Swarm Intelligence in Strategy Games

Auguste Antoine Cunha
auguste.cunha@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2015

Abstract

This work is an attempt to answer the problems resulting from *predictable* Artificial Players — a common issue with scripted implementations — and to improve its *adaptability* — taking advantage of the *emergent behavior* resulting of the Swarm Intelligence concepts. In this work, we designed and implemented an algorithm that combined *Swarm Intelligence concepts* with the traditional decision mechanisms of modern Artificial Intelligent Players — specifically those used in *Strategy Games*. Our main objective was to assert the adequacy of Swarm Intelligence current knowledge to the Artificial Intelligent Player requirements, followed by the development of a test algorithm in itself. The basic concept was to distance our implementation from a common *centralized solution*, into a *decentralized solution*, and complementing it by applying some of the currently documented Swarm Intelligence notions. The resulting algorithm was especially responsible for the means of *communication between the units* of an Artificial Intelligent Player. A centralized and scripted Artificial Intelligence was used as benchmark for our Swarm Intelligence based solution.

Keywords: *Artificial Intelligent Player*, *Strategy games*, *Swarm Intelligence*, *Algorithm Communication*, *Emergent behavior*

1. Introduction

Swarm Intelligence (SI) is the sub-field of AI that studies how a group of AI agents may work in a self-organized and coordinated way, without the use of a centralized control. The algorithms developed in this area, are a derivative of Nature studies, often from insect colonies (such as ants, bees, wasps, or termites). Perfected by nature, these algorithms have become an interesting part of AI study, giving new insights to the way people approach each problem, and even changing the way we perceive *'intelligence'*.

The use of SI-based algorithms have already began to help with *real world* problems[5]. However, the use of SI in computer games is limited, and usually only applied to *under the hood* features — uses not directly visible to the player. For this reason, a question is left floating — *How can we take advantage of these field developments in gaming AI?*.

1.1. Problem Description

Even though we understand that AIs may come in many forms, we will focus on SG and AIs that are responsible for the decisions of an opposing player (as if they were another human player in the same game). As players become more and more aware of the lack of good AI present in some games — either by seeing constant bad AI decisions or AIs that simply let them win — they are becoming more demanding in that respect[4]. Our objective will always be to develop AIs capable of entertaining the player — which can have multiple interpretations or definitions as different people are entertained by different things (e.g. *challenge seeking* vs *always have to win*). In this work we will try to do just that, as we will develop a new AI to control the units of an army in the *Multi-player Online Turn-Based Strategy Game* Almansur[1]. For this reason, and as stated before, our study focus will be on AIs used in SG, namely *Turn-Base Strategy Games* (TBS).

Going one step further, our goal will be to test the viability of using the current knowledge of *Swarm Intelligence* (SI), adapting it to improve the decision process of an AI in Almansur. With this approach, we will attempt to solve some of the issues that traditional AI faces in a TBS environment. More concretely, we will develop and apply an algorithm based on our SI research to the military decision process of the current AI of Almansur[2] — a scripted and purely reactive AI. We expect the use of a SI based AI will improve the quality of the AI itself, improving it's adaptability to unpredictable situations and constant changes in the environment.

With this work, we aim to answer the question

[1]Almansur — http://www.almansur.net/ — Almansur LDA, Last Accessed on 27 November 2013

left in the previous section, bringing some of the values of SI to a concrete case of video-game AIs, putting it in charge of the military decisions in a TBS environment.

As a final note, we'd like to reinforce that our goal is to improve the quality of an existing AI by applying a developed SI algorithm to the decision process of said AI. In order to validate our goal, we will run multiple scenarios between the two AIs, as well as with real players to validate our application. We expect to see a more responsible, a more adaptive, a more competitive and — generally speaking — a more *intelligent* AI form, followed by an improvement in Game Experience for the player.

2. Related Work

2.1. AI-Player in Strategy Games

In the Strategy genre, an AI-Player is responsible for making the same decisions a human player has to make — commanding the various units at its disposal, to achieve its own end goals. Knowledge on how to design a Player-AI, went through many stages and many techniques were tested. In older games, an artificial player that cheated was a very common practice. This could mean, for example, instantly generating resources or units required to counter another player's attack. While this technique is quite appealing efficiency-wise and presents decent results — this is, the challenge presented to the player is *adequate* — it's flawed execution may be the cause of a feeling of injustice in the player, for ruining the illusion of being challenged by an equally skilled opponent. This means the technique could lead to a negative experience and, therefore, should be avoided.

Developers took a step forward when they started to consider the strategic knowledge required from human players and began to introduce such notions in the AI-Player. However, the development of an AI with strategic knowledge, capable of coherent planning against a human player is a complex process. This complexity is partially due to the fact that a good AI will not resort to the same strategy every game[8]. A human player would notice the repetition after a few games, and would work to counter it, instead of learning from his own mistakes. In short, this would lead to an exploitation of that AI strategy by the player, and it would, ultimately, leave the player bored or uninterested in the game — exactly the opposite of the intended purpose. Such solutions would also fail to adapt to the different player decisions and would eventually fail to perform at the desired level.

