Decentralized and Emergent NPC Coordination using Behavior Trees

Tiago Nunes Henriques
tnh@ist.utl.pt
Instituto Superior Técnico, Taguspark Campus
Av. Prof. Dr. Aníbal Cavaco Silva, Oeiras, Portugal

Abstract - Artificial Intelligence in videogames is being more used nowadays than it was before and an area that needs to have better AI is definitely coordination between individual characters. The aim of this project is to use Behavior trees, a recent structure introduced in the game industry, to enable coordination in a decentralized way, giving more focus on the characters behaviors. The StarCraft commercial game was used to show and evaluate our implementation.

Keywords - Behavior tree, emergent coordination, Non Player Character, behavior, game AI, StarCraft

Introduction

In the videogame industry, a game to succeed should have good graphics, animations and gameplay. For some types of games, those aspects can be enough, but nowadays, almost every game needs to have some kind of artificial intelligence (AI). AI technics are already being used in videogames for a long time, for instance the A* algorithm (1), used for characters path finding and Finite State Machines (FSM) (2), used to control characters’ behaviors in a game. For instance Lionhead Studios makes the difference with their games¹ because of the AI they have, which is the case of Black and White and Fable series.

Fact is that when the players see more realistic graphics they also expect more realistic behaviors. For instance when Crytek released Crysis², it had the most amazing graphics, but even though it was not bad, the character’s AI was not at the same level.

More recently, in an attempt to improve AI in game development, Behavior Trees (3) were introduced in the industry. This new structure is an attempt to improve FSM and they seem to be earning an important place in the industry, by simplifying and making easier to implement behaviors for in-game characters. And this is a good improvement because it helps a lot the behavior designers, because is a lot easier to make new behaviors when using behavior trees, especially if a good editor is used.

However, there are still some issues that are not trivial to resolve with behavior trees. One of these problems is the coordination (4) between two or more Non Player Characters (NPC), which current implementations do not handle well, because they always require an additional structure to deal with the problem.

Our goal is to achieve a behavior tree implementation that controls the actions of a NPC and make them cooperate with each other without external structures to support it. In other words, we want to have decentralized coordination and at the same time create interesting behaviors that rely only on the knowledge of the unit itself without receiving explicit orders from another character or entity. This decentralization is important because it makes the coordination more flexible and oriented for each NPC enabling each character to have control over themselves. This can lead each character to worry about things that are not usually seen in videogames where the AI is centralized, such as the health points of the unit and behave according to that.

It is also our intention to show that this approach can give the players a different and more satisfying game experience then the AIs that we usually see in games like real time strategy games. In order to do that, we will experiment our implementation in a well-known video game in an array of scenarios and also do some play testing to take conclusions at the efficiency level and the entertainment level.

¹ Http://lionhead.com/
² Http://crytek.com/games/crysis/overview
Behavior Trees

A behavior tree is a form of hierarchical logic, mostly based in Hierarchical Task Networks (HTN) (5), in which the root is the overall behavior, the non-leaf nodes represent simpler behaviors and the leaf nodes are atomic tasks that work as the interface between the behavior tree and game engine. Behavior trees have two types of leaves. Conditions, used to check something in the game world, for instance checks if the character has a long range weapon and Actions, which correspond to the actual actions inherent from the game, for instance shoot the enemy.

The non-leaf nodes are therefore composite tasks which can be composed by atomic tasks and other non-leaf nodes. They are used to add complexity to the behavior tree by choosing and managing which branches of the tree should be executed. To do this we have Sequences, this kind of node is used when we want the child tasks to be executed one by one in a specific order, i.e. the first child runs and if it succeed, the next child is executed, if not, the node fails. If the last child runs and succeeds the node also succeeds and Selectors, which are used when we want to have more than one child to choose from, depending on the situations, so the selector chooses one of the children to execute, if it succeed the node also succeeds, if not, the selector chooses another child to execute, if there are no child left, the node fails.

One big advantage of using behavior tree is that they are easily understood even for non-programmers people, because they are goal oriented. With this in mind, it is possible to structure the tree in smaller sub-trees and even use the same sub-tree in more than one branch of the overall tree.

