigger picture. The parent is tired and thinking about many other things than the current noise coming from the child. The parent is thus liable to consider a few sharp words an optimal way to handle the immediate situation. Then just a few heartbeats later when the parent sees the reaction of the child and realizes its error, the parent is liable to wonder how on earth he could ever have even considered saying such things.
A similar case is with the gambler the momentary thrill and pleasure of gambling greatly overweights the perceived distant pain of losing a large portion of his possessions and diminishing his families trust.

A different perspective is needed when looking at the person who corrects a clerk when he gives him to much change. It is evident that the person does not have immediate gain from it, he loses money. There must be another gain that he considers to outweigh the financial gain. An obvious one is that he does not care to consider himself a thief. It does not matter if no one else notices, it is his own perception of himself that matters. To have such values affect our daily decisions and reactions we need to have some princips or ethical makeup that dictates what will pain or pleasure us in such circumstances. He could also be thinking of the clerk in question and worry that he would need to take the blame or he could value correctness over a small financial gain etc.

As can be seen, especially from the last example, is that in determining a rational action people can have quite a range of values and ways to calculate rationality.

4.2 Computers and rationality

Now we have a definition of good and bad as a measurement of the resulting pleasure and or pain. The pain and pleasure can be considered to be the prior calculations that a Bayesian net can do. The Bayesian net needs to have the significant qualifiers of what yields pleasure and pain. It can then calculate with a degree of certainty both immediate and remote pains and pleasures of an action. These prior calculations can then be set up in a game by the terms of Game Theory where each possible action is compared against each other. The most optimal action is then the action that has the most resulting pleasure or the least resulting pain. This will be the rational action to play and a character that chooses an action by these means will be rational. A more explicit discussion of how the Bayesian net, Game Theory and rationality is used in my MSc thesis is in section 9.4.

The computer today already far surpasses our computing skills but even as fast as it is and will become it can not calculate every possible outcome it will always have to approximate and it, just as we, will always have a limited set of data and degrees of certainties to base its calculations on.

Computers in fact do not act rationally for it is their irrational behavior that makes their behavior inhuman. The only times that computers act rationally is by chance or when they have been programmed to do so. They can not act rationally of their own accord until they have been designed to do so. This is certain to happen in the not to distant future. We create in our own image.

At present computers lack sufficient knowledge to reason and set things in context to be able to act rationally. They are in fact unable to perceive pain and pleasure and thus determine by themselves what is good and what is bad. The computer needs a sense of awareness to be able to act rationally such that it needs to evaluate each action as being a measurement of good or bad and then choose the optimal action.

When computers will act in a fully rational way of their own accord then we will be able to call them autonomous and to posses a sense of self-awareness and consciousness. Then computers can be considered to be intelligent.

Chapter 5

Propp's Morphology

Vladimir Propp (St Petersburg, April 29, 1895 - Leningrad August 22, 1970) was a Russian structuralist scholar who analyzed the basic plot components of Russian folk tales to identify their simplest irreducible narrative elements. His Morphology of the Folk Tale was published in Russian in 1928, but was generally unnoticed in the West until it was translated in the 1950s.

Propp observed that the Russian fairy tales, classified by Aarne index 300 to 749 (Aarne, 1911), are constructed of repetitive themes where actions and functions stay the same but the names and attributes of the persons change. In this case a function is defined as an act of character and viewed for its significance in the course of an action. This means that each function is independent from the rest of the narration and is only recognized for its effect on characters each time and for its part in completing an action. From his careful observations Propp was able to propose the following 4 rules for the makeup of the set of Russian fairy tales:

1. Functions of characters serve as stable, constant elements in a tale, independent of how and by whom they are fulfilled. They constitute the fundamental components of a tale.

2. The number of functions known to a fairy tale is limited.

3. The sequence of functions is always identical.

4. All fairy tales are of one type in regard to their structure.

Propp numbers the functions and states correctly that no function may happen after a function of a higher number has happened. This is to ensure a coherent time line. Propp managed in this way to map a sort of unintentional logical time line for the folk tales.

Some of Propp's functions are paired such that one will logically follow another. Yet the functions are not causally dependent such that the former has to have come before the latter. The latter function can happen independently from the former (Propp, 1968).

Example 1 (The Proppian function.) *Fairy tales frequently start with an interdiction that is addressed to the hero:*

- *you dare not look into this closet or*
- *don't pick the apples*

This is followed by a violation, the hero does what he had been warned or forbidden to do. If there is no prior interdiction then the hero is seen either violating some common norm, like being late or oversleeping, or he ventures out because of need or because of the urgings of some other character.

In respect to Propp's Morphology I implemented the motive for committing a murder as constraints on the Bayesian net rather than explicit functions. These constraints conform to the ideology that there is a certain structure necessary to create a mystery story and that this structure is independent of actual objects or specific individuals. Instead they are fulfilled by specific roles.

There are three roles defined; victim, murderer and suspect. The murderer has the most detailed knowledge of the crime and he alone will satisfy the constraints that define a murderer. These constraints are: all evidence match, has motive, is connected to the victim and had opportunity. There is only one murderer in the generated plot. The suspect has less knowledge of the crime than the murderer and a suspect will fail to satisfy one or more of the constraints that define a murderer. All of the characters are either related to the victim or have a motive to kill the victim or both.

In order to have a set of probable motives I used some of Agata Christies murder mystery novels for good ideas. The result from a light review are the following motives.

Swindle. Either the murderer is swindling the victim and the victim threatens to tell or the victim was swindling the murderer and the murderer wanted revenge. Either way the swindler should be rich.

Blackmail. Either the murderer was blackmailing the victim and feared that the victim would reveal it or the victim was blackmailing the murderer. In either case it is necessary that the one being blackmailed has some dark secret or else he would not be blackmailed.

Wedlock. In Agatha Christie's time it could sometimes be difficult and or bad publicity to get a divorce. This can also apply today that people do not wish to go through the divorce process and lose half their belongings and possibly custody of the children. The constraints in the net for wedlock is that the murderer is the victim's spouse.

Inheritance. This is self explanatory and has the necessary constraint that the victim must be rich and the murderer must be an heir.

The victim was having an affair with the murderer's spouse. This is pure jealousy and the necessary constraints are that the victim needs to be having an affair with the murderer's spouse.

Revenge. The murderer believes that he has just cause for revenge. The victim must have harmed the murderer in some sense in the past.

Dept. The murderer owed the victim lots of money and could not pay. This has the necessary constraint that the murderer is not rich.

The first rule of Propp's Morphology asks for functions that are stable, constant elements in a tale, independent of how and by whom they are fulfilled. This is what the constraints above accomplish they are constant elements that are filled by constant roles such as murderer and victim rather than being filled by specific characters. The constraints are limited in number as Propp's second rule asks for. The third rule really sets the feeling of a past history that one would expect such as; there is a victim because it was murdered by a murderer that has a motive which is the result from some past interactions with the victim. Here the time line was just recounted in reverse order. It could never hold that a murderer murders someone for no reason and then afterwards is swindled by the dead victim. Finally the net with it constraints is a single structure which all the plots are generated from and thus the fourth rule also applies.

Chapter 6

Game Theory

In this Section I describe those essentials of game theory that are necessary to understand the problem domains tackled by this thesis and the solution proposed. When the players are evaluating what to say then they use game theory to model the problem of finding an optimal sentence to speak. Then they use game theory to find an equilibrium, that is to find the sentence that gives them the highest gain.

6.1 Preliminaries

As previously mentioned in Section 2 I will use the terminology found in (Fudenberg & Tirole, 1991) to describe game theory, which I now introduce by means of an example.

Example 2 (Aumann's example) *Here player 1 and 2 want to find an optimal strategy in a game where they both have identical payoffs.*

		Player 2	
		L	*R*
Player 1	*U*	*5,1*	*0,0*
	D	*4,4*	*1,5*

Each game in Game Theory has at least a set of players, $i \in \mathcal{I}$, and a pure strategy space S_i, for each player $i \in \mathcal{I}$. In example 2 there are two players and two strategy spaces each with two pure strategies: $S_1 = \{U, D\}$ and $S_2 = \{L, R\}$. Strategy space S_i is the set of actions available to player i.

So player 1 can play pure strategy U or D and player 2 can play pure strategy L or R. If Player 1 plays U then player 2 may possibly play L. This will give player 1 a payoff (also known as utility) of 5, denoted $u_1(U, L)$, and player 2 a payoff of 1, denoted $u_2(U, L)$.

Each game has some *equilibrium point* which is the point that has a higher or equal payoff to any other point in the game. Such an equilibrium point is defined by the pure strategies and the probability that they are played. A pure strategy that has 0 probability of being played is not included in the definition of a point. This means that the marginal of the probability of any strategy defining a point is strictly positive. This simple fact is important when defining the Bayesian equilibrium in Subsection 6.3. A proper definition of equilibrium is given in Subsection 6.2 and Subsection 6.3

Each player has a payoff function for each of the strategy profiles that he can play. There are two types of strategy profiles: a *pure strategy* profile and a *mixed strategy* profile. The pure strategy profile has a payoff function u_i which is the utility $u_i(s)$ for each profile $s = (s_1, ..., s_I)$ of strategies and is dependent on the probability distribution of the opponents' strategies. Because the equilibrium in example 2 is to play either pure strategy with equal probability then the payoff functions are:

$$u_1(U) = 0.5 * 5 + 0.5 * 0 = 2.5 \, and \, u_1(D) = 0.5 * 4 + 0.5 * 1 = 2.5$$

$$u_2(L) = 0.5 * 1 + 0.5 * 4 = 2.5 \, and \, u_2(R) = 0.5 * 0 + 0.5 * 5 = 2.5$$

A player can also have a set of *mixed strategies*. A mixed strategy for player $i\sigma_i$ is a probability distribution over the set of pure strategies to which σ_i assigns positive probability. I write $\sigma_i(s_i)$ for the probability that σ_i assigns to s_i, for the profile $s = (s_1, ..., s_I)$ of pure strategies. Then i's payoff for profile $\sigma = (\sigma_1, ..., \sigma_I)$, using the payoff function $u_i(s)$ for each pure strategy, is the weighted sum over all profiles $s \in S = S_1 \times S_2 \times \cdots \times S_n$

$$u_i(\sigma) = \sum_{s \in S} \left(\prod_{j=1}^{I} \sigma_j(s_j) \right) u_i(s) \tag{6.1}$$

It is best to understand this equation with an example, so we will calculate a mixed strategy for example 2 for Player 1. We assign probability 0.5 to each of player 1's pure strategies because I now that the optimal strategy is playing either strategy with the probability 0.5. So player 1's mixed strategy σ_1 is a vector (σ_1(U), σ_1(D)). The payoffs for profiles σ_1 = (0.5,0.5) and σ_2 = (0.5,0.5) using equation 6.1 are

$$u_1(\sigma_1, \sigma_2) = \quad 0.5(0.5 * 5 + 0.5 * 0) + 0.5(0.5 * 4 + 0.5 * 1)$$

$$\begin{aligned} &= 0.5(u_1(U)) + 0.5(u_1(D)) \\ &= 2.5 \end{aligned}$$

Similarly for player 2

$$u_2(\sigma_1, \sigma_2) = 2.5$$

An *opponent* in Game Theory is a common denominator for the other players, he can be a competitor but is not generally viewed in such a light. The opponent is denoted $-i \in \mathcal{I}$ where $-i \neq i$.

Rationality in Game Theory is when a player chooses the strategy that will give him the optimal payoff no matter what his opponents play, given the information he has at the time. Every player is considered to be rational and thus to act in a rational manner at all times. If the player does not act rationally then he is considered to be making a mistake (for example, that he pushes a wrong button because his hands are trembling).

There are various levels of information that a player can have in any game. *Completely informed* player is someone who knows all the possible payoffs and everyone knows that he is completely informed and he knows that everyone knows that he is completely informed and so on. A game is considered a *completely informed game* if all the players are completely informed. If a player lacks information on his opponents' payoffs then he is considered to be *incompletely informed.* If a player that is incompletely informed can by calculation estimate the payoffs of his opponents then he is considered *imperfectly informed.* This is commonly done when finding Bayesian equilibria. The game that this thesis deals with is an incomplete information game and a large part of the problem domain is to convert this incomplete information game into an imperfectly informed game. The success of that enterprise is essential for solving the problem dilemma of the thesis.

Non cooperative game is when the players have no prior binding agreements. This holds true for most real world scenarios that the players in question do not have priorly signed contracts with their opponents. This means that although some opponents may value some sort of cooperation, it is not organized and the player can't depend on it. The game that this thesis deals with is a non cooperate game. The players may consider that cooperating with others is beneficial but there will never be a binding agreement.

6.2 Nash equilibrium

A Nash equilibrium is a profile of strategies such that each player's strategy is an optimal response to each of the other players strategies, e.g. there is no incentive to deviate.

Definition 1 (Nash Equilibrium) *(Fudenberg & Tirole, 1991).*

A mixed-strategy profile σ^* is a Nash equilibrium if for all players i, and for all strategies $s_i \in S_i$,

$$u_i(\sigma_i^*, \sigma_{-i}^*) \geq u_i(s_i, \sigma_{-i}^*) \tag{6.2}$$

Nash equilibrium holds for the *extended* game. Meaning that Nash equilibrium holds true when playing repeatedly against the same set of players, as proven by J. Nash (Nash, 1951). It is important to realize that because of this it is sufficient to show that an equilibrium is equivalent to Nash equilibrium to show that it to will hold for the *extended* game.

There are three Nash equilibriums in example 2. Two pure strategies (U,L) and (D,R) and one mixed strategy ((0.5*U,0.5*D),(0.5*L,0.5*R)). Nash equilibrium is essentially a maximization of the utilities. By this I mean that to find nash equilibrium you need to find the set of strategies that has a higher or equal payoff to any other strategy.

Example 3 (Finding Nash Equilibrium) *In this example we will find the pure strategy equilibrium (U,L).*

- *First, $u_1(U, L) = 5$, this is the highest so far for player 1, so lets ad it a set $\{(U, L)\}$ for player 1.*
- *Second, $u_1(U, R) = 0$, this is $<$ than $u_1(U, L)$ so it is not added to player 1 set.*
- *Third, $u_2(U, L) = 1$, this is highest so far for player 2, so lets ad it a set $\{(U, L)\}$ for player 2.*
- *Finally, $u_2(U, R) = 0$, this is $<$ than $u_2(U, L)$ so it is not added to player 2 set.*

Now we have effectively found a pure strategy equilibrium point, it is in the intersection of the two players sets $\{(U, L)\}$.

From example 3 it can be seen that it is essential for finding equilibrium to find what would be the opponents optimal moves. In this example it is easy because both players happen to have identical payoffs. If player 2 would for instance get $u_2(U, R) = 2$ instead of $u_2(U, R) = 0$ then (U,L) would not be an equilibrium because playing L is then not an optimal strategy for player 2.

6.3 Bayesian equilibrium

The traditional representation of game theory (Fudenberg & Tirole, 1991) is strictly mathematical, exact and explicit. The resulting high abstraction does not have a very transparent connection with real world scenarios. It is a non-trivial endeavor to model even the simplest of human interactions using game theory. Harsanyi discusses this point in great detail in (Harsanyi, n.d.-c), (Harsanyi, n.d.-a) and (Harsanyi, n.d.-b). He proposes the Bayesian equilibrium as a way of calculating Nash equilibrium for incomplete information games, where the agents lack complete information about the game world. This technique is well described by Fudenberg and Tirole (Fudenberg & Tirole, 1991) and Harsanyi (Harsanyi, 1995) where the type centric approach is especially well discussed. In J. C. Harsanyi own words (Harsanyi, n.d.-c) p:163:

> In our own view it has been a major analytical deficiency of existing game theory that it has been almost completely restricted to complete information games, in spite of the fact that in many real-life economic, political, military, and other social situations the participants often lack full information about some important aspects of the “game" they are playing.

Let us first consider the dynamics of an incomplete information game, which is a game where some players lack full information of the game. We will assume that each player knows his own payoff function and state to keep the complication within reasonable bounds and we will only look at two player games. Imagine, for instance, two warring factions; they would know very well their own state but not the state of their opponent. Still they will not be completely ignorant. They will know what states the opponent could possibly be in because for all practical purposes one can assume that there are only finitely many states for the opponent to be in. Examples of such states are that the opponent has a secret weapon, has depleted its sources of food in an area, has powerful tanks etc. Moreover each warring faction will have an intelligence service that informs it of what states the opponent most probably is in.

