

# Player Adaptive Cooperative Artificial Intelligence for RTS Games

T.J.A. Jansen

June 13, 2007

## Abstract

Developing an Artificial Intelligence can be a difficult task. Instead of applying static rules to which the AI has to adhere, the AI is able to adapt to its environment. This paper will focus on the cooperative aspects of AI development. Our approach to creating a cooperative AI focusses on creating a model of the AI's companion. Thusing opponent-modelling techniques, analysing the partners weaknesses and strengths, the AI will have to adapt itself to the partner player, to support and complement the player in every move he makes.

## 1 Introduction

This introductory section provides some background information about Artificial Intelligence development. The problem statement is described here and an overview of the paper is given. Also, opponent-modelling will briefly be explained.

### 1.1 Background Information

Artificial Intelligence is a challenging field which combines several disciplines with different game types (for example, RTS games, FPS games etc.). Mathematics, physics and computer science are just a few of the disciplines this field contains. AI also has to be able to determine for different play styles during a session of a game. A session can be played aggressively or friendly, you can have an opposing AI player or a cooperative AI player. Current Artificial Intelligences do not seek the limits of current hardware, this makes us believe that hardware does not pose a limitation in the field of AI development, the development itself however does. The intelligence of the AI ideally is adapted to the players intelligence. Meaning, the AI has to be a worthy opponent, or a decent companion.

In this paper we will research cooperative AI for RTS games. Cooperative AI is capable of supporting a player, the artificial intelligence has to be a cooperative player. In other words, the artificial intelligence has to show cooperative skills. In most current computer games however, artificial intelligences do not really cooperate with

their companion. They co-exist besides their partner, doing their own tasks, without attacking the partner player. Some cooperative traits are defending each other, and together attack an opponent.

For the duration of this paper, we will place the AI in a perfect information environment. Eliminating all of the problems that may arise when working in an imperfect information setting.

In general an AI is developed with winning in mind. Finding the most efficient way to win a session is most typically the goal of an AI. However when a cooperative AI is developed, it doesn't have to win. In this case the AI has to complement the player, support his moves, and work together with the player to achieve their common goal. But developing an AI which can cooperate requires more analysis. One constantly has to monitor the human player, determine what kind of action he is taking at that moment, whether it be an aggressive action, a defensive action, an explorative action or an action to extend your base. More specific, how does one classify these different types of actions? How does one determine that the player is just exploring the map instead of trying to find the opposing player, whilst in the back of his/her mind getting preparing a fast attack strategy? Keeping this in mind indicates that player modelling has a high priority when developing a cooperative AI.

For this paper it is important that we want to develop an artificial intelligence which can detect changes in pattern made by a player, and make a choice to match those changes. In case a player gets attacked, the AI should notice that the unit count decreases, and start defending the player. In case a player largely increases his offensive unit count, it should determine that that player wants to become aggressive, and in order to match that also start producing offensive units to aid him in his attack.

When thinking about how to be able to analyse this, player modelling comes to mind. There are already several approaches on how to model the behaviour of the player. But the most commonly used technique is opponent modelling. Using this technique you can create a model of a specific player, and in turn be able to deduce that players weaknesses/strengths. If all the required information then is gathered, you can then predict the players current game plan and adapt the AI to match

that, resulting in a cooperative AI.

## 1.2 Problem Definition

The main task that needs to be focussed on is developing an artificial intelligence that can cooperate with his partner player in real-time. This task will be tackled in stages. Starting with the opponent-modelling technique, we establish a player model and be able to determine the weaknesses/strengths of the AI's partner. Ultimately, to adapt the AI to match the player so they form a team in which both players support each other.

To achieve this we use machine learning to train the artificial intelligence, using a neural network to make decisions when the cooperative intelligence is in use. A decision tree will also be used, to determine to which type of units a unit belongs to (defensive, neutral or offensive). The experiments to test the generated cooperative AI will be conducted in the open-source real-time strategy game called Total Annihilation Spring. As mentioned earlier, all experiments will be conducted in a perfect-information scenario.

