Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2009, Article ID 129075, 10 pages
doi:10.1155/2009/129075

Research Article

Combining AI Methods for Learning Bots in a Real-Time Strategy Game

Robin Baumgarten,[1] Simon Colton,[1] and Mark Morris[2]

[1] *Department of Computing, Imperial College London, London SW7 2AZ, UK*
[2] *Introversion Software Ltd., Glastonbury BA6 BWF, UK*

Correspondence should be addressed to Robin Baumgarten, robin.baumgarten06@doc.ic.ac.uk

Received 31 May 2008; Accepted 21 October 2008

Recommended by Kok Wai Wong

We describe an approach for simulating human game-play in strategy games using a variety of AI techniques, including simulated annealing, decision tree learning, and case-based reasoning. We have implemented an AI-bot that uses these techniques to form a novel approach for planning fleet movements and attacks in DEFCON, a nuclear war simulation strategy game released in 2006 by Introversion Software Ltd. The AI-bot retrieves plans from a case-base of recorded games, then uses these to generate a new plan using a method based on decision tree learning. In addition, we have implemented more sophisticated control over low-level actions that enable the AI-bot to synchronize bombing runs, and used a simulated annealing approach for assigning bombing targets to planes and opponent cities to missiles. We describe how our AI-bot operates, and the experimentation we have performed in order to determine an optimal configuration for it. With this configuration, our AI-bot beats Introversion's finite state machine automated player in 76.7% of 150 matches played. We briefly introduce the notion of ability versus enjoyability and discuss initial results of a survey we conducted with human players.

1. Introduction

DEFCON is a multiplayer real-time strategy game from Introversion Software Ltd., for which a screenshot is provided in Figure 1. Players compete in a nuclear war simulation to score as many points as possible by hitting opponent cities. The game is divided into stages, beginning with placing resources (nuclear silos, fleets of ships, airbases, and radar stations; see Figure 1) within an assigned territory, then guiding fleet manoeuvres, bombing runs, fighter attacks, and finally missile strikes.

The existing single-player mode contains a computer opponent that employs a finite state machine with five states which are carried out in sequence:

(1) placement of ground units and fleet,

(2) scouting by planes and fleet to uncover structures of the opponent,

(3) assaults on the opponent with bombers,

(4) a full strike on the opponent with missiles from silos, submarines, and bombers,

(5) a final state, where fleets of ships approach and attack random opponent positions.

Once the state machine has reached the fifth state, it remains in that state for the remainder of the game. This results in a predictable strategy that may appear monotonous to human players.

We have designed and implemented a novel two-tiered bot to play DEFCON: on the bottom layer, there are enhanced low-level actions that make use of in-match history and information from recorded games to estimate and predict opponent behavior and manoeuvre units accordingly. The information is gathered in influence maps (see also [1]) and is used in synchronous attacks, a movement desire model, fleet formation, and target allocation. On top of these tactical means, we have built a learning system that is employed for the fundamental long-term strategy of a match. The operation of this system is multifaceted and

Figure 1: Screenshot of DEFCON.

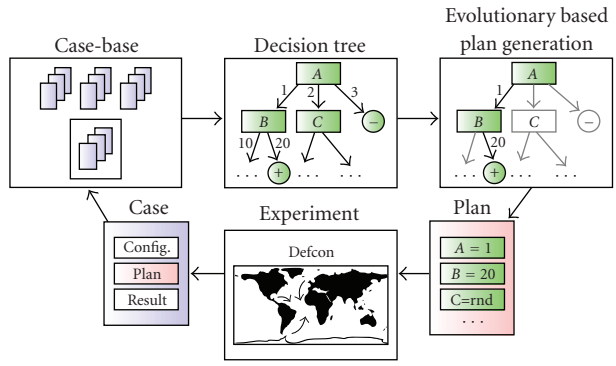

Figure 2: Overview of the system design.

relies on a number of AI techniques, including simulated annealing, decision tree learning, and case-based reasoning. In particular, the AI-bot maintains a case-base of previously played games to learn from, as described in Section 2. It uses a structure placement algorithm to determine where nuclear silos, airbases, and radars should be deployed. To do this, the AI-bot retrieves games from the case-base, ranks them using a weighted sum of various attributes (including life span and effectiveness) of the silos, airbases and radars in the previous game, and then uses the ranking to determine placement of these resources in the game being played, as described in Section 3.

Our AI-bot also controls naval resources, organized into fleets and metafleets (i.e., groups of fleets). Because these resources move during the game, the AI-bot uses a high-level plan to dictate the initial placement and metafleet movement/attack strategies. To generate a bespoke plan to fit the game being played, the AI-bot again retrieves cases from the case-base, and produces a plan by extracting pertinent information from retrieved plans via decision tree learning, as described in Section 4.

During the game, the AI-bot carries out the metafleet movement and attack plan using a movement desire model which takes its context (including the targets assigned to ships and opponent threats) into account. The AI-bot also controls low-level actions at game-time, such as the launching of plane bombing runs, attempting to destroy incoming missiles, and launching missile attacks from fleets. As described in Section 5, we implemented various more sophisticated controls over these low-level actions. In particular, we enabled our AI-bot to synchronize the timing of planes when they attack opponent silos. We also implemented a simulated annealing approach to solve the problem of assigning bombing targets to planes and opponent cities to missiles.

We have performed much experimentation to fine-tune our AI-bot in order to maximize the proportion of games it wins against Introversion's own player simulator. In particular, in order to determine the weights in the fitness function for the placement of silos and airbases, we have calculated the correlation of various resource attributes with the final score of the matches. We have also experimented with the parameters of the simulated annealing search for assignments. Finally, we have experimented with the size of the case-base to determine if/when overfitting occurs. With the most favorable setup, in a session of 150 games, our AI-bot won 76.7% of the time. We describe our experimentation in Section 6.

