Universidade Técnica de Lisboa

Instituto Superior Técnico

Departamento de Informática

Agents for Massive On-line Strategy Turn Based Games

Luís Miguel Landeiro Ribeiro

Dissertation for the degree of

Master in Science of Information Technology and Computer Engineering

Research Advisors:

Pedro Santos

Rui Prada

Setember 2007

Resumo

Nos jogos de estratégia existem tipicamente dois factores predominantes que se encontram presentes em quaisquer jogos. Um jogador veste a pele de um estratega em duas fases, ao mais alto nível controlando a gestão de recursos, investimentos tecnológicos e militares, economia. Ao mais baixo nível assume o controlo das decisões operacionais como a disposição de exércitos, coordenação e movimentos militares, compra e venda de recursos. O nível mais alto requer decisões estratégicas de médio e longo prazo, enquanto o nível mais baixo lida com decisões de curto prazo. Com introdução constante de simulações mais detalhadas os jogos tornam-se cada dia mais complexos. Complexos por serem mais difíceis de desenhar e implementar, e pela necessidade de se tornarem cada vez mais realistas.

As arquitecturas de agentes providenciam os meios para desenvolver jogos com interfaces múltiplas em ambientes complexos. Este trabalho pretende demonstrar como se pode melhorar o panorama actual de jogos de estratégia massivos on-line, com a introdução de um motor baseado em agentes. Para produzir um ambiente mais realista e divertido, cumprindo os requisitos de performance e as particularidades impostas por este tipo de jogos.

Palavras chaves

Agente, Inteligência Artificial, Jogo de Estratégia por Turnos, Jogo Multi-jogador

Abstract

Strategy games have two predominant factors that are present in every games. At the highest level, the player impersonates a strategist, managing resources; military and tecnological investiments; global economy. At the lowest level, the player assumes control over operational decisions, e.g. setting up the army disposition; coordinating armies; buys and sells resources. Mid-long term decisions are addressed at the highest level and imediate decisions are addressed at the lowest level. With the continuous introduction of details, to high and low levels simulations, the strategy games become more complex every day. The need for more realistic simulations, makes this kind of games harder to design and implement.

This work pretends to demonstrate, how massive multiplayer strategy on-line games can be improved by the introduction of an agent oriented engine. These engines make the development of complex interactions more manageable, create a more realistic and amuzing environment, keeping under control the performance requirements.

Keywords

Agent, Artificial Intelligence , Turn-based strategy game, Multiplayer game

Acknowledges

To my wife for her endless support and love, for pushing me when I lacked self motivation.

To my family for being so understanding even when I become progressively more distante.

To my friends for being there.

To my cats for always offering the right distraction when needed.

To Pedro Santos for his guidance even when the hours were long.

To Rui Prada for his technical pragmatism.

To PDM&FC and Almansur for providing the means to create great things.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Things are what they are, and behind them there is ... nothing

Sartre

"AI and emergent gameplay is demonstrably more realistic than anything that could practically be implemented in a hardcoded way" [1]

1.1 Motivation

Believability is the buzzword in the computer games artificial intelligence [1] , games focus on creating the illusion of reality. The longer it takes the players to realize the limitations of the virtual world the longer they will play.

Massive multiplayer on-line games [2] face specific problems which affect them harder than other genres. Scalability is often cited as the main issue, with thousand even millions of virtual entities (player controlled or non-player controller) roaming around a virtual world it is hard for performance to be good.

The multi-agent paradigm is increasingly more important in MMOGs because it addresses the main problems, the creation of a virtual world and the distribution of load. Agents act as individuals and constitute the most dynamic parts of the virtual world. Agents who are independent and autonomous can be decoupled from the world core, solving the scaling needs just by offloading their processing to other machines.

Developing a virtual world with the old game paradigms (everything centralized on the main *thinking* cycle), becomes a cobersome task. To ease up this task, game developers often develop/reuse frameworks to deal with recurrent core issues. This lets them spend more time solving the specific game issues.

1.2 Problem

Turn based strategy MMOGs have a peculiar set of problems to solve. Turns are processed at specific dates and therefore have a fixed time duration, during which time only player actions are collected. The

[1] AI - Artificial Intelligence is the science and engineering of making machines, with special emphasis on intelligent computer programs [2]

[2] MMOG - Massive multiplayer on-line games are played over the internet with thousands of human players at the same time

game processing only happens at the end of turn. All the processing is done together, which means the turns can take a very long time to process. Furthermore, turn processing time increases with the number of player inputs and the number of game "days" each turn represents.

A third issue arises by applying multi-agent systems to turn-based strategy massive multiplayer online games (TBSMMOGs) [3] .Agents are dormant for most of the time, only becoming active at the turn processing. This adds the need for each agent to save any internal state (at the end of turn processing) into a solid state support, and to load said state when the new turn processing begins.

Almansur is a turn based game played on-line by multiple players. Being a TBSMMOG, Almansur faces the issues presented previously and a few more specific problems:

- No single player mode, new players lack a learning playground
- No artificial intelligence, stopping a single player mode from being developed
- Players drop-out, without AI there is no quick replacement for quitters
- Repetitive tasks
- Static world
- Perfomance Issues

The major challenge of this work is to provide an alternative implementation of the Almansur, that surpasses the shortcommings of the actual version.

1.3 Goals

This work lays down an agent oriented architecture to introduce AI into Almansur. The architecture has the following objectives:

- **Flexibility** - The framework should allow for the development of new content and agents without significant addicional effort
- **Performance** - The framework performance must be good enough to support thousands of agents running at the same time
- **Ease of use** - No special configuration should be needed for things to work out of the box
- **Configurable** - The framework should provide the means to create special unique agents only by customizing their behaviors
- **Human Interaction** - The human players should be able to influence how their subjects agents's behavior

The developed architecture is applied to Almansur to create a second version of the game. Version 2.0 aims to surpass the original version on three levels: flexibility, performance, simulation realism.

[3]The combination of turn-based strategy games with massive multiplayer online games

1.4 Contribution

The main contribution of this work is the introduction of AI capabilities into turn-based strategy massive multiplayer on-line games. This is new ground, since no such game is known to date. The ability to influence the agents behavior by direct orders of the players is also a contribution to this active research field of human-computer interaction. A new domain specific language, simple yet powerfull, where the agents behavior can be configured without changing the core code or, even more powerfully, it can be changed by the agent itself, to simulate the learning of new abilities or even create offsprings.

1.5 Outline

On Chapter 2 the core concepts behind games artificial intelligence and academic research are explained. These are the bases required to understand the main problems addressed by this thesis. Chapter 3 focuses on games with outstanding AI, which are some of the best development of game AI in the last years. Analyses the AI techniques commonly used in games. On chapter 4 the version 1.0 of Almansur which serves as base for the current's work implementation is presented. Chapter 5 addresses architecture developed to introduce multi agent processing into TBSMMOG in general, the implementation of this architecture can be seen on chapter 6 Chapter 7 details the test taxonomy and tests performed followed by the results. Chapter 8 presents the conclusions and brings suggestions for further work.

Chapter 2

Common Techniques

2.1 Introduction

The main focus of academic research is to build smart AI and ultimately create an AI that can fool a human inspector on the Turing test. The Turing test is a proposal made by Alan Turing in the 1950's, consisting in having a human expert inspector interviewing two parties through natural language, one comprised by a human and the other by a computer. The test would be successfully passed by the AI if the human inspector was unable to determine which party was the computer by the end of the test. Up until now this kind of high level cognitive ability has not been delivered by the academic research and is considered by many the Holy Grail of AI research, a computer intelligent enough to fool the human inspector could be declared truly intelligen and at a human level on the intelligence scale.

For the past 50 years the AI field has been blooming and producing many techniques in an attempt to simulate human-level reasoning, such as finite state machines, rule based systems, planning. These techniques lacked the ability to learn progressively the way a human brain does, giving rise to other techniques which incorporated direct learning and adaptation. Most noteworthy are the neural networks (an attempt to simulate the human brain inside a computer), decision trees (highly efficient during both the learning phase and the look-up phase) and Bayesian networks (which cope well with uncertainty).

In the past 20 years the behavior architectures have changed the landscape of the AI field. The new generation authors recognized the failure in previous attempts of the classical AI and started creating small basic behaviors and allowing the complex high level behaviors to emerge naturally (less is more) instead of embarking upon a megalomaniac venture (modeling human-like cognitive abilities). The main reasons for their success are the simplicity of development and the cost effectiveness. A psychological factor is also determinant: the highly motivating force of seeing something work. This happens because it is easier to produce a working prototype with behavior oriented architectures than with classical methods.

In the next sections the most common techniques used in games will be outlined.

2.2 Common Game Techniques

Some common techniques are frequently found in games. They are generic algorithms or techniques that are adequate for a large set of problems. Before developing any AI such techniques should become familiar to the developer for they solve complex problems and prevent one from reinventing the wheel. On the following subsections the predominant techniques used on strategy games are presented, each

with a citation reference recommended for those in need of deeper knowledge.

2.2.1 A*

A* is a generic algorithm that searchs trough a graph and finds the path from the start node to the end node. A* is very competitive with other search algorithms, it fast and easy to understand. It uses an heuristic that estimates the cost to reach a node from the current point. The heuristic determines the best node to explore in order to potentially find the optimal path. The better the heuristic the faster the search will be processed. A good heuristic is one that never overestimates the cost to reaching a node and provides a good estimate and is fast to process. [5]

2.2.2 Hierarchical AI

The hierarchical AI follows the military structure to decision making. It divides the decision process into a chain of command, each level having to deal with problem at its own granularity level. High level(Strategic decisions) are planned at the top level, which then delegates local tactics on a middle level sergeant which in turn passes the orders along to the individual troops. The lower the level the more immediate the decisions should be, and the less decision power is available there is. Decisions made at each level are isolated from other levels, this divide and conquer technique represents a big problem (control of an army) and is divided into smaller problems and conquered bit by bit. [6]

2.2.3 Terrain Analysis

Terrain analysis is a generic name given to specialized map analysis, it basically turning in-game terrain details into manageable data. Every RTS [1] needs to implement some kind of terrain analysis to identify strategic locations, resource rich zones, military ambush points, and choice of facilities or walls location.

A common problem with strategy games is the recognition of strategic dispositions and identification of the strategic dispositions of armies in a landscape. For a human player it is relatively easy to spot the location of weak and strong points self or enemy armies. It is far more difficult for a computer to do so. Influence maps can help, providing an estimate of the current army strength present in the known areas of the map. They overlap a grid over the map and create multiple layers on each grid cell. Each layer obeys a custom formula whose common principle is to sum the strengths of self and allies and subtract the strength of enemies. After the raw data is computed, negative values represent zones where the enemy is stronger and high positive values represent zones where friendly units prevail. Each layer can provide information on a different aspect of the game, *e.g.*, we can have an openness layer whose paths are free to be used; a military layer can identify where our armies exert influence and where the enemy is stronger or weaker; an area occupancy layer can identify the fog of war. Information provided by these layers can be used to make strategical decisions as to which zones to attack or which weak points should be reinforced. [7]

To assist in higher level decision making it is common to propagate the influence of some cell to his neighbors, which provided a better picture of what is may potentially happen. On the example provided by figure 2.1 the gray cells represent neutral influence areas. Blue cells represent allied units strength (The darker the stronger), and red cells represent the enemy units strength [6]

[1] Real-time strategy, often abbreviated (RTS), is a genre of computer games characterized by war games which take place in real time, with resource gathering, base building, technology development and direct control over individual units as its key components.[12]

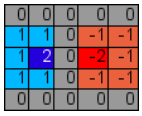

Figure 2.1: Influence Map

This is an example of symmetrical forces, if the forces are unbalanced then the resulting gradient will lose its symmetry. Areas with the greatest gradient represent the center of the mass of army. This can be used to determine the position of a unit in the battlefield (front, reserve, flank, etc..) [8]

2.2.4 Event System

An Event System manages all the event that can happen in the world environment and registers which agents want to be notified by each event type and location. Having all events centralized in an event system avoids all the issues related to agent polling. Polling requires that each agent directly contact the world to find out any events he might have interest in. Furthermore the querying agent might be required to query every other agent in its vicinity to acquire complete information about an event. It is CPU intensive task, as well as terribly inefficient. [6]

The best approach is to keep everything centralized in one place, when events occur the event system notifies all interested agents. If no one is interested, then nothing needs to be processed at all. Additionally some event processing can be cut by grouping agents together. Imagine a battlefield with all types of soldiers, if some high level event occurs that requires the commanding officer to act instead of sending this event to everyone, only the highest ranking units needs to receive it. [9]

2.2.5 Scripting

Scripting languages are extensively used in games because they are less complex to use than C or C++ and allow for faster development. They hide many complexity issues of the game, and allow for non-programmers to develop content more easily. Offloading the AI into scripting languages allows tuning of AI to be done on the fly without requiring the recompilation of the code. This tuning can also be handled by designers and facilitate the AI programmers's task since the scripting languages are usually simple to use and do not require advanced programming skills. Due to their dynamic nature it is fairly easy to provide external APIs that allow the AI to be improved without changing the game. This can be very interesting for the fan-based community, and allows for greater longevity for the game, different AI providing different experiences.[6]

Some problems persist, mainly regarding the lack of performance and the difficulty to debug custom-made script languages without the proper tools already available for more established programming languages. Offloading the work to designers can turn into a double-sided razor because their weaker programming abilities can lead to spaghetti code, which is very hard to maintain. The time to support an extra language has to be factored into the development costs as well.

2.2.6 Finite State Machine

Finite-state machines (FSMs) are finite automata with states and transitions. Each state represents an action or behavior, while the transitions represent the event that causes the active state to change to

another state. The state transition has some pre-conditions which determine whether the trigger can be activated. Since there is only one transition trigger at a time FSMs are used to encode deterministic behavior. FSMs are commonly represented with UML [2] state transition diagrams. State transition diagrams depict states as circles and transitions from state to state as directed arrows which ultimately provide a directed asserted graph. [22]

Example:

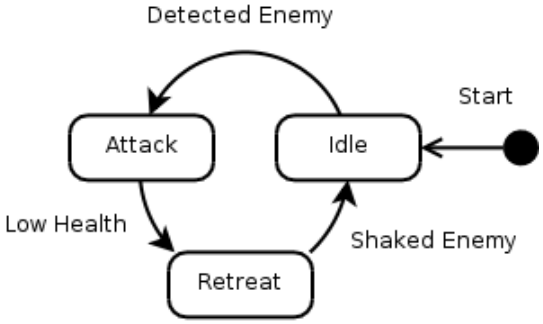

Figure 2.2: Finite State Machine Example

An agent following this state machine would start in an idle state and when an enemy is spotted the agent attacks it. When its health drops below a lower bound the agent retreats until it loses his enemy. This example shows how easy it is to build a simple deterministic behavior with a FSM.

2.3 Academic Research

Figure 2.3 shows a classification of the prominent research branchs of the Artificial Intelligence field. These fields will be described and analyzed for usefullness to produce the AI for a game throughout the next sections.

2.4 Classical AI

Traditional AI is strongly connected to knowledge representation and symbolic manipulation to produce more knowledge. This section describes the classical methods for knowledge inference.

2.4.1 Rule Based Systems

RBS were considered the state of the art for AI research in the 70's and are also know as expert systems. A RBS is built using a database to contain the facts known by the AI (expert knowledge, hence the name) and a set of IF-Then rules. The RBS matches the rules against the knowledge available in the database. More than one rule can match so a mediator is necessary to decide which rule has higher priority. The best known mediator method is the algorithm RETE [3] and is the key component for the performance of the system as a whole. The power of an RBS is directly related to its inference engine, which is also the most complex and inefficient component.

[2]The Unified Modeling Language (UML) is OMG's most-used specification and the way the world models not only application structure, behavior, and architecture but also business process and data structure. OMG is a not-for-profit computer industry specifications consortium [21]

[3]RETE is an algorithm that provides a generalized logical description of an implementation of functionality responsible for matching data tuples ('facts') against productions ('rules') in a pattern-matching production system.

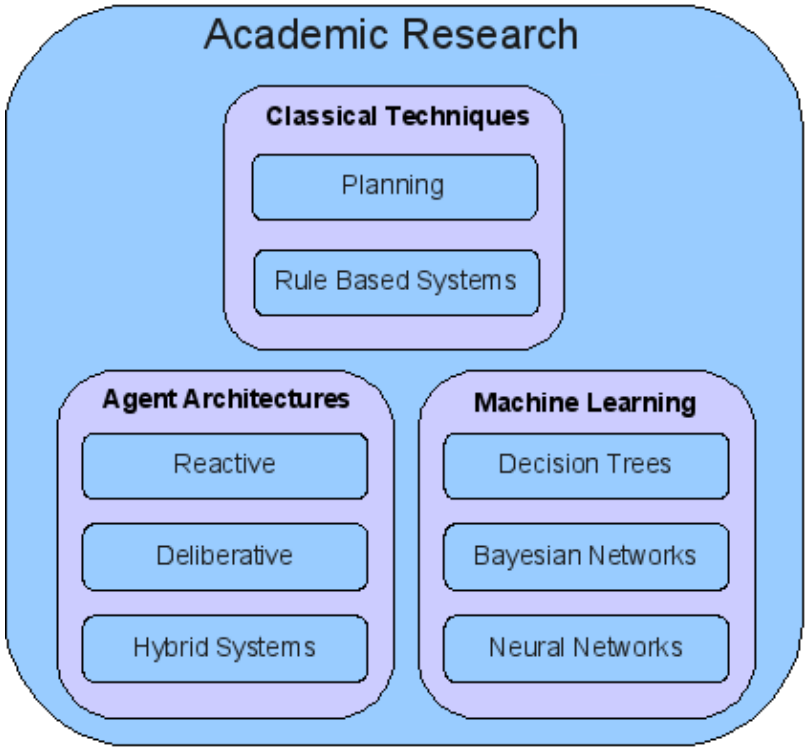

Figure 2.3: Academic Research Overview

2.4.2 Planning

During planning each agent has their goals and actions and they search through theirs means (actions) in order to attain their objective (goals). Planning can be viewed as the declarative way of creating AI.

The world state is represented by logical symbolisms, which represent the knowledge available to the AI. A planner is used on the world current knowledge to infer more knowledge. The planner is responsible for discovering which actions must be performed to execute a plan or if their plan is untenable.