## 1.3 Overview

The outline of this paper is as follows; our approach is described in section 2. There, we will cover all techniques that are used to achieve the goal of the project: opponent-modelling, machine learning, neural network, AI adaptability and the completed AI. We will also explain how these approaches are applied in the artificial intelligence, and other details about the inner workings of some of the algorithms.

In the next section (Chapter 3), we will describe the experiments that we have done, and explain the found results. In section 4 we will present a short summary of the report as well as the conclusion with respect to the goal of this project. Finally, in the last section, section 5, we will discuss the obtained results, and provide the concluding remarks.

## 2 Approach

In this section all of the techniques used to develop a cooperative artificial intelligence are described. We start with the opponent-modelling technique, next go on by describing machine learning, finally neural networks. These techniques are used to create a player model, analyse the player, teach the intelligence the style the player is using, and then to have it make its own decisions about which actions to take in order to complement and support the player. Followed by the AI's adaptability. In order to support/complement a partner player in the actions he/she takes, it is necessary to constantly monitor the players actions. One has to recognise behavioural patterns, and if a change in a repeating pattern occurs, adapt the artificial intelligence to match those changes.

## 2.1 Opponent Modelling

Opponent Modelling is a technique which is used to create a model of the opponent which in term can be used to predict the moves the opponent is making. This model then is used to predict the next move the opponent will make, and allow the player to beat the opponent. In case of cooperative AI development however this model will be used to predict the moves the 'partner' player is making. Allowing the AI to support the player even before he makes a specific move, simply because that move can be predicted. To create a model of a player, the following items are taken into account

1. Number of defensive structures built
2. Number of defensive structures destroyed
3. Number of defensive units built
4. Number of defensive units destroyed
5. Number of neutral structures built
6. Number of neutral structures destroyed
7. Number of neutral units built
8. Number of neutral units destroyed
9. Number of offensive structures built
10. Number of offensive structures destroyed
11. Number of offensive units built
12. Number of offensive units destroyed
13. Percentage of map uncovered

Each of the previous listed items will be stored, and be used to create a player model. By looking at the partitioning of the units, if one type of units is dominating over the other types, you can determine the main play style the AI's partner player is adhering to. By constantly monitoring this partitioning you can almost instantly predict an upcoming change, this because of the fact that the dominating type changes. For example; if the amount of offensive units increases drastically, you can deduce that the player is getting ready to take the role of aggressor.

By looking at the relations between the units built, destroyed and currently available of a type, you can analyse a players weaknesses/strengths. For example; if a player has a high number of defensive units built, as well as a high number of units destroyed, you can deduce that the particular player has a weakness in his defense. This analogy also applies to the neutral and offensive unit count. When looking at the structures, it is possible to analyse a players weaknesses and strengths, however this also is linked to the unitcount. For example; if a player has a high number of defensive units, but many of his structures are destroyed, whilst only few defensive units are destroyed, he does not necessarily have a problem

with his defending structures, but more with his ability to respond to actions the opponent makes, or with unit placement.

The percentage that the map is uncovered will for the duration of this paper not be taken into account, since the fact that in this paper we are working with a perfect information setting. However, in imperfect situations this is also an important issue. For example; if only 10% of the map is uncovered, but a lot of units already are destroyed, you can deduce that the opposing player's base is near.

In this paper we will focus more on the positive use of opponent-modelling. Instead of using the determined weaknesses/strengths to eliminate a particular player, we use it to support the player.

## 2.2 Supervised Learning

[2] Supervised learning is a machine-learning technique for creating a function from training data. The training data consist of pairs of input object, and desired outputs. The output of this function is used to predict the class label of the input object, in other words, classifying the input object. The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples (i.e., pairs of input and target output). To achieve this, the supervised learner has to use inductive bias, so it can generalize from the presented data to be able to predict unseen situations.

