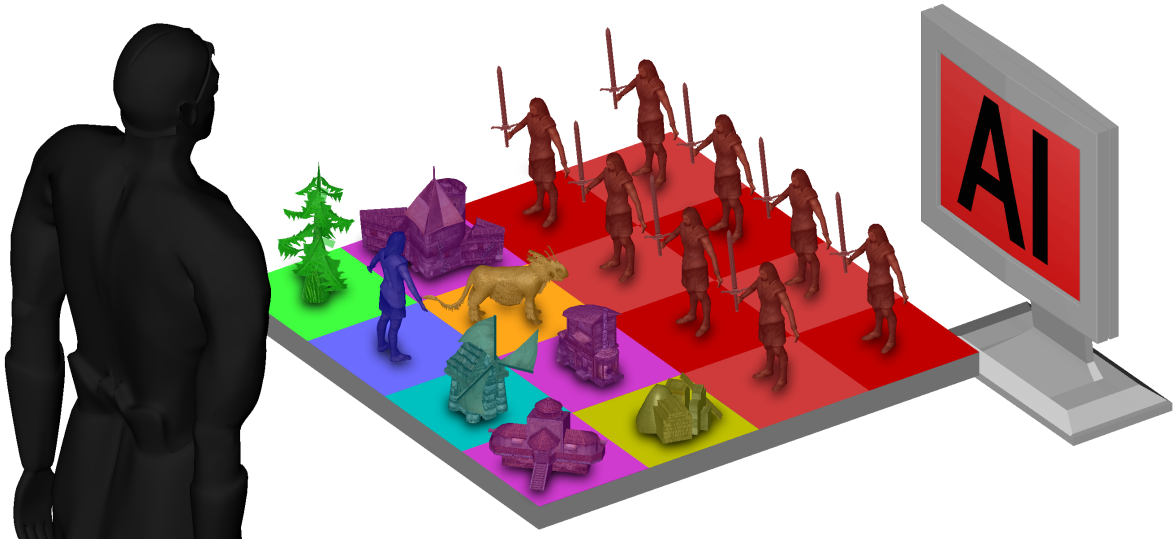
Using Genetically Optimized Artificial Intelligence to improve Gameplaying Fun for Strategical Games

Christoph Salge*
Adaptive Systems Group
University Hertfordshire

Christian Lipski†
TU Braunschweig

Tobias Mahlmann‡
Braunschweig School of Arts

Brigitte Mathiak§
TU Braunschweig



While human beta-testers instinctively try to find a balanced gaming style, an AI implementation is able to quickly find exploits in the game system.

Abstract

Fun in computer games depends on many factors. While some factors like uniqueness and humor can only be measured by human subjects, in a strategical game, the rule system is an important and measurable factor. Classics like chess and GO have a millennia-old story of success, based on clever rule design. They only have a few rules, are relatively easy to understand, but still they have myriads of possibilities. Testing the deepness of a rule-set is very hard, especially for a rule system as complex as in a classic strategic computer game. It is necessary, though, to ensure prolonged gaming fun.

In our approach, we use artificial intelligence (AI) to simulate hours of beta-testing the given rules, tweaking the rules to provide more game-playing fun and deepness. To avoid making the AI a mirror of its programmer's gaming preferences, we not only evolved the AI with a genetic algorithm, but also used three fundamentally different AI paradigms to find boring loopholes, inefficient game mechanisms and, last but not least, complex erroneous behavior.

CR Categories: I.2.1 [Artificial Intelligence]: Applications and Expert Systems—Games; I.2.m [Artificial Intelligence]: Miscellaneous—Genetic Algorithm; K.8.0 [Personal Computing]: General—Games

Keywords: fun, strategic games, genetic algorithm

*e-mail: c.salge@rise-of-atlantis.de
†e-mail:c.lipski@cg.cs.tu-bs.de
‡e-mail:t.mahlmann@tu-bs.de
§e-mail:mathiak@gmail.com

1 Introduction

1.1 Motivation

Developing a state of the art computer game today requires the investment of a lot of money and time. To minimize the risks of such an investment it is necessary to evaluate the possible success of the product at an early stage. Aside of factors that are not controlled by the developers, such as marketing, one wants to ensure that the game is attractive to the players, that it creates fun. This again depends on a combination of several factors, one being the actual game, or to be more precise: the game mechanics. If the game appears to be unbalanced, or unfair, it might not be well received. One way to avoid this problem is to copy existing game mechanics and then obfuscate that by moving them to a different theme setting. Another way, to ensure the quality of innovative game mechanics, normally involves a lot of game testers, thus costing time and money.

1.2 Idea

Our case study evaluates a different approach, where AIs are used to evaluate the game mechanics for critical errors. AIs are fast and cheap to operate, and they do not need a graphical representation, if a well defined interface exists. So, they can be used on early non-graphical prototypes. The costs are minimal as most games require the development of an AI anyway to serve as an opponent for the player. Our idea is to employ a genetic algorithm to adapt the AI and then analyze their winning strategies to find flaws in the game mechanics. The aim is not to show that AIs are able to determine if a game is actual fun to play, but if they are capable of finding flaws in the games that would ruin the game for human players.