Planning systems decouple actions from goals making it simple to implement layered behaviors. The decoupling allows for new behaviors to added without requering changes to the core system. The problems of planning systems are tied to the complexity of the planner engines, and to the difficulty of finding a symbolic representation that expresses all the concepts present in a virtual world.

2.5 Machine Learning

Machine Learning(ML), techniques are used to capture generic pattern from training data and create a model. This section describes the techniques that most frequently appear in games and the domains where each technique yields better results.

2.5.1 Bayesian Networks

A Bayesian network estimates the probability governing a group of variables with a specification of the conditional independence properties that apply on a subset of variables. In other words, the Bayesian network represents the probability distribution over a group of variables. The performance of Bayesian

networks has been shown to be comparable to other highly efficient ML methods such as decision trees and neural networks. Bayesian networks excel at classifying text and have achieved a notable reputation fighting email spam. In strategy games they can be used to classify incoming messages, attempting to determine the tone of the message (i.e. aggressive, friendly, neutral); in games that feature fog of war (FOW) they can predict what may exist beyond the known frontier, with a degree of probability, e.g. estimating the location of an enemy base.

For a complete reference on the algorithms used to implement Bayesian networks refer to the Machine Learning book by Tom Mitchell [19].

2.5.2 Artificial Neural Networks

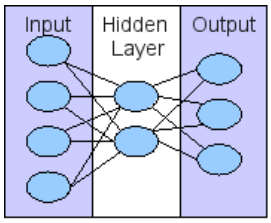

Figure 2.4: Artificial Neural Network Example

This is a method created for learning real, discrete, vector valued functions from some previously gathered train data. Inspired by biology, it tries to simulate a human brain by creating a complex network of interconnected neurons. Inside each neuron a function f($w0*x0, w1*x1..wn*xn$) = y evaluates the input to produce an output. Each neuron can receive multiple inputs (from sensors or other neurons) and only output on a single value. Typically each neuron has an associated weight (wi) with each input, plus a corrective value *x0*. The training of the network is done by modifying the weight values on each neuron to match the desired output [19].

ANN are best suited for problems where:

- The input data comes from a noisy source like a sensor camera. They excel in image processing problems.
- The time required to train the network is not an issue. This normally invalidates neural networks for in-game learning because it may take them a long time to converge into interesting models.
- The time required to evaluate the target function is restricted. After the network has modeled the detected patterns, feeding the network with custom data yields a quick response
- The model generated is not required to be human-readable. Neural networks generate complex models which most humans can not understand.

2.5.3 Decision Trees

Decision trees are a technique for estimating discrete functions by representing it in a decision tree form. The decision tree is learned from existing data and is able to produce credible results for a variety of domains even in the presence of noisy information. The Occam's Razor principle, which states that when choosing between two solutions of approximate value to the same problem the simplest one is usually the

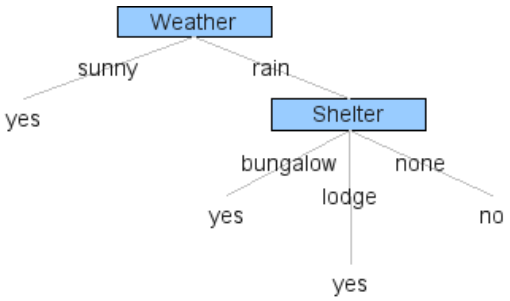

Figure 2.5: Example of a decision tree for a picnic friendly day

best, is used to choose between several different trees. This principle reduces the probability of choosing an overfitted tree for biased data and tries to capture the generic details as being the most important.[19]

Decision trees are best suited for problems where:

- The instances can be represented by key-value pairs such as "weather", which can take two values (sunny, rain).
- Discrete target functions, like "yes" or "no". Decisions trees can be expanded to support real value functions but their performance diminishes.
- The training data are error-prone or, in some instances, misses some attributes such as performing data mining over incomplete forms.
- The training data are small and the learning times are necessary to be short

2.6 Agent Architectures

2.6.1 Agent

Wooldridge and Jennings define Agent [20] as

> ... a hardware or (more usually) software-based computer system that enjoys the following properties:

autonomy	agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state
social ability	agents interact with other agents (and possibly humans) via some kind of agent-communication language
reactivity	agents perceive their environment, (which may be the physical world, a user via a graphical user interface, a collection of other agents, the INTERNET, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it
pro-activeness	agents do not simply act in response to their environment, they are able to exhibit goal-directed behavior by taking the initiative

Table 2.1: Agent Properties

An agent can be caracterized by several more properties, but for us this are the most important for the current work.

Pattie Maes has a another definition:

> A Software Agent is a computational system which has goals, sensors, and effectors, and decides autonomously which actions to take, and when

While there is no univeral definition of what an agent is, in this work an *agent is something that perceives information from an environment, possesses some kind of effectors that allow him to perform actions that modify the surrounding environment.* An agent is, therefore, the composition of its architecture and the mind that controls its actions.

2.6.2 Agent Types

There are a variety of agent designs, most noteworthy [28]:

Reflex agents	they respond immediately with some action for a given perception
Reflex agents with internal state	they keep record of passed events to help guide them on their action decision
Goal-Based agents	sometimes percepts are insufficient to determine the proper path of action. This happens when the right action is dependent of a specific goal
Utility-Based agents	when several actions are possible at a given situation the agent mind needs a way to choose the best one. If it can estimate the potential value of the next state, it can determine what action would bring the higher return. These agents possess an utility function that maps a world state in a numeric value. The agent uses these values to determine the quality of the competing actions
Learning Agents	they can become better as they become more experienced. Every agent can be turned into a learning agent by evolution means. For example, a reflex agent can dynamically change his tripping points to perform better under different environments

Table 2.2: Agent Types

2.6.3 Environment

The environment is the world where the agent lives. Each environment can be classified on the following properties

Accessible vs Inaccessible	On an accessible environment, the agent's sensors are able to fully determine all the relevant properties. Environment can be full observable, partially observable or inaccessible
Deterministic vs Non-Deterministic	An environment is deterministic if the next state of the environment can be completely determined by the current state and the action of the agent

Episodic vs Sequential	An episodic environment means that the actions only affect one state, and that states are totally independent
Static vs Dynamic	Static environments do not change while the agents are deliberating. Semi-dynamic environments do not change, but affect the agent's performance as time passes.
Discrete vs Continuous	Environments are discrete if the number of distinct percepts and actions is finite and limited by the environment
Single vs Multi-Agent	If more than one agent lives on the same environment, its a multi-agent environment, otherwise it's single agent. Multi-agent environment can be competitive or cooperative or both

Table 2.3: Environment Properties

As an example the real world is partially observable, stochastic, sequential, dynamic, continuous, multi-agent. [28]

2.6.4 Multi-agent Systems

Multi Agent systems are platforms for several agents to interact at the same time. Two academic definitions exist for the term multi-agent system (MAS). The first one from Distributed Artificial Intelligence (DAI)

> "As seen , a multi-agent system is a loosely coupled network of problem-solver entities that work together to find answers to problems that are beyond the individual capabilities or knowledge of each entity" [3]

The second and more generic and recent one

> "All types of systems composed of multiple autonomous components showing the following characteristics "[4]:
>
> - each agent has incomplete capabilities to solve a problem
> - no global system control
> - data is decentralized
> - computation is asynchronous

2.6.5 Brook's Subsumption Architecture

Developed by Rodney Brooks [20] as an alternative form of creating AI agents with the limited resources available, his work is based upon three fundamental bases and two main ideas

1. Intelligence without symbolic representation
2. Intelligence without abstract reasoning
3. Intelligence as an emergent property from simples systems interacting together to provide complexity

The ideas surface from their fundamental bases, and Brooks advocates that real intelligence can only be seen in conjunction with a world body in a specific situation. As a result, only embodied agents with real actions can be considered intelligent, automatic theorem provers are just programmed machines. The second idea relates to emergence behaviors, AI cannot be considered a property of some sort, but it should be new unexpected behaviors should be able to surface from an embodied agent facing unanticipated situations in a virtual or real world.

The Figure 2.6 picture exemplifies the generic subsumption architecture.

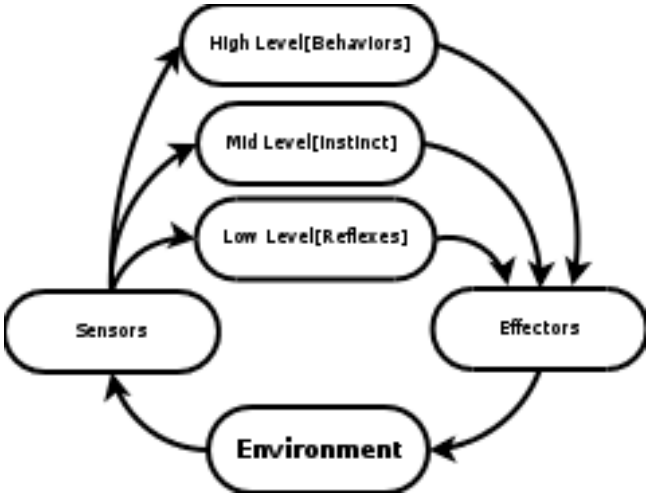

Figure 2.6: Subsumption Architecture

The agent's decisions are chosen from a set of task-oriented behaviors which are built over FSMs without any symbolic world representation. The world changes at such a fast rate that any model would always be outdated. Each behavior maps a situation given by the sensor to a set of actions that are mapped on the effectors. The lower levels provide reflex-like behavior, have the highest priority and deal with more concrete matters. The higher levels become progressively more abstract and low/mid term behaviors. Because the lower levels have higher priority, if they are always being used, the higher levels may suffer starvation (are blocked from use due to more immediate actions required).

This architecture shines in its simplicity, ease of development and robustness against failure. If a component fails the others will fill in, albeit with lesser capacity. The greatest novelty brought by this architecture is the emergence behavior that is reckoned with intelligence.

Without any world models only local information is available which hinders the elaboration of long-term plans. This, added to the fact that the agents are purely reactive, makes this architecture ill-suited for non-local actions or creating adequate emergent intelligence for long-lasting plans.

2.6.6 Belief, Desire and Intention (BDI)

Developed by Michael Bratman it is an attempt to model human reasoning. This architecture is characterized by the three properties an agent has:

Beliefs – The information about the environment surrounding the agent. This is subject to error, it being the world as the agent can see it.

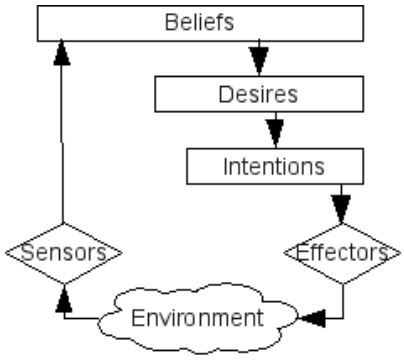

Figure 2.7: Generic BDI

Desires – The agent's goals

Intentions – The commitments made by the agent to achieve certain goals, the plans currently in execution.

2.6.7 Hybrid Architectures

Hybrid agent architectures have some multiple layers, which can be classified into three different types: reactive, deliberative and social. The main difference for the subsumption architecture lays in the higher level layers, which can include world representation and planning capabilities. The hybrid systems tries to explore the best part of each architecture and solve the shortcomings of subsumption. The deliberative layers can provide long-term planning abilities, and the social layers provide the share of communication between different entities.[20]

Two types of layer stacking are possible: horizontal and vertical.

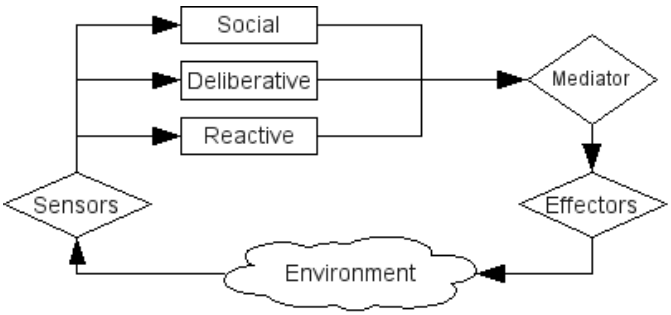

Figure 2.8: Horizontal Architecture

In the horizontal approach each layer is connected to the inputs and runs in parallel, and one output per layer can be suggested. It is up to the mediator to decide which behavior should be executed. The main advantages of this approach are the conceptual simplicity and the possibility of ensuring each layer has only one behavior. The major drawback is the need for an explicit mediator, which becomes the bottleneck.

In the vertical approach each layer has an input and output interface with the direct higher layer and direct lower layer. The perceptions are received by the reactive layer, with each layer having the responsibility of either forwarding the perceptions upwards or deciding upon

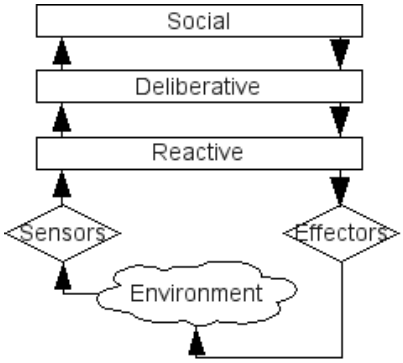

Figure 2.9: Vertical Architecture

a course of action. The higher layers delegate the plan implementation details they create on the lower layers, but because the lower layers have a higher priority they can sometimes ignore the orders they received if more pressing matters need to be addressed. Each layer processing is done in serial mode and a layer can filter the information from the higher layers. Under the right conditions this behavior can starve the higher level layers. This approach is less flexible than the horizontal layers because the 1-layer 1-behavior mapping is lost.

Chapter 3

Case Studies

3.1 Introduction

To create a next generation agents for a strategy game it is wise to analyze what other companies have done in the past, learn from their experience and establish a high standard to be met. Many games drive the AI innovation with fun as the primary motivator, from the high paced first-person shooters [1] to the patient turn based strategy games [2] To create a rich virtual world where the players like to interact with agents, nothing less than the current best will suffice. Players are more demanding by the day and require extreme in-game realism, both in graphics and AI.

In the next sections an analysis of some successful cases is presented in the hopes of determining which technique is best suited for a strategy game. The focus of the analysis will be on three different game genres. Firstly, the FPS are introduced as they are the game genre where higher competition exists, which calls for outstanding innovation in the AI field to allow them to stand out from all others. Secondly, RTS [3] games are described, which are the main genre for strategy games and still share all fundamental problems with the third genre, TBS.

3.2 FPS Games

3.2.1 Introduction

First-person shooters are among the genres which recently started requiring smarter AI. The new generation games Fear and Far Cry both achieved success within the game community with innovative AI implementations. Far Cry agents possess group behavior capabilities akin

[1] First-person shooter (FPS) is a genre of computer and video games which is characterized by an on-screen view that simulates the in-game character's point of view and a focus on the use of hand held ranged weapons [11]

[2] A turn-based game, also known as turn-based strategy (TBS), is a game where the game flow is partitioned into well-defined and visible parts, called turns or rounds. For example, when the game flow unit is time, turns represent units of time such as years, months, weeks or days. A player of a turn-based game is allowed a period of analysis (sometimes bounded, sometimes unbounded) before committing to a game action, ensuring a separation between the game flow and the thinking process, which in turn presumably leads to more optimal choices. Once every player has taken their turn that round of play is over, and any special shared processing is done. This is followed by the next round of play. [13]

[3] Real-time strategy, often abbreviated (RTS), is a genre of computer games characterized by war games which take place in real time, with resource gathering, base building, technology development and direct control over individual units as its key components.[12]

to models used in real military chains of command[14]. Fear stands out by using real time planning along with a good enough performance. Doom 3 arrived at nearly the same time with an old school zombie type AI and was quickly dismissed by many players who were critical of its poor AI [15]. The analysis itself will focus on Fear, as time and space constraints do not allow for a detailed analysis of every each individual game previously referred to.

3.2.2 Fear

Description

Figure 3.1: Fear Screenshot

> "F.E.A.R., as a first-person shooter, focuses on combat taking place in the first-person perspective. The game is entirely witnessed through the protagonist's eyes. The protagonist's body is fully present, allowing the player to see his character's torso and feet while looking down; also, within scripted sequences, such as rising from a lying position or fast-roping from a helicopter, the hands and legs of the protagonist can be seen performing the relevant actions" [10]

F.E.A.R.'s AI allows computer-controlled characters a large degree of action; enemies can duck in order to move through crawlspaces, jump through windows, vault over railings, jump down to a lower level, climb ladders, push over large objects to create cover, and flank players. Various opponents may act as a team, taking back routes to surprise the player, and using suppressive fire or taking cover if under fire.[16]

Architecture

Like the majority of games, under the wood F.E.A.R. uses a FSM to control character. Fear goes a step further by including a planner to dynamically compute the best course of action, based on a goal oriented architecture. The planner is a modified version of STRIPS [4] with a custom made knowledge representation based on the formalized logic symbolisms specified by STRIPS. The FSM for characters in F.E.A.R has only three states:

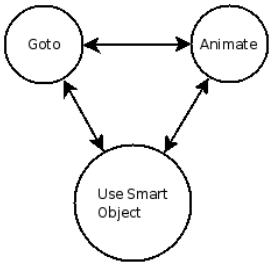

Figure 3.2: Fear Finite State Machine

Goto	Moves to some location
Animate	Default action on some target entity
UseSmartObject	Custom version of animate

Table 3.1: Fear Finite State Machine's States

In concept it is only a two state FSM because UseSmartObject is as a special version of Animate. The AI decision making is just moving or playing animations, the logic behind the state transitions (when to move from one state to another) is where the AI team of F.E.A.R. innovates.

Instead of hard-coding the AI logic, the F.E.A.R team created a data-driven architecture and stored the tunable data on an external repository so it could be changed without requiring recompilation of the core code. The AI logic remained in the data that are inserted into the planning system [5] (Data-oriented AI).

The planning system keeps the complexity of evolved behavior manageable. If every behavior were to be implemented in hard coded FSM the number of states and transitions would be terribly high.