### Overview

Supervised learning can generate models of two types. Most commonly, supervised learning generates a global model that maps input objects to desired outputs. In some cases, however, the map is implemented as a set of local models (such as in the nearest neighbour algorithm). In order to solve a given problem of supervised learning one has to consider the following steps:

1. Determine the type of training examples. Before doing anything else, the engineer should decide what kind of data is to be used as (learning) samples. In our case, a single unit, a collection of units, or a complete unit type.
2. Gathering a training set. The training set needs to be characteristic of the real world use of the function. Thus, a set of input objects is gathered and corresponding outputs are also gathered, in this case, from the decision tree.
3. Determine the input feature representation of the learned function. The accuracy of the learned function depends strongly on how the input object is represented. Typically, the input object is transformed into a feature vector, which contains a num-

ber of features that should not be too large. (because of the curse of dimensionality), but should be large enough to accurately predict the output.

4. Determine the structure of the learned function and corresponding learning algorithm. In our case a neural network.
5. Complete the design. The engineer then runs the learning algorithm on the gathered training set. Parameters of the learning algorithm may be adjusted by optimizing performance on a subset (called a validation set) of the training set, or via cross-validation. After parameter adjustment and learning, the performance of the algorithm may be measured on a test set that is separate from the training set.

Another term for supervised learning is classification. A wide range of classifiers are available, each with its strengths and weaknesses. Classifier performance depend greatly on the characteristics of the data to be classified. There is no single classifier that works best on all given problems, this is also referred to as the "No free lunch theorem".

## 2.3 Neural Networks

[1] A neural network is an interconnected group of artificial neurons that uses a mathematical model or computational model for information processing based on a connectionist approach to computation changes its structure based on external or internal information that flows through the network.

In practical terms neural networks can be characterised as decision making tools. They are used to model complex relationships between inputs and outputs or to find patterns in data.

### 2.3.1 Background

There is no precise agreed definition among researchers as to what a neural network is, but most agree that it involves a network of simple processing elements (neurons) which can exhibit complex global behavior, determined by the connections between the processing elements and element parameters. The original inspiration for the technique was from examination of the central nervous system and the neurons which constitute one of its most significant information processing elements. In a neural network model, simple nodes are connected together to form a network of nodes. While a neural network does not have to be adaptive per se, its practical use comes with algorithms designed to alter the weights of the connections in the network to produce a desired signal flow.

These networks are also similar to the biological neural networks in the sense that functions are performed

collectively and in parallel by the units, instead of having a clear delineation of subtasks to which various units are assigned. Currently, the term Artificial Neural Network (ANN) tends to refer mostly to neural network models employed in statistics, cognitive psychology and artificial intelligence. Neural network models designed with emulation of the central nervous system (CNS) in mind are a subject of theoretical neuroscience.

In modern software implementations of artificial neural networks the approach inspired by biology has more or less been abandoned for a more practical approach based on statistics and signal processing. In some of these systems neural networks, or parts of neural networks (such as artificial neurons), are used as components in larger systems that combine both adaptive and non-adaptive elements. While the more general approach of such adaptive systems is more suitable for real-world problem solving, it has far less to do with the traditional artificial intelligence connectionist models. What they do however have in common is the principle of non-linear, distributed, parallel and local processing and adaptation.

**2.3.2 Models**

Neural network models in artificial intelligence are usually referred to as artificial neural networks (ANNs); these are essentially simple mathematical models defining a function

$$f : X \rightarrow Y$$

Each type of ANN model corresponds to a class of such functions.

**2.3.3 The network in a neural network**

The word network in the term "neural network" arises because the function  $f$  is defined as a composition of other functions  $g_i$ , which can further be defined as a composition of other functions. This can be conveniently represented as a network structure, with arrows depicting the dependencies between variables. A widely used type of composition is the nonlinear weighted sum,

$$f(x) = K \left( \sum_i w_i g_i(x) \right)$$

where  $K$  is some predefined function, such as the hyperbolic tangent. It will be convenient for the following to refer to a collection of functions  $g_i$  as simply a vector

$$g = (g_1, g_2, \dots, g_n)$$

Figure 1 depicts such a decomposition of  $f$ , with dependencies between variables indicated by arrows. These can be interpreted in two ways.