Behavior trees were created to try to bridge the gaps that other hierarchical logic approaches used in games have, like FSM or scripting and less common HTN planner. The problem with scripting is the fact that introspection is difficult, so if something goes wrong it is not trivial what should be done. In other hand FSM takes a lot of work to edit, because of the big number of states and transitions. Planners are very heavy computationally because there is no easy way to re-plan. With behavior trees we not only manage to avoid other approaches cons as we can also avail their pros in one single structure.

Coordination

Agent coordination is a vast and complex subject and there have been many attempts to solve this problem even more nowadays because it is very common to see games where NPCs work together with each other or with the player itself.

Coordination is a very important subject in videogames (6) because the more times it happens and the better it is done more realism it gives to the scenes and more immersive will be the player experience.

Examples of scenarios where NPCs need to cooperate can be, Movement coordination, where two or more NPCs have to move in the terrain in a squad formation, Possible conflict tasks, where two or more NPCs need to go through the door and are trying to do so at the same time, but there are space just for one at a time, Tactic maneuvers, where in a battle two or more NPCs have a strategy to flank the player and Social interactions, where
a handshake between two human NPCs or a conversation where one participant talks while the other listens and vice versa. All of these examples are already present and can be seen in video games, but most of the times this is accomplished by scripting these behaviors manually, i.e. there is no real coordination because everything is preprogrammed.

In order to ensure coordination between agents it is necessary that each participant recognizes the behavior it is about to execute and knows what must be done at each moment. There are some approaches that have been developed to handle these problems and there are two basic ways of coordinating NPCs in a game, Centralized and Decentralized.

In a decentralized approach each agent has to, in a certain way, communicate with the others. They can do it in an explicit way or not. Explicit means sending a message to a particular agent or to everyone. Implicit means to leave marks in the environment so other agents can sense and act based on them, which is also called stigmergic coordination. Usually the explicit way is more used, but if it would make sense in the context of the game, an implicit approach may be used.

A centralized approach has the advantage of not being too complex and it is easy to program because there is just one entity controlling every NPC, but that implies that it cannot handle well the situation of a NPCs not being able to execute the order, for instance collisions with the scenario. Besides that with a centralized approach we are giving focus to a group, so if we have a game where we want to focus on individual NPC’s behaviors it is necessary to give it specific behaviors, and let the coordination emerge. In other hand, with emergent behaviors we introduce another problem, which is the communication between the agents.

What is usually seen in video games is a mixture of both approaches, particularly squad based games, where each NPC has to follow the squad’s goal and orders but it has its own decision making to avoid obstacles for example.

Original StarCraft AI

StarCraft AI is originally made to play in a complete game scenario, where it has to manage not only the attack, but everything inherent to an RTS game, like buildings, upgrades, units and resources. This is not the case of our scenarios, as our goals are only focused on unit attack and survival management. But as we which to compare our implementation with the original, we had to solve this little problem.

Originally StarCraft have different types of AI prepared to be used in the scenarios. There are custom AIs and campaign AIs. Campaign AIs are used for the campaign scenarios, where the objective is not to overwhelm the player but in fact, to let the player win easily. For this type of AI there are different levels of difficulty, going from easy to insane. Custom AIs are more like what we expect as an AI opponent for a head-to-head scenario, in which it tries to really win the game.

For our interests, the Custom AIs are not suitable because they expect to have a starting base, workers, i.e. units that construct new buildings and gather resources, and resources to collect, so we decided to use the Insane Campaign AI. We choose the insane one, because unlike the other campaign AIs, it really wants to win the game and it gives all it got, expecting nothing from the scenario.

But for the most scenarios we will use, to make a right and useful comparison the behavior of this AI is not enough. It needs to start an attack against the opposite player, which it does not since it stays still waiting for the player to attack. Fortunately there is a special script that we can use to complement the AI, which orders all units to make a suicide attack, i.e. it chooses an enemy unit and instructs them to attack it until all the opposite units are dead.

**BWAPI framework**

Since StarCraft is a commercial and closed game it is not open for anybody who wants to make AI experimentations on it. Fortunately there is an open-source framework which enables anybody to experiment their AI implementations. BWAPI is a free C++ framework with which we can retrieve information about the game and issue commands like a human player would do in the game. In addition to BWAPI we also used bwapi-mono-bridge, a wrapper for everybody who wants to use BWAPI, but prefers to program in C# instead of C++.