To perform the planning the developers used a modified version of STRIPS. The planner uses a specific knowledge representation using logic formalized symbolisms. The major concepts are behind planning, states, actions, goals. A state is a current view of the environment, an actions is an update on the environment only possible if some pre-conditions are met, a

[4]STRIPS was developed at Stanford University in 1970 and the name is simply an acronym for the STanford Research Institute Problem Solver. STRIPS consists of goals and actions, where goals describe some desired state of the world to attain and actions are defined in terms of preconditions and effects. An action may only execute if all of its preconditions are met, and each action changes the state of the world in some way [16]

[5]Planning is an AI technique that formalizes the process of transversing a search space looking for a sequence of actions which accomplish a desired goal

goal is the objective state (what is required to be accomplished). Each state of the world is represented by an assignment over a set of variables. Each goal is also represented by a partial or full assignment over a set of variables and can be understood as the target state, and each action is defined as a set of pre-conditions represented by a state or partial state (partial assignment of the set of variables) and its effects (assignments of variables it changes). To help visualize it here are some examples:

Logical Formalism:	Enemies(None)^At(Warehouse)	
Assignment:	(Enemies,At,Killed)	= (None,Warehouse,0)
Meta State:	(Enemies,At,Killed)	
State:	(Enemies,At,Killed)	= (None, Warehouse,0)
Goal:	First Kill (Killed)	= (1)

Table 3.2: Definitions

Action	Meta State	Preconditions	Postconditions
Goto(X,Y)	(PosX,PosY)	Not (X,Y)	(X,Y)
Attack	(Visible,Killed)	(Enemy, 0)	(Not Enemy, 1)

Table 3.3: Actions

Action	Meta State	Preconditions	Postconditions
Goto(1,1)	(PosX,PosY,Visible,Killed)	(0,0,-,0)	(1,1,-,0)
Goto(2,2)	(PosX,PosY,Visible,Killed)	(1,1,-,0)	(2,2,-,0)
Attack(Enemy)	(PosX,PosY,Visible,Killed)	(2,2,Enemy,0)	(2,2,-,1)

Table 3.4: Plan

To reach a goal the planner will search through a sequence of actions which can change the current world state to the goal state. F.E.AR.'s AI is given a set of goals each with its own priority, the goals are fed to the planner which then produces a plan to satisfy the highest priority goal attainable. To satisfy this goals a set of possible actions is specified for each agent's personality. This allows for different personalities with the same goals to react differently when facing the same situations.

Planning can be a CPU expensive task as the search space grows to an unmanageable size with just a few dozen states. To work around this issue the F.E.A.R. team added a cost to each action. This allows the introduction of A* to guide the search through the cost metric and find the lowest cost sequence of actions which meet a goal [16].

3.3 Strategy Games

3.3.1 Introduction

Most strategy games follow the 4X axiom. 4X refers to a genre of strategy game, usually a computer game, with four primary goals: explore, expand, exploit and exterminate. A 4x game can be turn-based or real-time. The 4X stands for the following goals [17]:

- **explore** – The player starts off with a limited knowledge of the playing area and must send some units into unknown areas to map it. The map can start all blacked out or partial information may be visible
- **expand** – The player needs to conquer more territories to do his bidding, create new settlements or improve the existing ones
- **exploit** – The player uses owned resources to produce the highest benefits possible
- **exterminate** – The player conquers his neighbors lands as part of his expansion plan which in turn enables him to exhaust their resources instead of their own

4X games fall into one of two categories, real time or turn-based strategy games, both with specialized needs but sharing most of the AI problems. The following games provide the state of the art for the RTS and TBS genres.

3.3.2 Empire Earth

Description

Empire Earth is a 4X strategy game. It is an RTS game with multiple historic scenarios. The game has some remarkable innovations, namely the morale system that allows for better military combat simulation and a hero system that allows for unique specialized units which award combat bonuses to any army they command.

Architecture

Civilization Manager - responsible for developing economy. Coordinates the build, unit, resource sand research managers giving them their budgets. Handles the exploration and expansion as well as buildings and units upgrades. Also decides when the AI should advance to the next era.

Build Manager - responsible for choosing the location where facilities and walls will be built. Other managers request him to select the optimal place for a given facility. Uses terrain analysis to provide facts that serve as the basis for the decision process.

Unit Manager - responsible for training units and for tracking the total unit count. This module tries to match the human players current strength to provide an adequate challenge. This prevents overwhelming human players with military units and them being dead in the water when the human player finally attacks.

Figure 3.3: Empire Earth, in-game screenshot

Resource Manager - responsible for selecting which resources the villagers should gather. Also responsible for exploring new areas. The priority of resource gathering is updated with the requests arising from other managers. For instance the build manager wants to build a given facility but lacks the necessary resources, it then asks the resource manager to gather the missing resources at high priority.

Research Manager - responsible for determining which technologies should be pursue by the AI by estimating their usefulness per cost ratio

Combat Manager - responsible for commanding units on the field. It coordinates group attacks and defense of weak or valuable positions. Performs multiple terrain analysis to identify weak spots on the enemy stronghold and its own base, identifies resource-rich locations and finds potentially hazardous locations where units can be ambushed.

Details

Each manager has a channel of communication to every other manager. And through this channel it can send a request that is available to the other manager public API. This architecture assumes every modules is trustworthy and that everyone works for the common good. A single disloyal manager could bring all cooperation to a halt. The managers are given the liberty to accomplish the tasks they are responsible for. For tasks dependent on multiple managers all managers need to agree on the same task. The tasks that are not immediate

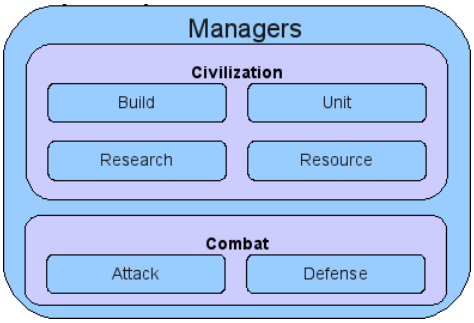

Figure 3.4: Empire Earth Architecture [9]

require some wrapping up in the form of a callback to notify the starting manager that they have been completed.

3.3.3 Axis & Allies

Description

Figure 3.5: Axis & Allies, in-game screenshot

Axis & Allies is a 4X RTS & TBS game where players will relive and experience the most epic struggle in the history of mankind, World War II. Players will be able to direct the military and economic destiny of any one of the world's most powerful countries - United States, England, Germany, The Soviet Union or Japan. Confronted with the strategic and

tactical situations experienced by the top generals and national leaders of the period, players will have to make critical decisions that determine the fate and the destiny of world [18]

Architecture

Figure 3.6: Axis & Allies Architecture [24]

Goal Manager - Responsible for making the strategic decisions. Determines which goals to pursue, which actions are required to achieve the selected goals and the location where said actions should be undertaken.

Actors - Entities that are directly or indirectly controlled by the player or AI, that have some in-game effect. Examples: Buildings, Units.

Goals - Represent every action the player or AI can undertake while following the game rules. Examples: Explore, Attack, Recruit, Build, Repair. Each goal has four properties: list of resources, status, base priority, current priority.

- List of Resources, the limited entities or commodities currently assigned to a goal
- Status, the current status of the goal, can be one of the following:
  - Active – The goal is ready to be executed
  - Inactive – The goal can not be executed right now, some preconditions is missing
  - Selected – The goal is in executions
  - Finished – The goal can no longer repeat itself and can be deleted
- Base Priority, the priority of a goal at the current world state (discarding the resources it requires to be executed)
- Current Priority, base priority weighted with the resources required for the goal to be executed, e.g. if a building is under attack a goal for that building has the base priority of the building value, and the current priority equals the building values multiplied by the unit force ratio between attacking and defending units .

Resources - Represent any commodity that is limited and therefore act as a strategic factor (e.g. Gold, Iron, Wood).

Details

The Axis & Allies use a hybrid AI architecture [6], with intelligent behaviors (Strategical Plans) provided by deliberative AI and immediate decisions and plans details worked out by the reactive AI. The reactive layer is always on, while the deliberative layer only runs once or twice a minute.

Terrain Analysis - The AI needs to retrieve information from the world convert into into facts it can understand and act on those facts. The conversion is done by three different terrain analysis techniques, region analysis, path-finding and influence maps. The region analysis determines the space and dimension of the a map zone with the same terrain topology (water, mountain, plain) this information is then used by the path finding and influence maps to optimize them. Path finding is done hierarchically to connect conceptual regions, which breaks down the global path finding into smaller less expensive region path findings. The influence maps determine the military balance of a certain region.

Egos - Even tough the AI decides on its own what actions to take, a set of configurable parameters influence the AI deciding thresholds. An assignment of variables over this set of parameters is called an ego, and it is simple to create multiple egos and in-game diversity. Each ego can represent a different strategy with, say, some focused on expansion and others on economic development. The AI chooses which ego estimates will fare better in the current situation.

Deliberative AI - The goal engine is checked at regular intervals because each iteration over the think cycle is CPU intensive. This window of time can not be too short or it will bring the game to a halt. The window slide can be specified by each ego but some exceptions can be created using an event-driven system.

Main Cycle:

1. Select the ego for this iteration
2. Delete finished goals and cull inactive goals from the decision list
3. Compute the base priority for each active goal
4. Assign initial resources to goals. A greedy algorithm that selects higher priority goals first and gives just enough resources to match the base priority or all the available resources if the first can not be accomplished
5. Optimize resource allocation. Tries to move resources around active goals to maximize the total current priority
6. Marks each goal with current priority >= base priority with selected status

The key behind this architecture is the goal priority selection, and the prioritizing mechanism presents three problems:

- Balance between predictable and random. If the priority is always computed in a deterministic way, the AI will always play the same way and will be highly predictable. To

[6] See Agent Architectures for more details on hybrid architectures

work around this issue a small random value is added to each base cost. If this value is too small the AI remains highly predictable, on the other hand, if the value is too high the AI will simply exhibit have random behaviors. A delicate balance must be established by trial and error.

- Flipping around same priority goals. To ensure that the AI sticks to a plan a small offset is added to the base priority of each selected goal. This prevents AI from switching back and forth between equal priority goals. Again, a delicate balance needs to be achieved for if the value is too high the AI can become stuck on a previous goal even after the world situation has changed dramatically.
- Long time goals (bigger than one main cycle iteration). If no special mechanism is used the long-term goals are never selected by the AI because they lack the necessary resources spent on cheaper goals. To solve this issue the AI determines how it needs to save resources to be able to select a long-term goal and if this value is lower than a value stipulated by the ego. The long-term goal is selected to acquire the necessary goals but it is executed only when the full amount of resources is reached.

Reactive AI - While the ego is dormant between being main cycle iterations a faster reactive AI takes the lead. This reactive module is responsible for carrying out the orders laid out by the ego and dealing with immediate problems.

3.4 Analysis

3.4.1 Taxonomy

The three game genres presented (RTS, TBS, FPS) will be analyzed in 4 areas:

- Complexity - How elaborate the gameplay is. Complexity indicates how hard it will be to play the game.
- Learning Curve - How long it will take to learn to properly play the game. Complex games typically have higher learning curves.
- Satisfaction - How long it will take for the player to feel good about playing a game. This measures some sense of achievement.
- Results - How strong the satisfaction will be.

The previous areas help us understand the different nature of each game genre. The following areas provide insight into the different AI implementations and what we can expect from AI.

Each game will have a detailed comparison on the following criteria:

- Success - How important was the AI was for this game success?
- Fun - How much does the AI contribute to this game fun factor?
- Scripts - Does it support scripted scenarios well? Are specific game flows supported?
- Adaptation - How does the AI fair on new worlds?

3.4.2 RTS vs TBS vs FPS

The next table provides a comparison between the two genres.

Genre	*RTS*	*TBS*	*FPS*
Complexity	Medium	High	Low
Learning curve	Gradual	Gradual slower	Steep
Satisfaction	Fast	Slow (Turn Time)	Immediate
Results	Moderate	Strong	Weak

Table 3.5: RTS vs TBS vs FPS

Complexity

FPS normally have a straightforward story and are quite simple to play so their complexity is low. RTS have a fair amount of complexity but are usually fairly easy to learn. TBS are complex by nature and take a longer time to get used to.

Learning Curve

On TBS games the learning curve is gradual and slow, the players only becoming comfortable with the game after having played it a few times since these games are complex and hard to master.

RTS games are very similar to each other so the players already are somewhat familiar with the gameplay, and even though they take a little time to master they are easier than TBSs.

The fast-paced action of FPS makes it necessary for the gameplay to be very easy to learn so the learning curve is very steep and the players are able to master the controls in a very short while.

Satisfaction

Complex games require a greater time investment to produce satisfaction but also can provide the most intense return. TBSs are then amongst the games with slower retribution of satisfaction due to their slow nature and complex gameplay.

RTS games are faster to provide satisfaction than TBS, they are, after all, played in real time but the player still needs to finish a level to feel some sense of achievement.

At the other end of the satisfaction specter are the FPS, which you derive satisfaction from from the very first kill.

Results

On FPS games, results come in small doses of instant satisfaction. RTS provide stronger satisfaction but it takes longer to achieve it. TBS requires the longest time to provide the high satisfaction levels.

3.4.3 Fear vs Empire Earth vs Axis & Allies

The next table provides a comparison between the two genres.

Game	*Fear*	*Empire Earth*	*Axis & Allies*
Success	Medium	High	Very High
Fun	High	High	Very High
Scripts	Poor	Good	Good
Adaptation	Good	Excellent	Very Good

Table 3.6: Fear vs EE vs AA

Success

AI is crucial for any non-multiplayer TBS, the player's only objective is to beat the computer. The computer must put up a decent fight to give the player the feeling of achievement or else the player will lose interest. Even on multiplayer TBS it is important to have some kind of AI to create better simulations and to replace drop-out players.

On RTS games AI also plays an important role, in single player games the military part of the game is very tactical and requires a degree of intelligence to be stimulating.

FPS games can stand out with good AI implementations but the graphics still typically hold the key factor for the game's success.

Fun

In strategy games AI is fundamental, either it cheats to be competitive or it performs well on the same level as the human player. Without AI you do not have a game and this is even truer for turn-based games.

FPS can depend on other features because they provide satisfaction in lesser and more frequent doses. Still, a good AI is funnier to play with than dumb zombies. Because good AI and large numbers can be too much for a player to handle, FEAR and FPS dumb down the AI. This dumbing down can easily increase the fun of an FPS. Methods used in FPS to improve fun when facing multiple enemies are:

- Kung-Fu style - only some of them attack the human player, the others are honorable enough to await their turn
- Reduced damage - opponents‘ damage is proportionally reduced with the number of enemies
- Reduced accuracy – opponents' accuracy is proportionally reduced with the number of enemies

Scripts

FEAR's planning system makes it very hard to follow specific behaviors on some scenes. The AI plans are computed dynamically by a planner. Scripted actions are not easilly supported by them, hence the difficulty to support custom behaviors.

Empire Earth's AI supports templates, which are particulary useful for specifying building combinations. Axis & Allies needs custom events to support historical gameplay so scripts are

well supported at the core. Any game attempting to recreate past events requires scripting of some kind.

Adaptation

All three games have adaptative AI's. It is less of an effort to create an AI that does well in random situations than to hardcore AI for each level of a game. Also for games with multiplayer modes it is very important for AI's to be able to cope with different and unexpected scenarios.

3.5 Conclusions

In this chapter we analysed three case studies: F.E.A.R., Empire Earth, Axis & Allies. F.E.A.R. was selected for being a successful FPS game with a publicly acclaim AI. Empire Earth is an RTS that possesses many interesting AI concepts, like hierarquical AI that can be applied to TBS games as well. Finally we analysed Axis&Allies to help us understand what special problems a TBS game has to overcome.

A few techniques prove fundamental on every genre. Path-finding always requires an A* star of some kind to be efficient. Creating an in-game storytelling still requires scripting to work out the rigid details. It is important to have knowledge of those techniques and they are absolutely necessary to avoid reinventing the wheel.

F.E.A.R shows that it is possible to use academic research techniques in FPS games, techniques which once where considered CPU intensive now provide decent results in real time games, even though some changes needed to be introduced to make the planning more efficient on the domain at hand. This sort of innovation shows the best bonding between game industry and academic research and has opened up the door for future collaboration. If any pertinent conclusions can be drawn from F.E.A.R we can foresee that the proper way to bring innovative research from academia into the games industry will be by slightly modifying the researched techniques to solve the right problems for games. In the game industry the resources available to AI are much scarcer (they need to be shared with other parts of the game), thus requiring greater resource efficiency.

RTS are among the game genres where it is harder to design good AI's. Empire Earth developers took a wise path and designed a clean hierarchical AI that serves as reference for any RTS. Each manager solves a domain of problems, avoiding the cluttering of centralizing everything. This allows for emergent behavior to occur: if the combat manager sends a message to the unit manager requesting a unit from the next epoch the unit manager recognizes the restrictions involved in building such a unit and sends off a request to the build manager to consider the epoch advancement as a goal.

Axis & Allies mixes RTS and TBS and as such deals with the unique challenges presented by turn-based games. In TBS the AI cannot be predictable because the game has fixed turn times and the human player can replay the exact same game in different time frames. Where is the fun in that? But a greater problem surface: a dedicated player can reverse-engineer all AI decisions by trial and error, which eventually leads to the discovery of exploits, creating shortcuts to finish the game not anticipated by the developers. For these reasons it is necessary to introduce some random factors into the decision-making process. A&A has done it with

two different methods: firstly, they created multiple egos for the same entities and one ego is chosen at the beginning of each think cycle. This variation factor is controlled by the designers and can be tested deterministically. The other factor is the introduction of small random values to the base cost of each goal, ensuring the player, receives a different experience every single time.

Chapter 4

Almansur 1.0

Almansur is a massive multiplayer on-line turn-based strategy game. Being a web game, it requires an internet connection and a browser to be played. It focuses on the operational warfare with deep influences from classical board games. The players face each other in historical and fantasy worlds on multiple games at the same time. Each player can play any number of games in parallel and, each game server can run thousands of games at the same time.

4.1 Description

> "Almansur is a strategy game of politics, economy and war, set in the early middle ages and in some scenarios, fantasy world. You play the role of a lord, to whom fell the job of guiding your land to greatness and glory. " [25]

Version 1.0 was released in 2007, after a one year of beta testing. The game is targeted at hardcore board players. Players who prefer accuracy of simulation and details to trivial and over simplified games.