The first view is the functional view: the input  $x$  is transformed into a 3-dimensional vector  $h$ , which is

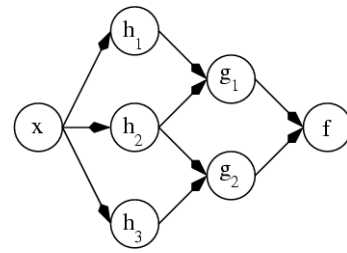


Figure 1: ANN Dependency Graph

then transformed into a 2-dimensional vector  $g$ , which is finally transformed into  $f$ . This view is most commonly encountered in the context of optimization.

The second view is the probabilistic view: the random variable  $F = f(G)$  depends upon the random variable  $G = g(H)$ , which depends upon  $H = h(x)$ , which depends upon the random variable  $x$ . This view is most commonly encountered in the context of graphical models.

The two views are largely equivalent. In either case, for this particular network architecture, the components of individual layers are independent of each other (e.g., the components of  $g$  are independent of each other given their input  $h$ ). This naturally enables a degree of parallelism in the implementation.

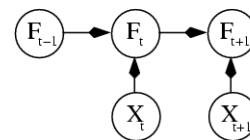
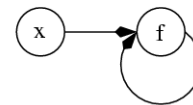


Figure 2: Recurrent ANN Dependency Graph

Networks such as the previous one are commonly called feedforward, because their graph is a directed acyclic graph. Networks with cycles are commonly called recurrent. Such networks are commonly depicted in the manner shown at the top of the figure, where  $f$  is shown as being dependent upon itself. However, there is an implied temporal dependence which is not shown. In practise this implies that the value of  $f$  at some point in time  $t$  depends upon the values of  $f$  at zero or at one or more other points in time. The graphical model at the bottom of the figure illustrates the case: the value of  $f$  at time  $t$  only depends upon its last value. Models such as these, which have no dependencies in the future, are called causal models.

### 3 Experiments

In this section we will discuss the experiments that have taken place. In the first section the setup of the experiments will be described. The experiments that will be conducted will be described in section 2, and the results of those experiments in the third section.

#### 3.1 Setup

To determine if the AI is the cooperative AI we require, we will perform several experiments. Those experiments will test the following three abilities of the AI;

1. Pattern recognition
2. Weakness/strength analysis
3. Neural network

During all of the experiments the cooperative AI will be placed in a perfect information environment. The cooperative AI will be put in different sessions with a human player, one or more opposing AAI players, and finally a cooperative AAI player. All experiments will take place on 4 different maps, and will be repeated 5 times per map. To ensure that the AI does not depend on a particular map, different maps will be used. Also, all experiments will be conducted 5 times to obtain accurate average results because of potential outliers during one of the sessions. The goal of these experiments is to find out how the cooperative AI performs performance wise. The cooperative AI has to support and/or complement his partner player within a predetermined timeframe. A lower response time results in a higher success rate, a higher response time in a lower success rate. If the timeframe gets exceeded, then the experiment will have failed. This timeframe however is influenced by several factors, such as;

1. Distance between the cooperative AI and the partner player
2. Distance between the cooperative AI and resources
3. Unit(s) used for the situation

The direct relation between the distance between the two players in a team and the timeframe is clear, if the distance increases, so does the time required to respond, and thus the timeframe. The relation between the distance between the cooperative AI and resources might however not be so clear. However, if the cooperative AI requires more time to gather resources, he automatically requires more time to build adequate units to match the partner players strategy, and thus we will have to increase his timeframe. The final factor that influences the timeframe are the unit(s) used for the situation; a plane will travel faster than a tank. Thus the timeframe will be influenced on that timeframe. This indicates that

the timeframe is different each time the cooperative AI has to respond.

The rate of success of a response will be measured by the following formula:

$$f = \frac{(U_s - U_d)}{U_d} * 100$$

wherein  $U_s$  stands for unit(s) sent, and  $U_d$  stands for unit(s) destroyed. If however the timeframe gets exceeded, then we can say that the test has failed. In that case, the rate of success will automatically be minimised to 0.

#### 3.2 Experiments

In this section all of the experiments will be described, as well as the desired outcome per experiment. As stated before, each experiment will be run on 4 different maps, and repeated 5 times per map.