In Game Theory terms this means that the players don’t know the opponents’ current payoff function but know all payoff functions that the opponent could be using. The latter is referred to as *exogenous data*, which means external data that is not derived from the players status at any given time but is a factor in the equilibrium-calculations. Examples of exogenous data are strategy spaces, pay off functions, types, and prior distributions. This is considered common knowledge among the players. Everybody knows them and knows that everyone knows them and so on. The $exogenous$ data is so named because it stands outside the influence of the equilibrium calculations, meaning that it will stay the same through out the game.

Let us first draw up what the players will need to do to get an equilibrium. Each player's *first order* expectations will be a subjective probability distribution $p_i^1(u_{-i})$ over all possible payoff functions u_{-i} that the opponent could have. Then each player will have a *second order* expectations of $p_i^2(p_{-i}^1)$ over all the first order expectations that the opponent will have and so on ad infinitum. In general terms, the kth-order expectations of player i will be a subjective probability distribution $p_i^k(p_{-i}^{k-1})$ over all alternative $(k-1)$th-order subjective probability distributions p_{-i}^{k-1} that the opponent may have (Harsanyi, n.d.-c). The game just described is a *player-centered* interpretation. The problem with this *player-centered* interpretation is that it contains *infinite recursion* and that one needs fixed point configuration in order to calculate equilibria.

There is also a *type-centered* interpretation and the two interpretations are always equivalent from a game theoretic point of view (Harsanyi, 1995). Meaning that they both find an optimal strategy for the player in an incomplete non cooperative game. The *type-centered* approach does not contain an *infinite recursion* in the same way as the *player-centered* approach. Harsanyi argues that the type of a player can represent the character of a player and his beliefs of other player's strategies, including their payoff functions. Assume that there are Θ different possible types that represent every possible makeup of the player. The type is any prior information that is not common knowledge and that may affect the player's payoff function, their decision making and their belief about other player's strategies and beliefs and so on. Some players may be better informed than others. Each player is considered to have a finite set of possible types Θ_i with an objective prior probability distribution p. I write $p(\theta_i)$ for the probability that player i has type θ_i.

In the previous example of warring nations, types could be for one of the warring faction θ_1 = "little food, has secret weapon" and then when food convoys have been opened up θ_1 = "enough food, has secret weapon". Moreover each player has a conditional probability about his opponents types $p(\theta_{-i}|\theta_i)$ where $\theta_{-i} = (\theta_1, ..., \theta_{i-1}, \theta_{i+1}, ..., \theta_I)$. It can also be assumed that the marginal, that is the lowest value of $p_i(\theta_i)$, on each type $\theta_i \in \Theta_i$ is strictly positive. Let $\sigma_i(\theta_i)$ be the possible mixed strategy that player i chooses when her type is θ_i, as each player can be assumed to choose her strategy dependent on her type. A player i may be able to compute his expected payoffs by using his beliefs $p(\theta_{-i}|\theta_i)$ if he knows the strategies $\{\sigma_i(\cdot)\}_{j\neq i}$ of the other player's as a function of their type. This would be the intelligence that the warring factions we described above would have acquired about its enemies.

The $exogenous$ data of the game is then the Θ, the set of all possible types, this includes all the possible payoff functions contingent to the types. Bayesian equilibrium then uses prior calculations commonly referred to as *ex ante* calculations. These *ex ante* calculations use Bayes rule to calculate the probability of each possible $type\ \theta \in \Theta$ that the opponents could be of, hence the name $Bayesian$ equilibrium. The reason for these fancy names as $exogenous$ data and *ex ante* calculations is to define tools that can be used as a means of valuating incomplete information from the world to use in the explicit calculations of game theory.

It is important to realize that since Bayesian equilibrium, like Nash equilibrium, is essentially a consistency check, players' beliefs about others' beliefs do not enter the definition – all that matters is each player's own beliefs about the distribution of types and his opponents' type-contingent strategies (Fudenberg & Tirole, 1991). A Bayesian equilibrium is a Nash equilibrium of the "expanded game" where each player $i \in \mathcal{I}$ has a space of pure strategies which is the set $S_i^{\Theta_i}$ of maps from the set of types for each agent Θ_i to S_i and a utility function that is contingent to the agent's types each time (Fudenberg & Tirole, 1991).

Definition 2 (Strategy profile) *(Fudenberg & Tirole, 1991)*

Given a strategy profile $s(\cdot)$, and an $s_i'(\cdot) \in S_i^{\Theta_i}$, let $(s_i'(\cdot), s_{-i}(\cdot))$ denote the profile where player i plays $s_i'(\cdot)$ and the other players, his opponents, play $s(\cdot)$, we let

$$(s_i'(\theta_i), s_{-i}(\theta_{-i})) = (s_1(\theta_1), ..., s_{i-1}(\theta_{i-1}), s_i'(\theta_i), s_{i+1}(\theta_{i+1}), ..., s_i(\theta_i))$$

- The probability of an opponent being of any given type $p_{-i}(\theta_{-i})$ is *strictly positive*.
- We use *ex ante* calculations to find the probability that the opponent is of a given type $p_{-i}(\theta_{-i})$.
- We use *ex ante* calculations to find the opponent's strategy for each of his possible types $s_{-i}(\theta_{-i})$.

Definition 3 (Bayesian Equilibrium) *(Fudenberg & Tirole, 1991)*

We find an equilibrium by maximizing the sum of player i's weighted payoff, conditional to his type θ_i, over the opponents' possible types θ_{-i}, for each of his strategies $s_i' \in S_i$ as is shown below.

$$s_i(\theta_i) \in \arg\max_{s'_i \in S_i} \sum_{\theta_{-i}} p(\theta_{-i}|\theta_i) u_i(s'_i, s_{-i}(\theta_{-i}), (\theta_i, \theta_{-i})) \tag{6.3}$$

Giving a small example is difficult. In Section 10.4 the algorithm for finding Bayesian equilibrium is described in detail and each step is explained in respect to its role in the Bayesian equilibrium.

Chapter 7

Bayesian networks

In this thesis I use Bayesian networks (Jensen, 2001) for three things. First the dynamic plot generating engine uses a Bayesian net to structure and determine the plot. Second the players use a Bayesian net to structure their knowledge base, this enables them to be able to add to the knowledge base and immediately refer new knowledge from the new addition. Finally Bayesian nets are combined with game theory to find an optimal sentence for the player to speak.

The graphical models described in this paper were created using the GeNIe modeling environment (*GeNIe modeling environment for graphical probabilistic model*, 2007). I use three types of variables, they are:

- Decision variables, $D \in \mathcal{D}$, represented with a square in the diagrams;
- Utility variables, $U \in \mathcal{U}$, represented with a diamond in the diagrams. Utility variables can only have parents, not children;
- Chance variables, $X \in \mathcal{X}$, represented with an oval in the diagrams;

Both decision and chance variables have a finite set of states that are mutually exclusive. Each variable X has a finite set $dom(X)$ of possible values called the domain of the variable. Decision variables are used to sum up the total cost of making a decision. In my prototype engine the decision is what sentence is optimal. The sentences are the states of the decision variable and those sentences that have the highest value are the optimal set of sentences. The utility variable calculates the value of each decision given the decision and the value of each factor that affects it. This is determined by setting values in respect to each of the utilities parents, (see Table 7.3).

There are other types of variables defined for Bayesian networks that I do not make use of, for more detail see (Jensen, 2001). I introduce the model by the following example. The example is an extension of an example that is used in (Jensen, 2001).

Let's assume that in Iceland the general population has the flu about one fourth of the year. The flu increases the probability of a fever and having a fever increases the probability of being sleepy. Moreover taking aspirin is very likely to reduce the fever to normal temperature, 37°C. Let us also assume that the Icelandic government has taken the initiative to distribute free aspirin to every Icelandic home in order to increase the productivity of the small population and that taking aspirin has no side affects.

Figure 7.1: Flu

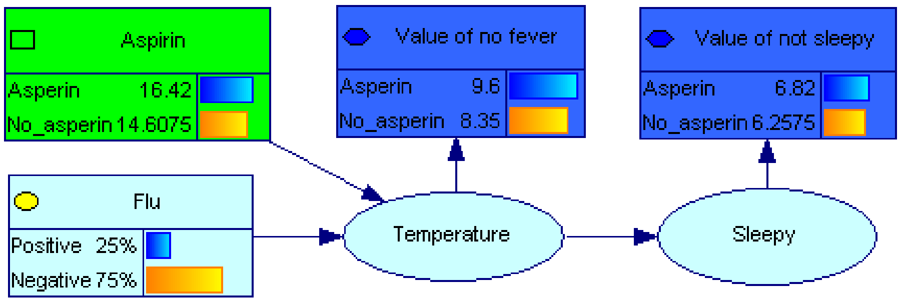


Table 7.1: Temperature

Aspirin	Aspirin		No aspirin	
Flu	Positive	Negative	Positive	Negative
Normal	0.9	0.98	0.4	0.98
Elevated	0.1	0.02	0.6	0.02

Table 7.2: Sleepy

Temperature	Normal	Elevated
Yes	0.3	0.75
No	0.7	0.25

In Figure 7.1 the Bayesian net showing the causal connections from flu to fever to sleepy is shown, it has the following sets of variables:

- $\mathcal{D} = \{Aspirin\}$
- $\mathcal{U} = \{Value\ of\ no\ fever, Value\ of\ not\ sleepy\}$
- $\mathcal{X} = \{Flu, Temperature, Sleepy\}$

Aspirin has two states: $\{Aspirin = to\ take\ Aspirin\}$,
$\{No\_Aspirin = not\ to\ take\ Aspirin\}$.

The flu variable has two states: $\{Positive\}$ with a probability of 0.25 and $\{Negative\}$ with a probability of 0.75.

Table 7.1 shows the table that defines the conditional probabilities of the temperature variable. The $Temperature$ variable is a chance variable and has two states: $\{Normal = 37°C\}$, $\{Elevated > 37°C\}$ and two parents, Flu and Aspirin. Reading Table 7.1 we see that if someone that has the flu takes Aspirin then she will have normal temperature with a probability of 0.9 and if she does not take Aspirin she will have normal temperature with a probability of 0.4. The top row in Table 7.1 shows the states of the decision variable $Aspirin$, the next row shows the states of the Flu variable and the next two rows show the states of the *temperature* variable and the respective conditional probabilities.

Table 7.2 defines the conditional probabilities of the *Sleepy* variable. In Figure 7.1 we can see from the two utility variables that the value of taking Aspirin to reduce fever is 9.6 and the value of taking Aspirin to reduce sleepiness is 6.82. The decision to take Aspirin carries then the gain of 9.6 + 6.82 = 16.42, and to not take Aspirin carries the gain of 8.35 + 6.2575 = 14.6075. The value of the utility variables are defined in respect to its parents just as the chance variables. The difference is that utility variables calculate value and the chance variables calculate probability. Each column in Tables 7.1 and 7.2 ads up to one because each of those columns define how probable the respective state is given the conditions declared in the rows above. Table 7.3 shows the definition of the utility variables.

Table 7.3: Utilities

Value of no fever

Temperature	Elevated	Normal
value	0	10

Value of not sleepy

Sleepy	yes	no
value	0	10

Chance and decision variables can be instantiated; this means that the variable has one of its values set to probability 1. In Figure 7.2 the Flu variable has the value of the state “Positive" set to probability 1. In Figure 7.3 the Flu variable has the value of the state “Negative" set to probability 1.

One reason for instantiating variables is to see the exact values when you assume some fact to be given. In Figure 7.2 the assumption is that the individual has flu and it is then clearly evident that there is higher gain in taking Aspirin since it yields a higher value, 15.55 to 8.3. In Figure 7.3 the assumption is that the individual does not have any flu and then there is no greater gain in taking Aspirin than not taking Aspirin. It is clear in this example that in general the gain in taking Aspirin is higher than in not taking it. It is also clear that if taking Aspirin has any cost or negative affects then that would affect the calculations.

This example is of course naive and has little correlation to reality but it shows how causalities can be drawn out and how the nets are read.

Figure 7.2: Instantiated to positive

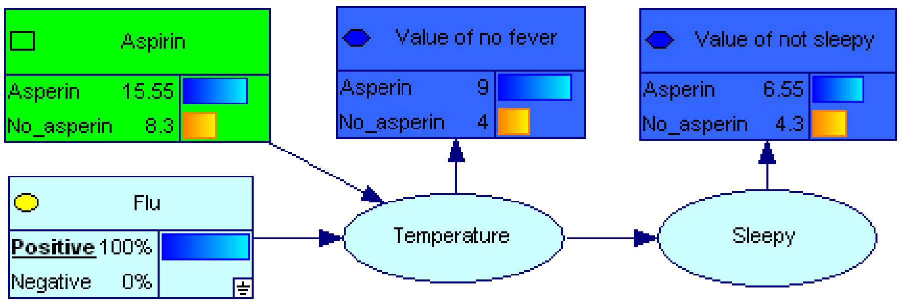
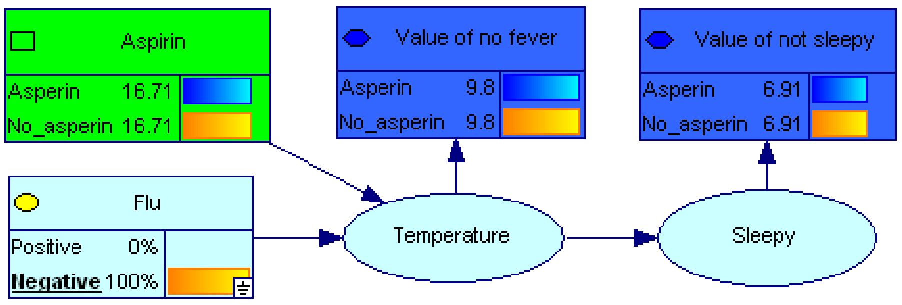


Figure 7.3: Instantiated to negative



7.1 Splitting and aggregating

The Tables that define the variables in the Bayesian nets have an exponential growth in respect to the number of parents the variable has. Looking again at the flu example, there can be any number of diseases besides flu that cause fever. If there are for instance 20 possible diseases, each with the two states $\{positive, negative\}$, then it results in a table with $2 * 2 * 2^{20} = 4,194,304$ entries. If n is the number of disease variables and noe is the number of entries and $s(X)$ is the number of states for a variable X then:

$$noe = s(Temperature) * s(Aspirin) * s(diseases)^n$$

There is a way to combat the exponential growth; it is done by splitting up the variables into many smaller variables. This splitting of variables was not known to me, I simply realized that it would be practical and necessary when I started to work with the Bayesian nets. This is not a new discovery though it is a well known technique (Jensen, 2006), although I only discovered that after having developed the splitting algorithm.

We can decide that the Temperature variable never has more than 8 disease variables as parents. This results in the following equation

$$noe = \begin{cases} n \; mod \; 8 = 0 & \Longrightarrow \quad n/8(2*2*2^8) \\ n \; mod \; 8 > 0 & \Longrightarrow \quad n/8(2*2*2^8) + (2*2*2^{n \; mod \; 8}) \end{cases}$$

If the number of disease variables is 20 then this results in 3 variables and 2112 entries. When the splitting of the variable is complete it is necessary to aggregate the result into one variable to get the overall probability of the individual having fever, see Figure 7.4.