The superiority of our AI-bot leads to the question of whether higher ability implies higher enjoyability for human players. To this end, we have proposed a hypothesis and conducted an initial survey, which we describe in Section 7. In Sections 8 and 9, we conclude with related work and some indication of future work.

2. Learning from Previous Games

2.1. Learning Cycle Overview. The design of the learning bot is based on an iterative optimization process, similar to that of a typical evolutionary-based process. An overview of the cycle is depicted in Figure 2.

Given a situation requesting a plan, a case-base of previous plan—game pairs is used to select matching plans according to a similarity measure described in the next subsection. This subset of plans is then used in a generalization process to create a decision tree, where each node contains an atomic plan item, as described in Section 4.1. The new plan is generated as a path in the decision tree, derived by starting at the root node and then descending to a leaf node by choosing each branch through a fitness-proportionate selection. The fitness function for the quality of this plan is the game itself, where the AI-bot plays according to the plan, described in Sections 3, 4, and 5. Together with the plan itself, the obtained game data and outcome form a new case. As the decision tree learning requires both positive and negative examples, the new case is retained regardless of the actual outcome of the match.

2.2. A Case-Base of Plans. We treat the training of our AI-bot as a machine learning problem in the sense of [2], where an agent learns to perform better at a task through increased exposure to the task. To this end, the AI-bot is able to store previously played games in a case-base, and retrieve games in order to play its current match more effectively. After a game is played, the AI-bot records the following information as an XML data-point in the case-base:

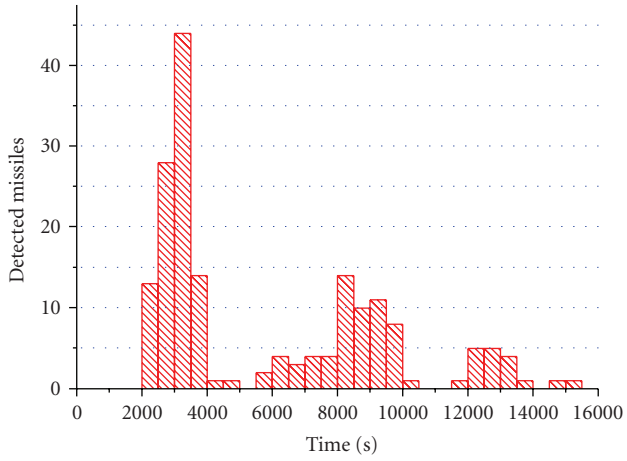

Figure 3: Launched opponent missiles during a match, showing wave-pattern of attacks. The time shown is in-game time.

(i) the starting positions of the airbases, radar stations, fleets, and nuclear silos for both players;

(ii) the metafleet movement and attack plan which was used (as described in Section 4);

(iii) performance statistics for deployed resources which are for nuclear silos the number of missiles attacked and destroyed and planes shot down by each silo, for radar stations the number of missiles identified, and for airbases the number of planes launched and the number of planes which were quickly lost;

(iv) an abstraction of the opponent attacks which took place; we abstract these into waves, by clustering using time-frames of 500 seconds and a threshold of 5 missiles fired (these settings were determined empirically, see Figure 3 for a typical attack distribution);

(v) routes taken by opponent fleets;

(vi) the final scores of the two players in the game.

Cases are retrieved from the case-base using the starting configuration of the game. There are 6 territories that players can be assigned to (North America, Europe, South Asia, etc.), hence there are ${}^6P_2 = 30$ possible territory assignments in a two-player game, which we call the starting configuration of the game. This has a large effect on the game, so only cases with the same starting configuration as the current game are retrieved. For the rest of the paper, we assume that a suitable case-base is available for our AI-bot to use before and during the game. How we populate this case-base is described in Section 6. Cases are retrieved from the case-base both at the start of a game—in order to generate a game plan, as described in Section 4.3—and during the game, in order to predict the fleet movements of the opponent. At the start of a game, cases are retrieved using only the starting territory assignments of the two players.

3. Placement of Silos, Airbases, and Radars

Airbases are structures from which bombing runs are launched; silos are structures which launch nuclear missiles at opponent cities and defend the player's own cities against opponent missile strikes and planes; and radar stations are able to identify the position of enemy planes, missiles, and ships within a certain range. As such, all these structures are very important, and because they cannot be moved at game time, their initial placement by the AI-bot at the start of the game is a key to a successful outcome. The AI-bot uses the previously played games to calculate airbase, silo, and radar placement for the current game. To do this, it retrieves cases with the same starting configuration as the current game, as described above. For each retrieved game, it analyzes the statistics of how each airbase, silo, and radar performed.

Each silo is given an effectiveness score as a weighted sum of the normalized values for the following:

(a) the number of enemy missiles it shot at;

(b) the number of enemy missiles it shot down;

(c) the number of enemy planes it shot down;

(d) the time it survived before being destroyed.

With respect to the placement of silos, each case is ranked using the sum of the effectiveness of its silos. Silo placement from the most effective case is then copied for the current game. The same calculations inform the placement of the radar stations, with the effectiveness given by the following values:

(a) the number of enemy planes detected;

(b) the number of enemy planes detected before other radars;

(c) the number of enemy ships detected;

(d) the time it survived before being destroyed.

Finally, the placement of airbases is determined with these effectiveness values:

(a) the number of planes launched;

(b) the number of units destroyed by launched planes;

(c) the time it survived before being destroyed.

To find suitable weights in the weighted sum for effectiveness, we performed a correlation analysis for the retaining/losing of resources against the overall game score. This analysis was performed using 1500 games played randomly (see Section 4.3 for a description of how randomly played games were generated). In Figure 4, we present the Pearson product-moment correlation coefficient for each of the AI-bot's own resources.