##### *Experiment 1*

The first experiment that will be done is the following. On a map with only two players, one controlled by the cooperative AI, and one by a human player. The human player starts by building 25 defensive units. Then over a time period of 10 minutes the human player will not build any unit. Next, he will build 50 offensive units. This experiment is performed to test the AI's ability. If the cooperative AI can detect the change in the pattern, it is able to adapt to it.

##### *Desired outcome*

First the cooperative AI start by building defensive units. When the human player changes his game style, and starts building offensive units, the AI also should start building offensive units. The cooperative AI player should respond within a time frame of 1 minute after it notices the change in pattern.

##### *Success rate calculation*

If the cooperative AI builds equal amounts of defensive and offensive units, it will have a 100% success rate. All other rates will be calculated by the following formula:

$$s = \left( 1 - \frac{Abs(U_h - U_c)}{U_h} \right) * 100$$

where  $U_h$  stands for the amount of units the human player built (75 in this experiment),  $U_c$  stands for the amount of units built by the cooperative AI.

##### *Experiment 2*

For this experiment, the setting will be as follow. Two teams compete with each other in a 2 vs. 2 situation. One team consists of an artificial intelligence controlled by the AAI intelligence. The other team consists

of a human player and the cooperative artificial intelligence working together. Instead of attacking the opposing AAI player with the human controlled player, we will wait until the AAI controlled player attacks the human player. Will the cooperative AI player start to defend the partner player? Even when the partner player is not defending himself? This experiment is done to check the weaknesses/strengths analysis of the cooperative AI.

#### *Desired outcome*

The cooperative AI should detect the attack on the human player. As soon as 1 defensive unit or building of the human player gets destroyed, the cooperative AI should start moving defensive units to the location where the attack takes place. When the units are in place the cooperative AI should attack the aggressor, and thus aid the human player in his defense.

#### *Success rate calculation*

The success rate of this experiment gets calculated by counting the number of units lost by the cooperative AI, comparing them against the units sent by the cooperative AI, and express that relation in percentages.

Both of the experiments will take place on the following 4 different maps:

1. Four Islands V2.1
2. CorePyramid
3. BlackStar
4. Nightscape V2

All 4 of the maps are chosen at random. The properties which the different maps provide also vary. So does BlackStar contain water spaces in the land. Four Islands V2.1 is surrounded by water, whilst being connected in the middle. NightScape V2 provides several mountain scapes, and finally CorePyramid provides everything on a single vast plane.

### 3.3 Results

In this section we will present all results of the experiments. Each experiment was conducted on 4 different maps, 5 consecutive sessions per map. Each of the tables show the indicated success rate per round per map.

#### *Experiment 1*

<i>Round</i>	<i>FourIslandsV2.1</i>
1	96%
2	91%
3	94%
4	89%
5	90%
<i>Round</i>	<i>CorePyramid</i>
1	100%
2	92%
3	89%
4	93%
5	92%

<i>Round</i>	<i>BlackStar</i>
1	92%
2	93%
3	96%
4	64%
5	95%
<i>Round</i>	<i>NightScapeV2</i>
1	90%
2	92%
3	97%
4	100%
5	96%

As can be seen in the first batch of experiments, the cooperative AI is able to match the human player in his game play. Most commonly the variations are made because of the fact that the AI does not build the exact amount of units the human player builds. In most 14 of the 20 rounds, the AI built more units than his human counterpart. In 4 of the 20 rounds, the AI built less units, and in 2 of the 20 rounds, the AI exactly matched the amount of units built. In 1 of the 20 rounds, more specifically Round 4 on the BlackStar map, the cooperative AI did not build any defensive units. The reason why the AI missed the building of defensive units is unknown. This round can thus be seen as an outlier in the results. *Average success rate of experiment 1: 92.05%*

#### *Experiment 2*

<i>Round</i>	<i>FourIslandsV2.1</i>
1	69%
2	78%
3	75%
4	72%
5	71%
<i>Round</i>	<i>CorePyramid</i>
1	81%
2	83%
3	76%
4	71%
5	80%
<i>Round</i>	<i>BlackStar</i>
1	51%
2	67%
3	33%
4	39%
5	61%
<i>Round</i>	<i>NightScapeV2</i>
1	42%
2	26%
3	81%
4	69%
5	54%

The second batch of experiments shows us a broad range of results. This range however can be explained, and will

provide us with several future improvements. The cooperative AI notices the attack on the human player. This results in it attacking the opposing player. However, it goes to the nearest enemy it finds. When the AAI player and the cooperative AI's bases are near, it will not defend the human player, but it will attack the base of the AAI player, while leaving the human player undefended. This results in the cooperative AI losing units.