Almansur in seven sentences [26]:

1. In the game, each player controls a Land, the objective being to increase the size, richness and power of that Land
2. Each Land is made of territories (represented by hexes on the map)
3. Territories are the economic and recruitment cells of the Land, and possess natural resources
4. Each territory can have a Facility of each type and different Populations living there
5. Facilities can be built in a territory and give economic or military benefits
6. Resource gathering facilities are only useful in territories which are rich in the specific resource
7. Armies are formed with contingents of troops. Contingents are recruited from territory populations

Figure 4.1: Almansur Screenshot

Almansur is targeted to be a next generation game [1] and it possesses some characteristics that may allow for that classification, namely:

- Uses web 2.0 technology, requires internet connection, browser with javascript, AJAX and flash capabilities
- Is a massive game, thousands of players can participate at the same time
- Focuses on skill over time instead of time over skill. The common MMOGs reward the players who spend the most time in front of the computer, in clear detriment of the player's skill

Still, some issues that prevent Almansur from being considered a revolutionary game.

- No single player mode, where the new players can learn the game at their own pace
- Almansur has no AI, which is a distinct trace of next generation games
- Players drop-out and without AI there is no quick replacement for quitters. This degrades other player's experience.
- Dull repetitive tasks
- An overly static world

This work is an attempt to address the previous issues. Below are the unique challenges that make the creation of AI for this game interesting:

[1] *Next generation games* is a term associated with innovative games that surpass current game in a set of features. Some games are so innovative they are considered revolutionary

- No MMOGTBS with AI exists so it is new ground
- Massive games have thousands of players playing at the same time, causing performance to be a major issue
- Strategy games are among the hardest games to develop AI, mainly because the world is quite complex and has only a partially accessible environment
- The combination of MMO + TBS makes it totally impossible to have millions of agents running at the same time, so they need to be dormant for a certain length of time period and will only wake up at turn processing time.

4.2 Architecture

To create playable game scenario, Almansur requires four steps. Step 1 consists of importing human-designed scenarios to the game. Once the scenario is imported the second step starts, each new game is a clone from an imported scenario. Step 3 is the periodic turn processing which starts when the game is cloned and stops when the game is finished (by sudden death or victory points). The web interface is the fourth step, it is the vessel used by the players to interact with the game.

4.2.1 Scenario Import

Each scenario is built by hand and later imported into the game by a web request. It is built from two files: a Lands file which determines the names and characteristics of the lands present game; and a Territories file, which specifies the properties of each hexagonal territory present in the game. Both files are in the coma separated values (csv) format. The first line represents the name of each column, the subsequent lines are the rows of data.

Lands Format

The lands format specifies the main properties of each land. No two lands can have the same name or the same capital. Below is the description of the format of this file and an explanation for each column.

Column	**Description**
name	The name by which the land will be known inside the game
name_male	The title of the ruler of the land
type	The political organization that rules the land, or the lands culture for historic scenarios
capital	The location of each land capital in the format *numberXnumber* where the first number is the horizontal location and the last one is the vertical location in the scenario
description	A short introduction that provides some background on the land

Table 4.1: Almansur's lands file format

Territories Format

The territories format specifies the properties for each territory in the scenario. For a complete format specification refer to table A.1 in appendix A.

The current format has some issues that impare the scenario creation.

- Inflexible format, the column names are fixed and must be present even if not in use.
- Limited format, not all combinations of populations with social states are supported by this format. Also those column-specified. It is also impossible to specify units present in each territory.

Because the scenario format is so rigid and limited the full game scenario expressiveness is not available during the import phase. This limits the quality of created scenarios. For instance it is impossible to recreate historical battles because the import format will not allow it.

4.2.2 Game Cloning

After a scenario is imported into Almansur it can be cloned any number of times. This allows for multiple games to be running at the same time on the same scenario. During this phase scenario topology is cloned and the game is initialized. The game initialization consists of:

- Allocating resources for each land (multiplied by a factor specified by the person cloning the game)
- Creating a default army for each land (multiplied by a factor specified by the person cloning the game)
- Initializing the market's offer/demand for each land
- Collecting the turn statistics for each land

To work around the importer flaws this game phase has been given extra powers. Along with choosing the unique characteristic for each game during this phase the following are determined:

- Will this game include alliances? This would make much more sense if it was present in the importer phase, where not only would it be possible to specify whether alliances are possible but also whether some initial alliance should already be present.
- Initial army multiplier. A factor that multiplies the default contingents a land should start with.
- Initial resources multiplier. A factor that multiplies the default resources a land should start with

The main problem with the augmented cloning phase is the loss of flexibility, compared to the extra functionally provided if those steps were done in the importer phase.

4.2.3 Turn Processing

The game turn processing is where all the player input is processed by the game and it is Almansur's main performance bottleneck. The process is monolithic and centralized on each game. All the players actions are processed at this time. Chaos effect of the combination of

different players actions is the where the fun of the game resides. The more actions a player can take, the greater number of possibilities exist for each game and the longer Almansur can last.

Turn processing flow:

1. backup all games - A complete database dump is done at the start of each game turn processing
2. for each day in days per turn
   (a) process battles - Enemy units in the same territories face each other. The winner stays in the current territory, the loser if alive retreats to another territory.
   (b) for each unit in game units
      i. for each contingent in unit contingents
         A. process current order - modify status, experience, quantity according to rules for the current order the unit has
      ii. kill unit if all contingents are dead
      iii. if unit still alive
         A. active the next order if new order starting today and unit without uninterruptible order
   (c) auto reject expired diplomacy proposals
   (d) process assaults - units in enemy territories with fortresses and with a conquer order, attempt to conquest the fortress by assault. If the assault is successful the territory changes owner
   (e) start sieges - units in enemy territories force populations to garrison inside the territory fortress
   (f) update sieges - populations of sieged territories consume the food for the day, if the food runs out the siege ends and the territory changes owner. The siege is lifted if current sieging units do not have enough man power to sustain the siege.
   (g) start conquers - units start conquering the enemy territory they are present if they have a conquer order in execution
   (h) update conquers - if all the population is subdued the territory changes owner. If current conquering units do not have enough man power to sustain the territory conquest, the conquer is suspended until reinforcements arrive.
   (i) process day events - any event scheduled to happen today is processed
   (j) if last day in turn
      i. process production - facilities send their resources to the land owner
      ii. charge credits - players are charged for credits
      iii. kill lands - lands without enough territories and without money are eliminated
   (k) if last day in month
      i. process market - the markets demand/offer values are updated

```
      ii. process resource growth - any live resources are reproduced
     iii. for each land in live lands
        A. process month economy - creates the economy resume for the current
           month
      iv. increase month count
   (l) increase day count
3. increase turn count
4. calculate next turn processing time
5. create turn statistics for each land
6. update map views - recalculates the fog of war based on the current units
   and territory ownerships
7. update alliance rulers - check is an alliance has a new leader
8. end game if finish rules satisfied - ends the game when any ending rule is
   satisfied
```

By itself turn processing limits the number of games a server can run simultaneously The formula(4.3) below illustrates how many games can be active at the same time on the same server

$$f_T(game) = \text{Number of minutes taken to process a game turn} \tag{4.1}$$

$$f_D(game) = \text{Number of real days between each game turn processing} \tag{4.2}$$

$$n = 60 * 24 * \frac{f_D(game)}{f_T(game)} \tag{4.3}$$

Where n represents the maximum number of games active before If a daily game takes 30 minutes to process then a server can only have at *n = 60 * 24 * 1 / 30 = 48* games at the same time.

This limit does not take into account the number of CPU cores, due to a limitation of version 1. No two games can be processed at the same time. This problem comes from two issues:

- Inflexible backups (all databases need to be dumped to backup a single game)
- Concurrency issues (when a game is being processed, other games can suffer currency issues)

4.2.4 Website

The website is the face of the game. It is has to be attractive or players will not be interested in finding out the games inner complexity. The main issues currently plaguing the game are:

- Complex interfaces, hard for new players to learn
- Too deep interfaces, some actions require too many steeps

In the next chapter an architecture will be introduced which deals with all the difficulties of this game. This work will focus the lower level of AI, introducing agent oriented processing and creating a more dynamic and immersive world in the process which better simulates the historical reality, as well as the foundations for higher level AIs to be developed.

Chapter 5

Almansur Agent Architecture

Man discusses, Nature acts

Voltaire

This chapter introduces the multi-agent architecture created to address the issues of the turn-based strategy massive multiplayer on-line game Almansur.

5.1 Introduction

The agent's mind follows the hybrid horizontal approach, which is a good fit for this game genre for the following reasons:

- Good performance
- Custom agents, the 1 layer - 1 behavior mapping provided by horizontal hybrid architectures make it easy to configure different agents, starting from the same template
- Allows for complex behaviors to be added when needed. Even though some agents can be only reactive, there is no hard restriction that forces them to be that way

Figure 5.1 shows the outline of the architecture. The environment is accessible to the agents through standard sensors. Also accessible through the sensors is the common knowledge pool, denoted *whiteboard*. The agents can try to modify the environment through their effectors. Effectors can either succeed or fail and when they fail none of their effects are propagated to the world. Through its effectors an agent can modify the content of his group whiteboard.

Each agent can have an internal state, this state can be shared with other agents if the agent allows it. Two types of sensors exist, the passive sensors which are received by the agent by event notification; and the active sensors which are used explicitly by the agent.

Agents can communicate with other agents in two ways. Passively, they simply write information in or pose questions to the whiteboard and check for changes later in pooling mode. Actively, the agent notifies another agent explicitly and waits for his response. The agents are grouped according to the master they work for and only have access to their group's whiteboard.

Intelligence is provided by the behaviors available to each agent, the combination of different behaviors specified and emergent higher level behaviors can be manifested by the in-

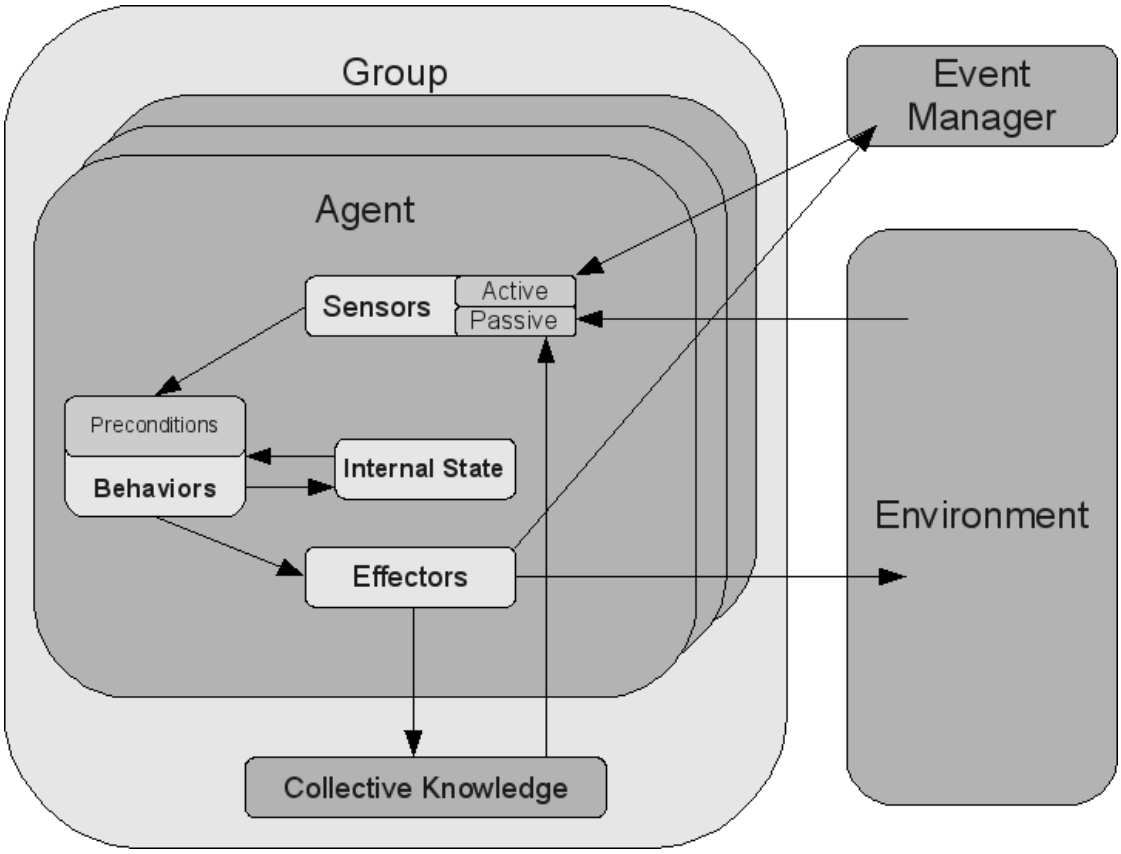

Figure 5.1: Agent Architecture

teraction of the simpler ones. Each behavior has a set of pre-conditions that need to be valid for the behavior to be enabled. The horizontal nature of the architecture allows multiple behaviors to be started at the same time if the proper conditions are met.

The event manager is the component responsible for creating the intelligent environment. Agents register themselves on the event manager to receive certain types of events. When such events arise the event manager selects the ones who should receive that event and in which order.

5.2 Environment

The virtual world of Almansur is a partially observable, non-deterministic, sequential, dynamic, discrete and multi-agent environment. It is not fully observable because other players actions or agents from outside groups are not known. The non-deterministic nature also derives from the actions of other players or competitive agents. The actions can have repercutions for many turns, making the environment sequential. It is a dynamic environment because multiple agents compete for the same actions at the same time, causing some agents' effectors to fail if they are done too late. While the number of percepts can be extremely large it is still a finite set, so it is a discrete environment. Several agents roam the Almansur world, creating a multi-agent environment. Agents are typically cooperative inside the same group and competitive against different agent groups.

5.3 Agents

Several types of agents exist in this architecture, which will be detailed in the next sections. All of them follow the core design, having a hybrid architecture. The internal state is optional, agents can be fully reactive whenever appropriate. Because the performance is of the utmost importance and because several thousands of games can be active at the same time it is not feasible to have millions of agents active at the same time. The agents thus need to be loaded only when needed and made dormant when there is nothing for them to do.

5.3.1 Behaviors

Behaviors are the core of the agent's mind, an agent being only as intelligent as its behaviors and the emergent interactions they provide. For instance, for an agent to be social it must possess some sort of behavior that enables communication.

The behaviors are loaded at each agent's initialization. Connected to the behaviors are the agent's sensors and effectors.

5.3.2 Sensors

Each Agent can have two types of sensors: passive sensors, with which the agent cannot control the action, it is notified when some event is detected and can choose to react to that event or simply ignore it; and active sensors that can be used whenever it deems it necessary to acquire information from the world or other agents or from the common knowledge pool. The agent's internal state is accessed directly, it is not necessary to go through its sensors. The sensors are only required to acquire information from outside the agent.

5.3.3 Effectors

Effectors are the standard way for an agent to influence the environment, other agents or to write to the group knowledge pool. They can have two outcomes: either they succeed and the environment reflects all the actions they represent; or they fail and none of the actions they represent are actually performed. This can happen when some other competitive agent performs a conflicting action at the same instant.

5.3.4 Domain Specific Language

The configuration of the behaviors requires some degrees of flexibility. To create a simple yet powerfull way of allowing configuration a custom domain specific language (DSL) was developed. Throughout this work will be refered to as Agent Modeling Language (AML).

This language is used to specify which behavior each agent can have, and which preconditions they have. For a complete reference of AML in BNF [1] refer to section B.1.

[1] BNF stands for either Backus-Naur Form or Backus Normal Form, it is a formal and precise metalanguage used to describe the grammar of a programming language [27]

E.g. of a population behavior configuration:

```
:revolt => {
        :EventType =>'end_of_turn',
        :Property => [
          { :name => :happiness, :max => REVOLT_THRESHOLD},
          { :name => :loyalty, :max => REVOLT_THRESHOLD},
          { :name => :in_fortress, :equals => false}
        ],
        :Sensor => [
          { :name => :territory_sensor,
            :method => :stability,
            :parameters => [ :territory_id],
            :max => REVOLT_THRESHOLD}
        ],
      },
```

5.4 Initialization

When the turn is processed, all the agents need to be loaded. Behaviors are grouped into modules by functionality. Loading a module into an agent causes the agent to learn how to execute all the behaviors in that module. But only behaviors that are configured are electable to be executed.

The loading process follows the algorithm for each agent:

```
1. Load custom modules or fallback to default modules
2. Load sensors for configured modules
3. Load effectors for configured modules
4. to_load_behaviors = Custom Behaviors
5. to_load_behaviors = Default Behaviors unless Custom Behaviors
6. agent_preconditions = new Hash
7. agent_behaviors = new Hash
8. for each behavior in to_load_behaviors
   (a) agent_behaviors[behavior] = new Array
   (b) for each precondition in behavior.preconditions
       i. agent_preconditions[precondition] ||= new Array
      ii. agent_preconditions[precondition].push behavior
   (c) end
9. end
```

If no custom behaviors are specified the agent will fall back to the default behaviors. This feature is useful to avoid repetition of customization of equal agents. A template can be created and all instances will share the same default configuration.

Each agent maintains a list of behaviors. The modules specify which sensors and effectors an agent must have to be able to execute the behaviors present in the module. This ensures an agent is always technically able to execute the configured behavior. When the module is loaded into the agent, the sensors and effectors are also injected into the agent.

For performance reasons, two hashes are also maintained by each agent. The first indexes each configured behavior according to the pre-conditions that he requires. This allows for behavior culling when processing passive sensors (event notifications). Instead of checking if all behaviors can be performed, the behaviors which lack the current event as a pre-condition can be trivially ignored. The second keeps record of all the pre-conditions required by a certain behavior. This is used when the agent needs to determine if the behavior can be performed or not.

5.5 Agent Types

Four types of agents are proposed to be used with this architecture. Reactive, proactive, manager and personality.

5.5.1 Reactive

Reactive agents react to the percepts they receive. They may or may not have an internal state, but they do not do any kind of reasoning or planning ahead. These are the simplest and fastest agents. They can be easily converted into another type of agent just by adding new behaviors.