We note that—somewhat counter-intuitively—the loss of carriers, airbases, bombers, fighters, battleships, and missiles is correlated with a winning game. This is explained by the fact that games where fewer of these resources were lost will have been games where the AI-bot did not attack enough (when attacks are made, resources are inevitably

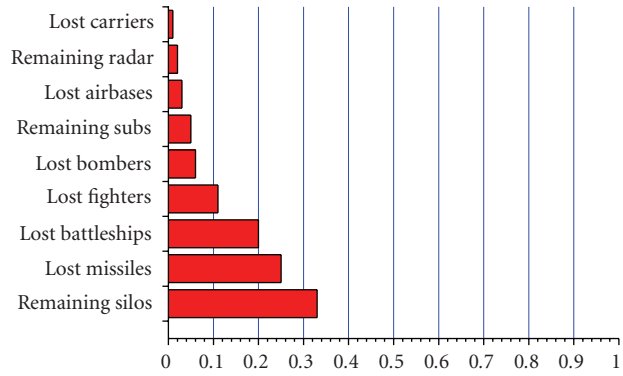

Figure 4: Pearson product-moment correlation coefficient for loss/retention of resources.

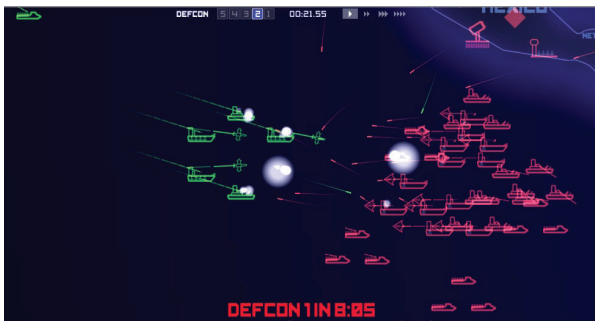

Figure 5: Two fleets attacking each other in DEFCON.

lost). For our purposes, it is interesting that the retention of silos is highly correlated with winning games. This informed our choice of weights in the calculation of effectiveness for silo placement: we weighted value (d), namely, the time a silo survived, higher than values (a), (b), and (c). In practice, for silos, we use 1/10, 1/3, 1/6, and 2/5 as weights for values (a), (b), (c), and (d), respectively. We used similar correlation analyses to determine how best to calculate the effectiveness of the placement of airbases and radar stations.

4. Planning Ship Movements

An important aspect of playing DEFCON is the careful control of naval resources (submarines, battleships, and aircraft carriers). We describe here how our AI-bot generates a high-level plan for ship placement, movement, and attacks at the start of the game, how it carries out such plans (see Figure 5 for a sea attack), and how plans are automatically generated.

4.1. Plan Representation. It is useful to group resources into larger units so that their movements and attacks can be synchronized. DEFCON already allows collections of ships to be moved as a fleet, but players must target and fire each ship's missiles independently. To enhance this, we have introduced the notion of a metafleet which is a collection of a number of fleets of ships. Our AI-bot will typically have a small number of metafleets, (usually 1 or 2) with each one independently targeting an area of high opponent population. The metafleet movement and attack plans describe a strategy for each metafleet as a subplan, where the strategy consists of two large-scale movements of the metafleet. Each subplan specifies the following information.

(1) In what general area (sea territory) the ships in the metafleet should be initially placed, relative to the expected opponent fleet positions.

(2) What the aim of the first large-scale movement should be, including where (if anywhere) the metafleet should move to, how it should move there, and what general target area the ships should attack, if any.

(3) When the metafleet should switch to the second large-scale movement.

(4) What the aim of the second large-scale movement should be, including the same details as for (2).

4.2. Carrying out Metafleet Movements at Game-Time. Sea territories—assigned by DEFCON at the start of a game—are split into two oceans, and the plan dictates which one each metafleet should be placed in. The exact positions of the metafleet members are calculated at the start of the game using the case-base, that is, given the set of games retrieved, the AI-bot determines which sea territory contains on average most of the opponent's fleets. Within the chosen sea territory, the starting position depends on the aim of the first large-scale movement and an estimation of the likelihood of opponent fleet encounter which is calculated using the retrieved games. This estimation uses the fleet movement information associated with each stored game in the case-base. The stored information allows the retrieval of the position of each fleet from the stored game as a function of time. For any given position, the closest distance from that position which each fleet obtains during the game can be calculated. The likelihood of enemy fleet encounter is then estimated by the fraction of games in which enemy fleets get closer to the observed position than a predefined threshold.

There are five aims for the large-scale movements, namely,

(a) to stay where they are and await the opponent's fleets in order to engage them later,

(b) to move in order to avoid the opponent's fleets,

(c) to move directly to the target of the attack,

(d) to move to the target avoiding the opponent's fleet,

(e) to move towards the opponent's fleet in order to intercept and engage them.

The aim determines whether the AI-bot should place the fleets at (i) positions with high-opponent encounter likelihoods (in which case, large-scale movements (a) and (e) are undertaken), (ii) positions with low-opponent encounter likelihoods (in which case, large-scale movements (b) and (d) are undertaken), or (iii) positions which are as close to the attack spot as possible (in which case, large-scale

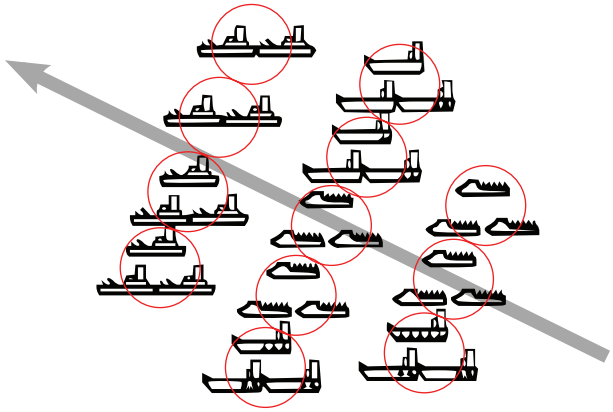

Figure 6: Fleet formation in DEFCON with front direction shown. Each circle indicates a separate fleet of up to three ships.