2.2. Implementing an AI-Player

> *The development of an AI for Strategy Games isn't an easy task. One of the reasons why, is their dependence on the underlying game world implementations — which present hard variations from game to game within the genres. This also means that the development of the AI is very dependent on having the game world up and running before hand.*
>
> (Forbus et al.[9])

There are generally two ways of implementing an AI-Player for a Strategy Game — using a *centralized* or a *decentralized* approach.

Some lower level units may hold some intelligence (like the know-how of path-finding). In some way, this means we are dealing with a case of multi-agent system. Moreover, we are dealing with a centralized multi-agent system, as we have a *controlling* unit and multiple *controlled* units. In a centralized approach[11] — the most common in Strategy games — a central unit, usually god-like (unseen and all seeing), holds the most knowledge and conveys its intentions to the other units. The units receive said intentions (e.g. *attack* this, *defend* that, or *moves* there), and execute the appropriate action to fulfill that intention. In this case, the lower units need to be less 'intelligent' than the controlling unit. One of the main characteristics of the centralized approach is the discrepancy between the levels of intelligence required from each AI, and this is the most common approach for AI-Players in Strategy games — since it requires less effort from the developers, as they may use the same unit intelligence for all AI-Players and all human players, and it is less complicated. This approach can be seen as a direct interpretation of player interaction — as we can see the controlling unit the 'player'.

By opposition, the decentralized approach[11] entitles each unit with a more complex thought process, allowing them to perform local planning and communicate exchanging requests and intentions at unit level. This more complex thought process is always built on top of the previous *know-how* the units already had — such as the previously mentioned path-finding skills and the like.

A correct implementation of a decentralized approach allows quicker reactions to unpredictable localized events. The downside, though, is the ill-suited nature of the decentralized approach for actions that require high coordination — like strategy execution ("I go this way, you go that way"). This means, units are able to request aid, or even request the execution of an action from another unit, but due to the nature of this kind of control (or this lack of full control), a unit is free to decide to help or not another unit. In order to solve this issue, it is necessary to pair each request with a certain priority value, to help each receiving unit plan its own action. Also, each unit may have a level of altruism/selfishness, making it more/less prone to help its companions.

There can be different levels of centralization/decentralization. There may be multiple agents acting towards the same ultimate goal (this is, win) but still have different sub-goals to achieve — e.g. dividing the responsibilities of different types of planning to different entities, such as *economic*, *military*, or *diplomatic*. Each objective-driven AI can be centralized or decentralized within the same system.

The advantages/disadvantages of each approach need to be weighted in before actually developing an AI-Player, and there is no right answer as it should be completely intention dependent — referring back to the *challenge* vs *defeat* the player intention. Despite the differences between these approaches, they are still both related to the implementation of a Player-AI. It makes sense that, in a way, in a game with multiple players (a mix or human and AI players), we would want them all to play with an approximate level of *human-like* intelligence, since this would create some balance and improve the player experience.

2.3. Swarm Intelligence

It is a well known fact that Man learned a lot from studying natural systems. This is also true for Computer Science[7], as the studies derived from natural system inspired the development of such algorithm models like artificial neural networks, evolutionary computation, swarm intelligence, artificial immune systems, and fuzzy state machines. These mentioned breakthroughs in computer science are the respective models of biological neural networks, evolution, swarm behavior of social organisms, natural immune systems, and human thinking processes.

From the studies on social organisms, it became evident that their ability to perform complex tasks had the interactions between the individuals of the swarm in its core. This means, that the complexity was not innate within any of the individuals, but rather present when analyzing their behavior as a whole. The interaction in these biological swarm systems may be direct — through the natural senses of touch, smell, hear or seeing — or indirect — through changes in the environment. The ability to perform complex tasks as a result of individual independent labor is called *emergence* and it is not easy to predict or deduct the complex resulting behavior from observing the simple behavior of the individuals. Engelbrecht et al.[7] define emergence as *the process deriving some new and coherent structures, patterns and properties (or behaviors) in a complex system* — structures, patterns and properties (or behaviors) that come to be without the presence of a central commanding unit delegating or tasks to the individuals.

2.4. Algorithms in Swarm Intelligence

Many are the SI-based algorithms and many are their applications in various areas. These algorithms have been proven very efficient in solving AI problems that range from optimization to clustering algorithms. We will focus on algorithms that have had some application in gaming.

The knowledge gained from studying ants has proven a great aid in the development of algorithms such as the Ant Colony Optimization[6], and its usefulness falls under various categories — e.g. routing problems (path-finding for distribution), assignment problems (distributing tasks to works, given some constraints), scheduling problems (allocation of resources over time), or subset problems (selecting items from a set that, together, form a solution).