---

3 [http://classic.battle.net/scc/faq/aiscripts.shtml](http://classic.battle.net/scc/faq/aiscripts.shtml)


order to use the mono-bridge, it is also necessary to install the mono .Net framework\(^6\) version 2.8.

To make the connection between BWAPI and StarCraft it is also needed a program that injects that code in the game. The program used is called “Chaos Launcher”. Basically this program injects a library (.dll) which contains the BWAPI and the AI implementation into the game, taking control of the human player when the game starts.

**Architecture**

The StarCraftBot class is the representation of the player in the game and therefore is essential to make the connection with the game. Basically it implements the IStarCraftBot interface provided by BWAPI and it is through the onFrame event that it calls the **execute** method on every unit present in myUnits list.

The Unit class that represents each agent controlling each ally unit in the game that belongs to the player. Each of which has several data stored that is updated every logical frame in order to help it make its choices. It stores the **Target**, a list of **Allies**, a list of **Enemies**, a **Map** and the **Base position**.

Besides these properties, the Unit class also has important methods, two of which being directly called by the execute method. The **chooseAction method**, where the behavior tree is executed and therefore where an action for the unit perform in the game is chosen. And the **update method**, where before the action choice, in each frame the agent updates its internal variables and state and this is where it is done.

The Unit class has two subclasses associated, the HighTemplar and the Attacker, which in turn also have two subclasses, the Archon and the Zealot. The differences between each one of them are the implementation of the virtual methods present in the original class and each one is used according to the unit type.

Each Unit object has reference for another object representing other units that we created and are named **Unit Record**. In each Unit Record we have the information of the **Unit ID**, its **Hit Points**, if the unit is **In Sight**, the **Unit Type**, its **Initial Position**, its **Last Position** and a **Frame Stamp**.

**Unit Behavior Trees**

At least as far as we know, every implementation previously tested on coordination between agents in video game scenarios was based on some centralized entity, mostly blackboards where all the agents store their knowledge and load new information based on the knowledge of the others. Examples of that are the Marc Cavazza (8) experiments in The Dropship Game or the Choreographic Coordination scenario by Lima (9). Another common solution is to use a super-agent which commands all the other agents in the world or even have some hierarchy between the units, which is somewhat the case with what Tan and Cheng (10). On their experiments, they came out with a framework capable of having agents cooperating with each other in a decentralized by, but for the strategic attacks they have a hierarchy of agents that give orders to the agents above. Nevertheless, it is the project that most resembles to ours in terms of decentralized agent coordination.

As it is one of our goals to achieve a solution without any kind of external entity, our implementation consists only in a large behavior tree made of smaller trees and nothing more. Therefore, all the information that each agent have is only known by it, until it is able to share it with the others. This communication between the agents occurs only when the correspondent units are in sight range.

All the decisions the agents take are based only on information received from others and on their own, not receiving direct orders from any other agent.

**Conceptual Model**

The tree in Figure 2 is the overall behavior tree of each agent. It is a selector with seven other sub trees each of which representing seven different types of behaviors that each unit can perform in the game. The priority for

---

\(^6\) www.go-mono.com
each child to run is from left to right and since this is a selector node, only one of them will run successfully.

The Defense behavior is used when the unit is in its base and is under attack. The Run Away behavior node is entered when the unit has enemies in sight but cannot attack them. The offense behavior is activated when the unit has enemies in sight and is ready to fight. The explore behavior is triggered when the unit does not know any alive enemy and therefore will explore the map to find some. The wait enemy behavior is used when there is an entrance point in the map for enemy units to enter in the match and make the ally units to wait nearby. The return to base behavior is used when the unit must retreat from battle or return to the base due a timeout.

**Figure 3 - “Offense” Behavior tree**

If the unit is not in danger and it do not need to defend at the base, then the unit must try to take the offense. The tree in Figure 3 shows how that behavior works. Before anything else it is necessary to check three things. If the agent knows any enemy unit that can be attacked by it, if the unit can attack and if it is not returning back to base. After that, the agent enters in a sequence of tasks. First, if the unit is not close to the enemy unit, it has to move towards its position. Then when the unit is about to engage in combat, i.e. is already near the enemy unit, it has to check if the unit is prepared to attack, i.e. if it thinks it has conditions to kill the enemy unit without being killed. If not, the unit must do whatever is necessary to be prepared. Otherwise it is now time for the unit to attack.