5.5.2 Proactive

Every agent that performs some kind of goal oriented thinking or uses some utility function to maximize performance is considered a proactive agent. Their main difference to reactive agents are the higher level behaviors that allow the agent to exhibit more intelligence. All proactive and reactive agents are bound to use sensors and effectors to interact with the environment.

5.5.3 Managers

Games have specific needs and the game rules need to be enforced by someone. Managers are a special kind of agent that enforces the game rules. They are responsible for the arbitrary decisions the game must take and they determine the outcome of conflits between competitive agents. As extentions to the game itself, these agent are totally impartial and can have God-like abilities. They know every agent in the game and can manipulate their internal state at will.

5.5.4 Personalities

Personalities are another special agent, although they possess abilities of their own, they can be placed in charge of other agents. When they hold such positions they absorb the capabilities of the subject agents and take control over the decision-making process. The acquired abilities

are only temporary, and when they leave the position they also lose them, even though smart enough personalities can incorporate learning abilities and be able to retain the knowledge from the positions they occupied.

Personalities try to simulate real persons, they have needs such as requiring a good salary or being employed, and when employed they have all responsibility and control over their decisions. Logically, once they leave their job they also lose responsibilities and control.

Each personality has 3 sets of characteristics:

- Status properties, volatile values that represent the internal state of the personality. For example, how much money it pocesses or how good its health is.
- Personality traits, fixed characteristics that determine the biased behaviors. A personality can be defensive or aggressive, tolerant or intolerant.
- Skills, are the abilities that a personality posssesses, which can be learned or improved. Skills have a proficiency level and this level improves whenever the skill is used, so the more an agent uses a skill, the better it becomes.

5.6 Communication

Fundamental to any multi-agent system is multi-agent communication. Communications can be either synchronous or asynchronous.

Synchronous communications imply that all parties involved must be present at the same time, which means the information is immediately propagated and an agent will wait for a response to each question even if it is only a reception acknowledgment.

Asynchronous communications simply require a common communication channel but do not require all parties to be present. The information can be written on the channel at the time instant X and be read by other members at instant Y.

The next subsection introduces the two methods available to communicate with other agent on this architecture.

5.6.1 Events

Events are the multi purpose system for communication. All events are under the control of the event manager module.

The event manager is responsible for keeping a record of which agents are registered for a given event type. Any agent can create new events and ask the event manager to process them. The event creator can specify filters that must determine a subset of agents entitled to receive such an event. The event can be filled with whatever information the event creator wants, so they can be used to transport all kinds of information back and forth in an object-oriented way.

Synchronous Events

When an agent registers a new event with the event manager he has the chance of specifying that the event be processed in a synchronous form. The workflow below exemplifies what happens:

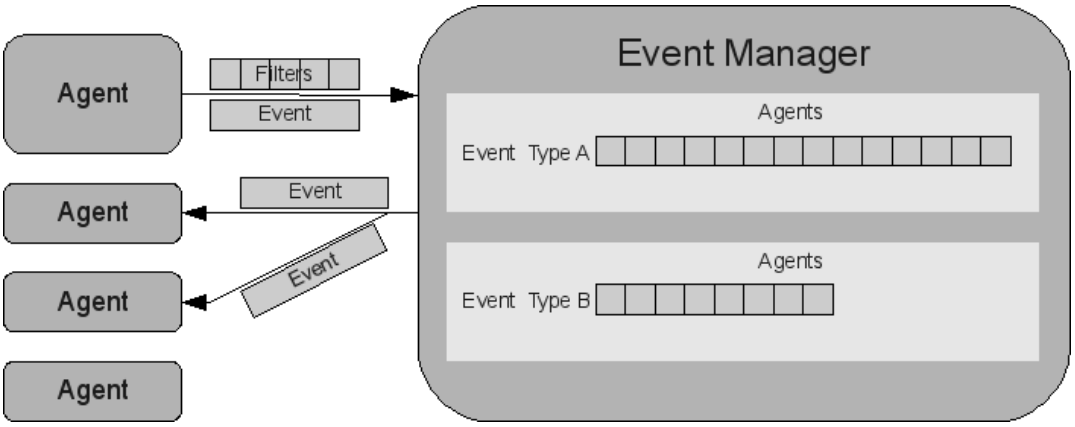

Figure 5.2: Event System

1. Agent creates a new event
2. Agent sends the event created in 1. to the event manager
   (a) Agent declares the event should be processed in a synchronous form
   (b) Agent sends out the filters each other agent much pass to receive the event
   (c) Agent is blocked until the event manager's execution is finished
3. The event manager receives the event and filters
4. Event manager retrieves all the agents registered to receive notification of event types with the same type as the received event
5. Event manager selects all agents from 4. that pass the received filters
6. Event manager notifies all agents from 5. with the received event
7. Each notified agent processes the received event
8. Event manager sends feedback to the caller agent
9. Agent unblocks

By the time the workflow ends the agent is sure everyone else was notified about the raised event.

Asynchronous Events

Asynchronous events do not block the caller agent while they are processed. Furthermore, the caller agent has no callback to notify that the event is indeed processed by everyone else. They follow the workflow below:

1. Agent creates a new event
2. Agent sends the event created in 1. to the event manager
   (a) Agent declares the event should be processed in a asynchronous form
   (b) Agent sends the filters each other agent much satisfy to receive the event
3. The event manager receives the event and filters
4. Event manager retrieves all the agents registered to receive notification of event types with the same type as the received event

5. Event manager selects all agents from 4. that pass the received filters

6. Event manager notifies all agents from 5. with the received event

Delayed Events

Delayed events are a particular type of asynchronous events that are only processed when all other events are finished. When the event manager receives this type of events it stores them in a queue. When the event manager knows that no more events are going to arrive, it fetches the queued events and processes them one by one using the asynchronous workflow. These events are particularly usefull for cleaning-up jobs.

5.6.2 Whiteboards

The second form of communication are the whiteboards. Whiteboards are simply a shared pool of knowledge that can be accessed by a group of agents. To use whiteboards the agents need to use sensors for reading and effectors for writing. There is no mechanism for controlling multiple agents writing the same information at the same time, so the persistent information will be the last to be written. Whiteboards are an indirection, so all communications that go through it are asynchronous.

Agents are denied access to whiteboards outside their group. They do not even know they exist so all whiteboard information can be considered if all members of the group are honorable and trustworthy.

5.7 Deletion

"In life only death is certain" – Anonymous

In many games agents can die, either because they outlived their usefulness or were slain by some other agent. To kill an agent in this architecture it is necessary for the agents to remove themselves from the event manager lists. They must notify the event manager that they no longer exist. This will help the event manager from sending events to dead agents.

5.8 Hibernation

In order for agents to be consistent between multiple turns, it is necessary for agents to save their internal state. When the turns are over, the current state is written to a persistent media. When the next turn is processed, all agents are loaded and initialized. During the initialization process any previous state or learned abilities are also loaded. After the initialization the agents are at the same state they were in immediately before hibernation.

5.9 Reproduction

"Humans live to breed" – Anonymous

Populations reproduce, it is innate and their only means of long time survival. To properly simulate that behavior the agents need to be able to reproduce as well.

5.9.1 Clones

To replicate itself a population can be divided by being cloned. After the clone is created it is simply initialized like a normal agent. From that point on two agents exist and their future experiences and states determine what they will be next, so diversification comes naturally. But, since reproduction begins with cloning all the parent knowledge is passed to the clone.

5.10 Event Processing

Event processing in multi-agent systems requires some form of arbitration to resolve conflicting actions. Imagine agent A and agent B are competing for the same objective. Both of them have to paint a wall. A wants to paint the wall red, B wants to paint the wall blue. Several possibilities exist:

- Ignore the problem
- Let both agent use the same resource at the same time, and allow the last agent to act to overwrite the first agent's actions
- Reserve the shared resource for the first agent to execute the action and force the second agent to fail

Solution number 1 brings chaos, all predictability and reproducibility being lost. Multiple runs of the same simulation would provide different results. Solution number 2 creates operational and coherency problems. It also is an unnatural solution, in the real world this would be impossible. Solution number 3 requires transactional support. Still it is the more natural solution. This architecture employs solution 3.

5.10.1 One-Phased cycle

For performance reasons, when it is clear that an event will not cause multiple action to conflict, the event manager can notify agents to perform a combined feel-effect cycle. Agents can act on their percepts without worrying about them getting outdated. This mode follows this workflow:

1. `Event manager receives the event and filters`
2. `Event manager retrieves all the agents registered to receive notification of event types with the same type as the received event`
3. `Event manager selects all agents from 2. that pass the received filters`
4. `Event manager notifies all agents from 3. with the received event and specifying its a feel-effect cycle.`
5. `Agents receive an event with a feel-effect cycle.`
6. `Agents uses their active sensors to extract information from the environment`
7. `Agents uses their effectors to perform changes on the environment`

5.10.2 Two-Phased cycle

When actions from multiple agents may be conflitcting, all agents need to be coordinated to acquire information from the environment at the same time. Otherwise, some agents would have an unfair advantage over others since the accuracy of the information gathered would change. Essentially, the processing is done in two steps: a feel phase where all agents update their information about the current state of the world; and an effect phase, where the agents use the knowledge acquired from the previous phase to select what actions to take, and to effectively send them to the environment. The agents can only act after all other agents have finished the feel phase.

This mode follows this workflow:

1. `Event manager receives the event and filters`
2. `Event manager retrieves all the agents registered to receive notification of event types with the same type as the received event`
3. `Event manager selects all agents from 2. that pass the received filters`
4. `Event manager notifies all agents from 3. with the received event and specifying its a feel phase.`
5. `Agents receive an event with a feel cycle.`
6. `Agents uses their active sensors to extract information from the environment`
7. `Event manager notifies all agents from 3. with the received event and specifying its a effect phase.`
8. `Agents receive and event with a effect cycle.`
9. `Agents uses their effectors to perform changes on the environment`

5.10.3 Usage

One-phased cycle is used for events that are conflict free. It's imperative that no two agents compete for the same resource. If the agents compete for the same resources or require synchronization, when responding to an event, a two-phased cycle is necessary. One-phased cycle has better performance and should be prefered whenever the events are guarateed to be conflict free.

Chapter 6

Almansur 2.0

After introducing the Almansur game in chapter 4 and the developed architecture in chapter 5, the application of the developed architecture to Almansur is described. The architecture was developed with the main objective of integrating AI into Almansur, but the known issues with Almansur were also tackled when possible. Version 2.0 will not bring any major improvement to the web interface of the game, it will focus on the game core. The improved parts of version 2.0 are, therefore, the scenario import, game cloning and turn processing. The turn processing modules suffered the most transformations, with the application of the proposed agent architecture. Over the next sections the modifications that produced Almansur 2.0 are explained, following the taxonomy from chapter 4.

6.1 Scenario Import

CSV files were kept as the file format for hand made scenarios but to overcome some of its problems the importer was refactored to support flexible headers and new requirements.

Lands Format

The new header format introduces the race column, It is the only difference from version 1.0. This modification is done by need, since in version 2.0 each land also has a new field that identifies the ruling race / culture.

Column	Description
name	The name by which the land will be known inside the game
name_male	The title of the ruler of the land
type	The political organization that rules the land, or the lands culture for historic scenarios
capital	The location of each land capital in the format *numberXnumber* where the first number is the horizontal location and the last one is the vertical location in the scenario
race	The name of the race/culture that rules this land
description	A short introduction that provides some background on the land

Table 6.1: Almansur's lands file format

Description

Territories Format

Major changes were done on the territories file. If a header does not have any data, then it makes no sense to require it. So in version 2.0 the concept of required and optional columns is introduced.

Required columns must be present in any territory file. They are the bare minimum that classifies a territory. Examples are the territory coordinates, without them it is impossible to place the territory on the scenario.

Optional columns can be present or absentin a territory file. They also have the interesting property of being in sync with the database data. For instance any new resource type that is introduced into the Almansur's database is immediately available in the importer. This type of flexibility creates a much more robust importer allowing the format to grow without actually changing any code.

Facilities	**Resources**	**Races**	**Social States**
Cattle_Raising	Gold	Barbarian	Free
City	Iron	Berber	Slave
Encampment	Stone	Christian	Noble
Farm	Wood	Dwarf	
Fortified_Villages	Fish	Elf	
Fortress	Trees	Goth	
Ironworks	Grain	Human	
Lumber_Mill	Salted Meat	Jewish	
Market	Salter Fish	Muwallad	
Mine_Gold	Horses	Orc	
Mine_Iron	Cattle	Roman	
Pig_Farm	Pigs	Uruk_hai	
Port	Wargs	Vandal	
Recruitment_Center	Wild Game		
Roads	Slaves		
Shipyard			
Stables			
Stone_Quarry			
Underground_City			
Warg_dump			
Church			
Mosque			

Table 6.2: Races, Social States, Facilities and Resources

Table 6.2. represents the possible substitutions currently available for the territory files. The name under brackets means the column name can assume any value shown on table 6.2.

The new format ends up being more powerfull and yet simpler to memorize. All the header names adhere to the same pattern, and follow the principle of least surprise[1].

Column	**Required**	**Description**
x	required	Horizontal coordinate for the territory
y	required	Vertical coordinate for the territory
mov_class	required	Type of territory, L - Land, S - Sea
altitude	required	Territory average distance to the sea level in meters. It's a positive value for shore and negative for water covered territories
relief	required	Average sharpness of the territory, a regular territory will have a low relief. While a cliff will be a very high relief
swampness	required	Average swampness of the territory
fertility	required	Average tax of reproduction of this territory
[resource_type]	optional	Quantity in tons or heads for resource naturally available in the wild, in this territory
terr_name	optional	Name of the territory (Empty for the name to be based on the territory coordinates)
[population]_ [social_state]	optional	Quantity for any combinations of populations with social_states present in this territory
[facility_type]_ [name]	optional	Name of for the facility of the type [facility_type](Empty for the facility name to be "[facility_type] [terr_name]") present in this territory
[facility_type]	optional	Level of the facility of type [facility_type] (Zero is no such facility exists) present in this territory
territory_resource_ [resource_type]	optional	Quantity in tons or heads for the resource present in this territory warehouses and populations
pe_name	required	Name of the land that owns this territory (must be an entry from the lands files under the column name)

Table 6.3: Almansur's territories file format

Description

6.2 Game Cloning

After changing the core paradigm to use the agent architecture presented on chapter 5, it becomes necessary to create the agents at the end of the game cloning phase. Because these agents use a whiteboard as a means of communication, the whiteboards for each agent group also need to be created and populated.

[1]The principle of least surprise states that, when elements of an interface conflict or are ambiguous, the behavior should be that which will least surprise the human user or programmer at the time the conflict or ambiguous situation arises.

The initialization procedure specified on section 5.4 shows that if the agent has no custom behaviors then the default behaviors are loaded instead. Since most of the agents belonging to the same type use the same behaviors if the default behaviors are chosen carefully nothing is required to create the game agents. So by sticking to the design pattern *Convention over configuration* [2] a lot of boiler plate code can be avoided.

6.3 Turn Processing

The most radical changes occurred on the turn processing part of Almansur. The version 2.0 not only introduces AI but also solves the issues that previously plagued version 1.0. By creating a framework that makes it possible to manage games individually (create, backup, delete) it is now possible to process multiple games at the same time without suffering any concurrency issues.

Such framework allows the revisiting of the formula 6.2, and introduces the number of CPU cores into the calculation.

$$f_C(computer) = \text{Number of processor cores a computer possesses} \tag{6.1}$$

$$n = 60 * 24 * f_C(computer) * \frac{f_D(game)}{f_T(game)} \tag{6.2}$$

Assuming the major bottleneck is the CPU processing[3] the number of processed games can now increase almost linearly with the number of processor cores available.

6.3.1 Agents

Version 2.0 of Almansur departs from the old central paradigm, where everything at turn processessing was pooled by the game. Now, at the core we have a virtual world that stimulates its agents to act. The game flow is controlled by what type of events the world generates, and the game impartial features are under the belt of special agents called managers. The managers can be seen as demi-gods in the virtual world, whose task it is to ensure the agents abide by their rules by any means necessary. For this reason the managers live outside the effectors/sensors jail and can manipulate any other agent's internal state directly.

For a complete reference on the agent minds, specified on the DSL presented on section 5.3.4 refer to appendix B.

6.3.2 World

The world is the environment where all agents are inserted. All agents interact with the world through sensors and effectors even managers. The game flow is determined by the events raised by the world. The next subsections document the parts that compose the world.

[2] Convention over configuration is a design pattern which privilege the use of the most common thread of execution as the default action to take. This allows for easier to setup frameworks without compromising the flexibility provided by configuration. [32]

[3] Ignoring IO wait eases the theoric calculations, and can be minimized in pratice by offloading the database load to a dedicated server, through a high speed connection

Events

Behind the agent architecture is an event system, every tidbit of game flow is generaly controlled by the type of events raised. Below is the exhaustive list of all events that can occur in version 2.0.

Event	Description
start turn	The beginning of a new turn (agents that have ahead of turn actions to do need to listen to this event)
upgrade	A facility has finished its upgrade
downgrade	A facility has finished its downgrade
conquer	A territory has been conquered by some land
tribute	A mandatory tribut needs to be paid
newcapital	A land has successfull moved its capital to a new territory
arrival	A unit has arrival at a new territory
start day	The beginning of a new day (agents that have daily actions to do, need to listen to this event)
production	Facilities have stocked enough production to send to the land ruler
reproduction	Alive agents are allowed to increase their quantity
economy	New financial data is available
end turn	The turn processed has ended (agents that need to do last minute actions, need to listen to this event)
market	The market stocks are being refilled
start siege	A siege action just started a specific territory
siege canceled	An undergoing siege action was canceled
diplomatic change	The diplomatic relation between two lands changed (for better or worst)
new population	A new population arrived at a territory
tax change	The tax collected by some territory changed
update siege	A new day has passed, the territory in question needs to feed the enclosed units

Table 6.4: Events (Almansur 2.0)

Turn flow 2.0

Turn flow uses the introduced events to get the core going. During the processing of the core events the agents can raise their own events and create a recursive feedback loop. This effect needs to be taken into consideration when electing an agent to raise events, or else we can raise an infinite feedback loop and bring the game down.

The Almansur 2.0 core is run from the simple pseudo code below.