However, SI systems are not absolutely reliable[3], as it isn't trivial to predict their behavior when faced with an unexpected event. There is also the issue of defining an adequate benchmark, suitable for SI testing. SI systems' performance shines when acting in dynamic environments, and so dynamically changing problems. In order to create a benchmark for such an adaptive system, implies that we would know what to expect from a generic adaptive system. How could we evaluate the performance of a system (what would be the metrics)? All in all, there are multiple ways of being dynamic, but it could be possible to show various systems with similar properties in terms of how difficult it would be to solve their corresponding dynamic problem.

2.5. Artificial Immune System — A Defense Mechanism

The biological immune system is a good metaphor for anomaly detection systems in general. In 2002, Matzinger offered his views on what he called *Danger Theory*(DT)[10], something that has become increasingly popular. The DT states that the biological immune response is triggered by sense of *danger* and not by sensing *foreign* (or ”non-self”) entities. There are still arguments concerning the validity of the theory from a biological stand-point, however, this knowledge suffices to develop Artificial Systems. Creating a bridge to the Strategy game scope, this would be equivalent to an invasion by some or many other player's units. This theory, DT, has been used to develop AI algorithms for *Spam Detection*[12] as well as Intrusion Detection within a network[1]. The scope of these algorithms seems larger than the one present in strategy games, however we feel as though it is *natural* for an AI-Player to react under *fear* and sense *danger*, as there is some correlation to the human reaction.

3. Solution

3.1. Algorithm Design

Our algorithm's primary goal is to be improve the communication between the units of a swarm, in order to reach a consensus — this is, a set of *intentions* or *intents* per unit that every unit agrees on. So a unit can be seen as a particle of the swarm, and in order to make a intent final, that intent needs to somehow approved by all other units.

In order to accomplish this, our algorithm was divided in two stages:

- **Selfish phase** — in which every unit, analyzing the surrounding environment, decides on the best course of action for itself;
- **Negotiation phase** — in which every unit, communicates its intention to every other unit for consideration and evaluation. This allows each unit to reconsider their intentions, and instead follow another unit's.

Figure 1 shows these two phases as idealized.

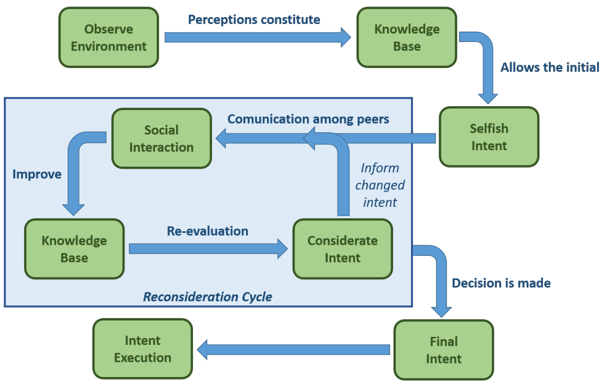

Figure 1: Conceptual design of the algorithm per swarm unit

Intent — Definition

An intent or intention is a structure that represents each unit's *thought* of what their *ideal action* would be. An action is an interaction with the environment, that is will be the cause for some desired outcome.

In terms of requirements, in regards to our algorithm, the intent must have an *heuristic value* and the *originating unit's identification*. These are the sufficient conditions for our algorithm to produce some results — even though not optimal in most cases.

However, both these parameters are completely dependent on the context implementation, especially for the heuristic value.

Another relevant variable that an intent can have is a *type* — however this is not a hard requirement. An intent type allows considering multiple different intents over a same target — e.g. *looking out of*, *opening*, or *closing* a window. Having a type is completely context dependent, and, as previously stated, it is not are not mandatory.

Considering studies such as the *Danger Theory*, referenced in **Related Work**, it became clear that there could be a requirement for raising the level of importance of a certain intention — in practical terms, when one unit detected a threat, or an very valuable target. For this reason, we considered and included the *help flag*. This feature, allows any unit to artificially increase the value of its target, in order to call the attention of others to it. But once again, the algorithm does not make existence of this flag mandatory.

On a side note, all these optional parameters were included in our final implementation of the solution.

3.2. Selfish Phase

Coming back to the algorithm itself, the first part is really straight-forward — each unit considers each available target in the environment, and marks the intent with the highest heuristic value found as its selfish intention.

Same as the intent types and heuristic values, the perceptions of the world (reference as available targets above) are entirely context dependent.

In **Algorithm 1** we represent a simplified version of the first step of our algorithm — the *selfish planning*[2]. This *pseudo-code* references intent type, as we believe there are more cases that require it than not.

[2]This and the examples that follow are a simple representation of the actual process — split up for explanation purposes. The complete algorithm is present in appendix, chapter **Complete Algorithm**, and is nothing more than these examples combined.