Figure 7.4: Fever

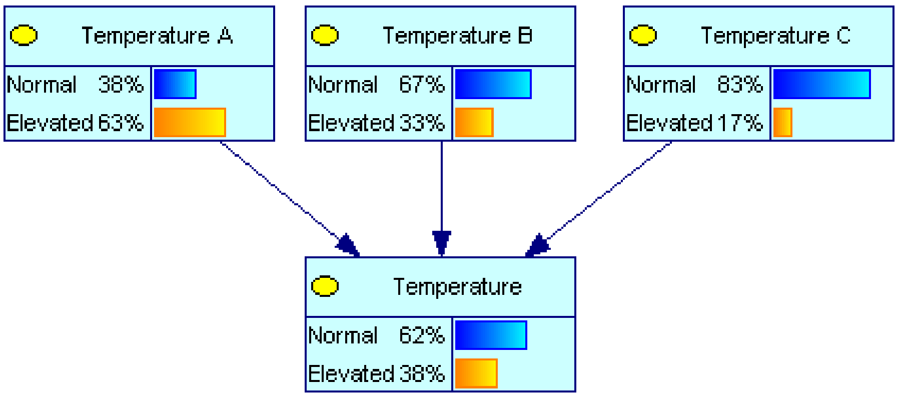


Table 7.4: Fever

Temp A	Normal				Elevated			
Temp B	Normal		Elevated		Normal		Elevated	
Temp C	Normal	Elevated	Normal	Elevated	Normal	Elevated	Normal	Elevated
Normal	3	2	2	1	2	1	1	0
Elevated	0	1	1	2	1	2	2	3

Table 7.5: Temperature normalized

Temp A	Normal				Elevated			
Temp B	Normal		Elevated		Normal		Elevated	
Temp C	Normal	Elevated	Normal	Elevated	Normal	Elevated	Normal	Elevated
Normal	1	0.666667	0.666667	0.333333	0.666667	0.333333	0.333333	0
Elevated	0	0.333333	0.333333	0.666667	0.333333	0.666667	0.666667	1

To define the Temperature variable in Figure 7.4 it is first necessary to aggregate the states as is shown in Table 7.4 and then to normalize the column as has been done in Table 7.5 where the values of every column add up to 1.

The result of splitting up the variables and then aggregating into one variable is in this example $20/8(2 * 2 * 2^8) + (2 * 2 * 2^{20\%8}) + (2 * 2^3) = 2,128$ entries, which is 0.05% of the 4,194,304 entries and the growth is no longer exponential.

I wrote a general algorithm that aggregates relations in this way, see algorithm 5. This is in a class Bayes that extends the Network class of the Smile library (DSL, 2007b).

First the algorithm decides whether to split or not based on the number of parents of a variable. The input to the algorithm is the variable id, an *array of parents* and *maximum number of parents*. If number of parents exceed the limit, set by max number of parent then the node is split and aggregated.

Algorithm 1: Define and split variable

```
Input  : String id, String[] parents, int maxParents
1  begin
2      # If there are to many parents then split and aggregate.
3      if parents.length > maxParents then
4          variables = splitVariable( id, maxParents, parents )
5          for each ∈ variables do aggregateEqualValues( each )
6      else
7          # else just add parents and aggregate.
8          for each ∈ parents do addArc( each, id )
9          aggregateEqualValues( id )
10 end
```

If the decision is to split the node then algorithm 2 handles the actual splitting. The Algorithm starts by creating a set of all the variables it needs and then connecting them and finally aggregating the values. First it checks, by modulus, whether there is need for a node for leftovers see line 4. If there is then that variable needs to be added to the variable array at the back just before the main variable that everything will then be aggregated to. Next the algorithm adds all the variables it will need to the array see line 9. Then, starting in line 15, all the parents are added to the variables. For each variable the algorithm first checks in line 16 whether this is a variable that will get maximum number of parents added to it. If the number of the variable is less than the division of number of parents by maximum number of parents then it should have a full set of parents, line 17. On the other hand if the number of the current variable is equal to the same division as before, see line 21, then this variable is the second last variable in the array and should have the rest of the parents added to it. The number of parents that the second last variable gets, when it applies, is equal to the number of parents modulus maximum number of parents, line 22. Finally for each variable add it to the main variable, line 26, as long as it is not the main variable.

Algorithm 2: Split variable

```
   Input  : String id, int max, String[] parents, String[] states, String name, String
            submodel, coordinates: x, y
   Output: String[] variables
 1 begin
 2     div = parents.length / max
 3     mod = parents.length mod max
 4     if mod > 0 then
 5         #add an extra variable when needed
 6         addChanceNode(newID, name, states )
 7         variables[ div, div+1 ] = newID, id
 8     else variables[div] = id
 9     for int i = 0; i < div; ++i do
10         #add all the new variables needed
11         addChanceNode( newID, name, states )
12         variables[i] = newID
13     end
14     count = 0
15     for int each = 0; each < variables.length; ++each do
16         if each < div then
17             for int i = 0; i < max; ++i do
18                 addArc( parents[count], variables[each] )
19                 ++count
20             end
21         else if each = div then
22             for int i = 0; i < mod; ++i do
23                 addArc( parents[count], variables[each] )
24                 ++count
25             end
26         if variables[each] != id then addArc( variables[each], id )
27     end
28     return variables
29 end
```

The last step of the split and aggregate algorithm is to aggregate equal values in all the variables. Algorithm 3 takes the id of a variable as input and aggregates the values. It first queries the number of parents and states and the size of the Table and then calls a recursive algorithm 4 to set the correct values in the value array before setting the value array as the defined set of entries for the variable.

Algorithm 3: Aggregate equal values

```
   Input  : String id
 1 begin
 2     size = getParentIds( id ).length
 3     states = getOutcomeIds( id )
 4     d = new double[tableSize(id)]
 5     aggregateEqualValues( size, d, 0, new String[size], states )
 6     setNodeDefinition( id, d )
 7 end
```

In order to make the aggregation fully generic then the actual aggregation algorithm is a recursive function. The algorithm is initialized with the number of parents as pow and an empty array $values$ for the changeable states that the parents have. Moreover an *array for the defining values*, a *counter* that tells the current position and the *states* of the variable.

Algorithm 4: Aggregate equal values

```
   Input  : int pow # initially number of parents, double[] d, int count, String[] values #
            initially empty, String[] states
   Output: int count
 1 begin
 2     if pow = 0 then
 3         for state ∈ states do
 4             for value ∈ values do
 5                 #add 1 when contradiction values match
 6                 if value = state then ++d[count]
 7             end
 8             ++count
 9         end
10         normalize( d, count, states.length )
11     else
12         idx = values.length-pow
13         - - pow
14         for each ∈ states do
15             values[idx] = each
16             count = aggregateEqualValues( pow, d, count, values, states )
17         end
18     return count
19 end
```

First in the $else$, line 11, of the main if function the pow is decremented until it reaches 0. For each step of the count down of pow a for loop is activated that runs through all the parents states and sets each of them in the initially empty $values$ array at the current level according to pow, see line 15. The recursive call rests inside the for loop so that for every state of the parent and for every parent the recursive function is called and when the bottom level is reached, $pow = 0$ in line 2, then the second face of the algorithm is entered, that of setting the aggregated value in its place. Then for each of the variables state and for each of the parents value, if they are equal then increment, see line 6. Finally call a normalize function that simply adds up all the values in the current column and weighs each value by dividing it with the total so that the column as a whole sums up to 1.

7.2 Relational aggregation

It is sometimes necessary to aggregate relations between two different parts of the network. As an example, when calculating the level of risk the player takes from lying as is described in Section 9.3, it is necessary to find the correlation between each *reply* or *sentence* and the aggregation of each *Risk* variable, see Figure 7.7. Each of the *Risk* variables are aggregating the risk of each sentence or reply depending on whether this is the reply or sentence setup. This is shown in example 4

Example 4 (Relational aggregation) *Let's assume that the domains of the* Reply *variable and the two* Risk *variables is a result of an aggregation from the net and are as follows:*

- *Dom(Reply)* = $\{10\%, 90\%\}$.
- *Dom(Risk 1)* = $\{40\%, 60\%\}$.
- *Dom(Risk 2)* = $\{75\%, 25\%\}$.

The trick here is to get the conditional probability in respect to each reply by setting 1 in each entry that corresponds to a Reply *and the respective* Risk *variable. Then the conditional probability of the* Reply *and the* Risk *variables has been joined in a single variable.*

Table 7.6: Relational aggregation

Reply	Reply_1				Reply_2			
Risk 1	True		False		True		False	
Risk 2	True	False	True	False	True	False	True	False
True	1	1	0	0	1	0	1	0
False	0	0	1	1	0	1	0	1

In Table 7.6 we see that Reply_1 *and* Risk 1 *are joined by setting 1 in the entry corresponding to the states of the* Risk 1 *variable when the* Reply *variable is in state* Reply_1.

The same is done for the Risk 2 *variable when reply is in state* Reply_2. *In column 1 state* true *is set to probability 1 because* Risk 1 *is true and* Reply_1 *is the active state. In column 6 state* false *is set to probability 1 because* Risk 2 *is false and the active state is* Reply_2.

It can be seen in Figure 7.5 and 7.6 that when the reply variable is instantiated, to *Reply_1* or *Reply_2* respectively, then the respective conditional probability is perfectly matched in the *Risk* variable. In Figure 7.7 we further see that the resulting variable is effectively mapping the conditional probability of the replies. The value for *Reply_2* is 90% so the result is much closer to *Risk_2* than *Risk_1*.

Figure 7.5: Reply 1

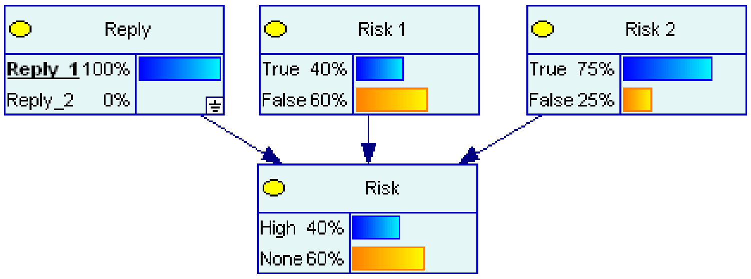


Figure 7.6: Reply 2

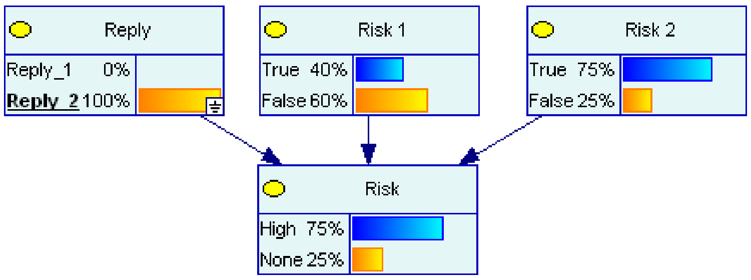
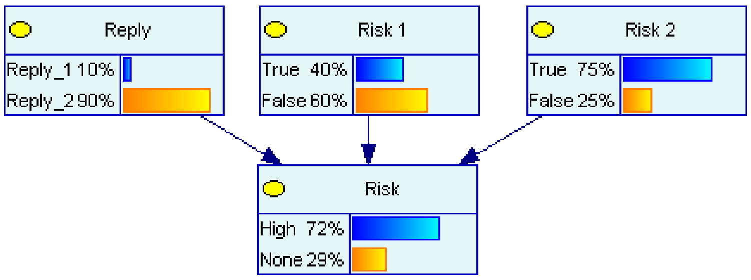


Figure 7.7: Relations



I wrote a general algorithm that aggregates relations in this way, see algorithm 5. This is again in a class Bayes that extends the Network class of the Smile library (DSL, 2007b). Algorithm 5 needs the id string of the variable that will handle the relational aggregation as input, e.g. the *Risk* variable in example 4, and aggregates the values in respect to their relations to the deciding variable, e.g. the $reply$ variable is a deciding variable in example 4. It first queries the number of parents and states and the size of the table and then calls a recursive Algorithm 6, inside a for loop that loops for the number of states in the deciding variable. Algorithm 6 sets the correct values in the value array before setting the value array as the defined set of entries for the variable.

Algorithm 5: Aggregate relations

```
   Input: String id
 1 begin
 2     parents = getParentIds( id )
 3     decisionLength = getOutcomeIds( parents[0] ).length
 4     parentStates = getOutcomeIds( parents[1] )
 5     states = getOutcomeIds( id )
 6     d = new double[ tableSize( id ) ]
 7     count = 0
 8     for int i = 0; i < decisionLength; ++i do
 9         count = aggregateRelations( decisionLength, d, count, new String[
           decisionLength ], i, parentStates, states )
10     end
11     setNodeDefinition( id, d )
12 end
```

In order to make the aggregation fully generic then the actual aggregation algorithm is a recursive function. The algorithm is initialized with the number of parents as pow and an empty array $values$ for the changeable states that the parents have. Moreover an *array for the defining values*, a *counter* that tells the current position, the *sentences*, *parents* and the *states* of the variable.

First in the $else$, line 9, of the main if function the pow is decremented until it reaches 0. For each step of the count down of pow a for loop is activated that runs through all the parents states and sets each of them in the initially empty $values$ array at the current level according to pow see line 13. The recursive call rests inside the for loop so that for every state of the parent and for every parent the recursive function is called and when the bottom level is reached, *pow = 0* in line 2, then the second face of the algorithm is entered, that of setting the aggregated value in it's place in respect to the state it should represent. In the second stage then for each of the variables state and for the current parents state, if they are equal then set that entry of that sentence to 1, see line 5.

Algorithm 6: Aggregate relations

```
Input  : int pow, double[] d, int count, String[] values, int sentence, String[] parents,
         String[] states
Output: int count
begin
    if pow = 0 then
        for state ∈ states do
            # set 1 when sentence and contradiction value match
            if values[sentence] = state then d[count] = 1
            ++count
        end
    end
    else
        idx = values.length - pow
        - - pow
        for parent ∈ parents do
            values[idx] = parent
            count = aggregateRelations( pow, d, count, values, sentence, parents, states )
        end
    end
    return count
end
```

Chapter 8

MAID (Multi-Agent Influence Diagram)

Koller and Milch (Koller & Milch, 2003) propose a very good way to tackle the same problem as discussed in subsection 6.2 namely dealing with the incomplete information of every day decision making. They propose a multi-agent influence diagram (MAID). This MAID is very useful to the prototype engine that I built because it enables the character to evaluate the sentences in respect to what he expects his opponent to reply. Moreover the complexity of this approach is linear. A MAID is an extended Bayesian net and is defined using the following:

- $\mathcal{I}$ is the set of agents.
- $\mathcal{D}$ is the set of decision variables. $\mathcal{D}_i$ is the set of decisions variables for each agent $i \in \mathcal{I}$.
- $\mathcal{U}$ is the set of utility variables. $\mathcal{U}_i$ is the set of utility variables for each agent $i \in \mathcal{I}$.
- $\mathcal{X}$ is the set of chance variables.
- Each variable X has a finite set $dom(X)$ of possible values, called domain.
- $\mathcal{V}$ is the complete set of variables $\mathcal{D} \cup \mathcal{U} \cup \mathcal{X}$.
- $Pa(X) \subset \mathcal{D} \cup \mathcal{X}$ is the parents of each $X \in \mathcal{X}$.
- for X, Y $\in \mathcal{V}$ there is an edge X $\rightarrow$ Y iff X $\in$ Pa(Y).
- $Pa(D)$ is the set of variables that the agent i knows when choosing a value for D for each $D \in \mathcal{D}_i$.

- (CPD) is the *conditional probability distribution* of each $X \in \mathcal{U} \cup \mathcal{X}$: a distribution $Pr(X|pa)$ for each instantiation pa of $Pa(X)$.
- for all D $\in \mathcal{D}$ The CPD for D is σ(D).
- $U(pa)$ denotes the value of variable U that has probability 1 when $Pa(U) = pa$ for $pa \in dom(Pa(U))$.
- A *decision rule* to a decision variable D is a function that maps each instantiation of pa of $Pa(D)$ to a probability distribution over $dom(D)$.
- An assignment of decision rules to every decision $D \in \mathcal{D}_i$ for a particular agent $i \in \mathcal{I}$ is called a *strategy*.
- An assignment σ of decision rules to every decision $D \in \mathcal{D}$ is called a *strategy profile*.
- A *partial strategy profile* σ_ε is an assignment of decision rules to a subset $\varepsilon \subset \mathcal{D}$.
- $\sigma_{-\varepsilon}$ is the restriction of σ to those decision rules not in ε.

Now we can define a utility function to calculate the payoff for each $i \in \mathcal{I}$ for any assignment of decision rules.

Definition 4 (Utility for player $i \in \mathcal{I}$) *(Koller & Milch, 2003)*

If $\mathcal{M}$ is a MAID and σ is a strategy profile for $\mathcal{M}$, then the joint distribution over $\mathcal{V}$ defined by the Bayesian net is $P_{\mathcal{M}[\sigma]}$, denoting the joint distribution for $\mathcal{M}$ induced by σ. Suppose that $\mathcal{U} = \{U_1, ..., U_m\}$.

$$EU_a(\sigma) = \sum_{U \in \mathcal{U}_a} \sum_{u \in dom(U)} P_{\mathcal{M}[\sigma]}(U = u) \cdot u \tag{8.1}$$

The utility function can now be used to reach an optimized decision.