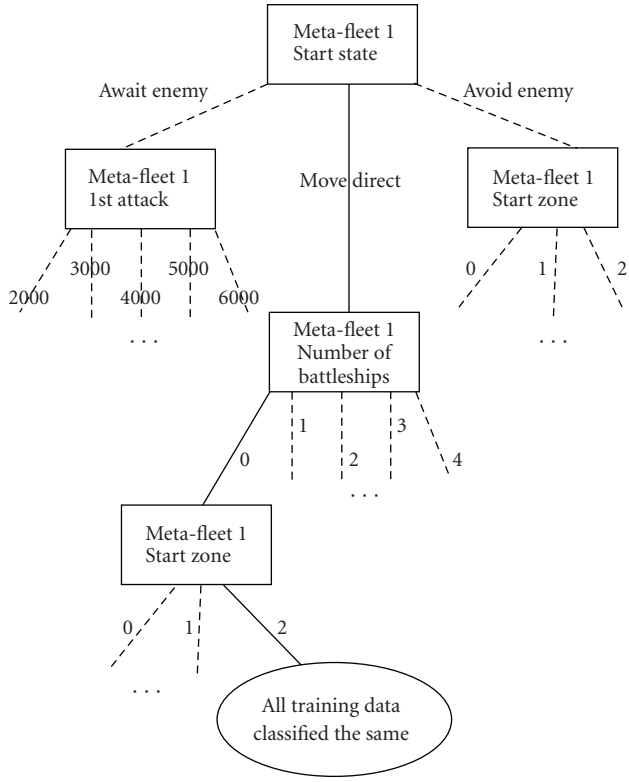

Figure 7: Full selected path through the decision tree. Chosen path is highlighted in bold, not chosen branches are truncated. Remaining plan items are filled in randomly, in this case this is second large-scale movement, first attack time, and number of carriers in metafleet.

movement (c) is undertaken). To determine a general area of the opponent's territory to attack (and hence to guide a metafleet towards), our AI-bot constructs an influence map [1] built using opponent city population statistics. It uses the map to determine the centers of the highest population density, and assigns these to the metafleets.

We implemented a central mechanism to determine both the best formation of a set of fleets into a metafleet, and the direction of travel of the metafleet given the aims of the large-scale movement currently being executed. During noncombative game-play, the central mechanism guides the metafleets towards the positions dictated in the plan (see Figure 6), but this does not take into account the opponent's positions.

Hence, we also implemented a movement desire model to take over from the default central mechanism when an attack on the metafleet is detected. This determines the direction for each ship in a fleet using (a) proximity to the ship's target if this has been specified (b) distance to any threatening opponent ships, and (c) distance to any general opponent targets. A direction vector for each ship is calculated in light of the overall aim of the large-scale metafleet movement. For instance, if the aim is to engage the opponent, the ship will sail in the direction of the opponent's fleets.

The movement desire model relies on being able to predict where the opponent's fleets will be at certain times in the future. To estimate these positions, our AI-bot retrieves cases from the case-base at game-time, and looks at all the various positions the opponent's fleets were recorded at in the case. It then ranks these positions in terms of how close they are to the current positions of the opponent's fleets. To do this, it must assign each current opponent ship to one of the ships in the recorded game in such a way that the overall distance between the pairs of ships in the assignment is as low as possible. As this is a combinatorially expensive task, the AI-bot uses a simulated annealing approach to find a good solution, which is described in more detail in Section 5. Once assignments have been made, the five cases with the closest assignments are examined and the fleet positions at specific times in the chosen retrieved games are projected onto the current game to predict the future position of the opponent's fleets. The five cases are treated as equally likely, thus fleets react to the closest predicted fleet positions according to the aim of their large-scale movement, for instance, approach or avoid it.

4.3. Automatically Generating Plans. As mentioned above, at the start of a game, the starting configuration is used to retrieve a set of cases. These are then used to generate a bespoke plan for the current game as follows. Firstly, each case contains the final score information of the game that was played. These are ranked according to the AI-bot's score (which will be positive if it won, and negative if it lost). Within this ranking, the first half of the retrieved games are labeled as negative, and the second half are labeled as positive. Hence, sometimes, winning games may be labeled negative and, at other times, losing games may be labeled positive. This is done to achieve an example set that generates a more detailed decision tree using the *ID3* algorithm, as described below.

These positive and negative examples are used to derive a decision tree which can predict whether a plan will lead to a positive or a negative game. The attributes of the plan in the cases are used as attributes to split over in the decision tree, that is, the number of metafleets, their starting

sea territories, their first and second large-scale movement aims, and so on. Our AI-bot uses the *ID3* algorithm [2] to learn the decision tree. This algorithm builds a decision tree by iteratively choosing the attribute with the highest information gain (i.e., the amount of noise reduction when splitting the dataset according to the attribute) to split the data. *ID3* is a greedy algorithm that grows the decision tree top-down until all attributes have been used or all examples are perfectly classified. By maximizing the entropy, that is, making sure that there are roughly the same number of negative and positive examples, *ID3* generates a deeper tree, because it takes more steps to perfectly classify the data. As we see below, this is beneficial, as each path in the tree represents a partial plan, with longer paths dictating more specific plans.

We portray the top nodes of an example tree in Figure 7. In this example, we see that the most important factor for distinguishing positive and negative games is the starting sea territory for metafleet 1 (which can be in either low, mid, or high enemy threat areas). Next, the decision tree uses the aim of the second large-scale metafleet movement, the attack time, and the number of battleships in metafleet 1.

Each branch from the top node to a leaf node in these decision trees represents a partial plan, as it will specify the values for some—but not necessarily all—of the attributes which make up a plan. The AI-bot chooses one of these branches by using an iterative fitness-proportionate method, that is, it chooses a path down the tree by looking at the subtree below each possible choice of value from the node it is currently looking at. Each subtree has a set of positive leaf nodes, and a set of negative leaf nodes, and the subtree with the highest proportion of positive leaf nodes is chosen (with a random choice between equally high-scoring subtrees). This continues until a leaf node is reached. Having chosen a branch in this way, the AI-bot fills in the other attributes of the plan randomly. The number of randomly assigned attributes depends on the size of the case-base, for 35 cases this is about 3 attributes.