1.3 Related Work

AI research shows, that it is possible to use genetic algorithms to improve the performance of AIs in computer games [Ponsen et al. 2005]. Several works exist, that explore the different strategies in designing and adapting AIs, such as reinforcement learning [Andrade et al. 2005], or coevolution [Demasi and de O. Cruz 2003]. These works also illustrates that AIs can be applied to different game mechanics, such as action games or real time strategy games. Also, the underlying model or paradigm of their approaches varies from the control of single units to high level decision making, similar to our approach of using and comparing several models. But unlike us, they solely focus on improving the game play of the AIs, to make them more challenging, or human like opponents.

The question of what makes a game fun is also discussed in [Koster 2005], although it does not introduce a formalism to approach it systematically. A good overview and synthesis of the current game definitions is presented in [Juul 2003]. Metric measurements of game fun were also attempted by [Yannakakis and Hallam 2006], introducing psychological qualities such as curiosity into an AI, and measuring it's satisfaction. Our approach now tries to combine the evolution of better AIs with the measurement game features related to fun.

1.4 Overview

For a systematic analysis it is necessary first to define what a computer game actually is, which parameters it has, and how they can be measured. This will be done in the first section, where we also discuss how those parameters correlate with the player having fun. We focus especially on the negative cases, since it is more convincing to argue, what a player will definitely not enjoy. This idea was tested during the development of "Rise of Atlantis"[1], a turn based strategy game developed at the TU Braunschweig to investigate several questions related to game development.

The following three sections introduce the key aspects of the game, and how those are important to our project. First, an overview of the game mechanics is given. The game was designed to reach a level of complexity that makes it impossible to find an optimal strategy, both by mathematical analysis and by evaluation of an experienced player. For space constraints only the key aspects are explained in more detail.

The next section describes the general program structure of the game, describing how to interface the AI with the game server without using a graphical representation. The advantages of a client-server approach in combination with our method are discussed, and we show that the AIs are only able to get the same amount of information that a player would receive. The graphical interface was mainly developed as a case study for a different question, but serves us as a tool to interface with the game.

The AI section presents the three types of AIs that were used to evaluate the game mechanics of "Rise of Atlantis". It describes in detail how the AIs were evolved with genetic algorithms, and how we determined the fitness of the resulting AIs. We then take a look at the resulting strategies and examine them to find the critical flaws in the game mechanics of "Rise of Atlantis". Several errors of different types are presented and discussed. The paper closes with some conclusions.

2 Theory of Games

This section will provide some of the theoretical background needed to analyze a game. The Encyclopaedia Britannica defines games as "a universal form of recreation generally including any activity engaged in for diversion or amusement and often establishing a situation that involves a contest or rivalry". We use a less holistic definition from Juul [Juul 2003], that not only defines what a game is, but does so by naming several features a game has.

2.1 Definition

Juuls definition [Juul 2003] identifies games by six parameters.

1. Fixed Rules: A game has to be described by a set of unambiguous rules. This is also an absolute necessity, if a game is programmed on a computer, since everything has to be systematically described before implementation.

2. Variable and Quantifiable Outcome: A game has to have more then one possible ending, which are distinguishable by the players.

3. Valorisation of the Outcome: Some of the outcomes have to be better then others.

4. Players Effort: The actions of the player have an effect on the game, usually it is harder to achieve a more positive outcome.

5. Attachment of the player to the outcome: The player wants to achieve the better outcome. If he does, it makes him feel good

6. Negotiable consequences: The game itself should not have consequences on the world, apart from winning or loosing. Even so, consequences can be assigned to the different ends of the game, by betting.

2.2 Strategy Games on Computers

This definition also includes the special case of a turn based strategy computer game. The fixed and unambiguous rules, which are necessary to implement a game on a computer, also determine the other five factors of this definition. They are normally split into three groups of rules, regarding the interaction between players, between the player and the game and describing the inner workings of the game. The first set of rules, which governs the interaction between the players is neglected, our focus lies on the second two groups.

The rules that govern the interaction between player and world are equivalent to the interface on a computer game. To formalize this, we assume that every turn the player picks an action from a finite number of options. If this is extended to assume that several choices per round are made, and that not doing anything can be an option, this formalism is able to describe most interactions with turn based

[1] http://www.rise-of-atlantis.de

computer games. But for the case of a strategic game, this formalism appears natural, since it is the skill of decision making that dominates this game, rather then agility or fast reactions.

The third set of rules that governs the inner workings of the game is associated with the game mechanics. It determines most of the games features, and improving its quality is our goal.

2.3 Measurable Parameters

Since the player is attached to the outcome, most games are played in order to win the game, meaning to achieve one of the better outcomes for the players. In many cases, this is difficult because the valorisation of the different outcomes if different for the players, and thereby a conflict is generated. Overcoming this conflict and still being able to win in spite of non favourable opponents creates a feeling of fulfilment and fun, and is therefore desirable when creating a game. The goal here is to determine those parameters that would spoil this process.