**Figure 4 - “Prepare To Attack” Behavior tree**

On the other hand, if the unit is not yet prepared to attack, it must prepare itself. In Figure 4 we have the tree that represents that behavior. If the unit is a High Templar then it just needs to wait until it is prepared, because it is just a matter of time. Although if the unit is an Attacker, i.e. a Zealot or an Archon, it must prepare according the situation.

- If the unit realizes that it needs the help of the High Templars, it returns to the base hoping to find a High Templar at it or in the way. If it happens to meet one, then it makes the request for it to transform into an Archon.
- If the unit is excessively close to the enemy unit but should not attack it, which is the case, then it must step back, i.e. move away a little bit by moving the unit to a close but further position from the enemy.
- If the unit has allies in sight range it can either approach them in order to form a bigger group, or if a timeout expires must move towards the base, searching for more ally units.
- If none of the three situations above is the actual situation, then the unit must move to the base.

The “approach group” action consists in calculate the centroid of the units around it and move towards it.

Finally, if the unit type is neither a High Templar nor an Attacker, it should do nothing, but it is opened for possible new implementations of other types.
Evaluation

In order to know if our solution has achieved the proposed goals, we made two types of evaluations:

- **Measurements at the end of each match:** Represented by the outcome of the corresponding game match. We recorded the time spend to the end of the match and the number of dead ally and enemy units. It was also taken note of the behaviors of each AI’s units, namely how they searched their enemies, which enemy they chosen to attack, how they attacked (e.g. if they used always the same method), if they used all of their potential, i.e. used all their abilities and if they coordinated with each other, same and different types of units.

- **Tests with users:** With a specific scenario we gathered a group of individuals to play against our AI and the original StarCraft AI, so they could compare each other and tell their opinions on their performances. At the end, we asked the testers to answer to a questionnaire regarding the experience they had, in order for us to evaluate them and take our conclusions about both AI’s performances.

Every scenario consists in two teams (player 1 and player 2), not necessarily with the same units, in an open area and there objective is to win the match, i.e. eliminate all opponent units. Player 1 always has its base at the bottom of the scenario and is controlled either by our implementation or the original AI. Player 2 always has its base at the top of the scenario and is controlled by the StarCraft AI with different behaviors depending on the scenario. Player 1 always controls Protoss units, whilst player 2 controls Protoss or Terran units.

Regarding what is known by each player, our AI only knows beforehand the static map information, i.e. the terrain, and does not know anything about the opposite player. Because of this, it needs to explore the map to know what exists in the world, even ally units are unknown. On other hand, the original StarCraft AI always has all map information, like static map information and the position of every unit in the map. This applies if it is controlling either player 1 or player 2. Unfortunately we cannot do anything to change this, so we have to carry out with this disadvantage.

Looking at the measurements made in all scenarios, including the information in Table 1, we learned interesting things. It is noticeable how our implementation outperformed the original AI in terms of unit survival. On the offensive scenarios, our AI managed to end their matches with more than half of its units alive, while the original AI had 80% of dead units in the end.

However, this survival advantage comes with a cost of time consumption. We can see that the original AI completes its goal faster in every scenario and on the overall it lasted more than twice then the original AI to finish all the matches. Nevertheless there are scenarios where the original AI does not finish its goal because it stops acting or just because it lost all units, whereas our implementation always finishes the matches, almost every time being the winner. The main motive for our AI not perform better in this scenario, is due the collisions between the units that are running away, with the ones that are going in the direction of the enemies. In more recent RTS games, collision avoidance technics are used (e.g. in StarCraft 2) but in StarCraft they are not. Therefore, every time groups of units are going on a collision course with each other, they lose too much time redefining their routes, giving time to the enemy units to attack them.