5. Synchronizing Attacks

In order for players not to have to micromanage the playing of the game, DEFCON automatically performs certain actions. For instance, air defence silos automatically target planes in attack range, and a battleship will automatically attack hostile ships and planes in its range. Players are expected to control where their planes attack, and where missiles are fired (from submarines, bombers, and silos).

5.1. Attacks as Assignment Problems. As mentioned above, the AI-bot uses an influence map to determine the most effective general radius for missile attacks from its silos, submarines, and bombers. Within this radius, it must assign a target to each missile. This is a combinatorially difficult problem, so we frame it as an instance of the assignment problem [3], and our AI-bot searches for an injective mapping between the set of missiles and the set of cities using a simulated annealing heuristic search. To do this, it calculates the fitness of each mapping as the overall population of the cities mapped onto, and starts with a random mapping. Using two parameters, namely the starting temperature S and the cool-down rate c, a pair of missiles is chosen randomly, and the cities they are assigned to are swapped. The new mapping is kept only if the fitness decreases by no more than S times the current fitness. When each missile has been used in at least one swap, S is multiplied by c and the process continues until S reaches a cut-off value.

Figure 8: Synchronized attack in DEFCON.

For most of our testing, we used values $S = 0.5$, $c = 0.9$, and a cut-off value of 0.04, as these were found to be effective through some initial testing. We also experimented with these values, as described in Section 6. Note that the mapping of planes to airbases for landing is a similar assignment problem, and we use a simulated annealing search process with the same S and c values for this.

Only silos can defend against missiles, and silos require a certain time to destroy each missile. Thus attacks are more efficient when the time frame of missile strikes is kept small, so we enabled our AI-bot to organize planes to arrive at a target location at the same time.

To achieve such a synchronized attack, our AI-bot makes individual planes take detours so that they arrive at the time that the furthest of them arrives without detour (see Figure 8 for an example of a synchronized attack using this method). Basic trigonometry gives two possible detour routes and our AI-bot uses the influence map to choose the route which avoids enemy territory the most.

6. Experimentation

We tested the hypothesis that our AI-bot can learn to play DEFCON better by playing games randomly, then storing the games in the case-base for use as described above. To this end, for experiment 1, we generated 5 games per starting configuration (hence 150 games in total), by randomly choosing values for the plan attributes, then using the AI-bot to play against Introversion's own automated player. Moreover, whenever our AI-bot would ordinarily use retrieved cases from the case-base, uninformed (random) decisions were made for the fleet movement method, and the placement algorithm provided by Introversion was used. The

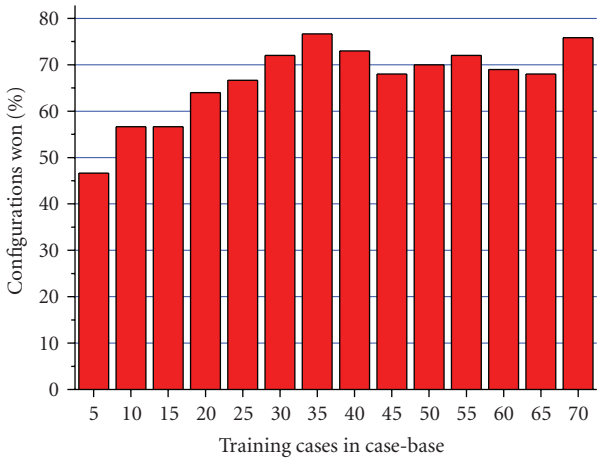

Figure 9: Number of winning configurations versus the size of training data.

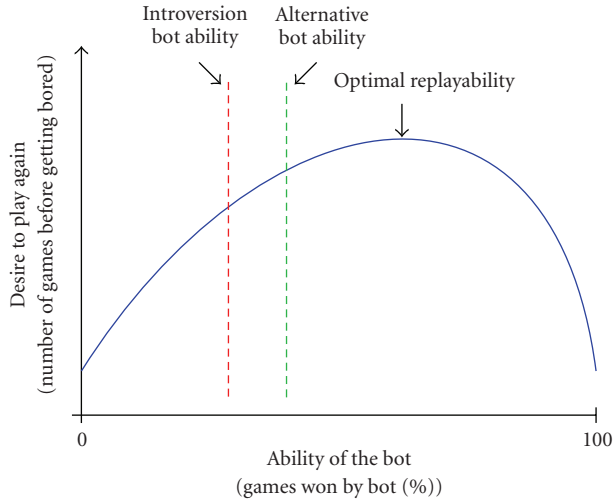

Figure 10: Graph for enjoyability as a function of bot ability (hypothetical).

Table 1: Performance versus simulated annealing parameters.

S	c	Games won (%)	Mean score differential
0	0	53.3	13.3
0.3	0.75	73.3	22.7
0.5	0.9	76.7	33.2
01.01.00	0.95	69.0	34.9
01.01.00	0.99	73.3	33.0

five games were then stored in the case-base. Following this populating of the case-base, we enabled the AI-bot to retrieve and use the cases to play against Introversion's player 150 times, and we recorded the percentage of games our AI-bot won. We then repeated the experiment with 10, rather than 5 randomly played games per starting configuration, then with 15, and so on, up to 70 games, with the results portrayed in Figure 9.

We see that the optimal number of cases to use in the case-base is 35, and that our AI-bot was able to beat Introversion's player in 76.7% of the games. We analyzed games with 35 cases and games with 40 cases to attempt to explain why performance degrades after this point, and we found that the decision tree learning process was more often using idiosyncracies from the cases in the larger case-base, hence overfitting. We describe some possible remedies for overfitting in Section 8.