1. Create a new Event Manager
2. Create a new Agent Manager
3. Request the Agent Manager to load all Agents

4. Loads whiteboards for the current game from the persistent support
5. Raise a synchronous *start turn* event
6. Retrieve current day from Game Agent
7. For each day in number of days per turn
   (a) Raise a synchronous *start day* event with current day
   (b) Request event manager to raise all in-game events scheduled to happen at current day
   (c) Increments current day
8. Raise a synchronous *production* event
9. Raise a synchronous *market* event
10. Raise a synchronous *reproduction* event
11. Raise a synchronous *economy* event
12. Raise a synchronous *end turn* event
13. Request event manager to raise all queued events
14. Save modifications to whiteboards to a persistent support

6.3.3 Units

Units represent a container of contingents, whose main purpose is to conduce the military warfare. Each unit allows one general, this job can be filled by a personality. A simbiotic relations is created between a personality and a unit. The personalities gain all the unit's behaviors (and gains the change of using its skill and gain more experience). Units gain bonus from the personalities skill modifiers. Table 6.5 presents all the behaviors a unit can have. For the details on the preconditions necessary for each behavior to be exectuded refer to appendix B.

Behavior	**Description**
upkeep	Request the salary for the current turn
eat	Request the pounds of food necessary for the current turn
revolt	When the loyalty is too low, the unit will no longer obey the land ruler
send to land	Send the captured resources to the central resource pool
arrival	Move to another territory
force rest	When the status is low rests for a few days to avoid high territory attrition
identify scouts	Attempts to discover any scouts present on the current territory
resume	When status is recovered, attempts to resume the current order

Table 6.5: Unit behaviors

6.3.4 Contingents

The contingents are members of units, they have simple behaviors and therefore do not provide jobs for personalities to fill. Also due to the possibility of having thousands of contingents

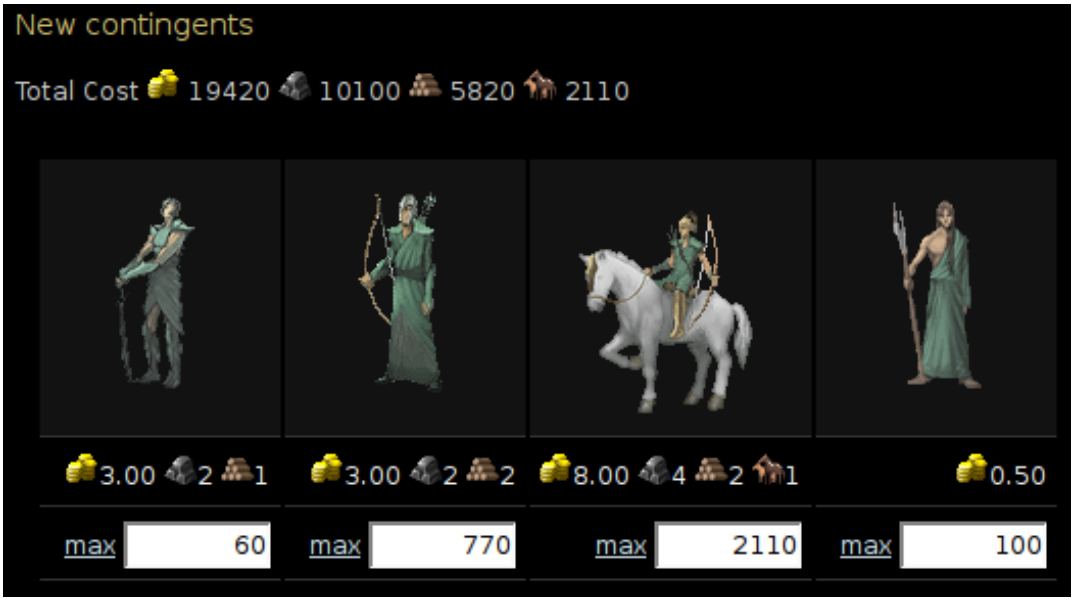

Figure 6.1: Almansur new contingents

for each land, it would become very hard to manage so many jobs. Below on table 6.6 the contingent's behaviors are specified.

Behavior	**Description**
process day	Apply the consequences of any queued orders
upkeep	Request the salary for the current turn
eat	Request the pounds of food necessary to eat for the current turn
disband	Converts contingent into settlers at the current location

Table 6.6: Contingent behaviors

6.3.5 Populations

Populations represent different cultures and social states, that live on the territories. On historical scenarios they are all human, but on fantasy scenarios multiple races are present. Table 6.2 presents all the different races a population can be.

These agents are now far more intelligent that in version 1.0. To introduce an historical background, down the Almansur lane the players started to act with less and less honour. And betrayals and massive taxing were now the rule. On version 1.0, the populations got less and less loyal, which meant they simply decreased the number of law abiding citizens. Which in turn would progressively provide less taxes and less recruitable forces. Still they suffered such abuses without complaining, which lead to several game exploits.

Creating more intelligent behaviors to cope with the situation was deemed necessary. Now on version 2.0, the new behaviors allow unhappy populations to migrate to better lands, searching for better opportunies, not unlike humans do in real life. Also populations tend to prefer pacific lands, and like the diplomatic stability they provide. To better simulate this, populations now react negatively to war, and positively to peace.

Table 6.7 presents all the behaviors implemented for the population agents.

Behavior	Description
reproduction	Increase population numbers
resource reproduction	Increase live resources for the population
merge	Attempts to merge with another population of the same race and social state
revolt	Uprise against the territory owner if unhappy and the territory is not protected
react to tax change	Decreases loyalty and happiness if taxes are raised, has a 1/3 inverse effect if the taxes are lowered
react to diplomatic change	Decreases loyalty and happiness if diplomatic ties with another land degrade. Increases loyalty and happiness when ties improve
react to conquer	Adjusts loyalty and happiness in regard to new territory owner
process siege	Requests daily supplies of food from the territory warehouses
prepare for siege	Seeks refuge in a protection building if enough space is available

Table 6.7: Population behaviors

6.3.6 Territories

A key element of the stategy games is the land occupied, they provide the resources necessary to sustain wars or invest in development. Still they are mainly reactive agents with little need for intelligent behaviors.

Table 6.8 presents the territory behaviors.

Behavior	Description
produce	All production facilities in this territory release stocked resources to land owner
resource reproduction	Reproduced the live resources, that belong to the territory
react to diplomatic changes	Changes territory stability according to the geo-political situation
react to conquer	Diminishes territory stability as a result of the current conquest
update stability	Recalculates the turn's territory stability
update available emergency recruits	Attempts to train more citizens to be part of the men-at-arms emergency force
update whiteboard	Updates territory information on the group whiteboard
react to unit arrival	Reports the arrival of known units
upgrade facility	Upgrades a facility
downgrade facility	Downgrades a facility

Table 6.8: Population behaviors

6.3.7 Personalities

Personalities are special agents that can have jobs. When a personality is employed it temporarily gains new behaviors depending on the job it has. A personality can be allocated to different jobs at different times, but may only have 1 job at a time.

Personalities have an addicional three sets of property:

1. Status, properties that can change frequently represent the internal state of the agent.
2. Traits, properties that represent the personality. These properties specify the way in which they approach a situation and are immutable over time.
3. Skills, specials abilities that get better with usage. Each time a personality uses a special skill it increases. When certain threshold is reached, new skills are learned.

Status

The status properties are dynamic characteristics used to represent the personality internal state. Table 6.9 show a description of all these properties.

Name	**Description**
Status	The current physical status of the personality
Loyalty	How loyal a personality is to his land
Happiness	How happy a personality feels
Salary	How much the personality earns per turn
Land	To which land the personality is currently associated
Title	Which title the personality has achieved
Territory	In which territory the personality is
Market	When unemployed on which job market a personality is
Time on market	How long the personality is unemployed

Table 6.9: Personality properties

Traits

The personality traits are fixed characteristics that influence the way a personality agent decides. A personality with strong militarism will invest most of its money in armies, while weak militarism will cause a preference for economical development and peace. These characteristics are listed and described in table 6.10

Name	**Description**
Birthday	The birthday of the personality
Birth location	The territory where the personality was born
Race	The personality's race
Aggressiveness	The balance between offense and defense that a personality possesses
Tolerance	How tolerant a personality is towards other races
Charisma	Whether the personality is reclusive or social by nature
Militarism	Whether the personality prefers armies or economic development

Table 6.10: Personality properties

Skills

Each personality starts with three random skills. The skills improve with usage and when the personality's total experience [4] reaches a certain threshold a new skill can be learn. At most, personalities can have a total of five skills.

Table 6.11 lists all the skills available and their description.

Name	**Description**
Trade	Improves the market power
Build	Reduces the time needed to build a facility
Produce	Increases resource production
Tax Collect	Improves tax collect effiency
Recruit	Improves status and experience of new recruits
Espionage	Gives unique informations about other lands
Rest	Improves the status recuperation associated with a rest order
Train	Improves the speed of training
Garrison	Improves fortress defensive abilities
March	Moves faster between territories
Scout	Improves the changes of a scout to remain undetected
Defend	Improves the military bonus for the defend order
Conquer	Speeds up the time required to conquer a territory
Siege	Reduces the military penalties for sieging armies
Attack	Improves the military bonuses for attacking armies
Range	Improves the damage done during the range combat phase
Shock	Improves the damage done during the shock combat phase
Charge	Improves the damage done during the charge combat phase
Melee	Improves the damage done during the melee combat phase
Pursuit	Improves the damage done during the pursuit combat phase
Assault	Improves the damage done during assaults

Table 6.11: Personality skill list

As stated before, personalities have the special ability to capture behaviors from other agents. When a personality is given a job it automatically gains the behaviors associated with that job. Still all personalities have some core behaviors that belong to them always.

[4] The sum of all its skills

Table 6.12 lists and describes such behaviors:

Behavior	Description
upkeep	If the personality is employed, it requests his/her salary from the employer
accept job	Evaluates all jobs proposals and accepts the best offer that statisfies this personality demands. If no such proposal exists, all existing proposals are rejected
change market	When a personality is unemployed for too long on the same local market[5], the personality moves to the global job market.[6]

Table 6.12: Population behaviors

6.3.8 Battle Manager

This special agent is responsible for all game battles. When enemy units occupy the same territory at the same day, this agent is responsible for determining the outcome of the conflict. Units and contingents present in the conflict have their internal state modified by the battle manager to ensure they abide by the game rules. If some units or contingents die in battle the battle manager is also responsible for removing the respective agents from the game.

Currently there is only one behavior, since only battle are implemented in the version 2.0. In a near future a second behavior to process game assaults needs to be implemented.

Behavior	Description
process day battles	Simulates the battles of enemies in the same territory

Table 6.13: Battle Manager behaviors

6.3.9 Hostile Action Manager

Similar to the battle manager, the hostile action manager controls the military conflicts that arise from conquests and sieges. Both types of conflict are processed in a similar maner, each day this manager checks for starting conflicts and notifies the parties involved. This allows, for instance, for populations to take refuge in fortresses when a territory is under siege. Afterwards, each ongoing hostile action must verify the minimal conditions to continue, end in failure or end in success. When success is achieved the territory suffering the hostile action changes hands.

6.3.10 Game

The game agent is an extention of the world that processes logic common to all lands. Its behaviors could very well be implemented by the world and destroy this agent altogether but for the sake of flexibility and future evolution they are separated modules.

[5] When on local markets only the land that owns the market can place work proposals
[6] When on the global market, and lands can place work proposals

Only two behaviors are yet needed. One for the start of turn actions, and another for end of turn updates.

Behavior	**Description**
process start turn	Resets all the land's production counters
process end turn	rejects expired diplomacy proposals; charges credits; kills dead lands; increments number of turns; selects next turn time; updates global market prices; updates alliance rulers; ends game if winning conditions are met

Table 6.14: Battle Manager behaviors

6.4 Website

Minor changes were done on the website, just the bare minimum to support the evolutions introduced by version 2.0. Most noteworthy is the introduction of a new set of views that allow the current interaction the personalities agents now available.

6.5 Tests and Results

In this Chapter version 2.0 is tested under the same environment of version 1.0 and both are compared in performance issues and simulation issues.

6.5.1 Methodology

To test the application of the architecture to the version 2.0 of Almansur, two different types of tests are presented: simulation and performance.

With the introduced AI, the existing agents are supposed to help simulate real behavior better and make the game easier to develop. In version 2.0 the simulation should be closer to what humans do than it was on version 1.0. These tests are hard to quantify and subjective by nature. To compare the evolution of behaviors from version 1.0 to version 2.0, the same player actions are taken, on scenario 2PL, in both versions. Afterwards three turns are processed and the outcome results are analyzed.

Performance tests measure the difference of clock time regarding the same actions in version 1.0 and version 2.0. These tests show whether agent-oriented processing has a negative or positive impact on game waiting times. Between each test run the test environment is rebooted so that each test starts out under the same conditions. Rebooting the test environment means resetting the database to the same point and restart the webserver to avoid any caching mechanisms which may distort results.

6.5.2 Test Machine

All the tests where done under the same environment.

Hardware	Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz 2GB of RAM, DDR2 667MHZ.
OS	Linux final 2.6.17-10-server #2 SMP Tue Dec 5 22:29:32 UTC 2006 i686 GNU/Linux Ubuntu 7.04
Ruby	ruby 1.8.5 (2006-08-25) [i486-linux]
Webserver	lighttpd-1.4.13 (ssl) - a light and fast webserver
Frameworks	*actionmailer* (1.3.3) - Service layer for easy email delivery and testing *actionpack* (1.13.3) - Web-flow and rendering Model-view-controller framework *actionwebservice* (1.2.3) - Web service support for Action Pack *activerecord* (1.15.3) - Implements the ActiveRecord pattern for ORM *activesupport* (1.4.2) - Support and utility classes used by the Rails framework *cgi_multipart_eof_fix* (2.3) - Fix an exploitable bug in CGI multipart parsing *daemons* (1.0.7) - A toolkit to create and control daemons in different ways *fastthread* (1.0) - Optimized replacement for thread.rb primitives *gem_plugin* (0.2.2) - A plugin system based only on rubygems that uses dependencies only *money* (1.7.1) - Class aiding in the handling of Money *mongrel* (1.0.1) - A small fast HTTP library and server that runs Rails, Camping, Nitro and Iowa apps *mysql* (2.7) - MySQL/Ruby provides the same functions for Ruby programs that the MySQL C API provides for C programs

paypal (2.0.0) - Paypal IPN integration library for rails and other web applications
PriorityQueue (0.1.2) - This is a fibonacci-heap priority-queue implementation
rails (1.2.3) - Web-application framework with template engine, control-flow layer and ORM
rake (0.7.3) - Ruby based make-like utility
sources (0.0.1) - This package provides download sources for remote gem installation

6.5.3 Taxonomy

Each scenario can be qualified under the following metrics:

- Number of territories
- Number of populations
- Number of lands
- Number of units
- Number of personalities
- Number of contingents
- Number of agents
- Number of days per turn
- Number of facilities
- Number of resources

Below are presented some additional metrics which pertain to version 2.0 alone:

Behaviors	The number of behaviors analyzed by the game engine
Processed	The number of behaviors actually run
Unprocessed	The number of behaviors whose preconditions failed and were blocked

Table 6.16: Test metric version 2.0

6.5.4 Performance

For a performance analysis the 3 major issues related to scenarios need to be measured. They consist of:

- Scenario Import
- Game Cloning
- Turn Processing (this issue will be analyzed over the course of three runs, since results can differ slightly between different runs)

Scenario import measures the time needed to import a hand-made scenario into the game. It is a one-time only operation and not executed very often. The need for fast importing is particularly crucial once the scenario tuning has been accomplished because it often is re-imported until deemed adequate.

Game cloning happens once for each game but each scenario can be cloned up to infinity. It is a fairly common operation, but since it happens during the first step of the scenario and without any human participation it does not cause players to have to wait for it to be able to play. The players' anxiety felt during turn processing can be totally ignored here.

Turn processing is the most crucial operation. It happens multiple times for each game and while it is on the players are locked out of the game, so the smaller the processing times the lesser the impact on the players. Take too long to process and players will simply lose interest. Turn processing brings feedback on the players' actions, and on important turns (e.g. a large battle) some anxiety builds up among the players due to the desire to know the outcome of their actions combined with the actions of other players.

6.5.5 Scenarios

On table 6.17 the scenarios used for performance tests are classified according to the previously introduced taxonomy.

The smallest scenario, 2HNoSea, has a reference point for a bare minimum scenario to be played in a multiplayer set. The timings achieved in this scenario are the best and represent the weight the architecture carries.

Game	Populations	Contingents	Units	Territories	Lands	Facilities	Resources	Personalities	Agents	Behaviors	Processed	Unprocessed
FTS4	45	28	4	99	4	12	159	44	223	1854	1359	495
CM40	2362	190	40	1989	40	80	2036	440	5024	26404	14723	11681
AB101	876	515	101	1881	101	2130	10790	1111	4487	34779	28038	6741
WOT20	589	130	25	1188	25	849	4025	275	2210	13138	8681	4457
Crejak10	341	40	10	528	10	227	1321	110	1032	5546	3040	2506
QI	980	644	107	1815	107	1285	8798	1177	4726	39584	29395	10189
Final	2144	351	60	1419	60	1025	9479	660	4637	29885	19789	10096
Juri	84	84	12	340	12	312	984	132	655	5350	3838	1512
OF6	54	33	6	280	6	72	492	66	442	2911	1810	1101
OF20	637	110	20	990	20	320	2916	220	1980	11548	6794	4754
2PL	17	14	2	140	2	26	190	22	198	1312	840	472
2PLT	2	14	2	56	2	14	46	22	99	916	690	226
2HNoSea	2	14	2	2	2	14	46	22	45	700	636	64

Table 6.17: Test scenarios

Figure 6.2: Scenario 2PL

The test begin with a vanilla scenario, it has two lands slice and dice. The player ruling slice increases all territory taxes to 99% while dice maintains the taxes at the default value 5%. Three turns are processed on each version of the game, after each turn the different behaviors are interpreted. Because populations are the most affected from the tax change, a visual representation of the number of citizens living in each territory is presented. This visual representation varies from dark orange (low population density) to bright yellow (high population density). Figure 6.3 represents the initial distribution of the populations.