Definition 5 (Optimize for player $i \in \mathcal{I}$) *(Koller & Milch, 2003).*

We say that σ^*_ε is optimal for the strategy profile σ if, in the induced MAID $\mathcal{M}\,[\sigma_{-\varepsilon}]$, where the only remaining decisions are in ε, the strategy σ^*_ε is optimal, i.e for all strategies $\sigma^{'}_\varepsilon$:

$$EU_a((\sigma_{-\varepsilon}, \sigma^*_\varepsilon)) \geq EU_a((\sigma_{-\varepsilon}, \sigma^{'}_\varepsilon)) \tag{8.2}$$

Now that the formal definitions are over it is good to put them into a slightly more general and less technical light. A MAID is essentially a Bayesian net with two or more decision variables D where one decision variable needs to consider the *decision rule* of some other decision variable D' in order to optimize its own decision. This is called *strategic* *relevance*. Such a net is mapping the decision of a player in respect to the decisions of other players in a *non cooperative* game. Having defined that there is strategic relevance it becomes possible to map the strategic relevance with a directed graph called a *relevance graph*. It has already been defined that σ_ε is a partial strategy profile where ε is a subset $\subset \mathcal{D}$. This partial strategy is used to induce a new MAID $\mathcal{M}_{[\sigma_\varepsilon]}$ where each Decision variable $D \in \varepsilon$ is a chance variable. This means that the only decision variables in the induced MAID $\mathcal{M}_{[\sigma_\varepsilon]}$ are decision variables $D \in -\varepsilon$ that is decision variables that are not in $\notin \varepsilon$. The restriction of variables $\notin \varepsilon$ is $-\varepsilon$. A MAID $\mathcal{M}_{[\sigma_\varepsilon]}$ such as just described is created for every decision in the relevance graph. The relevance graph is then used to evaluate each MAID $\mathcal{M}_{[\sigma_\varepsilon]}$ in a topological order such that the MAID $\mathcal{M}_{[\sigma_\varepsilon]}$ that is first evaluated is the MAID $\mathcal{M}_{[\sigma_\varepsilon]}$ that does not depend on any other MAID $\mathcal{M}_{[\sigma_\varepsilon]}$. The results of an evaluated MAID $\mathcal{M}'_{[\sigma_\varepsilon]}$ are then inserted into the chance variables that correspond to the decision variables $D \in \varepsilon$ that the MAID $\mathcal{M}_{[\sigma_\varepsilon]}$ has and are decision variables D in the $\mathcal{M}'_{[\sigma_\varepsilon]}$. This holds true for a *single agent decision problem*.

This is better explained by the means of an example, such an example is given in section 10 where this method is used to find an optimal sentence for a player to speak.

Chapter 9

NPC Alias Player

In this section the general makeup of the player is described. For clarity it is good to look at two figures, first Figure 9.1 that shows the sentence setup for the player and then Figure 9.2 to see how the player envisions his opponents reasoning process in deciding what would be the optimal reply

Figure 9.1: Overview of the sentence setup

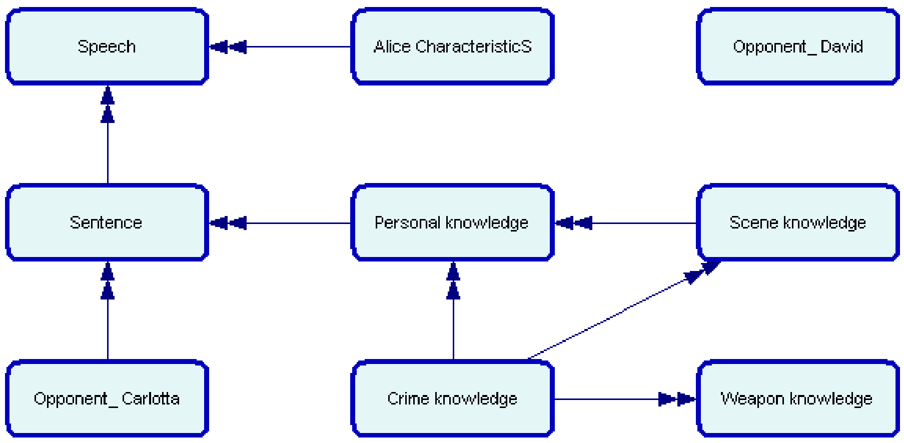


Figure 9.1 shows the sentence setup for the player. There are 9 subnets in the Figure. The player is speaking to Carlotta and not David which means that the *opponent_David* subnet is not connected to the others. There are two important subnets, the $speech$ and $sentence$ subnet. The sentence subnet contains all the sentences that are currently being evaluated (see subsection 9.3 for details). The speech subnet contains the variables used to calculate the value of each sentence (see subsection 9.4 for details). The player's name is Alice and her characteristics will influence how she calculates an optimal sentence. The subnet *Alice characteristics* is connected directly to the speech subnet so that it can be used in the calculations; this is described in subsection 9.1. In Figure 9.1 Alice is evaluating what

to say about her own relations to the victim which means that she takes into account, amongst others, what she thinks that Charlotta knows or assumes about Alice's relations to the victim, thus the *opponent_Charlotta* subnet is connected to the sentence subnet. Alice uses the assumption that she herself has of what assumptions Charlotta has of the world to evaluate sentences. This is explained in detail in subsection 9.3. There are also four $knowledge$ subnets; these represent the knowledge that Alice has of herself and the world. The knowledge subnets are explained in subsection 9.2.

Figure 9.2: Overview of the reply setup

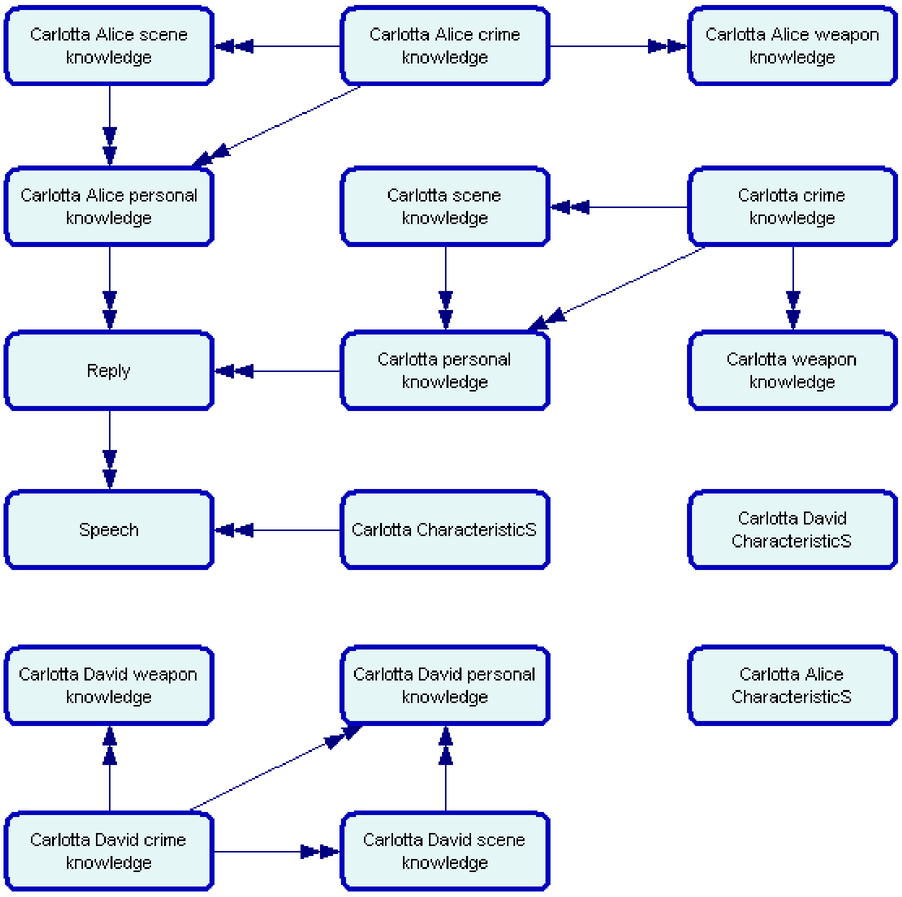

Carlotta Alice scene knowledge
Carlotta Alice crime knowledge
Carlotta Alice weapon knowledge
Carlotta Alice personal knowledge
Carlotta scene knowledge
Carlotta crime knowledge
Reply
Carlotta personal knowledge
Carlotta weapon knowledge
Speech
Carlotta CharacteristicS
Carlotta David CharacteristicS
Carlotta David weapon knowledge
Carlotta David personal knowledge
Carlotta Alice CharacteristicS
Carlotta David crime knowledge
Carlotta David scene knowledge

Figure 9.2 shows the overview of the reply setup which is used to calculate the optimal reply. All the nodes that are connected to the $reply$ or $speech$ subnet in Figure 9.2 influence the calculations used for finding the reply. As in Figure 9.1 Alice is talking to Carlotta and is wondering what Carlotta would reply based on what Alice assumes that Carlotta knows and on what sentence Alice chooses to say. If the speech and reply subnets are excluded then there are 3 sets of subnets in Figure 9.2, They are:

- Subnets that describe what Alice assumes is Carlotta's knowledge and characteristics.

- Subnets that describe what Alice assumes is Carlotta's knowledge of Alice's knowledge and characteristics.
- Subnets that describe what Alice assumes is Carlotta's knowledge of David's knowledge and characteristics.

David is not participating in the conversation and thus there are no connections between subnets starting with *Carlotta David* and the speech and reply subnets. Moreover Alice does not need to use what she assumes Carlotta believes of Alice's characteristics and thus that subnet is also unconnected. As before the characteristics subnets are described in subsection 9.1. The Knowledge subnets are described in subsection 9.2. The Reply subnet is described in subsection 9.3. Finally the speech subnet is described in subsection 9.4.

9.1 Characteristics and emotions

The character subnet consists of all the variables that define the player's characteristics such as: Gullible, Honest, Reliable and Integrity. These are an example of a vast variety of characteristics that might by considered. $Gullibility$ affects how likely the character is to believe hearsay (see Section 9.2). $Honesty$ affects how willing he is to tell the truth; an honest character will put a higher value on telling the truth than others. $Reliability$ reflects how likely he is to be a man of his words; the more reliable he is thought to be the more likely he is to be believed, (see Sections 9.2 and 9.3). Moreover a reliable player does not want to be known to be incorrect. $Integrity$ affects how much he values ethical principles such as honesty and reliability.

As this is a one person project I have needed to pick and choose what to emphasize on. The goal and research question of this project is whether it is possible to create players that can make rational decisions using Bayesian nets and game theory. This means that adding characteristics and emotions to the players' decision model is a secondary goal that I did not have time to develop.

9.2 Knowledge

As can be seen in Figures 9.1 and 9.2 there are four distinct types of knowledge subnets, they are: $personal$ knowledge, $crime$ knowledge, $scene$ knowledge and $weapon$ knowledge. A player has his own set of knowledge subnets that he uses to evaluate the world around him.

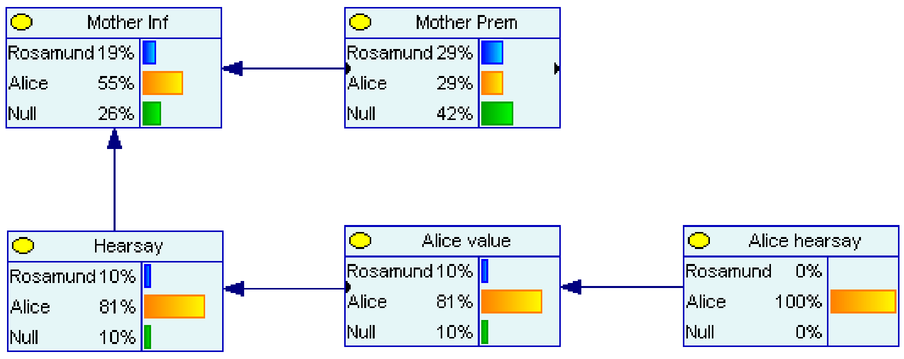
$Personal$ knowledge is everything that has to do with the individual such as hair color, shoe size, name, relations to others etc. The player has a personal knowledge subnet where he maps everything he knows about himself. He also has personal knowledge subnets to map what he believes is true about his opponents. Moreover he has personal knowledge subnets to map what he assumes his opponents know of their opponents.

$Crime$ knowledge is everything to deal with the crime such as the victim, murderer, murder weapon, wound etc. It is divided into three main parts: *murder weapon* knowledge, $murderer$ knowledge, $victim$ knowledge. The player has a crime knowledge subnet where he maps everything he knows about the crime. He also has crime knowledge subnets to map what he believes his opponents know of the crime. Moreover he has crime knowledge subnets to map what he assumes his opponents believe that their opponents know of the crime.

$Scene$ knowledge is everything to do with each scene in the game such as description of a room, the furniture in it and possible clues hidden in each scene. The player has a scene knowledge subnet where he maps everything he knows about the different scenes in the game. He also has scene knowledge subnets to map what he believes his opponents know about different scenes. Moreover he has scene knowledge subnets to map what he assumes his opponents assume their opponents know about the different scenes.

$Weapon$ knowledge is everything to deal with various weapons found in the game and whether the could be the murder weapon. The player has a weapon knowledge subnet where he maps everything he knows about different types of weapons. He also has weapon knowledge subnets to map what he believes his opponents know about weapons. Moreover he has weapon knowledge subnets to map what he assumes his opponents assume that their opponents know about weapons.

Figure 9.3: Knowledge



Each of the subnets that have just been described above are divided into many small subnets. Each of these small subnets contain two or more variables that describe the state of some piece of knowledge. In Figure 9.3 one such subnet is shown. The subnets name is $mother$ and it belongs to the $victim$ knowledge subnet in the crime knowledge subnet. This is the piece of knowledge of who, if any, of the suspects is the victim's mother. Each of the variables have three states, two possible female names and null. The null represents the possibility that none of the suspects is the victim's mother. The two female names are suspects and possibly the murderer. The top right variable *mother prem* is connected to other $...prem$ variables in the net. Each of these knowledge piece subnets have exactly one such variable. This is the $premises$ variable that is drawn from the plot generated at the start. some of these premises variables are instantiated at the start of the game to represent what each player knows in respect to the generated plot. Each player's personal knowledge subnet has for instance all of its premises variable instantiated in respect to the plot because the player will have complete information about himself. This is described better in section 10.4 and 10.5 where the process of finding equilibrium is described. This *mother prem* variable has *wife prem*, *victim prem* and *female adulterer prem* as parent variables and *murderer prem* and *parent prem* as children variables. The wife, victim and adulteress variables are needed as parents to make sure that none of these characters gets also chosen as a mother. The victim's wife cannot be the victim's mother etc. The child nodes check for the murderer and suspect whether they are parents of the victims and thus draw inference from that knowledge.

The top left variable *mother inf* is an inference variable. This variable represents the inferred knowledge of both actual data from the premise variable and information that the player has been told by an opponent. The *Alice hearsay* variable shows what the opponent said. In this example the player was talking to Alice and Alice claims to be the victims mother. The *Alice value* variable shows how much value the player puts on Alice's claims. The value variables are influenced by the players gullibility and by how reliable he believes his opponent to be, see the characteristics section 9.1. The hearsay variable aggregates all hearsay so that the inference is simpler. If the player then talks to other opponents and these opponents also give information on this piece of knowledge then that is added in a similar way. Thus the player can receive conflicting evidence from its opponents and needs to evaluate each hearsay in respect to the characters reliability. Since some variables may have many states and the opponents can be multiple then the hearsay variable sometimes needs to be split and aggregated as is described in section 7.1.

The premise and inference variables are used to generate sentences which is described in section 9.3. There is ample opportunity to play around with possible characteristics and emotions when handling the knowledge variables. Intelligence would for instance influence how much precedence, if any, the inference node would have over the premises node. This could also be influenced by gullibility and some emotions. Moreover the hearsay value variable could take more factors into account such as emotions or connections to the opponent or the type of knowledge being handled. The player might put a higher value on the opponent's claim of being the parent than on the opponent's claim of not having a dark secret.