<table>
<thead>
<tr>
<th></th>
<th>Dead Allies</th>
<th>Time (min:sec)</th>
<th>Wins</th>
<th>Finished Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our AI</strong></td>
<td>673</td>
<td>06:20:32</td>
<td>50</td>
<td>60</td>
</tr>
<tr>
<td><strong>Original AI</strong></td>
<td>1169</td>
<td>02:46:45</td>
<td>29</td>
<td>41</td>
</tr>
<tr>
<td><strong>Max</strong></td>
<td>1470</td>
<td>∞</td>
<td>60</td>
<td>60</td>
</tr>
</tbody>
</table>

Table 1 – Totals from offensive scenarios

Looking at the totals on the defensive scenarios (Table 2), one more time our implementation managed to perform better than the original AI. As expected though, our implementation lasted more time in play and in consequence killed more units. This shows us that independently of the scenario, if both AIs have the same conditions, our implementation has better control over the units than the original AI. This is mostly caused by the fact that the units retreat when they have their shield low and also because they communicate with the ally units that are waiting near to the base, when they enter in their sight range, which makes them to know of the existence of the enemies. These behaviors result in an emergent cooperation between these units, where new units take the place of the units retreating, thereby maximizing the total life points of all of them.
To complement the results of the measures, it is necessary to have a more qualitative evaluation. For that reason, a play testing session was prepared and we let 20 players play against our AI and the original StarCraft AI, so that in the end they could respond to a questionnaire about their game experience. The reason that made us choose play testing over showing some videos for people analyze was because we wanted to have the opinions from their interaction with the AI. By seeing videos of gameplay, the person would not be sufficiently immersed in the game to analyze the scenario as a player would, which is the objective of these tests.

So the testers can have a baseline of comparison, they have the possibility to play against our AI and also the original AI. But since it is important that the testers do not know if they are playing against the original AI or our AI, they are referenced as AI A for the original and AI B for ours. It is great that the testers have this possibility, but to do it we need to be careful with the order in which each is played, because the first will undoubtedly influence the strategy of the tester when he plays against the second. To surpass this problem a commonly used solution is to divide the testers in two groups, where the sequence in which each AI is played is inverted, so group 1 plays against AI A first and AI B second and group 2 plays against AI B first and AI A second. In our case we were intercalating the testers making the group of the testers with even numbers and other group with the odd numbers.

To introduce the game controls, the scenario and the units and their abilities, an extra phase is added before the actual test begins. In this way we can make the player more acquainted to game and consequently when he is in the real test, he can focus more of his attention on the behaviors of the AI opponent.

To really evaluate and make conclusions on their experiments, two questionnaires were elaborated. One is about the tester itself about their gaming habits, experience and opinion. The other is about the match experience and their opinions on the AI opponent performance. This questionnaire is answered two times, one for each AI.

In order to make a good questionnaire about artificial intelligence we looked at the Godspeed questionnaires (11) to take some ideas. The article mention questionnaires to test robots and therefore most of them are useless to our work, except for the perceived intelligence questionnaire, which has exactly the questions we were looking forward.

This questions are a five point scale answer and on each one is asked to rate the impression of the robot on its Competency, Knowledge, Responsibility, Intelligence and Sense. In our case we decided to split the Knowledge question into six, because we wanted to have a more detailed opinion. So, we made our questionnaire with the Godspeed questions being the core questions, where the Knowledge question was split in knowledge about the location, the hit points and the capabilities of the ally and enemy units.

According to the article, it is important to have more questions mixed within to mask the intention. For that we looked into GameFlow (12), a model for evaluating player enjoyment in games and we saw that an important element is that a game should be challenging. Since one of the challenges in games is the AI opponents, we added five more questions about gaming, such as if the AI was challenging and if it surprised the player.

20 people attended to our test sessions and with them we were able to have 10 testers for each group, which is a reasonable number of testers for this case. Each session took about 25 minutes to be completed.

As we have done in the test scenarios, at the end of each match we took note of the number of ally and enemy units killed.

<table>
<thead>
<tr>
<th></th>
<th>Dead Enemies</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our AI</td>
<td>2403</td>
<td>05:56:49</td>
</tr>
<tr>
<td>Original AI</td>
<td>1390</td>
<td>02:56:01</td>
</tr>
</tbody>
</table>

Table 2 - Totals from defensive scenarios

To introduce the game controls, the scenario and the units and their abilities, an extra phase is added before the actual test begins. In this way we can make the player more acquainted to game and consequently when he is in the real test, he can focus more of his attention on the behaviors of the AI opponent.

To really evaluate and make conclusions on their experiments, two questionnaires were elaborated. One is about the tester itself about their gaming habits, experience and opinion. The other is about the match experience and their opinions on the AI opponent performance. This questionnaire is answered two times, one for each AI.