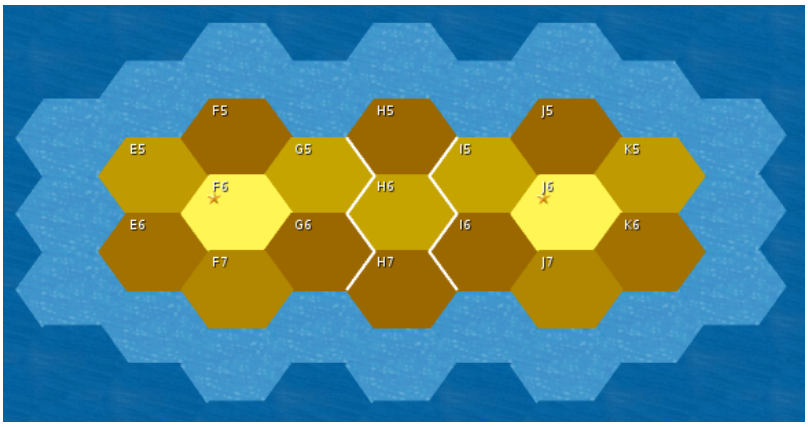

Figure 6.3: Initial population distribution

To better compare the inner works of the game, the following properties are measured between each turn processing:

- Lawfull - The number of law abiding citizens in the territory [0 - everybody is a criminal, 1 - everybody follows the rules]
- Loyalty - How loyal a population is to its territory [0 - rebelion about to happen, 1 - they love the territory where they live]
- Population - The number of citizens living in the territory
- Stability - How peaceful and stable the territory is
- Tax Base - The potential value a territory can collect in taxes
- Tax - The actual tax value for the territory

6.5.7 Version 1.0

Turn	Territory	Lawfull	Loyalty	Population	Stability	Tax Base	Tax
0	”H6”	0.95	1	20000	1	19900	0.05
0	”F6”	0.95	1	30000	1	29900	0.99
0	”G5”	0.95	1	20000	1	19900	0.99
0	”I5”	0.95	1	20000	1	19900	0.05
0	”J6”	0.95	1	30000	1	29900	0.05
1	”H6”	0.96	1	20517	1	20619	0.05
1	”F6”	0.84	0.5	30848	0.59	27356	0.99
1	”G5”	0.84	0.5	20517	0.53	18157	0.99
1	”I5”	0.96	1	20517	1	20619	0.05
1	”J6”	0.96	1	30848	1	31058	0.05
2	”H6”	0.97	1	21047	1	21361	0.05
2	”F6”	0.62	0.25	31479	0.36	21006	0.99
2	”G5”	0.6	0.25	20881	0.3	13474	0.99
2	”I5”	0.97	1	21047	1	21361	0.05
2	”J6”	0.97	1	31719	1	32257	0.05

Table 6.18: 2PL scenario test on version 1.0

After the third turn the population distribution is still very similar between Slice and Dice as it can be seen in figure 6.4. Stability and Loyalty fall by similar amounts, which the lawfull property decreases slower. Still after three turns the population reproduction difference is small and the only the tax base is much better for the Dice player. Slice has collected much more money in taxes, and the populations keep on paying even though on smaller percentages.

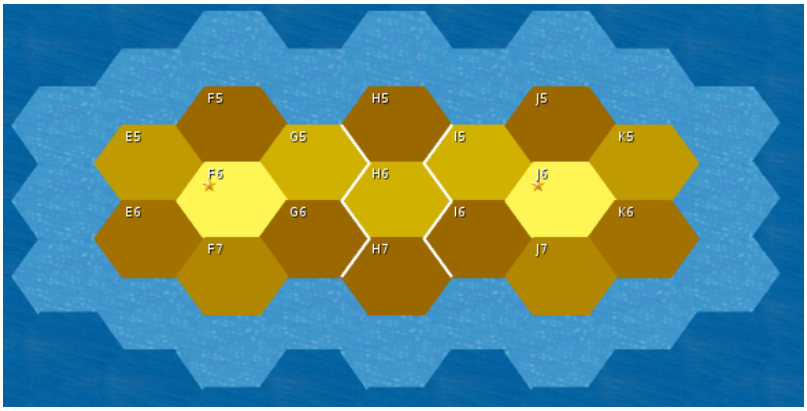

Figure 6.4: Population distribution for version 1.0 after 3rd turn

6.5.8 Version 2.0

Version 2.0 starts with the same population distribution, but because the population effects are much more noticeable, a visual representation after each turn is provided.

Turn	Territory	Lawfull	Loyalty	Population	Stability	Tax Base	Tax
0	"H6"	0.95	1	20000	1	19900	0.05
0	"F6"	0.95	1	30000	1	29900	0.99
0	"G5"	0.95	1	20000	1	19900	0.99
0	"I5"	0.95	1	20000	1	19900	0.05
0	"J6"	0.95	1	30000	1	29900	0.05
1	"H6"	0.95	0.95	21734	1	21484	0.05
1	"F6"	0.73	0.03	32058	0.59	24803	0.99
1	"G5"	0.73	0.03	18487	0.53	14396	0.99
1	"I5"	0.96	1	20517	1	20596	0.05
1	"J6"	0.96	1	30819	1	30986	0.05
2	"H6"	0.94	0.93	23391	1	22903	0.05
2	"F6"	0.57	0.02	33225	0.36	20328	0.99
2	"G5"	0.55	0.02	16390	0.3	9914	0.99
2	"I5"	0.97	1	21047	1	21315	0.05
2	"J6"	0.97	1	31658	1	32108	0.05
3	"H6"	0.95	0.96	24007.89	1	23745.5	0.05
3	"F6"	0.42	0.01	2151.76	0.23	2298.16	0.99
3	"G5"	0.39	0.01	131.28	0.18	951.2	0.99
3	"I5"	0.97	1	21589.55	1	21841.86	0.05
3	"J6"	0.97	1	32518.03	1	32942.49	0.05

Table 6.19: 2PL scenario test on version 2.0

Because version 2.0 is more complex and harder to analyze, it's necessary to check the log files to explain the current state. Partial log from turn 1:

```
37 migrating from F5 to E5
61 migrating from F5 to F6
56 migrating from F5 to G5
Territory (F5) @ The Humans are unhappy, and 349 of them decided to migrate
Territory (F6) @ The Humans are unhappy, some of them are tempted to migrate
```

After the first turn is processed, the populations of Slice start migrating to territories with better taxes. Since Slice's capital is surrounded by other territories with high taxes, they don't have where to go, and that explains why the populations increases in territory F6. Loyalty now falls much faster than in version 1.0, stability varies slower than loyalty but still faster than in version 1.0. The percentage of lawfull population also changes faster than in version 1.0, but slower than loyalty or stability.

Figure 6.5 shows a slight darker color on the slices side, a very small difference yet.

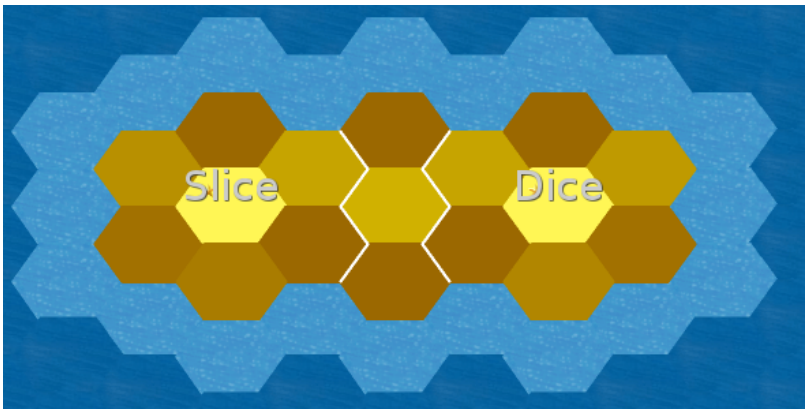

Figure 6.5: Population distribution of version 2.0 after turn 1

Partial log from turn 2:

```
Territory (F5) @ The Humans are unhappy, and 2407 of them decided to migrate
Territory (F6) @ The Humans are unhappy, some of them are tempted to migrate
Territory (E5) @ The Humans in E5 are revolting against you
Territory (E6) @ The Humans in E6 are revolting against you
Territory (F7) @ The Humans in F7 are revolting against you
Territory (F5) @ The Humans in F5 are revolting against you
```

The second turn the populations start to revolt against Slice, figure 6.5.8 show a massive contrast from the previous turn. Also the number of populations migrating increased to much larger numbers.

Partial log from turn 3:

```
Territory (E5) @ The Humans in E5 are revolting against you
```

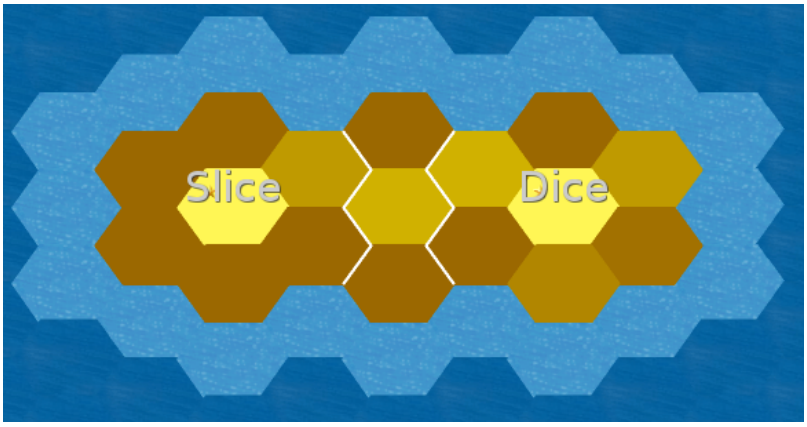

Figure 6.6: Population distribution of version 2.0 after turn 2

Figure 6.7: Population distribution of version 2.0 after turn 3

```
Territory (G5) @ The Humans in G5 are revolting against you
Territory (G6) @ The Humans in G6 are revolting against you
Territory (F7) @ The Humans in F7 are revolting against you
Territory (F6) @ The Humans in F6 tried to revolt against you, armies loyal to you stopped th
Territory (E6) @ The Humans in E6 are revolting against you
Territory (F6) @ The Humans in F6 are revolting against you
```

After the third turn with a taxes at 99%, almost all the population of Slice has abandoned him. They also formed armed militias that are now enemies of Slice. That's why the population dropped so much on every territory. In F6, Slice has it's army and it's able to stop smaller revolts from taking place, but the large numbers of revolting populations are finally able to defeat Slice's army.

Figure 6.7 shows now a darker picture for Slice's side, it possesses almost no population, and will soon disappear.

6.6 Performance

6.6.1 Version 1.0

Game	Import	Clone	Turn 1	Turn 2	Turn 3
AB101	100.87	439.27	286.32	290.63	289.67
QI	51.75	215.85	191.24	191.57	191.85
Final	60.37	178.93	146.87	146.34	146.16
CM40	61.89	92.42	92.9	92.38	92.05
WOT20	38.12	91.95	94.25	92.93	92.55
OF20	21.17	43.62	42.56	42.54	42.61
CJK10	10.56	21.15	20.33	20.34	20.33
Juri	7.21	17.88	20.69	20.72	20.41
OF6	3.85	8.31	10.1	10.2	10.17
FTS4	1.68	4.93	7.44	7.42	7.55
2PL	1.66	4.04	5.15	5.07	5.11
2PLT	0.53	1.84	3.55	3.52	3.56
2PLTNoSea	0.4	1.59	3.61	3.51	3.52

Table 6.20: Version 1.0 times in seconds

The heavier scenarios are, off course, the larger ones. AB101, QI, Final and CM40 are the slowest to import. AB101 shows an abnormal import time when compared to the other scenarios. The dominant metrics for the import times seems to be populations, facilities, territories, lands and resources, which makes sense since those are the data present in the importer files.

Cloning times show a similar picture, AB101 is by far the heaviest scenario to clone. Here the dominant metric is the number of facilities. AB101's high number of facilities and resources explains why it is so slow to clone.

Because all the tests are done over vanilla scenarios (without player's orders) the true height of units and contingents is disguised. Overlooking that fact, we can deduce by comparing the results of QI with AB101 that the reason AB101 takes more than 100 seconds to process is the high number of facilities, because this is the only metric where AB101 significantly wins over QI.

6.6.2 Version 2.0

Game	Import	Clone	Turn 1	Turn 2	Turn 3
AB101	74.99	311.18	300.87	295.79	290.29
QI	57.17	182.78	201.82	188.25	187.42
Final	60.85	128.12	161.76	160.19	161.71
CM40	42.52	80.91	146.95	160.48	144.68
WOT20	35.85	85.44	97.61	96.87	96.48
OF20	23.99	44.17	54.56	53.82	53.79
CJK10	12.74	21.69	26.81	26.37	27.36
Juri	8.44	20.56	19.6	19.59	19.93
OF6	5.15	11.12	11.8	11.88	11.95
FTS4	2.06	6.02	5.98	5.99	6.12
2PL	2.52	5.4	6.02	6.78	6.76
2PLT	0.99	3.08	3.13	3.08	3.07
2PLNSea	0.44	2.22	2.03	2.04	2.04

Table 6.21: Version 2.0 times in seconds

Version 2.0 shows a picture that is very similar to that of to version 1.0. To better understand the differences, each column is compared independently in the next subsections..

6.6.3 Importer times

Game	Version 1	Version 2	Speedup	Percentage
AB101	100.87	74.99	1.35	35%
QI	51.75	57.17	0.91	-9%
Final	60.37	60.85	0.99	-1%
CM40	61.89	42.52	1.46	46%
WOT20	38.12	35.85	1.06	6%
OF20	21.17	23.99	0.88	-12%
CJK10	10.56	12.74	0.83	17%
Juri	7.21	8.44	0.85	-15%
OF6	3.85	5.15	0.75	-25%
FTS4	1.68	2.06	0.81	-19%
2PL	1.66	2.52	0.66	-34%
2PLT	0.53	0.99	0.54	-46%
2PLNSea	0.4	0.5	0.81	-19%
Total	100.87	117.51	0.86	-14%

Table 6.22: Scenario Import Comparison

A direct comparison between import times from version 1 to version 2 reveals a rather curious fact: the small scenarios are slower to import on the new version but the larger ones are faster! Several reasons could be responsible for this:

- On version 2.0 the tables have more data so writing and reading records takes a tad longer. This can partially explain why the small scenarios are slower, because they write a few records only any small increase in each record is more noticeable.
- The database schema of version 2.0 is simpler as far as the facility and territories-related tables go. This allows for speedups in scenarios with lots of facilities or territories (which tend to be the slower to import anyway).

Globally, version 2.0 is 14% slower than version 1.0. Still, if only the larger scenarios are considered version 2.0 comes up further ahead.

6.6.4 Cloning times

Game	Version 1	Version 2	Speedup	Percentage
AB101	439.27	311.18	1.41	41%
QI	215.85	182.78	1.18	18%
Final	178.93	128.12	1.4	4%
CM40	92.42	80.91	1.14	14%
WOT20	91.95	85.44	1.08	8%
OF20	43.62	44.17	0.99	-1%
CJK10	21.15	21.69	0.97	-3%
Juri	17.88	20.56	0.87	-13%
OF6	8.31	11.12	0.75	-25%
FTS4	4.93	6.02	0.82	-18%
2PL	4.04	5.4	0.75	-25%
2PLT	1.84	3.08	0.6	-4%
2PLNSea	1.59	2.22	0.72	-28%
Total	1121.77	902.7	1.24	24%

Table 6.23: Scenario Import Comparison

The gains of version 2.0 are more noticeable because the slowest map is 41% faster to clone. By removing AB101 from the comparison, the total times would come to about the same, but, since version 2.0 is faster on the other big maps, it would still be preferable.

6.6.5 Turn processing times

Game	Version 1	Version 2	Speedup	Percentage
AB101	289.67	295.65	0.98	-2%
QI	191.85	192.5	1	0%
Final	146.16	161.22	0.91	-9%
CM40	92.05	150.71	0.61	-39%
WOT20	92.55	96.99	0.95	-5%
OF20	42.61	54.06	0.79	-21%
CJK10	20.33	26.85	0.76	-24%
Juri	20.41	19.71	1.04	4%
OF6	10.17	11.88	0.86	-14%
FTS4	7.55	6.03	1.25	25%
2PL	5.11	6.52	0.78	-22%
2PLT	3.56	3.09	1.15	15%
2HNoSea	3.52	2.04	1.72	72%
Total	925.54	1027.25	0.9	-10%

Table 6.24: Turn Processing Comparison

During the turn processing phase version 2.0 is consistently slower than version 1.0. The added overhead from the agent's framework can be shaken off by the improvements at the schema level. CM40 is the scenario with the highest agents:lands and agents:territories ratio, and it demonstrates most heavily the agent's overhead. The global picture looks promising for, even though many new behaviors have been added, version 2.0 is only 10% slower.

Another important factor to consider is, version 2.0 can process multiple games at the same time. On a machine with multiple cores version 2.0 can theoretically provide an almost linear improvement on the formula 4.3.

For the sake of reasoning, let us imagine that a hypergame exists whose turn time processing represents processing all the presented scenarios. Making it a daily game, on a dual core like the test server, for version 1.0 we have:

$$n = \frac{60 * 24 * 60s}{925.54s} <=> n = 93.35 \tag{6.3}$$

For version 2.0:

$$n2 = \frac{60 * 24 * 60s * 2}{1027.25s} <=> n2 = 168.21 \tag{6.4}$$

Computing the speedup we have:

$$speedup = \frac{n2}{n1} = 1.80 \tag{6.5}$$

This means that version 2.0 can theoretically run 80% more games on a dual core than version 1.0.

To verify this assertion two Q1 games were cloned from the Q1 scenario and processed in parallel. The final results are, one Q1 processed in 200.14s and another in 214.94 seconds,

and their average is $\frac{200.14s+214.94s}{2} = 207.54s$. Comparing that to the 192s a single Q1 took to process in the single process test we can calculate the real speedup per extra core and adjust formula 6.2 with the experimental factor.

$$\frac{192}{207.54} = 0.925 \tag{6.6}$$

And now we have a formula adapted to the experimental data.

$$n = \frac{60 * 24 * f_C(computer) * 0.925^{(f_C(computer)-1)} * f_D(game)}{f_T(game)} \tag{6.7}$$

Using result 6.4 we can extrapolate how many human players can be served by a single server with the same characteristics has the test server. This only works for vanilla scenarios and therefore represents an upper bond on the maximum number of players.