9.3 Sentences and replies

Each time the player wants to say something she needs to generate a set of possible sentences. The sentences need to be based on some prior knowledge that the player has and on her intended purpose for speaking. It is also necessary to generate all the possible replies of the opponent so that she can evaluate the affect of her intended sentence. This is discussed in more detail in section 10.3

The sentence and reply subnets contain the sentences that the active character could say and the replies she would expect from her opponent respective.

It is necessary to calculate the levels of contradiction and risk for each sentence and reply to get an evaluation of the over all risk of each sentence. First it is good to get a detailed description of what the contradiction and risk variables measure and then it is possible to put it in a more general context by an example.

$Contradiction$ is obtained when the knowledge that a sentence or a reply is based on is at odds with the assumed knowledge of the opponent.

To calculate the contradiction variable of a sentence or reply it is necessary to compare the knowledge $k^{s,r}$ associated with each sentence s or reply r with each of the respective opponent's knowledge denoted k^o. Such that $k_1^{s,r}$ is compared to opponent's k_1^o.

Contradiction is caused by inequalities between the values of $k_1^{s,r}$ and k_1^o

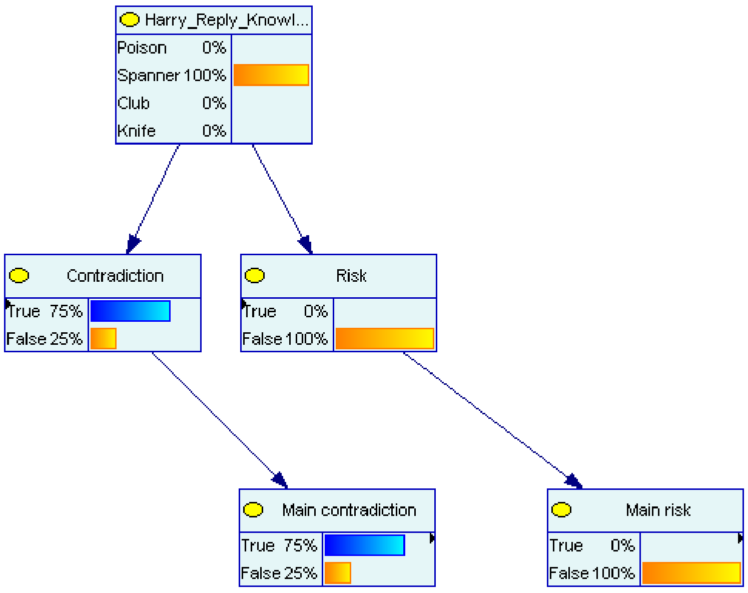
$$k_1^o = True \neq False = k_1^{s,r}$$

It may be reasonable to assume that there are degrees of contradiction due to other factors such as type of knowledge. As an example: if someone tells us he arrived at three o'clock but we know in fact that he arrived 15 minutes earlier we do not necessarily consider it a contradiction, or at least less of a contradiction than if the same person claimed that he never arrived. Such affects would influence the contradiction variable.

$Risk$ is when the knowledge that a sentence or a reply is based on contradicts the knowledge of the player. The level of risk of a sentence is estimated following the same lines outlined above for the level of contradiction.

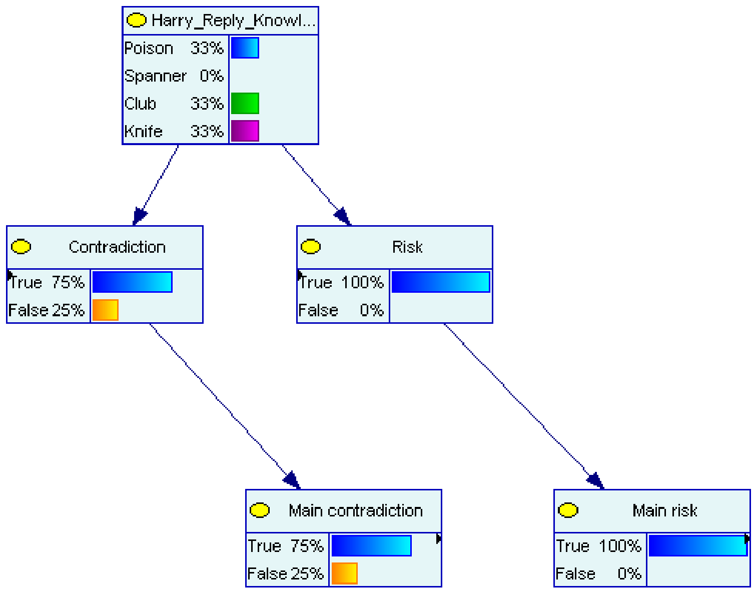
The *main contradiction* and *main risk* variables aggregate values from all the contradiction and risk variables respective in the same subnet. They may need to be split first as described in section 7.1 if there are numerous knowledge variables for any one sentence created. Having the main contradiction and risk variables as intermediate steps is necessary to simplify the calculations in the speech subnet see section 9.4.

Figure 9.4: Spanner true



As an example look at Figures 9.4 and 9.5 that show two possible replies: reply A “the murder weapon is spanner" and reply B “the murder weapon is not spanner". In this example Harry is evaluating what the opponent is likely to reply. Harry believes that his opponent is absolutely sure that the murder weapon is spanner and that the opponent believes that Harry is clueless about what the murder weapon is such that Harry has an equal value set on all states in his respective knowledge subnet. The top variable in Figure 9.4 has its probability for state spanner set to 1. and all other states set to 0. Since Harry beliefs that his opponent believes that Harry is clueless, Harry will assume that the opponent realizes that claiming that the murder weapon is spanner will contradict Harry’s knowledge by 75% but will be identical to 25% of Harry’s knowledge. On the other hand, Harry assumes that the opponent assumes that there is no risk involved in claiming that the murder weapon is a spanner because the opponent is absolutely sure that the murder weapon is spanner.

Figure 9.5: Spanner false



In Figure 9.5 Harry evaluates possible replies to the effect that the murder weapon is not a spanner; so the probability of spanner is set to 0 but all other states have equal probabilities. Just as in Figure 9.4 the contradiction is 0.75 because there is a contradiction of 75% essentially. But now the risk is very high because Harry assumes that the opponent will think that he, the opponent, is lying and will find it a very risky business.

9.4 The value of speech

As can be seen in Figure 9.1 and Figure 9.2 all paths lead to the speech subnet which this section deals with. To implement the Bayesian net as a MAID, as described in section 8, we need one net for each decision that would have been in a Bayesian net. In the prototype system described in this thesis there are two decisions: what the active character is going to say and what he expects as a reply. The result is two Bayesian nets; the difference between the two nets is in the sentence and speech subnets. The sentence subnet describes which knowledge variables affect the contradiction and risk variables.

Each speech subnet has a $utility$ node called *payoff* which calculates the utility of either the sentence or reply. All variables that influence the payoff have the same set of states $\{True, False\}$. This is to simplify calculations. *True* is bad and *False* is good. Each value of True gets 100 points. If the values would simply be added up then some of them could become really high. Moreover it would be very difficult to get a quick relative perspective of the payoff. So in order to gain greater transparency the value is divided by the number of parents. This means that the payoff is always between 0 and 100.

Figure 9.6: Sentence setup

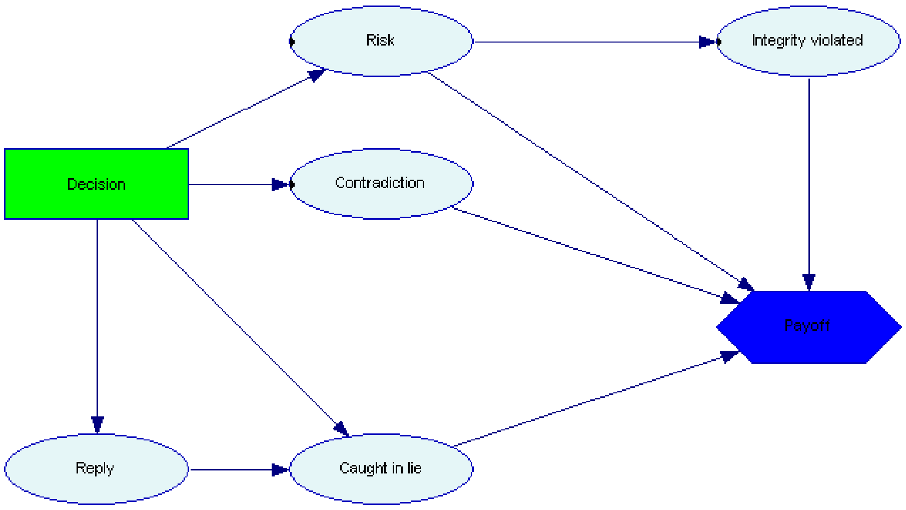
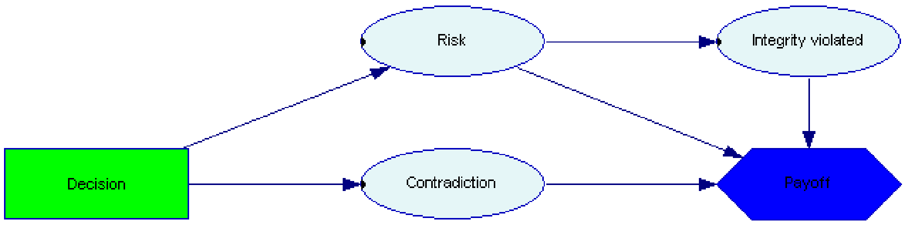


Figure 9.7: Reply setup



The $contradiction$ and $risk$ variables, in Figure 9.6 and 9.7, determine the overall contradiction and risk of each sentence or reply and influence the payoff. They aggregate all the main contradiction and risk variables from the sentence and reply subnets for the sentence and reply setup respective. The risk and contradiction need to be aggregated in respect to what sentences and replies they represent. Relational aggregation as is described in section 7.2 accomplishes this.

The *integrity violated* variable, in Figure 9.6 and 9.7, calculates how much the sentence violates the integrity of the player or opponent, in respect to his reliability and honesty.

The *sentence setup* in Figure 9.6, is used to calculate the payoff of each sentence that the player is considering.

$Reply$ variable, in the sentence setup in Figure 9.6, lists all the optimal replies that the player expects to get in respect to the sentence spoken. This is explained in more detail in section 10.4.

The *caught in lie* variable measures whether the reply indicates that the opponent suspects the player to be lying.

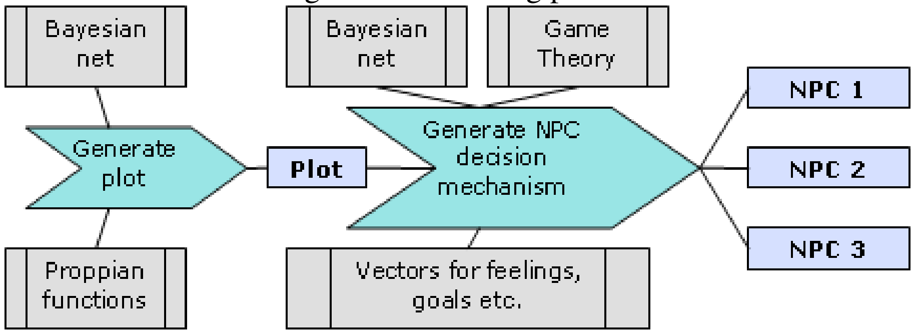
The *reply setup* in Figure 9.7 is used to calculate the possible payoff of each reply for the opponent.

Chapter 10

The Engine

This section describes the computational engine of my prototype; i.e. the actual mechanism from creating a new plot and new players to the act of finding a rational sentence to speak. In order to help readers to understand better what follows, it is best to present first a general overview of the computational engine.

Figure 10.1: The big picture



The engine is in three parts; each of the parts can be considered as a completely independent part. The parts are the DPGE (Dynamic Plot Generating Engine), the decision mechanism of the player and the game theoretic setup for finding equilibrium, see Figure 10.1.

The $DPGE$ is the first part of the engine. This part creates a new murder mystery plot complete with one and only one victim, one and only one murderer, one and only one murder weapon, a set of suspects, a set of possible murder weapons, a murder scene, a set of other scenes, motives for the murderer and each of the suspects, connections between the victim and murderer and suspects and of course some clues to connect murderer and murder weapon to the crime. The resulting plot is consistent and coherent. This means that the expected causal connections hold such that if the victim was the suspects mother then that suspect will not have some other relations to the victim and that suspect will most likely be a heir. It then follows that if someone is a heir to a rich victim then that someone has a motive. This is accomplished by combining a Bayesian net with Proppian functions, see sections 7 and 5. How this is accomplished is described in section 10.1.

The *decision mechanism* of the player is created for each new game from the plot that the DPGE generated. This means that the murderer and each of the suspects are created as independent players by the engine. Each of these players contain two Bayesian networks that have the playert's characteristics and a knowledge base as described in sections 9.1 and 9.2. Moreover each of them has the ability to generate sentences and replies and the contingent sentence and reply setups respectively. How they are created from the plot is described in section 10.2.

Finding equilibrium, essentially refers to the player finding an optimal sentence to speak at all times. In order to find any sentence at all the player first needs to generate a set of possible sentences. This he does in respect to his knowledge base and a seed that he gets. The seed is either something that the opponent just said or simply something used to start the dialog with. Each sentences is then used as a seed to generate all the replies that it is likely to provoke. The process of generating sentences and replies is explained in section 10.3.

Now that a player has been created with a knowledge base, characteristics and sets of sentences and replies it is time for that player to decide what, if anything, to say. The problem he faces is what is called a single player decision problem in a non cooperative game. This simply means that the player needs to make a decision based on his *exogenous* data and *ex ante* calculations. What matters to him is his beliefs about his opponent and about his surroundings and not other players beliefs about other players. He wants to find an optimal sentence in respect to what he believes that his opponent will reply. The player is trying to find a Bayesian equilibrium and I want to stress again the importance that since Bayesian equilibrium, like Nash equilibrium, is essentially a consistency check, players' beliefs about others' beliefs do not enter the definition – all that matters is each player's own beliefs about the distribution of types and his opponents' type-contingent strategies (Fudenberg & Tirole, 1991).

What the player will say is dependent upon what he expects as a reply. So we have two decision variables in a MAID. One for the optimal sentences D_s and another for the optimal replies D_r. Decision variable D_s is dependent on decision variable D_r. This means that we can draw a relevance graph such that $D_s \rightarrow D_r$. This means that there is need for two MAIDs to find an equilibrium. One induced by the replies $\mathcal{M}_{[\sigma_{D_r}]}$ and the other induced by the sentences $\mathcal{M}_{[\sigma_{D_s}]}$. There are then two MAIDs: $\mathcal{M}_{[\sigma_{D_r}]}$ where D_r

is a chance variable and $\mathcal{M}_{[\sigma_{D_s}]}$ where D_s is a chance variable. When these two MAIDs have been constructed the algorithm evaluates the two MAIDs by iterating through them in reverse order. This means that $\mathcal{M}_{[\sigma_{D_r}]}$ is calculated first and $\mathcal{M}_{[\sigma_{D_s}]}$ second induced with the results from D_r. The decision reached in $\mathcal{M}_{[\sigma_{D_s}]}$ is then an optimal decision. This is equivalent to finding a Nash equilibrium.

10.1 The DPGE (Dynamic Plot Generating Engine)

First the engine starts by creating the plot. This is done by first reading two text files. One of the files lists the identifying strings, names and positions for each of the variables needed to create the Bayesian net that the DPGE then uses to create the plot. The second file contains all names of characters that the DPGE will need. The reason for this is so that it is easier to make changes and to make the DPGE more autonomous. This is also to prepare for the possibility that a graphical interface could be built on top of the DPGE. This would be so that non-programmers could add to the plot setup.

The DPGE uses the information that it reads from the files to construct a Bayesian net and adding all the necessary constraints that are described in Section 5. This newly constructed net is then used to set the plot. First a victim is chosen randomly from the set of available names. Then a murderer is randomly chosen. The constraints in the net make sure that he satisfies all the necessary constraints. Any names that are left in the set read from the initial names file are now automatically set as suspects and the constraints of the net ensure that they will not be able to satisfy all of the constraints needed to be a murderer.