Maps like Four Islands V2.1 and CorePyramid, maps that provide a fairly even distribution of starting points, pose no problem for the AI. Maps like BlackStar and NightScape V2 are built up in a different way, those maps allow the human player to be further away than the AAI player. They do not provide an even distribution of starting points, resulting in unwanted responses by the Cooperative AI.

This implies in practise that in cases where the AAI and cooperative AI are more near to each other than the cooperative AI and the human player. The cooperative AI then makes the 'wrong' decision when it has to defend the human player. Also, if the cooperative AI player has to attack the AAI player, it seems to ignore the partner player of the AAI player. This has the following consequence, unit loss of the cooperative AI player due to attacks by the partner of the AAI player. To be more specific, the cooperative AI player does not plan the best route to the place it has to reach. Instead, it finds the shortest route, and takes that. *Average succes rate of experiment 2: 63.7%*

## 4 Conclusion

We discuss our experimental findings/results, provide concluding remarks, and finish by suggesting improvements for future research.

### 4.1 Main Conclusion

The obtained results show a different picture than the one we had in mind when developing the cooperative AI. We now are aware that the cooperative AI is able to do pattern recognition, and is able to analyse strong and weak points of a particular player. Using the decision tree the cooperative AI can determine whether a particular player is aggressive or defensive. Mimicing the game play by the partner player is done by the neural network.

We now can ensure that the cooperative AI can "mimic" the actions made by his partner. It however can not determine which of the possible actions might be the best one. This can best be noticed in unevenly distributed maps that are similar to NightScape V2. Or when a map is not used to its maximum potential, which was the case when experiment 2 was used on the BlackStar map.

The main conclusion we can make is thus the following. The cooperative AI is able to perform its most basic features such as pattern recognition and weakness/strength analysis. It still however does not perform in the most optimal way. As can be seen in the next section there are several improvements which can be done to ensure that the cooperative AI performs the way we expect to perform.

### 4.2 Future improvements

In the future, the first thing we need to focus on is making sure that the cooperative AI is able to determine the difference between the point of attack and the aggressor. This will then remove the random unit loss which is caused by maps which do not have an even distribution of starting points. Problems like the uneven distribution of starting points also will occur when a map is made for a specific amount of players, and you play with a different amount of players on that map. For instance, BlackStar, this map is made for 5 players, however in experiment 2 we only play with 4 players. This will result in an unwanted loss of units by the cooperative AI in situations where the path between the cooperative AI and the wanted attack point.

Further more, what also can be improved is the unit selection made by the cooperative AI. Currently it mostly matches the unit choice which is made by the partner player. The partner player however might not make the most optimal choice of units depending on the map and situation he is in. This problem can be made often by beginners. Ideally, the cooperative AI has to determine the most effective unit which the situation requires. This also can be a weakness of the partner player, but the cooperative AI does is unable to recognise this.

Lastly, we currently are working in perfect information scenario's. Working with imperfect information should also be incorporated in the cooperative AI. In that case it might be combined with several other techniques to scout the map, and determine then what might be the best course of action, and start to prepare for those possible situations.

## References

- [1] *Wikipedia: Artificial Neural Networks.*  
[http://en.wikipedia.org/wiki/Artificial\\_neural\\_network](http://en.wikipedia.org/wiki/Artificial_neural_network).
- [2] *Wikipedia: Supervised Learning.*  
[http://en.wikipedia.org/wiki/Supervised\\_learning](http://en.wikipedia.org/wiki/Supervised_learning).