Naturally, a player tries to use a strategy that would increase his chances of winning. So first of all, such a strategy has to exist. That means, some of the actions the player take have to be better then other, at least for the given situation. If this is not the case, or if the player never has the necessary information to do so, the game becomes completely random and the player is not able to put any effort into the game, since his actions don't really change anything.

So some actions have to be better then others. But if one action, or a certain combination of actions is always better then others, the player will eventually learn of this, and always pick this action. We will call this a dominant strategy. This is also adverse to the idea of a player effort, because the player will realize, that he is not really making smart decision during the game in order to win it, but is just executing a strategy that leaves him no choice. His actions are again without effect, the outcome is predetermined.

A similar problem poses an action, which is never beneficial. This "inferior choice" is not as problematic as a dominant strategy, but it also limits the choices a player can make. Therefore all actions should at least be useful in certain circumstances. Every game feature should be part of a successful strategy.

Therefore game designers should avoid dominant strategies and inferior choices. If the inner workings of the game are kept secret, it is harder for the player to discover such strategies. But if they are eventually found, the game is still spoiled. Complex games make it hard, even if the player knows the rules, to determine what a dominant strategy is, but it also makes it hard for the game designer to check if those strategies exist.

Finding such strategies is a multidimensional search problem, which genetic algorithms are able to approximate. So a genetically evolved AI, whose fitness function is defined by how well it plays the game, would consider a dominant strategy a local maximum. So if several AIs adapt a certain strategy and are then always able to beat the other AIs, their strategy can be considered dominant. Even if a certain game feature or action is chosen much more frequent then others a strong hint exist that this action might be to powerful. Respectively, if AIs never use a certain game feature, this feature is an inferior choice.

Another parameter that can be measured is the time a game would last if played by a human player. Since the game should not infringe upon the real life of the player, this time should be limited to a manageable length.

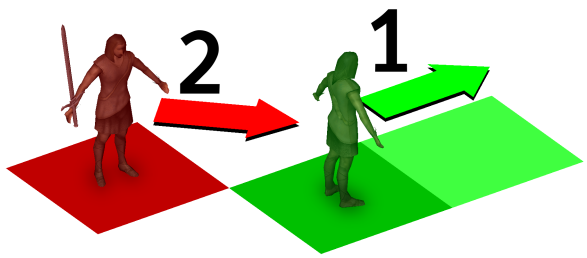


Figure 1: *Both units were ordered to move. The green unit is faster and thereby escapes the attack.*

3 Game Mechanics

To test our theories we designed a case study that was conducted during the development of the game "Rise of Atlantis". This section will present a short overview of the game mechanics and its basic features. The main goals here are to show that the game is too complex to find an optimal strategy with a simple mathematical analysis, and to explain some of the details that we will refer to in the results section.

"Rise of Atlantis" is a turn-based strategy game set in a pre-industrial world with some fantasy elements. The game world is represented as a grid of game tiles, each game tile being of a certain terrain, and possibly containing resources and local population. The player's goal is to hire people, explore the territory, amass resources and increase his power in order to either complete the goals set by the storyline, or to overcome his opponents.

3.1 Concurrent Turn Execution

The idea of a turn based game was in our case inspired by letter games that were common in the last century. Players would send in their turn to a game master, who would then evaluate them and send reports out to the players. In our game, every turn the player chooses the actions his troops should undertake, and those are then send to the server. The server evaluates the commands, places them in an order determined by the kind of action and the properties of the agent performing them, and then tries to execute them. Normally, there is no uncertainty, if an order can be executed, only when other players are involved the agents might fail. If, for example, a troop is send to attack an enemy unit but this unit was moved away in the same turn and is faster, the attack can not take place (cf. Fig. 1).

3.2 Interaction with the world

The interaction of the player with the world happens only through his agents, either single heroes, or troops of people he commands. All the actions the player can take are actions one of his agents can perform. There are two kinds of agents, heroes and regular troops. Regular troops are used for gathering resources, building structures, fighting enemies, processing goods and other tasks that are more efficient with several people (cf. Fig. 2). They can consist of a different amount of people and can be hired in villages.

Heroes are single individuals that can do all the things troops of people can do, but do so less efficiently, since they are alone. However, they have additional abilities that allow them to hire people, prospect for new resources, lead armies into battle, perform various kinds of magic and rally villages so they join your side. Since all

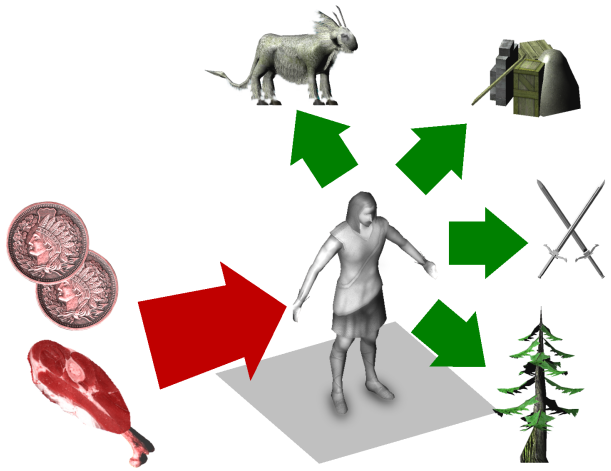


Figure 2: *The player interacts with the world through his units. Each turn they consume money and food, but they can perform various actions such as harvesting and fighting.*

the action in the world are performed as orders given to the units, all a player has to do is to give an order of what to do in the next round to every unit. This is very convenient for our AI testing, since it allows us to offer a limited amount of choices (the set of possible orders) for a finite set of agents to the AI to decide.