In order to make a good questionnaire about artificial intelligence we looked at the Godspeed questionnaires (11) to take some ideas. The article mention questionnaires to test robots and therefore most of them are useless to our work, except for the perceived intelligence questionnaire, which has exactly the questions we were looking forward.

This questions are a five point scale answer and on each one is asked to rate the impression of the robot on its Competency, Knowledge, Responsibility, Intelligence and Sense. In our case we decided to split the Knowledge question into six, because we wanted to have a more detailed opinion. So, we made our questionnaire with the Godspeed questions being the core questions, where the Knowledge question was split in knowledge about the location, the hit points and the capabilities of the ally and enemy units.

According to the article, it is important to have more questions mixed within to mask the intention. For that we looked into GameFlow (12), a model for evaluating player enjoyment in games and we saw that an important element is that a game should be challenging. Since one of the challenges in games is the AI opponents, we added five more questions about gaming, such as if the AI was challenging and if it surprised the player.

20 people attended to our test sessions and with them we were able to have 10 testers for each group, which is a reasonable number of testers for this case. Each session took about 25 minutes to be completed.

As we have done in the test scenarios, at the end of each match we took note of the number of ally and enemy units killed.

<table>
<thead>
<tr>
<th></th>
<th>Units Lost</th>
<th>Units Killed</th>
<th>Matches Won</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>%</td>
<td>mean</td>
</tr>
<tr>
<td>Original AI</td>
<td>27,30</td>
<td>91</td>
<td>24,15</td>
</tr>
<tr>
<td>Our AI</td>
<td>28,65</td>
<td>95,5</td>
<td>13,05</td>
</tr>
</tbody>
</table>

Table 3 - Results from all 20 tester's matches

By looking to the results detailed in Table 3, we can see that either against the original AI and our AI, on average the testers lost almost the same number of units. But if
we look to the number of matches won, we see that almost half of the testers managed to win the match against the original AI, whilst only 3 managed to win against ours. This happens because, if we look at the units killed, we see that the testers managed to kill much more units while playing against the original AI.

As mentioned in the session guide, the testers after playing a match, answered to a questionnaire regarding their thoughts about the opponent AI they just played against.

<table>
<thead>
<tr>
<th>Knowledge</th>
<th>Original AI</th>
<th>Our AI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resistance</td>
<td>3.85</td>
<td>4.50</td>
</tr>
<tr>
<td>Surprise</td>
<td>3.50</td>
<td>4.20</td>
</tr>
<tr>
<td>Competency</td>
<td>3.80</td>
<td>4.55</td>
</tr>
</tbody>
</table>

Table 4 - Results from the AI questionnaires (the higher the value the better)

In Table 4 we have the average answers to the questions made, including the Godspeed questions. In the individual behavior scale, values closer to 1 means that the orders were given to a group of units and values closer to 5 means the units make their own decisions and do not receive orders from anything.

As we can see, our AI has bigger values in practically all the categories, only the knowledge of the location of the enemy units and the individual behavior is the opposite. In the knowledge of the location of the units, it is easily explained with the fact that all the units with the original AI attack directly the player.

One measure we wanted to obtain was the perceived intelligence mentioned in the Godspeed questionnaires. To do that, we combined the six different scales regarding the knowledge questions and created the scale Knowledge. Then we did the same thing with the five perceived intelligence questions, in order to create one single scale, which is the Perceived Intelligence scale.

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>σ</th>
<th>Cronbach’s Alpha</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original AI</td>
<td>3.46</td>
<td>0.89</td>
<td>0.89</td>
</tr>
<tr>
<td>Our AI</td>
<td>4.28</td>
<td>0.51</td>
<td>0.78</td>
</tr>
</tbody>
</table>

Table 5 - Perceived Intelligence results

With a Cronbach’s Alpha over 0.7 seen in Table 5, which Nunnally (13) recommends being the minimum value, the results from both AIs are internally consistent. However both AIs are above the middle value 3, it is obvious the difference between them, with our AI being almost one value higher in the scale than the original AI and also with the standard deviation being lower.

In order to evaluate the significance of these results a paired difference test was made, where both AIs make a pair for each question. The Kolmogorov-Smirnov test, made with the results from the questionnaires, shows that all the pairs have at least one value under the 0.05 significance, except for the single scale Knowledge and the perceived intelligence. So, for these two measures, the test to do must be the Paired Student’s t-test, for the other ones, the test to make must be the Wilcoxon signed-rank test.