Algorithm 1 Simplified take on the selfish cycle step

```
1: input: perceptions
2: output: $selfishIntents \leftarrow map[unit, intent]$
3:
4: $selfishIntents \leftarrow newmap[unit, intent]$
5: $selfishIntent \leftarrow null$
6:
7: for all unit in swarm do
8:     for all target in perceptions.availableTargets do
9:         for all type in context.intentTypes do
10:            $intent \leftarrow newintent(type, target, unit)$
11:            if $intent.heuristic > selfishIntent.heuristic$ then
12:                $selfishIntent \leftarrow intent$
13:            end if
14:        end for
15:    end for
16:
17:    $selfishIntents[unit] \leftarrow selfishIntent$
18:
19: end for
```

3.2.1 Selfish Phase - Analysis

Considering we have three nested loops, the complexity of the algorithm can be calculated based on the number of times each of those loops runs. In the worst case scenario:

- **Main Loop - Unit Cycle** — runs exactly $U(\leftarrow swarm.length)$ times — or N in the usual nomenclature;

- **First Inner Loop - Target Cycle** — runs a number of times equal to the target for each unit — which is a constant, so t times;

- **Second Inner Loop - Intent Cycle** — runs a number of times equal to the number of intent types — which is constant, so i;

This allows us to conclude that the this phase of the algorithm will have a number of cycles equal to a constant (c = i*t) — which depends largely on the context — times the number of units (N) — making it of complexity $O(N)$.

3.3. Negotiation Phase

Immediately after concluding the first phase, the second one begins. In this *negotiation* phase, each unit will divulge its own intent to the rest of the swarm — consequentially, each unit will also receive every other unit's intention. This will allow for a reconsideration step.

3.3.1 Reconsideration — Definition

By *reconsideration* it is implied that a unit is rejecting its own selfish intent, and instead decided to follow another unit's intention.

A reconsideration is only valid if, by the end of the cycle, every unit is pointing to either:

- Its own selfish intention;

- Another unit's intention, and that intention's originating unit still intends on following it.

It is not expected that a unit maintains its own intent, even after someone reconsiders toward their intent. Our algorithm is also responsible for recovering from an invalid state. This is done through a *rollback*, which we will explain later.

3.3.2 Negotiation Phase — Division

As was made clear by the reconsideration definition, the negotiation phase can be further divided in two phases — *reconsideration*, and *validation / rollback*.

These two phases are ran in succession, and together they allow the swarm to reach a *consensus*. A consensus is met when there are no units reconsidering, and the end state is valid for all units. While there are any units reconsidering, the cycle continues. In **Algorithm 2** we can see the algorithm for the negotiation phase.

Even though there is a *while(true)* in the algorithm, this is only to make sure the algorithm only completes when there is a consensus, and every cycle of reconsideration is specially designed to work towards that goal.

3.3.3 Negotiation Phase - Analysis

Noting that the number of units directly reflects on the number of intentions, in the worst case scenario, these algorithms will run:

- **Reconsideration** — *Number of Units in the swarm* times the *number of intentions* — $O(U*I)$ or $O(N^2)$. Every unit listens to every other unit in order to understand if it can or should do something about their intention;

- **Validation-Rollback** — *Number of Units in the swarm times* — $O(U)$ or $O(N)$. It is a necessary condition to have a valid state before proceeding — any unit found with an invalid intent, will fall back to its original intent.

3.4. Algorithm Analysis

Algorithm analysis is usually done in terms of time complexity or resource allocation amount. However, this algorithm was directly developed on top of the previous game's AI, which made it especially

Algorithm 2 Negotiation algorithm - part 1

```
1: input: $selfishIntents \leftarrow map[unit, intent]$
2: output: $consensusIntents \leftarrow map[unit, intent]$
3:
4: $initStepIntents \leftarrow selfishIntents$
5:                                         ▷ Reconsideration
6: while $true$ do
7:     $hasReconsidered \leftarrow false$
8:
9:     for all $unit$ in $swarm$ do
10:        $myIntent \leftarrow initStepIntents[unit]$
11:
12:        for all $intent$ in $initStepIntents$ do
13:            $isValid \leftarrow finalIntents[intent.unit]$ != $null$ and $finalIntents[intent.unit]$ != $intent$
14:
15:            if ($intent.unit == unit$) or (not $isValid$ then
16:                continue
17:            end if
18:
19:            $myHeuristic \leftarrow myIntent.heuristicFor(unit)$
20:            $newHeuristic \leftarrow intent.heuristicFor(unit)$
21:
22:            if $newHeuristic$ ¿ $myHeuristic$ then
23:                $myIntent \leftarrow intent$
24:                $hasReconsidered \leftarrow true$
25:            end if
26:        end for
27:
28:        $finalStepIntents[unit] \leftarrow myIntent$
29:    end for
30:
31:    if $hasReconsidered$ then
32:        break
33:    end if
34:                                         ▷ Validation / Rollback
35:    for all $unit$ in $swarm$ do
36:        $intent \leftarrow finalIntents[unit]$
37:        $parent \leftarrow intent.parent$
38:
39:        $isValid \leftarrow (parent == unit)$ or $(finalIntents[parent] == intent)$
40:
41:        if not $isValid$ then
42:            $previousIntent \leftarrow initStepIntents[unit]$
43:            $previousParent \leftarrow previousIntent.parent$
44:
```