3.3 Unit Skill System

There are only two basic types of units, heroes and troops, but with time they can specialize by developing their skills. Both types gain skills in two ways by executing an activity that is related to that skill, or by practicing it in a school. Heroes learn faster and also have a higher skill cap. There are two main advantages of higher skills. First, all actions performed with that skill are increased in efficiency. While a group of people harvests wood, they gain skill points in the category wood, which increases their productivity. At certain point levels, they also gain additional bonuses and extras, such as the ability to plant trees and a bonus for using axes in battle.

Since heroes have a higher skill cap, some abilities and bonuses can only be reached by them. Also, some abilities require a high skill in more than one category. To avoid having too many skills, several actions are grouped under the same skill. A unit skilled in gathering wood is also a good carpenter. The main result of this is that as the game progresses, and the units gain more and more experience, the options for the player and the commands he can give grow more complex. There is also a trade-off to be considered: does the player want to have only few skilled units, or rather a large mass of cheap troops.

3.4 Resources

There is a wide range of resources available in the game world, which can be harvested. Some like wheat or wood grow back, while other, such as ore and gems will eventually be depleted. Some, such as ore, also have to be discovered first and then the player has to build a special building, a mine, to harvest them. The resources are used for either building new structures that enhance the players abilities, give his agents new options or are used to produce tools and weapons, that have similar effects. In general, all goods can also be traded to the local villagers to gain some money, which is one of the two basic resources. Money and food is used every turn by the agents, and if they do not receive enough they get unhappy or unhealthy respectively. Every person uses up one unit of food and troops uses up one piece of gold per person, while heroes demand 100 pieces of gold. So, to have a lot of people under you command, you would need a lot of food, while someone who wants to have a lot of heroes would need a lot of gold income.

3.5 Settlements

Settlements, even though they can not be built, play an important part in the game. Players can either persuade them with diplomatically skilled heroes to join sides, or take them by force. The second option will then create a fight and part of the population will be diminished. Either way, if the player gains control, the villagers will then allow him to recruit troops and heroes from its people, construct buildings in the settlement, and pay taxes every turn. The settlements thereby create a steady stream of income needed to support the troops.

The player can also sell and buy goods in settlement, where prices depend on local supply and demand. The settlements have an inventory of goods, just like the agents, and every turn they use up goods and produce others, depending on the local environment they are situated in and what their needs are. So a settlement in the middle of the forest might not be the best place to sell wood. Even if allied with the player, the settlements still have a high grade of autonomy. The agents can take actions to influence the settlements, but there is no complete control. Especially if the player takes actions the villagers dislike, such as sending their people to battle where they die, or not paying their salary. The settlement will become unhappy and eventually will stage a revolt. Then the player is faced with the choice of suppressing his settlements or keeping them happy.

3.6 Asynchronous Diplomacy

When playing with other human players a direct diplomatic exchange is not possible due to the turn based nature of the game. So, as another feature, all diplomatic exchange is conducted in a matter of offers and responses. The game offers several levels of diplomatic relations one can declare toward another player. The best one is being allied, where another player is able to use other player's facilities like his own. The levels then decent from neutral, to trade embargo, to closed borders and war. In the state of war, all your units attack the units of your enemy. Since you can change your status with the execution of your turn, it is possible to change you status to war, with a player that has another status toward you. So, your units will attack his, and he will be ambushed.

The same goes for peace, once two players are at war. Both sides have to change back to peace, otherwise the fighting will continue. In the late game, another powerful group will appear, that is supposedly a common enemy of the players. Some of the players will then get the offer to betray their allies, and win together with this faction. The element of asynchronous diplomacy adds to the suspense, because allies can betray one another at every turn.

4 Program Structure

Genetic evolution demands the AIs to play thousands of games as fast as possible. In order to do so, they have to interface with the game, or more specifically the game mechanics. This section describes how the game program is structured, and how that allows the AIs to interface with the program.

In a turn based strategy game the interaction with the game can be formalized as choosing an action from a given set of actions at every

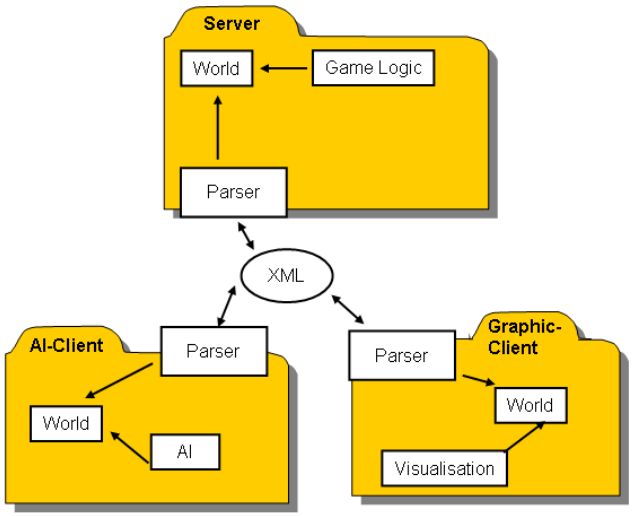


Figure 3: *UML package diagram of the game's internal structure, showing how both clients for AI and human players interact with the server.*

turn. In our game those decisions are the specific orders given to all units under the player's control. Note, that the order to do nothing is also an order, and will be assumed as the default order if no other is given.