Finally all variables in the net that have not been instantiated are now one by one instantiated. Each variable is instantiated by finding a random number between 0 and 1 and instantiating the variable to the state that matches the randomly chosen value. For instance if a variable has two states $\{True, False\}$ and $True$ is 25% likely then a value of 0.2 would result in the variable being instantiated to $True$.

10.2 Generating players

The plot that the DPGE creates is a complete description of everything that is needed to create the knowledge base of a player. First each player is given the basic set of characteristics that have randomly set values. It is not dependent to the plot whether a character is gullible or honest. The plot is then used to create the knowledge base of each of the player. The knowledge base is in four parts. Each part contains variables with the knowledge that the player will have in respect to their position in the plot. The four parts are:

- Knowledge of the $crime$ which contains knowledge of the murderer, murder weapon and victim. The murderer will know nearly everything about the crime while the suspects know very little except about the victim. Still all the players get all the nodes for the knowledges. The murderer gets his instantiated to what he knows while the other players just get the nodes as they appear before the plot has been set.
- Knowledge of $weapons$, each player gets the knowledge setup for each possible weapon that can be found in the plot so that he can use it to know the weapons and their attributes and to deduce which weapon could be the murder weapon
- Knowledge of $scenes$, each player gets the knowledge setup for each possible scene to be found in the game. This he uses to be able to know its attributes and how it connects to other knowledges.
- $Personal$ knowledge, each player gets personal knowledge about himself and also about each of the other players. He will have his personal knowledge instantiated to what he knows about himself but he will have fairly little knowledge about other players except for what is readily apparent. Such as hair color, name and hight.

A more complete explanation of the knowledge base is in section 9.2. When the knowledge base has been created for each player then the knowledge that he “knows" is specially instantiated. The murderer is then instantiated with murder specific knowledge, that is the knowledge that only he would know. The suspects, other players, are instantiated with much more general knowledge such as hight hair color and such, and then of course everything about them selfs. Finally each player is equipped with tools to generate sentences and replies using his knowledge base which is described in the following section. Moreover the player is also equipped with tools to create the speech setup for both the sentence and reply setups which is described in section 10.4.

10.3 Generating sentences and replies

The sentences are generated in the simplest possible manner as is shown in the following algorithms. First in algorithm 7 a $seed$ is used to find relevant variables, the seed is the basic identifying string of a variable. The string is joined to the name of the player and opponent. First the seed is used to generate sentences and then for each of the generated sentences the id of the sentence is used to generate replies for that sentence.

Algorithm 7: Generating Sentences and Reply

```
Input: String seed, String player, String opponent
begin
    # see algorithm 8, Creating the sentences
    sentences = createSentences( player + seed )
    String oldid
    Sentence[] replies
    for each ∈ sentences do
        # First check to see if identical set of replies exists
        if oldid = each.getId() then
            each.setReplies( each, replies )
            continue
        end
        oldid = each.getId()
        seed = each.getId( )
        # see algorithm 8, Creating the sentences
        each.setReplies( createSentences( opponent + seed ) )
    end
    # Update the global graph
    graph.setSentences( sentences )
end
```

Example 5 (Variables id) *The seed _s_relation along with the player's name and the opponent's name, e.g. player = Horace, opponent = Pilar, results in the two seeds:*

- *Horace_s_relation and*
- *Pilar_s_relation,*

where _s_ stands for suspect.

The second step, shown in algorithm 8, is to find the variable that matches the seed and to find all of its parents' variables and child variables. Since the net is causal it follows that those variables are in causal relations to the seed and that humans should find them relevant.

The third step, shown in algorithm 9, is to create a sentence for each possible state of all the collected variables. Each of the states of a variable is instantiated as being either $true$ or $false$. If a variable has only two states then only one of the states needs to be instantiated because the states are mutually exclusive. This removes some obvious redundancy. However this methodology of generating sentences is still very brute force and should be optimized or removed in order to obtain "more intelligent" sentence generating

Algorithm 8: Creating the sentences

```
Input  : String seed
Output: An array of generated sentences
begin
    String[] ids
    # Append the matching variable
    ids.append( seed )
    for each ∈ net.getChildIds( seed ) do
        ids.append( each ) # Append the children
    end
    for each ∈ net.getParentIds( seed ) do
        ids.append( each ) # Append the parents
    end
    # see algorithm 9, Create each sentence instances
    Return sentenceInstances( ids )
end
```

Algorithm 9: Create each sentence instances

```
Input  : String[] ids
Output: An array of generated sentences
begin
    Sentence[] sentences
    for id ∈ ids do
        states = net.getOutcomeIds( id )
        for state ∈ states do
            sentences.append( id, state, true, states )
            sentences.append( id, state, false, states )
            # Remove obvious redundancy
            if states.length = 2 then break
        end
    end
    return sentences
end
```

10.4 Finding equilibrium

To find a Nash equilibrium in the game that has been portrayed it is necessary to find the assumed strategy profiles of the opponent. It is important to realize that the game is sequential, e.g. the player makes a move and then his opponent makes a move. They do not make their moves in parallel which simplifies the computation of equilibrium significantly.

Recall definition 2 of a strategy profile, that is contingent to the player's type, where the player's strategy profile needs to take into account every strategy profile of the opponent. Given a strategy profile $s(\cdot)$, and an $s_i'(\cdot) \in S_i^{\Theta_i}$, let $(s_i'(\cdot), s_{-i}(\cdot))$ denote the profile where player i plays $s_i'(\cdot)$ and the other players, his opponents, play $s(\cdot)$, then

$$(s_i'(\theta_i), s_{-i}(\theta_{-i})) = (s_1(\theta_1), ..., s_{i-1}(\theta_{i-1}), s_i'(\theta_i), s_{i+1}(\theta_{i+1}), ..., s_i(\theta_i))$$

The MAIDs determine types in respect to the decisions that need to be taken to resolve the game. There are two decisions that need to be made by the player: The assumed reply and the sentence in respect to the assumed reply. This means that to determine the opponent's strategy profiles for each of the player's strategy profiles it is necessary to generate all the possible replies to each of the possible sentences that the player contemplates saying. This is accomplished in section 10.3, where I describe the process of sentence generating.

Now recall the Bayesian equilibrium, definition 3. That definition states the set of optimal strategies is the one that maximizes the accumulated player's utilities contingent on the opponent's strategy profile which in turn is contingent on the opponent's type. Formally,

$$s_i(\theta_i) \in \arg\max_{s_i' \in S_i} \sum_{\theta_{-i}} p(\theta_{-i}|\theta_i) u_i(s_i', s_{-i}(\theta_{-i}), (\theta_i, \theta_{-i}))$$

To find the players utility u_i it is necessary to find the strategy profiles that the opponent will play contingent to the opponents type, $s_{-i}(\theta_{-i})$, this is accomplished in algorithm 10 and 11.

Algorithm 10 finds the set of optimal replies to each sentence. The replies need to be evaluated in subsets of approximately ten for reasons of memory allocation and exponential growth in computing the net. The memory allocation problem results from the number

Algorithm 10: Calculating optimal replies per sentence

```
Input: Sentence[] sentences, Calculate calc, Bayes net, Player p, Player o
1  # Player o is the opponent
2  begin
3      for each ∈ sentences do
4          max = 10
5          while each.getReplies.length mod max = 1 do ++max
6          repeat
7              SentenceGenerator sg = new SentenceGenerator( p, max )
8              sg.replySetup( each )
9              Speech speech = new Speech( p, o, max )
10             speech.reply( each )
11             net.updateBeliefs( )
12             calc.bestReplies( each, net )
13             p.clearSentenceAndSpeech( )
14         until !each.incrementRepliesIndexes( max )
15     end
16 end
```

of replies that need to be evaluated. There are a few hundreds of replies that need to be evaluated and each of them corresponds to one state in a decision variable. A solution is to distribute the replies to many decision variables. This solves the memory allocation problem but introduces a different problem of exponential growth in calculating the net. This growth arises because all of the decision variables are then connected in such a way that they depend on each others' result. If there are 9 other variables, each having 10 states, that need to be evaluated, then the strategy function of each decision variable is 10^9 (Jensen, 2001), so the complexity for each decision variable is $O(ns^{nv-1})$ where ns is the number of states and nv is the number of variables that influence the decision. There are ways to use an aspect of the Bayesian net called d-separation (Jensen, 2001) to handle this to some extent but I chose to simply create a completely new reply and speech setup for each 10 replies evaluated. This is possible because the evaluation of one reply is not dependent on the evaluation of another reply.

The first step of the algorithm is to set max replies to 10 in line 4. Next it is necessary to make sure that there is never an attempt to create a variable with less than two states because a variable must, by definition, have a minimum of two states. This is accomplished with simple modulus of number of replies by max replies as is shown in line 5 (see example 6).

Example 6 (Two states minimum) *Assume that max = 10 and number nReplies of replies is 21. Then*

$$nReplies \; mod \; max = 1$$

This will produce a variable with only one reply and thus only one state to be created, leading to a failure in the program. To solve this problem I simply increment the variable max and then repeat the check to make sure that the modulus does not result in 1.

The function in line 14 will return false when it is unable to shift to the next subset of replies to evaluate. For each subset of replies the variables for those replies need to be generated and the Speech subnet needs to be repopulated in respect to the current subset of replies. This is accomplished by function calls in lines 8 and 10 respectively. The details of populating the subnets are omitted. Then the net is updated in line 11, e.g. all variables in the net that need to be evaluated are updated. Then, in line 12, the current set of optimal replies is found as shown in algorithm 11. Finally it is necessary to clear the sentence and speech subnets for the next round with a function call in line 13.

Algorithm 11: Finding optimal replies

```
Input: Sentence sentence, Bayes net
1  begin
2      optimal = sentence.getOptimalReplies()
3      replies = sentence.getNextReplies( )
4      decision = net.getNodeValue( DECISION )
5      high = sentence.getHighValue()
6      for int i = 0; i < replies.length; ++i do
7          if high = decision[i] then
8              optimal.append(replies[i])
9              continue
10         end
11         if high > decision[i] then continue;
12         if high < decision[i] then
13             high = decision[i]
14             optimal.clear( )
15             optimal.append(replies[i])
16         end
17     end
18     sentence.setHighValue( high )
19     sentence.setOptimalReplies( optimal )
20 end
```

Algorithm 11 is the function that returns the optimal set of replies per sentence. First the algorithm retrieves the former optimal sentences, decision values and high value. Then, for each reply, it appends the highest ranking reply to the set of optimal replies. If a reply value is equal to the high value then append it to the set of optimal replies as is shown in the if block starting in line 7. If a reply value is too low then continue (see line 11). If a reply value is higher than the current high value then the set of optimal replies is cleared and the high ranking reply is appended to it; see block starting at line 12. Finally store the result in the sentence object

The next step is to find the sum of the weighted player's utility dependent on the opponents type θ_{-i}, e.g. $\sum_{\theta_{-i}} p(\theta_{-i}|\theta_i)u_i(s'_i, s_{-i}(\theta_{-i}), (\theta_i, \theta_{-i}))$, and then maximize it with each of the player's strategy profiles, $s'_i \in S_i$, as argument. The player knows his type and he knows all the strategy profiles that he consider viable. What the player needs to find is which set of his strategy profiles, s'_i, is the optimum one, e.g. which subset of $s'_i \in S_i$ yields the highest payoff. Knowing that the probability that the opponent is of any given type θ_{-i} is strictly positive and that this is a sequential game then the player can use the sets of optimal replies, previously acquired in algorithms 10 and 11, to find the optimal sentence as is now shown in algorithms 12 and 13.

Algorithm 12: Calculating optimal sentences

```
Input: Calculate calc, Bayes net, Player p, Player o
 1 # Player o is the opponent
 2 begin
 3     max = 6
 4     while sentences.length mod max = 1 do - - max
 5     Speech speech = new Speech( p, o, max )
 6     SentenceGenerator sg = new SentenceGenerator( p, o, max )
 7     repeat
 8         sentences = p.getNextSentences( max )
 9         sg.sentenceSetup( sentences )
10         speech.sentence( sentences )
11         calc.equilibria( sentences )
12         p.clearSentenceAndSpeech()
13     until !p.incrementSentenceIndexes( max )
14 end
```

Algorithm 12 finds the set of optimal sentences in the following manner. The sentences need to be evaluated in subsets for the same reason as the replies. First max sentences is set to 6 in line 3, then just as with the replies it is necessary to make sure that there is never an attempt to create a variable with less than two states. This is accomplished with simple modulus of number of sentences by max sentences as is shown in line 4. Next repeat the following until there are no more sentences to evaluate. The function in line 13 will return false when it is unable to shift to the next subset of sentences to evaluate. For each subset of sentences the variables for those sentences need to be generated and the Speech subnet needs to be repopulated in respect to the current subset of sentences. This is accomplished by function call in lines 9 and 10 respective. The details of populating the subnets are omitted. Then, in line 11, the current set of optimal sentences is found which is shown in algorithm 13. Finally it is necessary to clear the sentence and speech subnets for the next round with a function call in line 12, details omitted.

Algorithm 13: Finding equilibrium

```
   Input: Player p, Bayes net
 1 begin
 2     setReplyValue( p, net )
 3     net.updateBeliefs()
 4     decision = net.getNodeValue( DECISION )
 5     optimal = p.getOptimal()
 6     high = 0
 7     if optimal.length != 0 then high = optimal[0].getValue()
 8     for int i = 0; i < decision.length; ++i do
 9         if high = decision[i] then
10             optimal.append( sentences[i] )
11             continue
12         end
13         if high > decision[i] then continue
14         if high < decision[i] then
15             high = decision[i]
16             sentences[i].setValue( high )
17             optimal.clear( )
18             optimal.append( sentences[i] )
19         end
20     end
21     p.setOptimal( optimal )
22 end
```

Algorithm 13 is the function that ultimately returns the optimal set of sentences. First the algorithm defines the reply variable with a function call in line 2, as is shown in example 7 This is really setting the grounds for calculating the weighted payoff, e.g. $p(\theta_{-i}|\theta_i)u_i(s'_i, s_{-i}(\theta_{-i}), (\theta_i, \theta_{-i}))$. The probabilities set in the reply variable represent the probability of each type and the optimal reply which is contingent to the type. The net is then updated in line 3 which sums up the weighted utility in the decision variable. The values of the decision variables are then retrieved together with the former optimal sentences if there were any. Also, in line 7, retrieve the former high value if there was one. Then, for each sentence, append the highest ranking sentences to the set of optimal sentences. If a sentence value is equal to the high value then append it to the set of optimal sentences as is shown in the if block starting in line 9. If a sentence value is to low then continue (see line 14). If a sentence value is higher than the current high value then the set of optimal sentences is cleared and the high ranking sentence is appended to it, see block starting in line 13. The end result is a set of optimal sentences.

Example 7 (The reply variable in the sentence speech setup) *Assume that there are two sentences, namely*

- *sentence S1 that has 1 optimal reply = { S1R1 } and*
- *sentence S2 that has 2 optimal replies = { S2R1, S2R2 }.*

Then S1 will receive the reply S1R1 with probability 1 and S2 will receive replies S2R1, S2R2 each with probability 0.5. This is defined in the Reply variable in the sentence speech setup as is shown in table 10.1.