<table>
<thead>
<tr>
<th></th>
<th>Significance (2-tailed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resistance</td>
<td>0.051</td>
</tr>
<tr>
<td>Surprise</td>
<td>0.018</td>
</tr>
<tr>
<td>Competency</td>
<td>0.021</td>
</tr>
<tr>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>Enemies</td>
<td>0.273</td>
</tr>
<tr>
<td>Allies</td>
<td>0.080</td>
</tr>
<tr>
<td>Hit Points</td>
<td></td>
</tr>
<tr>
<td>Enemies</td>
<td>0.163</td>
</tr>
<tr>
<td>Allies</td>
<td>0.012</td>
</tr>
<tr>
<td>Capacities</td>
<td></td>
</tr>
<tr>
<td>Enemies</td>
<td>0.073</td>
</tr>
<tr>
<td>Allies</td>
<td>0.793</td>
</tr>
<tr>
<td>Single scale</td>
<td></td>
</tr>
<tr>
<td>Responsibility</td>
<td>0.032</td>
</tr>
<tr>
<td>Intelligence</td>
<td>0.003</td>
</tr>
<tr>
<td>Sense</td>
<td>0.002</td>
</tr>
<tr>
<td>Individual Behavior</td>
<td>0.729</td>
</tr>
</tbody>
</table>

Table 6 - Paired difference test significance results for the questionnaire measures
In Table 6 we have the significance results from the Wilcoxon test and it confirms that almost every pair differences are significant, because they are under the maximum value, 0.05 derived from the 95% confidence interval. The results with two columns means that the result was above the 0.05 value, but since the results are 2-tailed we can divide the value by two, and the second column represents that value. This means that the individual behavior and the knowledge of the enemy location, hit points and the capacities of the allies are not statistically significant. Nonetheless, the most important measure we wish to see is the perceived intelligence and that one is undoubtedly significant.

Chart 1 - Perceived Intelligence

By looking to the mean values obtained in the questionnaire and there statistical results, we can say that the testers think our AI is more competent, knowledgeable, responsible, intelligent and sensible. Those five characteristics are part of the Perceived intelligence Godspeed IV questionnaire\(^7\) and with a Cronbach’s Alpha over 0.7, the results are coherent. Since the statistical Paired Student’s t-test results shows the questionnaire results are significant, is notorious the perceived intelligence that our AI has, compared to the original StarCraft AI (see Chart 1).

Conclusions

After all the experiments made and measures taken we gain knowledge about our AI that we could not obtain by any other way.

Looking at the scenarios and to both AIs, the biggest conclusion we made was that when the units coordinate with each other and have notion of their heath points, they can finish more scenarios being the winners and with less casualties. Although it was expected that our units would run away, since the AI was programmed to make the unit retreat if its shields were lower, it was great to see that a good strategy emerged and it is an efficient way to win a match. This behavior also proved to be the core of the strategy that ended to emerge in the overall attack process.

We can also compare our AI with the original by making an analogy to the evolution of the real battles over the centuries. The original AI is similarly to what used to happen in the past in those great and epic battles where the main weapons were swords, bows and arrows. On those battles the soldiers were in line formations and the ones in the back could not do anything until the soldiers in front were killed. Basically they were all like cannon fodder and each soldier was just a number. While our AI is more like what happens in more recent wars, where each soldier is the most important thing and everything must be done to win a battle with the fewer casualties possible.

From the results of the questionnaires, units lost/killed and commentaries made by testers about both the original AI and our AI, we conclude that the testers were very surprised with the behaviors of our AI and showed much more satisfaction playing against our AI then the original. It was also notorious that the testers understood the self-preservation “feeling” that the units have.

Unfortunately, the testers did not realized that our units were being controlled by a single entity, since the results to the question about their individual behavior was very close to the middle value and statistically it did not had significance. However, the results for the original AI were identical, so we cannot take great conclusion on this one, except that this is perhaps very difficult to analyze and also the behavior of the units can be deceiving.

In general we can say that our AI is superior to the original AI in almost every way. The results from the scenario tests between Ais show that is true. The play testing results and the confidence degree given by the statistical results based on the questionnaires, also tell us that it is true.

\(^7\) http://www.bartneck.de/2008/03/11/the-godspeed-questionnaire-series/
References