The diagram in Figure 3 shows the basic modularization. The game was realized as a client server application. The module "WorldData" on the server side contains all game data about the world and the players. The server module called "GameLogic" represents the rules, and is responsible for the evaluation of each turn. When all players hand in their turns, it executes their effects on the model in "WorldData". Two kinds of clients were created, graphical ones for the human players, and non-graphical for the AIs. In both cases, they receive their data from the "Parser" module that encodes the relevant game data in an XML file that would then in turn be used to update the world data on the clients. The module "Parser" only encodes the data accessible to the player through his units, so the AI does not receive more information than a normal player. The AIs and the players then give orders to their units in their "WorldData" Module after examining the game situation. Those are transmitted back to the server, again in the form of an XML File.

This modularization has several advantages. First, the AI as well as players can be constricted from viewing the complete world state of the game, thus preventing cheating. The communication of the orders in the form of a specified XML file makes it possible to play by TCP, email or any other kind of communication that is able to transport that file. It also gives a well specified format for the AI to express their decisions. It is also possible to completely omit the graphical representation, if only AIs are playing; accelerating the game in order to speed up the evolution process.

A genetic evolution of that complexity takes a lot of time. A high number of complete games have to be played to evolve even a single generation. Several generations are necessary to achieve even a basic level of evolution. All this takes considerable time, and it is not only the decision making of the AIs that slows down the process. Also the speed in which that decision can be transmitted and processed must be taken in account. Having a fast implementation of all factors included should not be neglected either.

5 Graphics

Although a graphical client does not necessarily belong to the basic requirements of the game, such a client was implemented at an early stage of the development process. The graphical client ensures that all programmers can easily inspect the current game state and compete with the AI. We also used the graphical game client to conduct a before-and-after comparison of the game with human players. To do so, some of the game mechanics that were designed to make the game accessible to AIs now had to be visualized to human players.

In another project that was part of the development of "Rise of Atlantis" a concept was created that allows efficient data representation, which also addresses the two most troublesome aspects of its game mechanic. The need to visualize the concurrent turn execution of orders, and the presentation of the different orders a unit can execute.

To address the first issue, we switched between two distinct screen layouts. The first layout presented the actual game state and allowed interaction with the game entities, such as giving orders. The second layout contains a wide screen display of the game world and shows the game progress in between rounds by following an automatically compiled timeline. The player can jump back and forth in this timeline layout to inspect all visible in-game events and he may even return to this timeline view when he entered the interaction layout.

In order to present all possible orders in a single, coherent view, the amount of information displayed was linked to the position of the camera. When the player zooms out and overlooks several game tiles in a birds-eye view, only very basic orders, such as movement, are possible. When viewed from a closer distance, the camera automatically changes its viewing angle, so that a single game tile is presented in an almost isometric view. More details of the scene are revealed and thus, the orders where the unit would interact with them are accessible by clicking the corresponding objects. By doing so it was ensured, that the player would have access to the same amount of possible orders the AIs had, without getting confused.

6 Genetic Optimization

Genetic algorithms are optimization of search algorithms that have been inspired by biological evolution [Goldberg 1989]. The idea is to have a genome space that is equivalent to the search space. Individual solutions are assessed by an evaluation function, also called fitness function. Fit individuals are allowed to procreate and are changed either through mutation or recombination. Much of the terminology used with genetic algorithms is taken from biology as its historical roots are in the simulation of evolutionary processes. Today they are used in many areas including artificial creativity [Pereira 2006], bioinformatics [Hill et al. 2005] and all general optimization problems without a clear direct approach.

6.1 Mathematical Definition

A genetic algorithm belongs to the category of random-walk algorithms. Given a multi-dimensional search space and a fitness function $f(\vec{x})$ the goal of the random-walk algorithm is to find the global maximum of the function f. A classic random-walk algorithm starts at a randomly determined point in the search space and from there on tries to find higher ground (e.g. the hill-climbing algorithm). Genetical algorithms vary this theme by starting at multiple places at once and sharing information on the heights globally, randomly trying to find even higher grounds. The point in the search space is represented through a set of chromosomes carried by each individual.

The genetic algorithm can be split into four phases: initialization, selection, reproduction and termination. In the initialization phase, random individuals are generated as the first generation. Next, for each of these individuals, the fitness function is calculated and the less fit individuals are removed from the process.

There are two ways to create new genomes, mutation or recombination (crossover). Mutation is modelled by randomly picking chromosomes and changing them. Recombination takes at least two fit individuals. Their genomes are split randomly and the pieces are recombined to form a new individual. Selection and reproduction are then repeated until the termination criterion is met, which might be anything from a target number of generations or fitness over lack in fitness increase to computational time constraints.