Algorithm 3 Negotiation algorithm - part 2

```
45:            $isPreviousValid \leftarrow (previousParent == unit)$ or $(finalIntents[previousParent] == previousIntent)$
46:            if $isPreviousValie$ then
47:                $finalStepIntents[unit] \leftarrow initStepIntents[unit]$
48:            else
49:                $finalStepIntents[unit] \leftarrow selfishIntent[unit]$
50:            end if
51:        end if
52:    end for
53:
54: end while
55:
56: $consensusIntents \leftarrow finalStepIntents$
```

hard to profile — forcing us to step away from the conventional time complexity analysis for now. Resource allocation was also an analysis without much potential, as most of the data is related to the game itself and the algorithm doesn't really look into it al that much.

In the end, we performed a simpler worst-case scenario analysis based on the number of iterations of the algorithm itself and our conclusions are summed up in the table below.

Phase				Complexity
1	Selfish			O(N)
2	Negotiation	2.1	Reconsideration	O(N x N)
		2.2	Validation / Rollback	O(N)

Table 1: Algorithm complexity for each of the phases

In order to consider the complexity of the algorithm as a whole we can consider it as a sequence of the three steps, and its complexity is the sum of the three — $O(N)+O(N^2)+O(N)$ or $O(2N+N^2)$. Simplifying it, we can state that the complexity of the algorithm is, in fact, simply $O(N^2)$.

3.5. Algorithm Implementation and Heuristic Development

The integration of our algorithm with the context of our problem followed the initial concept of the algorithm itself. A couple of details were immediately evident:

- **Type definition** — every possible action in the game needed to be translated into an understandable type;
- **Heuristic function** — every possible action needed to have a quantifiable for comparison.

Everything a player could do, every action he could perform over his army, needed to be translated into AI logic. Part of this effort had already been done in the AI in place, but we felt it deserved an overhaul. For this reason, we dropped the existing *Actions* for our new concept *Intents*, complete with a new *Intent Manager* — since our environment, Almansur, is a TBS, we didn't see a reason to create a complicated structure for each unit, and settled for a structure that would hold all the intents while they waited for further processing. In the end, we ended up having only four types of actions — *defending*, *attacking*, *conquering*, and *no action*. These intended to provide enough diversity for our needs.

Regarding the heuristic, we ended up implementing two — one for *value*, and one for *danger*. These heuristic functions were common to all units, and represented the way they perceived the environment — they are represented in the algorithms as *perceptions*. In order to only perform calculations once for every unit, we chose to implement these via *influence maps*. Each position, a territory unit in the map, contained an associated danger level — taking into consideration the number and strength of the enemies on and surrounding it — and a value — taking into consideration the amount of resources available per type in it.

We can say that danger was more floaty than terrain value. Enemy units tend to move, instead of idling in certain territories, which in turn causes their threat to be somewhat *dynamic*. Value, however, is static. Resources that can be gathered in one territory, and they don't necessarily enrich the neighboring territories.

Finally, we needed a way to improve communication, and signal for emergencies — a call for help or assistance. For this purpose, each intent carries a *help* package, which contains the amount of fear the source unit has when performing that intent — unit is more fearful, then it needs more help. As an example, when a unit notices an enemy army that is far stronger than it.

3.6. Integration Additional Notes

3.6.1 One unit, one target — two units, two targets

At first, we allowed every unit to freely choose whatever target they wanted for a selfish intent. The scope of the play — amount of units vs amount of space — lead to the conclusion that most of the time, units would choose the same target. This is an issue for two reasons. First, the swarm always converged, too fast, too inefficiently. Second, communication was pointless, since most of the time, everyone wanted and stated the same thing.

For these reasons, we decided to try and maximize the area of effect of our swarm, by making sure that every unit had a unique target. This meant, no two units would leave the selfish phase of our algorithm with the same target for intent. Results proved our initial hypothesis, and our swarm spread more evenly around the map, greatly improving our conquering and our sensing of danger abilities.

3.6.2 Sufficiency vs Efficiency

In our initial implementation, we allowed any unit to target any territory — even if it had one of our units there. After implementing our *one unit, one target* mechanic, this became an issue. Units that were on a territory and didn't have high rating in the hierarchy, were forced to choose other targets, which, more often than not, meant going to the opposite end of our territory — correctly, because of the resource distribution around the map.

This was generating a lot of internal turbulence, within the swarm, much like ants seem to behave when they are in their colonies, but the results were definitely not good. From this turbulence we only got slower response times to threats — as the units spent a lot of time aimlessly walking around already conquered territory.