Table 10.1: Reply variable

Sentence	S1	S2
S1R1	1	0
S2R1	0	0.5
S2R2	0	0.5

10.5 Example run

In order to exemplify the workings of the engine I described above, I now present an example run of my prototype system from the creation of a plot to the choice of a sentence by the player. The log-file is generated with the log4J library. In order to improve clarity I remove the log string that informs from where in the program the logging occurs. Instead I write that information string at the start of each new section. First the DPGE creates the plot from two text files

```
INFO main NPC.NPC - Start at Mon Apr 16 08:24:14 GMT 2007
```

First the DPGE creates the plot from two text files,
(INFO main DPGE.PlotGenerator)

```
Characters defined
Crime defined
```

```
Weapons defined
Murder scene defined
Suspects defined
Number of nodes:  268
Net created -----------------------------
Net Written
```

Here the plot has been set and a description of the plot is written out,
(INFO main DPGE.PlotGenerator)

```
--- Victim is Alice---
The victim is Medium, Female with shoe size Nr_41
Husband = Donald, Wife = Null, Male affair = Null,
Female affair = Null Mother = Linda, Father = Null,
Rich = True, dark secret = False
-----------------------------
--- Weapon is Spanner---
Material = Steal, Marks = Square, wound type = Blunt_wound,
needs strength = True, Hair on victim = True,
hair on weapon = True, blood on weapon = False
-----------------------------
--- murderer is Penny ---
Small, Strong, Female with shoe size Nr_37
Relation = Friends, Heir = False, Rich = True
In debt to victim = False, Has dark secret = False
The murderer has the following as possible motives for murder:
Inheritance = False, Blackmailer = False, Swindler = False,
Adultery = True Wedlock = False, Debt = False,
isSwindled = False, Revenge = True, isBlackmailed = False
-----------------------------
--- suspect 1 is Linda ---
Small, Weak, Female with shoe size Nr_37
Relation = Parent, Heir = False, Rich = True
In debt to victim = False, Has dark secret = False
The suspect has the following as possible motives for murder:
Inheritance = False, Blackmailer = False, Swindler = True,
Adultery = False Wedlock = False, Debt = False,
isSwindled = True, Revenge = True, isBlackmailed = False
-----------------------------
--- suspect 2 is Donald ---
Tall, Strong, Male with shoe size Nr_43
Relation = Spouce, Heir = True, Rich = True
In debt to victim = False, Has dark secret = False
The suspect has the following as possible motives for murder:
Inheritance = False, Blackmailer = False, Swindler = True,
Adultery = False Wedlock = True, Debt = False,
isSwindled = True, Revenge = True, isBlackmailed = False
-----------------------------
```

Now that the plot has been created the players are created from the plot,
(INFO main NPC.KnowledgeGenerator)

```
ADDING CRIME KNOWLEDGE for Linda
ADDING WEAPON KNOWLEDGE for Linda
ADDING SCENE KNOWLEDGE for Linda
ADDING PERSON KNOWLEDGE for Linda
ADDING CRIME KNOWLEDGE for Penny
ADDING CRIME KNOWLEDGE for Pennys opponent Linda
ADDING CRIME KNOWLEDGE for Pennys opponent Donald
ADDING WEAPON KNOWLEDGE for Penny
ADDING WEAPON KNOWLEDGE for Pennys opponent Linda
ADDING WEAPON KNOWLEDGE for Pennys opponent Donald
ADDING SCENE KNOWLEDGE for Penny
ADDING SCENE KNOWLEDGE for Pennys opponent Linda
ADDING SCENE KNOWLEDGE for Pennys opponent Donald
ADDING OPPONENTS PERSON KNOWLEDGE for Penny
ADDING OPPONENTS OPPONENTS PERSON KNOWLEDGE for Pennys opponent
Linda
ADDING OPPONENTS OPPONENTS PERSON KNOWLEDGE for Pennys opponent
Donald
ADDING CRIME KNOWLEDGE for Donald
ADDING CRIME KNOWLEDGE for Donalds opponent Linda
ADDING CRIME KNOWLEDGE for Donalds opponent Penny
ADDING WEAPON KNOWLEDGE for Donald
ADDING WEAPON KNOWLEDGE for Donalds opponent Linda
ADDING WEAPON KNOWLEDGE for Donalds opponent Penny
ADDING SCENE KNOWLEDGE for Donald
ADDING SCENE KNOWLEDGE for Donalds opponent Linda
ADDING SCENE KNOWLEDGE for Donalds opponent Penny
ADDING OPPONENTS PERSON KNOWLEDGE for Donald
ADDING OPPONENTS OPPONENTS PERSON KNOWLEDGE for Donalds opponent
Linda
ADDING OPPONENTS OPPONENTS PERSON KNOWLEDGE for Donalds opponent
Penny
The number of nodes in the net are:  3.311
--Knowledge generated for Linda--
-.-.- Knowledge is generated for Linda's reply setup
The number of nodes in the net are:  3.311
--Knowledge generated for Linda--
Written, elapsed time 18563 - 00:00:18
-.-.- Knowledge is generated for Donald's sentence setup
The number of nodes in the net are:  3.311
--Knowledge generated for Donald--
-.-.- Knowledge is generated for Donald's reply setup
The number of nodes in the net are:  3.311
--Knowledge generated for Donald--
Written, elapsed time 35157 - 00:00:35
-.-.- Knowledge is generated for Penny's sentence setup
The number of nodes in the net are:  3.301
--Knowledge generated for Penny--
-.-.- Knowledge is generated for Penny's reply setup
The number of nodes in the net are:  3.301
--Knowledge generated for Penny--
Written, elapsed time 53391 - 00:00:53
SEED S_name
```

Linda is talking to Donald, Start by creating sentences and replies,
(INFO main NPC.SentenceGenerator)

```
5 matches for Linda_S_name_Prem where found, (sentences)
5 matches for Donald_Linda_S_name_Prem where found, (replies)
```

```
5 matches for Donald_Linda_S_sex_Prem where found, (replies)
6 matches for Donald_Linda_S_spouce_Prem where found, (replies)
5 matches for Donald_Linda_S_parent_Prem where found, (replies)
5 matches for Donald_Linda_S_adulterer_Prem where found, (replies)
```

Now Linda evaluates the replies. The -.-.- represents logging of replies that are not shown, (INFO main NPC.NPC).

```
sentence nr 0:  1 optimal replies found of 16 replies evaluated,
elapsed time 61735 - 00:01:01
Linda says that Linda S_adulterer is not True, the value is 100,
→
Linda_Reply_Knowledge_Donald_Linda_S_adulterer_Prem_True_false
-.-.-
sentence nr 8:  3 optimal replies found of 36 replies evaluated,
elapsed time 130985 - 00:02:10
Linda says that Linda S_sex is Male, the value is 66,667, →
Linda_Reply_Knowledge_Donald_Linda_S_sex_Prem_Male_true
Linda says that Linda S_size is Tall, the value is 66,667, →
Linda_Reply_Knowledge_Donald_Linda_S_size_Prem_Tall_true
Linda says that Linda S_name is Donald, the value is 66,667,
→
Linda_Reply_Knowledge_Donald_Linda_S_name_Prem_Donald_true

sentence nr 9:  3 optimal replies found of 36 replies evaluated,
elapsed time 147328 - 00:02:27
Linda says that Linda S_sex is Male, the value is 66,667, →
Linda_Reply_Knowledge_Donald_Linda_S_sex_Prem_Male_true
Linda says that Linda S_size is Tall, the value is 66,667, →
Linda_Reply_Knowledge_Donald_Linda_S_size_Prem_Tall_true
Linda says that Linda S_name is Donald, the value is 66,667,
→
Linda_Reply_Knowledge_Donald_Linda_S_name_Prem_Donald_true
-.-.-
```

Now an optimal reply has been found for all 16 sentences that the player is evaluating. Next the player, Linda, evaluates the optimal sentence,

(INFO main Calculate.Calculate)

```
MAX_SENT 6, containers.size() 16
Linda is talking to Donald, 0 sentences of 16 have been evaluated,
slot size is 6
Sentences generated, elapsed time 221532 - 00:03:41
Linda says that S_name is Linda, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_name_Prem_Linda_true

Linda is talking to Donald, 6 sentences of 16 have been evaluated,
slot size is 6
Sentences generated, elapsed time 224500 - 00:03:44
Linda says that S_name is Linda, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_name_Prem_Linda_true
Linda says that S_sex is not Male, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_sex_Prem_Male_false
Linda says that S_spouce is not True, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_spouce_Prem_True_false


Linda is talking to Donald, 12 sentences of 16 have been evaluated,
slot size is 4
Sentences generated, elapsed time 227360 - 00:03:47
Linda says that S_name is Linda, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_name_Prem_Linda_true
Linda says that S_sex is not Male, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_sex_Prem_Male_false
Linda says that S_spouce is not True, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_spouce_Prem_True_false
Linda says that S_parent is True, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_parent_Prem_True_true
Linda says that S_adulterer is not True, the value is 90, →
Linda_Sentence_Knowledge_Linda_S_adulterer_Prem_True_false
```

Sentences have been evaluated. Linda vill randomly choose one of the optimal sentence and if she has not used it before then she will say it. If Linda has said the equivalent before then she will randomly pick another of her optimal sentences and check it. She will do this until she either finds something to say or realizes that she has nothing more to say and then the conversation will end.

```
id Linda_S_name_Prem, state Linda, position true, value 90,
knowledgeID Linda_Sentence_Knowledge_Linda_S_name_Prem_Linda_true
Linda says:  "My name is Linda".
```

The actual calculation of finding the optimal sentence took:

```
Calculations took:  elapsed time 177219 - 00:02:57
```

Now Donald gets S_name as a seed and starts the process of generating sentences and replies. This continuous until either of the character has nothing more to say.

Chapter 11

Results

The results are promising; firstly the players are able to calculate an optimal sentence using game theory and they are able to participate in a dialog. I show below four examples of dialog that are started by some seed. These dialog are between two suspects and between the murderer and a suspect. The dialogs are both about known and unknown knowledge.

Both Alice and Rosamund are suspects so they have only accurate information about themselves and not the crime. First Alice starts to talk to Rosamund. The seed that Alice gets is S_name which makes her start searching for all sentences connected to her name. The name variable has many connections so there are many matches.

```
Seed S_name, calculation time:  00:02:57,
optimal value = 100,
optimal sentences 5 of 24 sentences:
Alice_Sentence_Knowledge_Alice_S_name_Prem_Alice_true
Alice_Sentence_Knowledge_Alice_S_sex_Prem_Male_false
Alice_Sentence_Knowledge_Alice_S_spouse_Prem_True_false
Alice_Sentence_Knowledge_Alice_S_parent_Prem_True_false
Alice_Sentence_Knowledge_Alice_S_adulterer_Prem_True_true

Alice says:  I'm not Horace's wife.

Seed S_spouse, calculation time:  00:02:06,
optimal value = 100,
optimal sentences 4 of 18 sentences:
```

```
Rosamund_Sentence_Knowledge_Rosamund_S_spouse_Prem_True_false
Rosamund_Sentence_Knowledge_Rosamund_S_spouse_Inf_True_false
Rosamund_Sentence_Knowledge_Rosamund_S_relation_Prem_Parent_true
Rosamund_Sentence_Knowledge_Rosamund_S_wedlock_Prem_True_false


Rosamund says:  I'm Horace's parent.

Seed S_relation, calculation time:  00:02:58,
optimal value = 100,
optimal sentences 1 of 26 sentences:
Alice_Sentence_Knowledge_Alice_S_relation_Prem_Adulterer_true


Alice says:  Horace was having an affair with me.

Seed S_relation, calculation time:  00:02:57,
optimal value = 100,
optimal sentences 1 of 26 sentences:
Rosamund_Sentence_Knowledge_Rosamund_S_relation_Prem_Parent_true


Rosamund can't think of anything to say!
```

Second example: Rosamund starts to talk to Alice. The seed that Rosamund gets is $M\_victim\_knows\_of\_dark\_secret$ which makes her start searching for all sentences connected to that variable. There are not many connections and not many sentences to evaluate. Moreover she has no evidence to know whether the murderer had a dark secret or that the victim knew about it so she just makes an educated guess in respect to the initial structure of the plot engine. Alice does the same in replying.

```
Seed M_victim_knows_of_dark_secret, calculation time:  00:00:19,
optimal value = 71,875,
optimal sentences 3 of 6 sentences:
Rosamund_Sentence_Knowledge_Rosamund_M_victim_knows_of_dark_secret_Prem
Rosamund_Sentence_Knowledge_Rosamund_M_victim_knows_of_dark_secret_Inf_
```

```
Rosamund_Sentence_Knowledge_Rosamund_M_isBlackmailed_Prem_True_false


Rosamund says:  "The murderer was probably not being blackmailed
by Horace"

Seed M_isBlackmailed, calculation time:  00:00:13,
optimal value = 71,875,
optimal sentences 2 of 4 sentences:
Alice_Sentence_Knowledge_Alice_M_isBlackmailed_Prem_True_false
Alice_Sentence_Knowledge_Alice_M_isBlackmailed_Inf_True_false


Alice says:  "The murderer was probably not being blackmailed
by Horace"

Seed M_isBlackmailed, calculation time:  00:00:13,
optimal value = 71,875,
optimal sentences 2 of 4 sentences:
Rosamund_Sentence_Knowledge_Rosamund_M_isBlackmailed_Prem_True_false
Rosamund_Sentence_Knowledge_Rosamund_M_isBlackmailed_Inf_True_false


Rosamund can't think of anything to say!
```

Third example: Now Rosamund and Linda the murderer and the victim's wife are talking together. Rosamund gets seed S_parent.

```
Seed S_parent, calculation time:  00:02:31,
optimal value = 100,
optimal sentences 2 of 16 sentences:
Rosamund_Sentence_Knowledge_Rosamund_S_parent_Inf_True_true
Rosamund_Sentence_Knowledge_Rosamund_S_relation_Prem_Parent_true


Rosamund says:  I'm Horace's parent.
```

```
Seed S_parent, calculation time:  00:00:19,
optimal value = 100,
optimal sentences 2 of 4 sentences:
Linda_Sentence_Knowledge_Linda_S_parent_Prem_True_false


Linda says:  I'm not Horace's parent.

Seed S_parent, calculation time:  00:02:32,
optimal value = 100,
optimal sentences 2 of 16 sentences:
Rosamund_Sentence_Knowledge_Rosamund_S_parent_Inf_True_true
Rosamund_Sentence_Knowledge_Rosamund_S_relation_Prem_Parent_true


Rosamund says:  I'm Horace's parent.

Seed S_relation, calculation time:  00:03:13,
optimal value = 100,
optimal sentences 2 of 24 sentences:
Linda_Sentence_Knowledge_Linda_S_relation_Prem_Spouse_true
Linda_Sentence_Knowledge_Linda_S_relation_Inf_Spouse_true


Linda says:  I'm Horace's spouse.

Seed S_relation, calculation time:  00:03:14,
optimal value = 100,
optimal sentences 2 of 26 sentences:
Rosamund_Sentence_Knowledge_Rosamund_S_relation_Prem_Parent_true


Rosamund can't think of anything to say!
```

Final example: Now Rosamund gets seed $M\_revenge$ and having now knowledge of the actual motive believes that revenge is a probable motive. Linda knows better and contradicts Rosamund even though she knows it will contradict Rosamund's knowledge. Note that when both Linda and Rosamund have heard hearsay that contradicts their premises then their inferred knowledge stops having equal value to their premises and it is no longer in the set of optimal sentences.

```
Seed M_revenge, calculation time:  00:00:15,
optimal value = 84,375,
optimal sentences 2 of 4 sentences:
Rosamund_Sentence_Knowledge_Rosamund_M_revenge_Prem_True_true
Rosamund_Sentence_Knowledge_Rosamund_M_revenge_Inf_True_true


Rosamund says:  The murderers motive was probably revenge.

Seed M_revenge, calculation time:  00:00:14,
optimal value = 81,25,
optimal sentences 1 of 4 sentences:
Linda_Sentence_Knowledge_Linda_M_revenge_Prem_True_false


Linda says:  The murderers motive was probably not revenge.

Seed M_revenge, calculation time:  00:00:16,
optimal value = 84,375,
optimal sentences 2 of 4 sentences:
Rosamund_Sentence_Knowledge_Rosamund_M_revenge_Prem_True_true



Rosamund can’t think of anything to say!
```

Computing time. The computing times are impressive. Note that there was no specific optimization applied. Still over 75% of the sentences are computed within 3 minutes and more than 50% in less than 1 minute as can be seen in the boxplot in Figure 11.1. It helps that most of the time there were not that many senteces generated. Moreover in the plot in Figure 11.2 it can be clearly seen that the growth is roughly linear. The variations are due to irregular number of replies generated.