6.2 Using Genetic Algorithms to Train the AIs

The main problem in applying this method to our game it that the fitness function cannot be calculated directly. There is no explicit function that gives a fitness value for each individual. For the selection process, however, that is not exactly necessary. The fitness function is only used to compare the fitness of two individuals, but that can be done just as well by letting them compete against each other. This method of indirectly determining the fitness of the individuals is called "coevolution" and is one of the main advantages of genetic algorithms over classical search algorithms [Koza 1991].

Our implementation of the genetic algorithm with coevolution works as follows. Let G be the maximum generation size and S be the minimum generation size. During initialization, G individuals are build randomly. The individuals are randomly pitted against each other in a match. The losing individual is deleted. When the generation size reaches its minimum S, the reproduction begins. From the survivors again two are chosen randomly. They are crossed, using a random one-point-crossover and mutated at a rate of $1/numberofgenes$, which on average results in one mutation per generation. We stopped the process after about 24 hours of computation time, with $S = 32$ and $G = 128$.

When mapping behavior to a vector space, the naive approach is to give the reaction to all possible input possibilities. For "Rise of Atlantis", this approach would be fatal as even the map has more than trillions of variations. Instead of dictating the reactions to the input, we chose to use AI behavior models, for which the genetic algorithm provides the parameters. The three behavior models, we used, are introduced in section 7. All three follow a different paradigm.

7 AI paradigms

When building an AI, basic design questions have to be answered, before even considering implementation, optimization and integration. While studying typical decision-making processes in human players, we noticed that there are different paradigms concerning strategical and tactical play.

In the next three sections, we will discuss the AI paradigms we used for optimization of the game and their advantages and disadvantages.

7.1 Swarm AI

The most basic paradigm is that of swarm behavior. Every entity is seen independently and tries to perform best from its limited point of view. Their decisions are based on primitive motivations like hunger, loneliness, curiosity, fear and anger. It is randomly decided what motivation to pursuit. As you can see in figure 4, the motivations give motivation points toward the different actions, which add

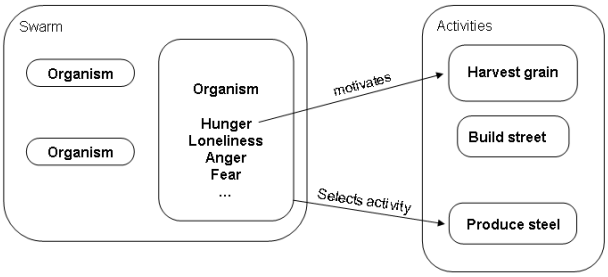


Figure 4: *Organisational diagram for the Swarm AI*

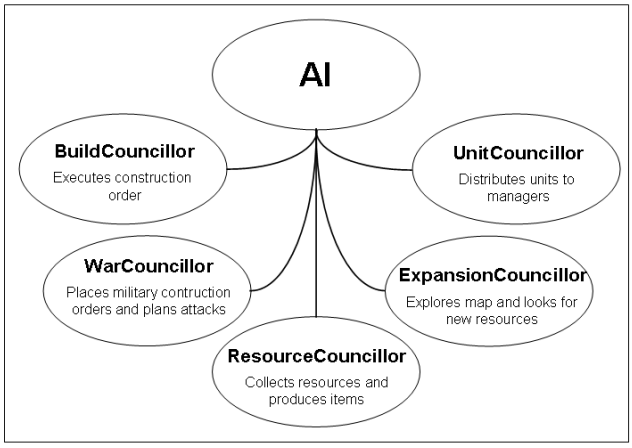


Figure 5: *Organisational chart for the Councillor AI*

up to the probability distribution of the actions. The precise parameters are determined by the genetic code. While most motivations are determined individually, like hunger and loneliness, others are set as global parameters like anger and fear.

Although each entity decides on their own, their behavior does not have to be egoistic. In fact, entities may give food away when meeting another entity with fewer resources. Each individual's prime goal is to win the game. They do not do, however, explicitly plan or coordinate on a global level.

We did, however, notice emergent behavior that did benefit the whole group. The individual's aversion to loneliness, leads to moving clusters like unit groups, which proved to be very effective in both war scenarios and resource management.

Other advantages where more technical in nature. The implementation of a motivation system is very simple. New actions and even new motivations can be added quickly, when the game logic is extended or changed. In our test games, the swarm AI usually beat its opponents, not only because its implementation is simpler, hence more robust, but also because it learned much faster. Since, we restricted learning time and the swarm AI could play faster due to its simplicity and could thus run through more generations.