Considering that the hypergame has a number of lands equal to the sum of all the test scenarios it adds up to 391 human players for hypergame, which means that the upper bond of players per server would be:

$$168.21 * 391 = 65770 \tag{6.8}$$

Now, using the live data from the Almansur 1.0 server, where a daily game like Q1 takes around 26 minutes to process, a more realistic estimate can be calculated using formula 6.7.

$$n = \frac{60 * 24 * 2 * 0.925^{2-1} * 1}{26} = 102 \tag{6.9}$$

$$102 * 107 = 10914 \tag{6.10}$$

To sum it up, at most 10914 can be supported with the current hardware with 107 player games. Being limited mainly by the speed of ruby and the multiple access to the database, the solution to improve the number of players per server may be a port of the game core to another language. Perhaps porting only the most affected parts is required, so further benchmarking and profiling are necessary to identify the turn processing bottlenecks.

Chapter 7

Conclusions and future work

All good things must come to an end

Chaucer

In the previous Chapters the game Almansur was introduced, the main identified issues with Almansur were:

- No single player mode, where the new players can learn the game at their own pace
- Almansur has no AI
- An overly static world
- Massive games have thousands of players playing at the same time, causing performance to be a major issue
- Complex world with partially accessible environment

To overcome these issues, Almansur Agent Architecture (AAA) is presented in Chapter 5. The developed architecture had the following objectives in mind:

- **Flexibility** - The framework should allow for the development of new content and agents without significant additional effort
- **Performance** - The framework performance must be good enough to support thousands of agents running at the same time
- **Ease of use** - No special configuration should be needed for things to work out of the box
- **Configurable** - The framework should provide the means to create special unique agents only by customizing their behaviors
- **Human Interaction** - The human players should be able to influence how their subjects agent's behavior

The AAA was applied to Almansur creating version 2.0 of the game. Two main areas where tested to compare version 1.0 and 2.0: simulation and performance. Simulation tests try to identify if version 2.0 really improves over the world of version 1.0. Performance tests are focused on the turn processing times, which is the main bottleneck of the game.

7.1 Simulation

The basic blocks for developing AI in Almansur were accomplished. The virtual world is now more dynamic and with the improved behaviors of populations and other agents, it's expected to be more interesting for players.

Version 2.0 resolved some of the main issues of simulation present in version 1.0.

- Populations were easy to exploit, now they have a mind of their own and players must keep them happy if they want populations to stay with them
- The contingents used to magically provide food for themselves, this lead for major balance issues. The richer lands could simply recruit huge stacks, until they overpower the poor ones. Now it's necessary to feed the contingents at the end of each turn, making it harder to sustain huge armies.

The other great advantage of version 2.0 is the clear separation of modules. New agents and behaviors can be implemented without changing the game core. It's now easier to produce new content for the game without having the risk of introducing new bugs.

7.2 Turn Processing

On the tests chapter, only the vanilla scenarios where tested. While this shades some light on what to expect performance wise, and allows to establish an upper bound on the maximum number of players using the same game server.

Even with the overhead introduced by AAA the game performance is roughly the same. The increased flexibility of version 2.0 allows for futher developments to be easily introduced, no more changes to the game core are necessary.

This work fixed the major issues with the turn processing of version 1.0, the major issue that remains is the time it takes to process each turn. A few hypotheses are possible to improve that area,

- Change the implementation language. Currently the game core is implemented in ruby [1] which is a beautiful and elegant programming language, but it is dead slow. Reimplementing the core on other language like c++ or java might be the only safe way to really scale the game to millions of players.
- Change the ruby interpreter. Multiple implementations of ruby are surfacing today, within some time they might solve the performance issues on their own.

7.3 AI

There are two fields where the AI can be improved. At the lower level, the world can be evolved to incorporate more core agents. The markets show potential to be regulated by agents, to more accurately simulate the real world demand/offer price relations. Also the already developed agents can be improved, new behaviors can be implemented to incorporate

[1] Ruby is a language of careful balance. Its creator, Yukihiro "matz" Matsumoto, blended parts of his favorite languages (Perl, Smalltalk, Eiffel, Ada, and Lisp) to form a new language that balanced functional programming with imperative programming.[30]

learning and better social skills. In particular personality agents can be improved to reproduce human emotions.

On the higher level, AI to replace the human player is surely one of the most important features. This would solve one of the most problematic areas of the game, drop-out players. It would also bring new possibilities of gameplay, like a human alliance against the computer.

7.4 Architecture

Due to ruby's green threading[29] each ruby application uses only one CPU at the time. So no 1-1 mapping between OS threads and ruby threads exist. Knowing that it makes no sense to create a multi-threading architecture for the current ruby interpreter. The new ruby version 2.0[30] and JRuby [2] support 1-1 native threads. This open new options for future implementations. With the growing number of cores per CPU, multi-thread programming paradigms will become more important, over the coming years. Agent architectures are already pretty good to apply multiple threads, if each agent can process in parallel. Each agent would then occupy a thread, but creating thousands of threads can hurt the system performance, so a thread pool system could be the best option.

[2] JRuby is an 100% pure-Java implementation of the Ruby programming language.[31]

Appendix A

Almansur 1.0

Column	Description
x	Horizontal coordinate for the territory
y	Vertical coordinate for the territory
mov class	Type of territory, L - Land, S - Sea
altitude	Territory's average distance to the sea level in meters. It is a positive value for shore and negative for water covered territories
relief	Average sharpness of the territory, a regular territory will have a low relief, while a cliff will have a very high relief
swampness	Average swampness of the territory
fertility	Average reproduction rate of this territory
arb	Average tree density of this territory
goldr	Gold naturaly available in this territory
ironr	Iron naturaly available in this territory
stoner	Stone naturaly available in this territory
pisc	Fish naturaly available in this territory
wildcav	Free wild horses available in this territory
wildgame	Small animals naturaly available in this territory
wildwarg	Free wild wargs available in this territory
terrname	Name of the territory (Empty for the name to be based on the territory coordinates)
pophuman	Number of free humans present in this territory
poporc	Number of free orcs present in this territory
popuruk	Number of free uruk-hais present in this territory
popdwarf	Number of free dwarfs present in this territory
popelf	Number of free elfs present in this territory
popbarb	Number of free barbarians present in this territory
slaves	Number of human slaves present in this territory
cityname	Name of the city (empty for the cityname to be "city territory name") present in this territory
citylevel	Level of the city (zero is no such facility exists) present in this territory

cityorclevel	Level of the orc encampment (zero is no such facility exists) present in this territory
citydwarflevel	Level of the underground city (zero is no such facility exists) present in this territory
portlevel	Level of the port facility (zero is no such facility exists) present in this territory
farmlevel	Level of the farm (zero is no such facility exists) present in this territory
walllevel	Level of the fortress (zero is no such facility exists) present in this territory
goldmine	Level of the gold mine (zero is no such facility exists) present in this territory
ironmine	Level of the iron mine (zero is no such facility exists) present in this territory
villagefortlevel	Level of the fortified villages facility (zero is no such facility exists) present in this territory
marketlevel	Level of the market facility (zero is no such facility exists) present in this territory
livcav	Number of horses present in this territory warehouses
livgado	Number of cattle heads present in this territory warehouses
livpigs	Number of pigs present in this territory warehouses
livwargs	Number of warg animals present in this territory warehouses
cereal	Quantity in tons for the grain resource present in this territory warehouses
saltfish	Quantity in tons for the salted fish resource present in this territory warehouses
saltmeat	Quantity in tons for the salted meat resource present in this territory warehouses
pe	Name of the land that owns this territory (must be an entry from the lands files under the column name)

Table A.1: Almansur's territories file format

Appendix B

Agent Minds

Every Arthur needs its Merlin

Hamilton

B.1 BNF for AML

```
<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<integer> ::= <digit> | <integer> <digit>
<number> ::= <integer> | + <integer>.<integer> | - <integer>.<integer>
<letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
<identifier> ::=
      <letter>
   |  <identifier> <letter>
   |  <identifier> <digit>

<symbol> ::= :<identifier>
<symbol_list> ::= <symbol> | <symbol_list> , <symbol>

<condition> ::= <equals> | <not_equals> | <min> | <max>
<condition_list> ::= <condition> | <conditions_list> , <condition>
<property> ::= "{"
                  :name => <symbol>, <condition_list>
               "}"
<property_list> ::= <property> | <property_list>, <property>
<properties> ::= :Property => "["
                     <property_list>
                    "]"
<sensors> ::= "{"
                  :name => <symbol>,
                  :method => <symbol>,
                  <property_list>,
```

```
                  :parameters => "[" <symbol_list> "]"
               "}"
<event> ::= <symbol> => "'"<identifier>"'"

<behavior> ::= <symbol> => "{"
                              [<sensors> ,]
                              [<properties> ,]
                              [<event>]
                           "}"
<behavior_list> ::= <behavior> | <behavior_list> , <behavior>

<module> ::= "{"  <behavior_list> "}"
<module_list> ::= <module> | <module_list> , <module>

<agent_mind> ::= "{" <module_list> "}"
```

Here are the mind specifications for Almansur 2.0 agents (using AML).

B.2 Unit

```
{
  :UnitBehavior => {

    :upkeep => {
      :EventType =>'end_of_turn'
    },

    :eat => {
      :EventType =>'end_of_turn'
    },

    :revolt => {
      :EventType =>'end_of_turn',
      :Property =>[
        { :name => :loyalty, :max => 0.2}
      ]
    },

    :send_to_land => {
      :EventType =>'end_of_turn',
      :Property =>[{ :name=> :ai, :equals => true }]
    },

    :arrival => {
      :EventType =>'arrival',
```

```
    },

    :force_rest => {
      :EventType =>'arrival',
      :Property =>[
        { :name => :ai, :equals => true },
        { :name => :status, :max => 0.2}
      ],
      :Sensor => [
        { :name => :territory_sensor,
          :method => :known_friendly_armies_size,
          :parameters => [ :territory_id],
          :max => 0}
      ]
    },

    :identify_scouts => {
      :EventType => 'arrival'
    },

    :resume => {
      :Property =>[
        { :name => :ai, :equals => true },
        { :name => :status, :min => 0.3},
        { :name => :status_plus_risk_taking, :min => 0.9},
        { :name => :rest?, :equals => true},
        { :name => :has_new_orders?, :equals => true},

      ],
    },

    :process_day => {
      :EventType =>'start_of_day'
    }
  }
}
```

B.3 Contingent

```
{
   :ContingentBehavior => {
     :process_day => {
       :EventType => 'start_of_day'
     },
```

```
    :upkeep => {
      :EventType =>'end_of_turn'
    },
    :eat => {
      :EventType =>'end_of_turn'
    },

    :disband => {
      :EventType =>'end_of_turn',
      :Property =>  [{ :name => :loyalty, :max => 0.2}]
    },
  }
}
```

B.4 Population

```
{
  :PopulationBehavior => {

    :reproduction => {
      :EventType => 'reproduction',
      :Property => [
        { :name => :in_fortress, :equals => false}
      ]
    },

    :resource_reproduction => {
      :EventType => 'reproduction',
      :Property => [
        { :name => :in_fortress, :equals => false}
      ]
    },

    :merge =>  {
      :EventType =>'new_population',
      :Sensor => [
        { :name => :territory_sensor,
          :method => :populations_by_race_and_state_size,
          :parameters =>[ :territory_id, :race_id, :social_state_id, :in_fortress],
          :min => 2 }
        ]
      },

    :revolt => {
```

```
  :EventType =>'end_of_turn',
  :Property => [
    { :name => :happiness, :max => REVOLT_THRESHOLD},
    { :name => :loyalty, :max => REVOLT_THRESHOLD},
    { :name => :in_fortress, :equals => false}
  ],
  :Sensor => [
    { :name => :territory_sensor,
      :method => :stability,
      :parameters => [ :territory_id],
      :max => REVOLT_THRESHOLD}
  ],
},

:migrate => {
  :EventType =>'end_of_turn',
  :Property => [
    { :name => :happiness, :max => HAPPINESS_THRESHOLD},
    { :name => :in_fortress, :equals => false}
  ]
},

:react_to_tax_change => {
  :EventType =>'tax_change'
},

:react_to_diplomatic_changes => {
  :EventType =>'diplomatic_change'
},

:react_to_conquer => {
  :EventType =>'conquer'
},

:process_siege_day => {
  :EventType =>'update_siege',
  :Property => [
    { :name => :in_fortress, :equals => true}
  ]
},

:prepare_for_siege => {
  :EventType =>'start_of_siege'
}
```

```
    }
  }
```

B.5 Territory

```
{
    :TerritoryBehavior => {

      :produce => {
        :EventType => 'production'
      },

      :react_to_diplomatic_changes => {
        :EventType =>'diplomatic_change'
      },

      :react_to_conquer => {
        :EventType =>'conquer'
      },

      :update_stability => {
        :EventType => 'end_of_turn'
      },

      :update_available_emergency_recruits => {
        :EventType => 'end_of_turn'
      },

      :update_whiteboard => {
        :EventType => 'end_of_turn'
      },

      :react_to_unit_arrival => {
        :EventType => 'arrival'
      },

      :upgrade_facility => {
        :EventType => 'upgrade'
      },

      :downgrade_facility => {
        :EventType => 'downgrade'
      }
    }
```

```
}
```

B.6 Personality

```
{
   :PersonalityBehavior => {

     :upkeep =>  {
       :EventType =>'end_of_turn',
       :Property => [{:name => :pe_id, :not_equals => nil}]
     },

     :accept_job => {
       :EventType =>'start_turn',
       :Property => [{:name => :pe_id, :equals => nil}]
     },

     :change_market => {
       :EventType =>'end_of_turn',
       :Property => [
         {:name => :turns_unemployeed, :min => MAX_TURNS_ON_MARKET},
         {:name => :market_id, :not_equals => GLOBAL_MARKET}
       ]
     }
   }
}
```

B.7 Battle Manager

```
{
  :BattleBehavior => {
    :process_day_battles => {
      :EventType => 'start_of_day'
    }
  }
}
```

B.8 Hostile Action Manager

```
{
  :HostileActionBehavior => {
    :process_day_conquers => {
      :EventType => 'start_of_day'
    },
```

```
      :process_day_sieges => {
        :EventType => 'start_of_day'
      }
    }
  }
```

B.9 Game

```
{
  :GameBehavior => {
    :process_start_turn => {
      :EventType => 'start_of_turn'
    }

    :process_end_of_turn=> {
      :EventType => 'end_of_turn'
    }
  }
}
```

Bibliography

[1] Benjamin wooton , *Designing for emergence*, AI game programming wisdom, *Thomson Delmar Learning*, page 53

[2] John McCarthy, *What is artificial Intelligence*, On-line,, *last accessed on 1 September of 2007*

[3] Durfee, E.H., Lesser, V.R. and Corkill, D.D. *Trends in Cooperative Distributed Problem Solving*. In: IEEE Transactions on Knowledge and Data Engineering, March 1989, KDE-1(1), pages 63-83.

[4] Roberto A. Flores-Mendez, *Standardization of Multi-Agent System Frameworks*, ACM Crossroads 5(4) 1999

[5] Http://en.wikipedia.org/wiki/A*, online, last access on 9th February 2007

[6] AI game programming wisdom 2, Steve Rabin, 2004, Charles River Media, ISBN 1584502894

[7] Remco Straatman, William van der Sterren, Arjen Beij,Killzone's AI: dynamic procedural combat tactics, GDC 2005 Procedings

[8] Dave Pottinger, Terrain Analysis in Realtime Strategy Games, Ensemble Studios, GDC 2000 procedings

[9] AI game programming wisdom, Steve Rabin, 2002, Charles River Media, ISBN 1584500778

[10] Http://en.wikipedia.org/wiki/F.E.A.R.,online, last access on 9th February 2007

[11] Http://en.wikipedia.org/wiki/First_person_shooters, online, last access on 9th February 2007

[12] http://en.wikipedia.org/wiki/Real-time_strategy, online, last access on 9th February 2007

[13] Http://en.wikipedia.org/wiki/Turn-based_strategy, online, last access on 9th February 2007

[14] Http://pc.ign.com/articles/452/452317p1.html, online, last access on 9th February 2007

[15] Http://www.kantor.com/blog/2005/01/why_doom_3_well_sucks.shtml, online, last access on 9th February 2007

[16] Jeff Orkin, Three States and a Plan: The A.I. of F.E.A.R, GDC 2006 procedings

[17] Http://en.wikipedia.org/wiki/4x, online, last access on 9th February 2007

[18] http://www.timegate.com/games.php, online, last access 19th June 2007

[19] Machine Learning, Tom Mitchell,1997, McGraw-Hill, ISBN 0070428077

[20] An Introduction to MultiAgent Systems, Michael Wooldridge, 2002, John Wiley and Sons, ISBN 047149691

[21] http://www.uml.org/, online, last access 19th June 2007

[22] Http://en.wikipedia.org/wiki/Finite_state_machine,online, last access on 9th February 2007

[23] http://en.wikipedia.org/wiki/Rete_algorithm, online, last access on 9th February 2007

[24] AI game programming wisdom 3, Steve Rabin, 2006, Charles River Media, ISBN 1584504579

[25] http://www.almansur.net/game/, online, last access 9th September 2007

[26] http://www.almansur.net/forum/viewtopic.php?pid=1554, online, last access 30th September 2007

[27] NAUR, Peter (ed.), "Revised Report on the Algorithmic Language ALGOL 60.", Communications of the ACM, Vol. 3 No.5, pp. 299-314, May 1960.

[28] Russell S, Norvig P (1995) Artificial Intelligence: A Modern Approach, Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey

[29] http://www.headius.com/rubyspec/index.php/Ruby_Threading, online, last access on 10th Setember 2007

[30] http://www.ruby-lang.org/en, online, last access on 10th Setember 2007

[31] http://jruby.codehaus.org, online, last access on 10th Setember 2007

[32] http://softwareengineering.vazexqi.com/files/pattern.html , online, last access on 10th Setember 2007