The solution was to give priority to any unit that is already on a territory, in a way that other units would only go to that target if the first unit requested assistance. This was the most *efficient* solution.

On the other hand, the unit at the position could not have enough strength to finalize its intention — *sufficiency*. In this case, it would request assistance from nearby targets.

3.6.3 Units can not help themselves

The help factor is quite important for the reconsideration process in allowing units to compensate for one another and work together towards one same goal. Regardless, we can not leave unmentioned that this factor only affects units other than the source of the intent. This means, if unit-1 has an intent with the help factor activated, the intent will have a greater heuristic value (multiplied by the help factor) than normal, but it will remain the same (unaffected by the help factor) for unit-1 during the reconsideration phase.

4. Testing methodology and data collection

Evaluating an algorithm is not an easy task. We could have done simulations and more complex analysis for time and resource allocation, but we believe testing the final integration is more interesting. For this reason, our tests can be divided in two components:

- **Duels with old AI** — multiple duels in different scenarios, resulted in fast games allowing for quick information;
- **Multi-player games** — between 15-20 players, both real and AIs, new players could enter in the middle of the game, and turns lasted one day.

The original AI was, since the beginning of the development, a benchmark for our AI. In the early stages of development, we used the duel system for debugging our new AI. In later stages, they were used to fine tune all parameters before taking the AI for real player tests.

We only took part in one multi-player scenario with real players, because of the game and turn lengths. However, we had three AIs in it, alongside two old AIs and ten real players — allowing us to retrieve some interesting data from the game.

As per metrics, we tried to use most of the information available from the game itself:

- **Victory Points (VPs)** — these represent the score in the game — player with most VPs wins;
- **Territory Owned** — being an important part of the game and of the military display, the evolution of territories owned during the game can be an interesting metric;
- **Army Power** — more than the ability to conquer, the ability to keep their own is an important task for the military agent — this metric should reflect this quality.

An final additional and important metric, was our reconsideration rate — this is, number of reconsiderations per number of actions taken.

5. Results

5.1. Static Scenario Test - Duels

During the development time, the new AI was matched against the old AI in various games. This was the easiest access benchmark for our implementation. Our main objectives with these matches were:

- Debugging the implementation of the algorithm;
- Identifying issues with the implementation, that could be directly linked to flaws in the algorithm itself;
- Asserting the correct evaluation of the perceptions of value and danger per territory;
- Improving the ability to conquer multiple territories per turn;
- Assessing the adaptability of our algorithm to a controlled environment, benchmarking with the old AI.

Almansur's duels are scenarios that place two players in equal footing, in a symmetric map. Each game had 24 turns and turns were processed after every player ended their respective turn — since both players were AIs in this scenario, turns were actually processed fast enough to play multiple games, making it easier to test and iterate on the implementation.

Due to the deterministic nature of the game and the AIs implementation, replaying the same match up, will generate the same actions from both players, producing the exact same result. For this reason, the analysis present in this section is based of the last iteration of tests made in this context.

5.1.1 Static Scenario Test Analysis

As seen in **Figure 2**, victory points steadily increase for both players, however there is a noticeable positive difference towards the new AI.

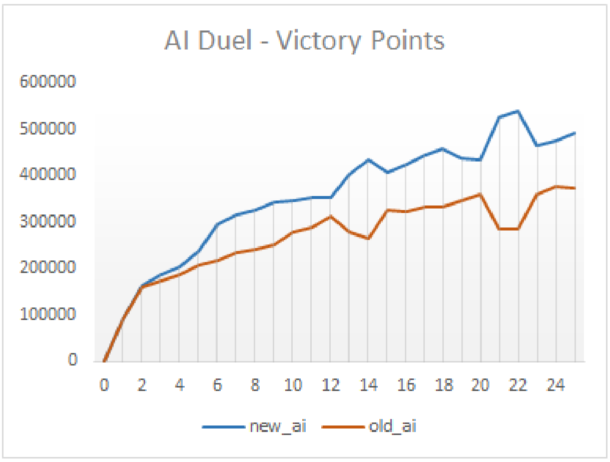

Figure 2: Graphic with the evolution of Victory Points for both players in the duel

Even though constant at first, there are a couple of moments when the victory points line abruptly changes — around turns 12-14, and 20-22. As can be seen by **Figure ??**, the evolution of victory points is directly connected to the evolution of territory victory points.

5.1.2 Static Scenario Test Conclusion

Being the easiest accessible data source, through the development process these duels provided the best setting for fine tuning our implementation. These allowed for the discovery of a few shortcomings, and consequent upgrades to our implementation itself — the following are examples of these upgrades:

- **Units of a swarm should be ordered by adequacy for their tasks** — otherwise, the *strongest* unit in the swarm could be forced to pick a less valuable target — which could lead to the demise of the whole swarm.