Using the 35 cases optimum, we further experimented with the starting temperature and cool-down rate of the simulated annealing search for mappings. As described in Section 5, our AI-bot uses the same annealing settings for all assignment problems, and we varied these from no annealing ($S = c = 0$) to very high annealing ($S = 1.0$, $c = 0.99$). As portrayed in Table 1, we recorded both the proportion of wins in 150 games and the score differential between the players. We see that our initial settings of $S = 0.5$, $c = 0.9$ achieved the highest proportion of wins, whereas the setting $S = 1.0$, $c = 0.95$ wins fewer games, but does so more convincingly.

In a final set of experiments, we tried various different metafleet planning setups to estimate the value of learning from played games. We found that with random generation of plans, our AI-bot won 50% of the time, and that by using hand-crafted plans developed using knowledge of playing DEFCON, we could increase this value to 63%. However, this was not as successful as our case-based learning approach, which—as previously mentioned—won 76.7% of games. This gives us confidence to try different learning methods, such as getting our AI-bot to play against itself, which we aim to experiment with in future work.

7. Discussion

We achieved the initial goal of building an AI-bot that can consistently beat the one written by Introversion and included with the DEFCON distribution. Its capability to learn and improve from previous experience makes it more competitive, and thus we can assign a higher *ability* to our bot. This observation leads us to the question of whether increased ability of a computer opponent translates into increased enjoyability for human players.

7.1. Ability versus Enjoyability. We define *enjoyability* in the context of computer games as *the desire to play again* after a match, that is, the number of games a user wants to play before he/she gets bored or frustrated and thus stops enjoying the game. We hypothesize that there is a correlation between ability and enjoyability.

In Figure 10, we portray a hypothetical graph with the ability of the bot (in terms of the percentage of games won against an opponent) plotted against the desire to play again (in terms of the number of games played before the player gets bored). Bots with a fixed strength are therefore points on the ability-axis, indicated by the dotted vertical lines. If our hypothesis is valid, a function similar to the blue graph might emerge. The rationale behind this is that too low or too high

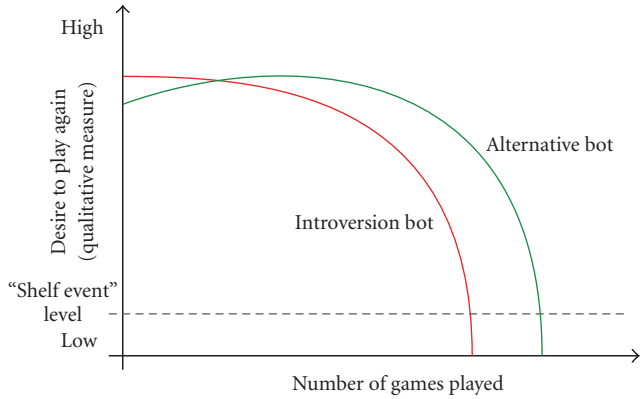

Figure 11: Graph for enjoyability as a function of time (hypothetical).

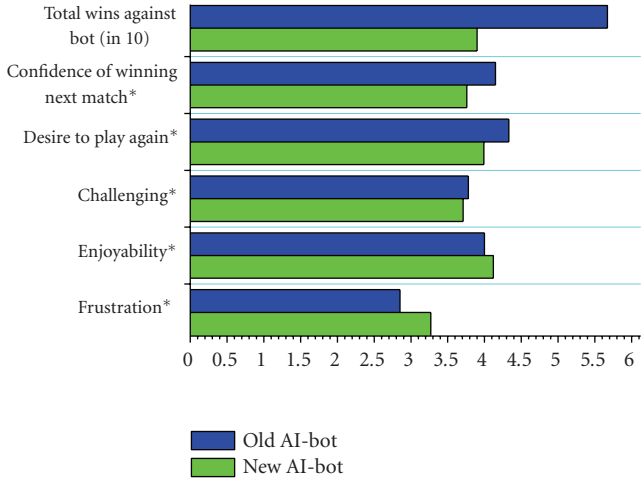

Figure 12: Results of the conducted survey, averaged over 10 games. Values marked with an asterix range from 1 = very low to 6 = very high.

bot ability will bore or frustrate the player and negatively affect his/her desire to play again. The maximum of this function is the ability of a bot that optimizes the enjoyability of playing against it.

Figure 11 shows a hypothetical progression of enjoyability (as a qualitative measure; the player is asked to rate his desire to play again after each match) over time (i.e., over the number of games played against a bot). As the Introversion bot is fairly predictable, we would expect the player to get bored of it after he/she learns how to defeat it quite quickly, indicated by a sharp drop of enjoyability. An alternative, more challenging bot might start with a lower enjoyability as it seems too hard to beat. However, the player's desire to play again should then rise as he/she gets better and learns how to win against the alternative bot. The raised difficulty is expected to delay the drop in enjoyability as it takes longer to consistently win against the bot. We indicate in Figure 11 the idea of a shelf event, that is, a user getting so upset (e.g., through frustration or boredom) with a game that he stops playing it and puts it on his shelf forever. This event can be associated with a very low desire to play again and is indicated as a line in the graph.

7.2. Player Survey. To approach the question of enjoyability versus ability in strategy games, we conducted a pilot survey with students, none of which had played DEFCON before. The sample size of 10 is too small to yield statistically significant results, but it provided valuable responses for further improvement of our AI-bot and helped us to remedy inaccuracies in the test protocol. The test was carried out as a blind test, that is, half the subjects played against the original AI-bot and the other half against our AI-bot. All other game parameters such as starting territories and game mode were identical. After each of the 10 successive matches against their computer opponent, the players were asked to rate their enjoyment, frustration, difficulty, desire to play again and confidence of winning the next match. The results are portrayed in Figure 12.