7.2 Councillor AI

The Councillor AI acts similarly to a Cabinet. The Councillors are specialized in topics like war, infrastructure, expansion and resource gathering. Each of the Councillors has a wish list of secondary goals, like "build workshop" or "more soldiers". The exact

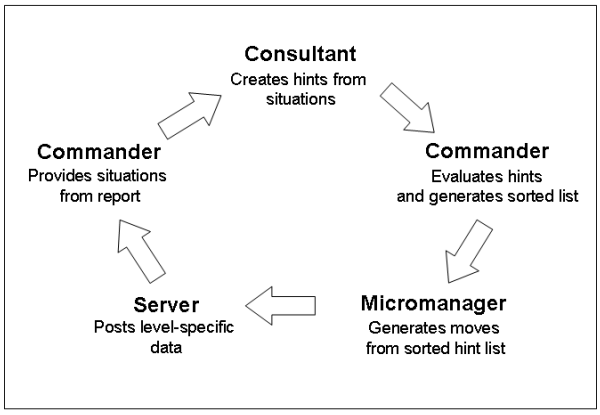


Figure 6: *Workflow of the Reactive AI*

content and priorization of these wishes is determined by the genetic makeup and some situational markers. Also, the Councillors are prioritized based on situation and genes, which influences the global priority of their wishes.

According to the priority of each wish, needed resources are distributed, missing resources are set on the resources gathering list or transported. Unlike the swarm AI, resources can be allocated even when the goal is not reachable in the current turn. That way, resources can be saved for more expensive buildings.

Besides resources, wishes need units to be fulfilled. They are assigned jobs to units like gathering and exploring. When redistributing the jobs according to the priorization, the algorithm tries to maximize the units staying in a job to take full benefit of the learning system of "Rise of Atlantis".

Unlike the swarm AI, this system needs a lot of handicraft, by formulating the wishes and splitting them into jobs intelligently. The main advantage is that decisions are easy to understand for humans and predictable. It makes a nice opponent and could easily be used as a recommender system for new players. The main disadvantage is that many crucial decisions are made hard-coded, without influence of genes, which seriously hampers the flexibility needed in the play testing phase, when rules are still changing. Tactically, we noticed occasional micro-management mistakes that were not covered through the jobs and wishes. The gameplay worked very well for building quickly an infrastructure, but did not cope well with sudden changes, rare situations and attacks.

7.3 Reactive AI

Much like the Councillor AI, the Reactive AI decides globally on how to proceed. Unlike the Councillors, it focuses much more on reacting to situations, than following its own plans. As you can see in figure 6 the decision process is split into four steps. The Commander analyzes the game report and looks for certain patterns, called situations. Typical situations are enemies on the border, resource scarcity and lacking exploration. From the detected situations, the Consultant generates general solutions or hints by consulting the genes, together with an alarm level of the situation. Next, the Commander sorts the hints by the weights given to them in the gene and their alarm level. The Micromanager converts the hints greedily into actual commands. Unused resources are given standard tasks that do not require much movement.

Again, the main problem is the amount of hard-coded intelligence required to make this AI run. This increases calculation time and decreases flexibility. The Reactive AI did a much better job of reacting to attacks and tactical micro-management than the Councillor AI.

8 Results and Discussion

In this section we will present the general results of the case study and discuss some of the findings in more detail. The AIs adapted well to the game mechanics and already twenty generations after the initialization the genomes would perform significantly better then the randomly initialized genomes. This suggests that the hard coded parts of the AIs were well chosen, giving the AIs enough flexibility to adjust, but also restricting them enough so they can evolve successfully.

It also shows, that the game mechanics, as a minimum, are not completely random itself, and that certain strategies improve the odds of winning. The next step is to look for dominant strategies, inferior choices and bugs.

8.1 Dominant Strategies

One example for a dominant strategy was the overproduction of grain. One of the AIs found a strategy in which it would hire as much troops as possible, and use all of them to harvest grain, an openly available and regrowing resource. The grain was then partly eaten, and the rest was sold for money at the closest settlement. The resulting money was enough to hire more troops, and the AI was thus able to increase its troops exponentially. The mechanism that adjusted the prices in settlements depending on supply and demand should have prevented this. Selling increasingly huge amounts of grain to the villagers, which would never use them up, should have decreased the prices dramatically and made this unprofitable. Unfortunately, the regular price for grain was only two gold units, and a price for anything could never fall below one gold piece. A change of this lower bound made the strategy disappear completely.

This illustrates, how a somewhat complex dominant strategy that involves several different actions (hiring troops, harvesting grain, transporting it to the settlement, selling it) could be successfully identified. An experienced player could have found that strategy as well, but might then have been forced to repeat those rather boring actions over and over, and control an exponentially large amount of troops. It was decided that this strategy is not beneficial for game-playing fun and it was thus removed from the game.

In a later evolution, many AIs decided to hire more troops in the beginning than they could sustain in the long term to gain a short term benefit in resources and protection. Since that strategy was not essential to win a game and actually seemed like a fun strategy for players, giving them more to do in the beginning, it was left as a viable possibility in the game.

8.2 Inferior Choices

The AIs where also able to identify several choices as inferior, and did not use them at all. For example, after additional cost and requirement for the production of iron where introduced, most AIs dropped the production of those good depending on iron completely from their strategies. Even so they where capable to produce iron goods, as was seen before the cost of the ore processing steps were increased. Something similar happened to the use of horses. In one version, their training depended on building a stable and the AI stopped to produce horses, because it was not profitable anymore.