- **Each target should only be picked by one unit** — otherwise, multiple units could (and would) point at the same target if it had great value, without having to communicate — invalidating the reconsideration process.

- **A target that has a unit from our swarm, should not be our target** — otherwise, a more suitable unit could pick it as a target, forcing the unit already at the location unit to move — most time would be spent traveling around the map, instead of actually picking up objectives (conquering or battling the enemy).

- **Great risks should only be considered together with great reward (*value*)** — otherwise, units would be reckless and attempt to conquer (or battle) any enemy territory in sight.

These concepts were evident after the first few duels between the AIs. They led to implementation decisions such as the inclusion of influence maps for value and danger. After these upgrades, the new AI behavior was greatly improved being able to easily out-duel the old AI.

From the figures present in this section, it is possible to conclude that our AI implementation was more successful in these closed scenarios than the old AI.

5.2. Dynamic Test

Our solution had to be tested in a more complex environment in order to raise more interesting metrics for our solution and potential shortcomings. One of the advantages of these dynamic scenarios is the possibility of adding players to the game while it is already running. For this reason, a scenario was created with a total of 14 players, distributed as follows:

- **9 active Human players** — 8 joined at the start of the game, and 1 mid game;

- **3 players with New AI** — 2 joined at the start of the game, and 1 mid game;

- **2 players with Old AI** — 2 joined at the started of the game.

At the time the data was analyzed, this game had completed 16 turns during the course of 3 weeks, processing 1 turn per day except weekends. Our objectives for this scenario were:

- Identifying flaws in the implementation, that could be directly linked to flaws in the algorithm itself;

- Assessing the capability of our solution, when compared with that of the old AI implementation;

- Assessing the adaptability of our solution to a dynamic environment, with multiple opponents and threats.

5.2.1 Dynamic Test Analysis

In **Figure 3** we can see the evolution of average *victory points* among the 3 types of players. Right after the second turn, we can see a clear separation between the line of the old AI and the other two. It is also clear that our solution is able to be on par with the human player until the sixth turn.

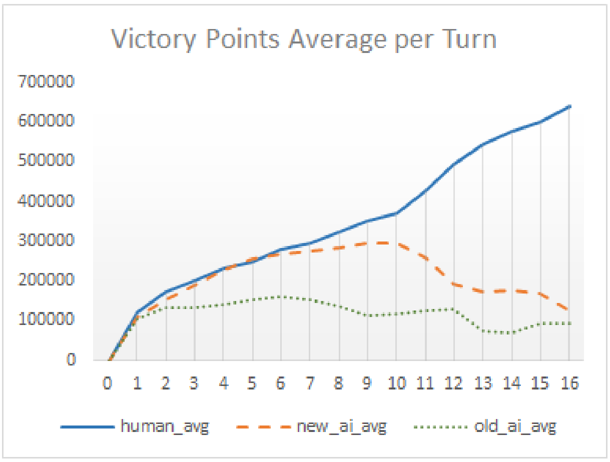

Figure 3: Graphic with the average evolution of Victory Points for the three types of players

5.3. Dynamic Test Conclusion

The greatest accomplishment for the new AI was being able to keep up with the human player during the first few turns of the game.

Considering territory victory points, the new AI displays a good perception of value and good prioritization of conquer targets. Despite the initial setback with one of the human players, the new AI was able to recover and stand equal to the human players in terms of this variable. The old AI was not able to reach such a level at any point in the game, falling short in the first couple of turns — displaying its inadequacy to prioritize and evaluate both target and self worths.

Our greatest validation should come from the comparison between the new and old AIs though. Noting that the AIs were similar in every aspect except the military agent responsible for issuing commands — conquering, attacking, and defending — the difference in victory points observed comes as a huge accomplishment.

Both types of AI, in this game, were playing without the aid of a complex diplomatic agent. This left them unable to properly interact with one and other, or with the player, at this level — making it impossible to form alliances, and forcing each AI to achieve their results on their own.

In respect to military stability, the new AI was able to come back from its loses, and to avoid further decline after an initial struggle. On the other hand, the old AI is unable to keep from decreasing strength, turn after turn. Although this allows us to conclude that it is possible that the new AI is better at picking its fights — having a better perception of danger, and a better judgment of its own ability — it is clear that there is also room for improvement on the AI that is responsible for recruiting additional military units.

6. Conclusions

In this work, we implemented an algorithm based on swarm intelligence concepts, that was responsible for an artificial player's decision process in a strategy game. Our main objective for this work, was to derive said algorithm from the current swarm intelligence knowledge, and apply it in a different than usual context.

Our implementation was built over an already existing AI, in the game Almansur. The original version was adapted and had its military decision process replaced by our new algorithm, ensuring most of the implementation was shared between both AIs. This decision allowed to benchmark the modified military decision process without too many influences from the other components — economic and strategic.

This work was based on some already proven theories:

- Heuristic functions based of influence maps are not new;
- Decentralized approaches for game AIs are not new.