The results indicate that the novices were overburdened with the increased strength of the new AI-bot, as they won 39% of the games against our AI-bot, while the test group

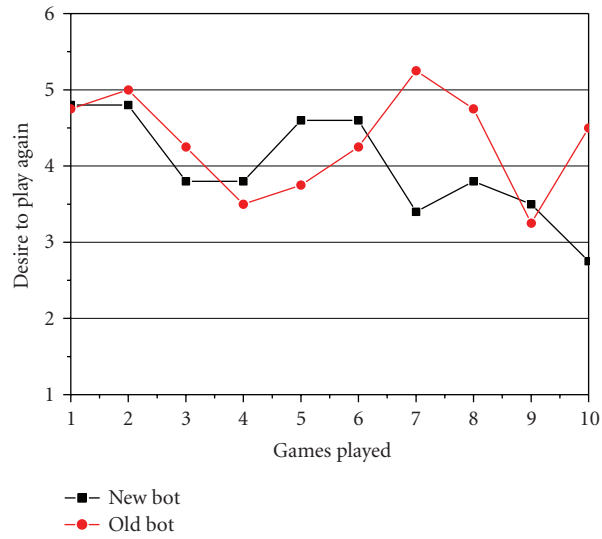

Figure 13: Comparison of survey results on the *desire to play again* (range from 1 = very low to 6 = very high) after each played game. Introversions bot is shown in red, while the new bot is shown in black.

won 56% of the games against the original DEFCON bot. This was reflected in the answers of the questionnaire, where the people playing the original bot were less frustrated (Figure 14) and more confident of winning (Figure 15), which resulted in an often higher desire to play again; see Figure 13. Also, we found that the choice of the starting configuration had a very strong impact on the perceived difficulty of the bots. For instance, in the test games, no player won as *Europe* against the AI-bot playing *South America*. On the other hand, the bots always lost playing *North Asia* against *Europe*. Therefore, great care has to be taken when choosing starting configurations for subsequent surveys.

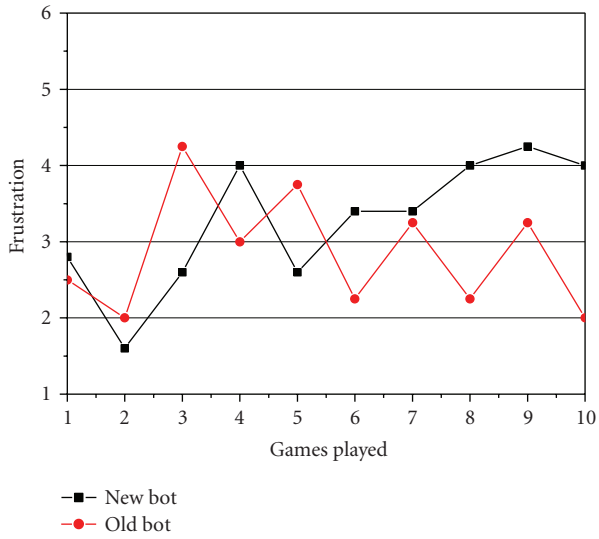

Figure 14: Comparison of survey results on the *frustration* (range from 1 = very low to 6 = very high) after each played game. Introversions bot is shown in red, while the new bot is shown in black.

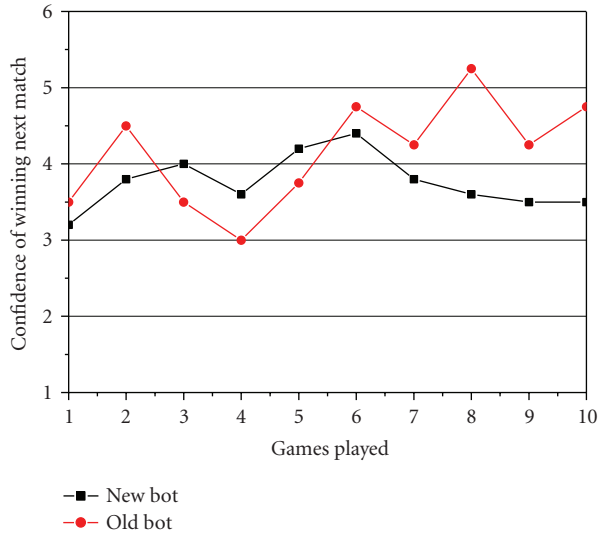

Figure 15: Comparison of survey results on the *confidence of winning the next match* (range from 1 = very low to 6 = very high) after each played game. Introversions bot is shown in red, while the new bot is shown in black.

Regarding our initial hypothesis from Section 7.1, we cannot draw any conclusive results yet, as the sample size and the time-span (i.e., number of games played in the survey) were too small to establish trends. This requires further testing and a survey with a bigger sample size, as described in Section 9. However, our initial survey does indicate that—at least for DEFCON—enjoyability and ability are not correlated in a simple positive manner, which is an interesting finding.

8. Related Work

The use of case-based reasoning, planning, and other AI techniques for board games is too extensive to cover, hence it is beyond the scope of this paper. Randall et al. have used DEFCON as a test-bed for AI techniques, in particular learning ship fleet formations [4]. Case-based reasoning has been used in [5] for plan recognition in order to predict a player's actions when playing the Space Invaders game. They used a simplified planning language which did not need preconditions and applied plan recognition to abstract state-action pairs. Using a plan library derived from games played by others, they achieved good predictive accuracy. In the real-time strategy game Wargus, a dynamic scripting approach was shown to outperform hand-crafted plans, as described in [6]. Moreover, hierarchical planning was tested in an Unreal Tournament environment by [7], who showed that this method had a clear advantage over finite state machines.

A comparison of artificial neural networks and evolutionary algorithms for optimally controlling a motocross bike in a video game was investigated in [8]. Both methods were used to create riders which were compared with regard to their speed and originality. They found that the neural network found a faster solution but required hand crafted training data, while the evolutionary solution was slower, but found solutions that had not been found by humans previously. In commercial games, scripting and/or reactive behaviors have in general been sufficient to simulate planning, as full planning can be computationally expensive. However, the Dark Reign game from Activision uses a form of finite state machines that involves planning [9], and the first-person shooter game F.E.A.R employs goal-oriented action planning [10].