Apart from that, the statistics could be used to investigate which features of the AI used. For most options, it could even be determined to what extend or frequency a certain feature was used. Most of the AIs, for example, settled for a moderate approach to exploration. No exploration appears as an obviously bad choice, since it limits the player's options heavily. But a maximum of exploration also proves to be bad, since it entails the risk of loosing several units due to the lack of support form the main troops, and early contact to the enemy.

A different factor of the game mechanics that could be determined was the actual time it would take a player to play a game. On average it took the AIs 300 turn to complete a game. This is too long, considering that a player would need 2 minutes for a turn the game would last 10 hours. It also revealed that especially the early buildings and tools took way to long to produce.

8.3 Bug Detection

Another interesting result we discovered during the case study was the ability of the AIs to identify bugs in the game mechanics module. Since the AIs were more likely to get "killed" if they lost a game, being able to crash the game was an advantage for the genetic selection process. Therefore, several AIs developed ways to crash the game. One was particular memorable, because it involved the combination of several complex actions to crash the game. These would have been hard to find by conventional beta testing, since it involved several phenomena human players would instinctively avoid.

Every troop unit has an integer number that represents the number of people in that troop. If, for any reason such as combat or bad health, a person in that unit dies, that number is decreased, if the number of people reaches zero in that way, the unit is erased. The problem was that if a unit was created as a troop of zero people, the reduction of one person put the counter to -1, and the unit would not be erased. So zombie units where created, which produced food and gold every turn, could not die and caused the game to crash in battle simulations. Naturally, units of that size should no be possible to produce, but the AI found a way. By not giving a hero food for a longer time, the unit gets very close to death, and its efficiency in different tasks suffers. When this hero hires a troop unit in a settlement, the amount of hired people is calculated in regard to the hero's general efficiency, which could yield a result of zero people.

9 Conclusions

In this paper, we studied the usefulness of using genetically optimized AIs as playtesters. More specifically, we compared the performance of three different AI paradigms in this situation. Each AI was designed and programmed by 6 to 7 undergraduate students over a time period of three months.

As discussed in section 8, the evolving AIs where able to develop winning strategies, revealing both dominant and inferior strategies. None of those strategies were obvious in superficial human play test, we conducted before training. It fact, many of those strategies used rather boring and repeated moves, or neglected certain aspects of the game. Human beta-testers usually try to avoid both. Degrading the value of those strategies made the game more interesting, by not forcing the players to use them in order to be effective.

Implementing the genetic algorithm was relatively fast and easy, given that the AI had been designed for it from the start. Since an AI had to be developed for the game anyway, the additional cost of making it optimisable was relatively low. Working with an unfinished and changing game logic was quite a challenge, though. While the SwarmAI could be changed relatively easy, both groups with more complex AIs had a hard time adjusting their AIs to the changes. To our surprise, the simplest AI became also the most effective, both in terms of finding strategies and bug detection, simply because they could run more and faster than the more complex AIs. Also, it turned out, that the most important ability of AIs is not to have the most efficient micro-management, but to react to threats and actions from the opponent and the surrounding area.

In conclusion, we believe this case study to be convincing proof of concept for our initial thesis. The human play tests we conducted before and after the modifications actually show the game is now more fun to play.

References

ANDRADE, G., RAMALHO, G., SANTANA, H., AND CORRUBLE, V. 2005. Automatic computer game balancing: a reinforcement learning approach. In *AAMAS*, ACM, F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, Eds., 1111–1112.

DEMASI, P., AND DE O. CRUZ, A. J. 2003. Online Coevolution for Action Games. *International Journal for Intelligent Games and Simulation 2*, 2, 80–88.

GOLDBERG, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, Boston, MA.

HILL, T., LUNDGREN, A., FREDRIKSSON, R., AND SCHITH, H. 2005. Genetic algorithm for large-scale maximum parsimony phylogenetic analysis of proteins. *Biochimica et Biophysica Acta*, 19–29.

JUUL, J. 2003. The game, the player, the world: looking for a heart of gameness. In *DIGRA Conf.*

KOSTER, R. 2005. *A theory of fun for game design*. Paraglyph press.

KOZA, J. R. 1991. Genetic evolution and co-evolution of computer programs. In *Artificial Life II*, C. T. C. Langton, J. D. Farmer, and S. Rasmussen, Eds., vol. X. Addison-Wesley, Santa Fe Institute, New Mexico, USA, 1990, 603–629.

PEREIRA, F. C. 2006. *Creativity and Artificial Intelligence: A Conceptual Blending Approach, Applications of Cognitive Linguistics*. Amsterdam: Mouton de Gruyter.

PONSEN, M. J. V., MUNOZ-AVILA, H., SPRONCK, P., AND AHA, D. W. 2005. Automatically Acquiring Domain Knowledge For Adaptive Game AI Using Evolutionary Learning. In *AAAI*, AAAI Press / The MIT Press, M. M. Veloso and S. Kambhampati, Eds., 1535–1540.

YANNAKAKIS, G. N., AND HALLAM, J. 2006. Towards Capturing and Enhancing Entertainment in Computer Games. In *Proceedings of the Hellenic Conference on Artificial Intelligence*, 432–442.