Figure 11.1: Sentences and Time in minutes

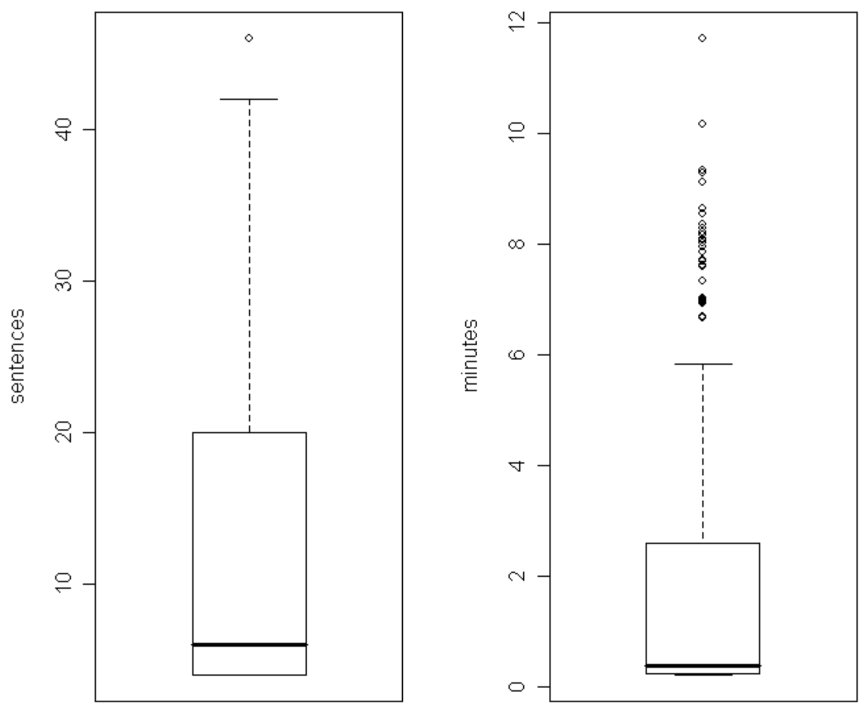
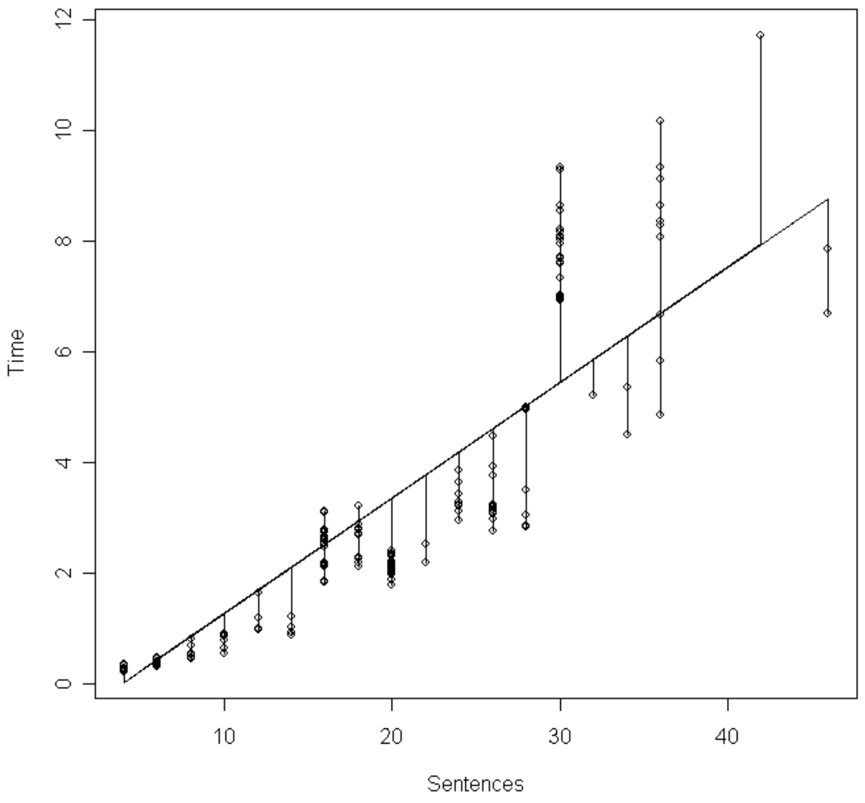


Figure 11.2: Time to calculating sentences



Chapter 12

Discussion and conclusion

In this Section I first discuss briefly the conclusions and then point out possible future work.

12.1 Conclusion

The specific problem investigated in the thesis is whether a computer generated player will interact with other players in a rational and goal driven way when given a past life and a decision support mechanism based on a causal network like a Bayesian network. Will the player adopt a strategy that will maximize its payoff?

The prototype engine that I have built successfully creates new murder mystery plots that are used to give the players past lives and connections with each other. It then uses the knowledge base of the player to generate sentences and replies and calculates an equilibrium by the terms of game theory. These results support the view that it is possible to have computer characters decide what to say in a rational way. This also means that they are able to evaluate the set of possible sentences not only by what they themselves think but also by how they expect their opponent may think and reply. Moreover the players are able to adapt to what they hear from their opponents. This they do by adding the hearsay nodes in respect to their opponent's input. Although this is only a prototype it is clear that there is basis to create in this way characters that can participate in a dialog and calculate on the fly a rational sentence to speak each time.

A very pleasant result is that they are able to evaluate the sentences and replies in a surprisingly little time. Even without many possible optimizations the time needed to calculate more than 75% of the sentence is within 3 minutes and more than half of the sentences are calculated in less than 1 minute. There are several possible optimizations possible and I mention the most obvious ones, such as in Section 12.2. Such as dynamic programming and optimizing sentence generation in Section 10.3.

Another pleasing property of the proposed solution is that the MAIDs have a linear growth and thus the complexity is linear in respect to number of sentences.

Finally this is a solution that uses inference and causality along with game theory to find an optimal sentence instead of the common approach of using a lookahead or the popular search techniques. This approach is more abstract and it is not dependent on as detailed prior descriptions of search spaces as for instance search techniques are. Moreover it does not have the computational problems of the search techniques or techniques that apply a lookahead.

12.2 Future work

There is considerable room for improving the engine and for extending it, here I will point out some of the most prominent directions.

Maturing the DPGE (Dynamic Plot Generating Engine). Although the plot generating engine is able to create complete plots it is still fairly limited. The reason being that it is still small and has fairly few options. At present only the most prominent motives are connected. There is very little supportive evidence to enrich the plot. A sophisticated way to improve the DPGE would be to take a set of murder mystery novels from a prominent murder mystery writer and analyze them in the same way as Vladimir Propp analyzed the Russian folk tales (Propp, 1968). This should result in a complete morphology of the murder mystery that then would complete the DPGE. The works of either Agatha Christie or Arthur Conan Doyle would be suitable for the task. The reason being that they are complete, the authors are deceased and will not add to the collection. Moreover the plots are well known and have been used as the basis for many other mystery novels, they are still highly popular and well regarded and have thus proven their worth.

Such work would of course require someone with considerable literary skills to correctly analyze the plots and all the relevant intricate details. Moreover this work would need to be done with the aid of someone with sufficient structural knowledge such that a proper morphology would result. This could be a MA project in Literature or even a Ph.D. thesis.

Generating sentences in a more logical and optimized way. At present the sentences are generated in a brute force way and this can of course be improved. The complexity of the algorithm for finding equilibrium is linear in respect to number of sentences evaluated. This means that refining the sentence generation process should significantly reduce computing time. One important refinement is to not generate sentences that are “equivalent" to previously spoken sentences if there is no added value in the restatement. Here “equivalent" means that the player's knowledge base has not changed in this respect in the meantime. For example player Donald should not say “My name is Donald" more than once to the same opponent unless he gets information that the opponent did not hear him correctly or has forgotten his name. Another important thing is to not generate sentences that are stating the obvious or that are just not said. An example of this is for instant that player David should not consider saying “I am male" or “I am not my own father". These are examples of actual connections in the net that will need to be excluded from sentence generation. These connections are necessary for causal integrity of the plot but are such commonly known parameters to humans that it is idiotic to state them in most scenarios. A simple way to resolve this is to mark the variables in the plot with whether they are appropriate to include in sentence generation. The limitations to that method is that sometimes we do need to state the obvious, for instance when filling out forms we are frequently asked whether we are male or female. A more sophisticated way would be to have a separate logical net to decide which sentences are relevant given the current state. All of this applies also to the generation of possible replies that the player expects from the opponent.

The player's decision mechanism. The knowledge base of the player can be greatly improved. The work was concentrated into areas that show the use of calculating Nash equilibrium using MAIDs, which means that the current variables considered by the player are whether to lie and whether the reply indicates that he has been caught lying. The players are currently influenced by hearsay but in a primitive way. That element needs to be developed especially in respect to how to evaluate hearsay from other players. By this I mean that there can be many aspects inferred from what the opponent says such as his apparent lack of knowledge or his surprisingly detailed knowledge. For instance if the opponent is assuming simple common knowledge about the murder weapon then that indicates that he is fairly ignorant of the crime but if he describes details that are not common knowledge then that should raise suspicion as to why he is so knowledgeable. Additionally the decision mechanism should be matured to take into account that there are many things that can affect the choice of what the player would want to say. The player may want to increase the opponent's opinion of himself or secure a trusted relationship.

He may want to offer the opponent something or request something from the opponent. All of this would make him interested in a different set of variables that deal with whether the opponent will be offended, impressed, disgusted, humored etc. Moreover the effect of characteristics and emotions needs to be improved. At present only characteristics that have to do with gullibility, honesty and integrity are in effect.

Optimizing calculations. There are a few fairly obvious optimizations that can be done to speed up the calculations.

First, dynamic programming can be applied by storing the value of each sentence in a hash-table by its premises so that it could be easily retrieved instead of recalculated when it is again relevant. It is highly likely that the stored value would still be perfectly valid which would make this a very efficient optimization, especially within a conversation when the player has not had any indication that the opponent's relevant knowledge has changed. When running the engine it becomes fairly clear that a large number of sentences are unnecessarily recalculated.

Secondly, it is not always appropriate to find an equilibrium. It is in fact only appropriate if the decision is dependent on the optimal reply. For instance introducing oneself is not dependent on the reply. It is somewhat dependent on the circumstances, whether one needs to be formal or not, but the opponent's reply will not influence the player's payoff. So some method should be developed for the player to decide whether finding an equilibrium is at all necessary.

Finally, if the player believes that his opponent is indecisive in some area then he can really just maximize his payoff from his own net assuming that the opponents reply is very predictable. For instance if the player beliefs that his opponents knowledge of the murder weapon is very general then he can simply use his own general knowledge of a weapon to decide what to say instead of finding an equilibrium for that group of sentences. This is because the player has then no reason to expect that what he says about the murder weapon would in any way challenge his opponent.

Add natural language generation and processing. If these characters will ever be implemented in a computer game then they need to be able to understand a human player and to be able to have a human player understand them. It is thus an interesting and necessary project to have the players generate natural language from the first order logic that they use to make sentences, and to be able to understand natural language.

Courtesy mode. In addition to adding natural language processing it would also be a good idea to add a module or modules that handle other aspects of speech, such as intonation and how to phrase the sentence. Does the character want to be demanding, polite or questioning? Example: “Close the window", “Please close the window" or “Isn’t it cold in here?" For graphical display it would also be necessary to add some modules for facial expressions and body language.

Bibliography

Aarne, A. (1911). Verzeichnis der märchentypen. *Folklore Fellows Communications No. 3.*

Arinbjarnar, M. (2007). Rational dialog in interactive games. In *proceedings of aaai fall symposium on intelligent narrative technologies.* Westin Arlington Gateway, Arlington, Virginia.

Arinbjarnar, M. (2008). Dynamic plot generation engine. In *proceedings of the workshop on integrating technologies for interactive stories.* Playa del Carmen, Mexico.

Bangsø, O., Jensen, O. G., Jensen, F. V., Andersen, P. B., & Kocka, T. (2004). Non-Linear Interactive Storytelling Using Object-Oriented Bayesian Networks. In *Proceedings of the international conference on computer games: Artificial intelligence, design and education.*

Chaplin, D. J., & Rhalibi, A. E. (2004). IPD for Emotional NPC Societies in Games. *Proceedings of the International Conference on Advances in computer entertainment technology.*

Davis, D. N., & Lewis, S. C. (2003). Computational Models of Emotion for Autonomy and Reasoning. *Informatica (Special Edition on Perception and Emotion Based Reasoning)*, *27*(2), 159-165.

DSL. (2007a). *Genie modeling environment for graphical probabilistic model decision systems laboratory, university of pittsburgh (http://dsl.sis.pitt.edudsl.sis.pitt.edu).*

DSL. (2007b). *SMILE, reasoning engine for graphical probabilistic model.* Decision Systems Laboratory, University of Pittsburgh (http://dsl.sis.pitt.edudsl.sis.pitt.edu).

Fairclough, C. R., & Cunningham, P. (2004). AI Structuralist Storytelling In Computer Games. *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education.*

Fudenberg, D., & Tirole, J. (1991). *Game Theory.* The MIT Press.

GeNIe modeling environment for graphical probabilistic model. (2007). Decision Systems Laboratory, University of Pittsburgh (http://dsl.sis.pitt.edudsl.sis.pitt.edu).

Harsanyi, J. C. (n.d.-a). Games with incomplete information played by "bayesian" players, i-iii. part ii. bayesian equilibrium points. *Management Science*, *14*(5), 320–334.

Harsanyi, J. C. (n.d.-b). Games with incomplete information played by "bayesian" players, i-iii. part iii. the basic probability distribution of the game. *Management Science*, *14*(7), 486–502.

Harsanyi, J. C. (n.d.-c). Games with incomplete information played by "bayesian" players, i-iii. part i. the basic model. *Management Science*, *14*(3), 159–182.

Harsanyi, J. C. (1995). Games with Incomplete Information. *The American Economic Review*, *85*(3), 291–303.

Heap, S. H., Hollis, M., Lyons, B., Sugden, R., & Weale, A. (1992). *The Theory of Choice*. Blackwell.

Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer-Verlag.

Jensen, F. V. (2006). (Private communication)

Koller, D., & Milch, B. (2003). Multi-Agent Influence Diagrams for Representing and Solving Games. *Games and Economic Behavior*, *45*(1), 181–221. (Full version of paper in IJCAI '03)

Lindley, C. A., & Eladhari, M. (2002). Causal Normalisation: A Methodology for Coherent Story Logic Design in Computer Role-Playing Games. *Proceedings of the International Conference on Computers and Games*, 292–307.

Nash, J. (1951, September). Non-Cooperative Games. *Annals of Mathematics*, *54*(2), 286–295.

Plato. (1967). *Plato in Twelve Volumes, translated by W.R.M. Lamb* (Vol. 3). Cambridge, MA, Harvard University Press; London, William Heinemann Ltd.

Propp, V. A. (1968). *Morphology of the Folktale* (2nd ed.). University of Texas Press.

Python. (2007). *The Python Programming Language.*

Reisig, W. (1988). *Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets*. Springer-Verlag.

Rhodes, B. J. (1996). *PHISH-Nets: Planning Heuristically in Situated Hybrid Networks.* Masters of Science thesis in Media Arts and Sciences at Massachusetts Institute of Technology.

Riedl, M. O., & Young, M. R. (n.d.). Character-focused narrative generation for execution in virtual worlds. In (pp. 47–56). Germany: Springer-Verlag.

Schneider, O., Braun, N., & Habinger, G. (2003). Storylining Suspense: An Authoring Environment for Structuring Non-Linear Interactive Narratives. *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision.*

Si, M., Marsella, S. C., & Pynadath, D. V. (2005). Thespian: Using Multi-Agent Fitting to Craft Interactive Drama. *In Proceedings of the International Conference on Autonomous Agents and Multi Agent Systems*, 21–28.

Silverman, B. G., Bharathy, G., O'Brien, K., & Cornwell, J. (2006). Human behavior models for agents in simulators and games: part II: gamebot engineering with PMFserv. *Presence: Teleoperators and Virtual Environments*, 163 – 185.

Sun. (2007). *Java^{TM} 2 Platform Standard Edition 5.0, Sun Microsystems.*

Theune, M., Faas, S., Nijholt, A., & Heylen, D. (2002). The Virtual Storyteller: Story creation by intelligent agents. *Proceedings of the Workshop on Storytelling in Collaborative Virtual environments at ACM Collaborative Virtual Environments*, 95–100.

Theune, M., Rensen, S., Akker, R., Heylen, D., & Nijholt, A. (2004). Emotional Characters for Automatic Plot Creation. *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment*, 95–100.

Tolk, A. (2002). Human Behaviour Representation - Recent Developments. *presented at the Lecture Series on Simulation of and for Military Decision Making.*

Verbrugge, C. (2002). A Structure For Modern Computer Narratives. *Proceedings of the International Conference on Computers and Games*, 308–325.