This work was also an experiment for testing theories in a different than usual concept (as previously mentioned):

- Use of influence maps to apply the *Danger Theory*, originated from immune system studies, in the context of Game Artificial Intelligence;
- The definition of semantics understandable by a decentralized system, or *swarm*;
- Designing an algorithm capable of defining rules for communication, in order to reach a beneficial consensus.

In this sense, the final solution presented is a mixture between the conventional AI and SI concepts.

Given the results presented in the previous chapter, it is possible to state that our solution has some very promising results. The final implementation displays some of the benefits of a SI system, such as *adaptability*, *unpredictability* (which is a good thing in a game AI) and associated *emergence* — providing results that rival those of a human (in our case, in terms of conquering territories).

We do not feel, however, that these results are sufficient for concluding the success of this work. At most, we can reiterate our claim and state that the results are promising. In order to fully commit to the success or failure of this solution, our implementation needs to go through more tests against users of different skill levels.

Considering the algorithm base has an agnostic nature, devoid of context, our algorithm needs to be implemented in other different contexts — other than Almansur — otherwise we are only able to conclude that this solution works for this case.

Finally, considering the positive results so far, this work will likely stay as a part of the Almansur game, allowing for more time of testing and potentially some further development in this area of studies.

7. Future Work

Regarding Almansur, specifically, there are a few changes that could be implemented in order to improve results. These would benefit both old and new AIs.

- **Implement a Diplomatic Agent** — the most relevant issue found in the resulting AI is its lack of ability to communicate with other players. Frequently, when the player's territories begin to overlap, in-game personal messages are sent to question the opponents and check if they are active before attacking. It is common to avoid being attacked by simply replying, or to form alliances with the surrounding players, attempting an alliance victory. The current AIs are unable to deal with these situations.

- **Communication between modules** — the current communication and is not optimized. For example, when evaluating the value of a territory, every resource is considered equal to every other. Depending on the needs of the player, the economic player should be able to increase the value of a resource that was required, or decrease the value if it was in excess;
- **Responsability for generating/maintining Influence Maps** — in the current implementation, there isn't a specific class responsible for generating the influence maps. These, depending on their nature, should be generated by specific agents. For instance, the Influence Danger Map could be generated by the Strategy agent (or by a new Diplomatic agent), and the Influence Value Map could be generated by the Economic agent. This would allow for the definition of clear objectives and improve the communication between all the AI agents (or components).
- **Propagate Value Influence in the Map** — when an enemy is present at a specific location, its strength is the value of danger on that position. However, that danger is propagated to nearby territories in order to take into account the possible movements of that enemy unit. The Value does not propagate this way, though. If it did, it could allow for the definition of optimal paths to a target — allowing to conquer every territory along the way.

As for the algorithm itself, its presence in this environment could determine some flaw in its core and, through repetition, allow to fine tune some of the logic behind it. In theory, however, the algorithm would benefit more from being implemented in a different context, providing further insight on its uses and its shortcomings. Ultimately, only different implementations would allow for a final confirmation of worth for this algorithm on its own.

References

[1] U. Aickelin and J. Greensmith. Sensing danger: Innate Immunology for Intrusion Detection. *Information Security Technical Report*, 12(4):218–227, Jan. 2007.

[2] A. Barata, P. Santos, and R. Prada. AI for Massive Multiplayer Online Strategy Games. *AIIDE*, pages 110–115, 2011.

[3] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm intelligence: From Natural to Artificial Systems.* 1999.

[4] M. Carneiro. Artificial Intelligence in Games Evolution. *Business, Technological, and Social Dimensions of Computer Games: Multidisciplinary Developments*, pages 98–114, 2011.

[5] W. Collins, M. Carpenter, and J. Shankle. The iEconomy! pages 1–78, 2013.

[6] M. Dorigo and M. Birattari. Ant colony optimization. In *Encyclopedia of Machine Learning*, pages 36–39. Springer, 2010.

[7] A. Engelbrecht. *Computational Intelligence: An Introduction.* 2nd edition, 2007.

[8] C. Fairclough, M. Fagan, B. Mac Namee, and P. Cunningham. Research Directions for AI in Computer Games. *Irish Conference on Artificial Intelligence and Cognitive Science*, pages 333–344, 2001.

[9] K. Forbus, J. Mahoney, and K. Dill. How qualitative spatial reasoning can improve strategy game AIs. *Intelligent Systems, IEEE*, 17(4):1–8, 2002.

[10] P. Matzinger. The Danger Model: a Renewed Sense of Self. *Science (New York, N.Y.)*, 296(5566):301–5, Apr. 2002.

[11] W. Van Der Sterren. Squad tactics: Team ai and emergent maneuvers. *AI Game Programming Wisdom*, pages 233–246, 2002.

[12] Y. Zhu and Y. Tan. A Danger Theory Inspired Learning Model and Its Application to Spam Detection. pages 382–389, 2011.