8.1. Other Applications. Although the developed bot is in itself already an application of the used techniques, the underlying concept of combining artificial intelligence methods to benefit from synergy effects is applicable to many problems, including, but not restricted to, other strategy computer games that have similar requirements of optimizing and planning actions to be able to compete with skilled humans.

In particular, the combination of case-bases and decision trees to retrieve, generalize, and generate plans is a promising approach that is applicable to a wide range of problems that exhibit the following properties.

(i) *Discrete attributes.* The problem state space must be discrete or discretizable. This is required for decision tree algorithms to build trees. Attributes with a low cardinality are preferable, as a high number of possible values can cause problems with the decision tree learning algorithm.

(ii) *Recognizable opponent states.* Problem instances must be comparable through a similarity measure, which is required for retrieving cases. In a game domain, it should be based on opponent attributes or behavior to allow an adaptation to take place.

(iii) *Static problem domain.* The interpretation of a plan has to be constant, or else the similarity measure might retrieve irrelevant cases that show similarity to an obsolete interpretation. This also means that, for a hierarchical planner, lower-level plans should not change much when reasoning on high-level plans, as the case-base is biased towards previously successful plans.

(iv) *Availability of training sets.* The problem has to be repeatable or past instances of problem-solution pairs have to be available to train the case-base.

8.2. Future Work. There are many ways in which we can further improve the performance of our AI-bot. In particular, we aim to lessen the impact of over-fitting when learning plans, by implementing different decision tree learning skills, filling in missing plan details in nonrandom ways, and by trying other logic-based machine learning methods, such as Inductive Logic Programming [11]. We also hope to identify some markers for success during a game, in order to apply techniques based on reinforcement learning. There are also a number of improvements we intend to make to the control of low-level actions, such as more sophisticated detours that planes make to synchronize attacks.

With improved skills to both beat and engage players, the question of how to enable the AI-bot to play in a multiplayer environment can be addressed. This represents a significant challenge, as our AI-bot will need to collaborate with other players by forming alliances, which will require opponent modelling techniques. We aim to use DEFCON and similar video games to test various combinations of AI techniques, as we believe that integrating reasoning methods has great potential for building intelligent systems. To support this goal, we are developing an open AI interface for DEFCON, which is available online at http://www.introversion.co.uk/defcon/bots/.

The initial results of the survey and the discussion of ability versus enjoyability raise another important point for the future direction of our research. Usually it is a practice in academic AI research to strive for an algorithm that plays at maximum strength, that is, it tries to win at all costs. This is apparent in the application of AI techniques to playing board games. Chess playing programs, for example, usually try to optimize their probabilities of winning. However, this behavior may be undesirable for opponents in modern video games. It is not the goal of the game to make the player lose as often as possible, but to make him/her enjoy the game. This may involve opponents that act nonoptimally, fall for traps, and make believable mistakes. This behavior is another aspect we hope to improve in our bot in the future. It also suggests further player studies, as it is imperative to evaluate the enjoyability and believability of a bot through player feedback.

9. Conclusion

We have implemented an AI-bot to play the commercial game DEFCON, and showed that it outperforms the existing automated player. In addition to fine-grained control over game actions, including the synchronization of attacks, intelligent assignment of targets via a simulated annealing search, and the use of influence maps, our AI-bot uses plans to determine large-scale fleet movements. It uses a case-base of randomly-planned previously played games to find similar games, some of which ended in success while others ended in failure. It then identifies the factors which best separate good and bad games by building a decision tree using ID3. The plan for the current game is then derived using a fitness-proportionate traversal of the decision tree to find a branch which acts as a partial plan, and the missing parts are filled in randomly. To carry out the fleet movements, ships are guided by a central mechanism, but this is superceded by a movement-desire model if a threat to the fleet is detected.

References

[1] P. Sweetser, "Environmental awareness in game agents," in *AI Game Programming Wisdom 3*, S. Rabin, Ed., vol. 3, Charles River Media, Boston, Mass, USA, 2006.

[2] P. Tozour, "Influence mapping," in *Game Programming Gems*, vol. 2, Charles River Media, Boston, Mass, USA, 2001.

[3] T. Mitchell, *Machine Learning*, McGraw-Hill, New York, NY, USA, 1997.

[4] M. R. Wilhelm and T. L. Ward, "Solving quadratic assignment problems by 'simulated annealing'," *IIE Transactions*, vol. 19, no. 1, pp. 107–119, 1987.

[5] T. W. G. Randall, P. I. Cowling, and R. J. S. Baker, "Learning ship combat strategies in the commercial video game DEFCON," in *Proceedings of the 8th Informatics Workshop for Research Students*, Bradford, UK, June 2007.

[6] M. Pagan and P. Cunningham, "Case-based plan recognition in computer games," in *Proceedings of the 5th International Conference on Case-Based Reasoning (ICCBR '03)*, vol. 2689 of *Lecture Notes in Artificial Intelligence*, pp. 161–170, Trondheim, Norway, June 2003.

[7] P. H. M. Spronck, *Adaptive Game AI*, SIKS Dissertation Series, Universitaire Pers Maastricht, Maastricht, The Netherlands, 2005.

[8] H. Hoang, S. Lee-Urban, and H. Muñoz-Avila, "Hierarchical plan representations for encoding strategic game AI," in *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '05)*, AAAI Press, Marina del Rey, Calif, USA, June 2005.

[9] B. Chaperot and C. Fyfe, "Motocross and artificial neural networks," in *Proceedings of the 3rd Annual International Conference in Game Design and Technology (GDTW '05)*, Liverpool, UK, November 2005.

[10] I. Davis, "Strategies for strategy game AI," in *Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Computer Games*, pp. 24–27, Palo Alto, Calif, USA, March 1999.

[11] J. Orkin, "Agent architecture considerations for real-time planning in games," in *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '05)*, AAAI Press, Marina del Rey, Calif, USA, June 2005.

[12] S. Muggleton, "Inductive logic programming," *New Generation Computing*, vol. 8, no. 4, pp. 295–318, 1991.