A Selective Move Generator for the Game Axis and Allies

David L. St-Pierre, Mark H.M. Winands and David A. Watt

***Abstract*— We consider the move generation in a modern board game where the set of all the possible moves is too large to be generated. The idea is to provide a set of simple abstract tactics that would generate enough combinations to provide strong opposition. The reduced search space is then traversed using the $\alpha\beta$ search. We also propose a technique that allows us to remove the stochasticity from the search space. The model was tested in a game called Axis and Allies: a modern, turn-based, perfect information, non-deterministic, strategy board game. We first show that a tree search technique based on a restrained set of moves can beat the actual scripted AI engine — E.Z. FODDER. We can conclude from the experiments that searching deeper generates complex maneuvers which in turn significantly increase the likelihood of victory.**

I. INTRODUCTION

Typically, modern board games offer challenges in the field of AI due to their high complexity. Such games often involve several phases during a turn which have non-deterministic (stochastic) properties or imperfect information.

Traditionally, finding the best move based on search can be generalized as: (1) traversal of the search space; (2) evaluation of the board position; (3) move generation [1]. The majority of the research focuses on the traversal of the search space. For instance, pruning algorithms are a common approach to improve the traversal [2], [3]. From classic $\alpha\beta$ [2] to forward pruning [4], these methods seek to increase the efficiency of the search space traversal.

The second research branch tackles problems regarding the evaluation function; from the elaboration of different features, to the trade-off between speed and the amount of knowledge [5], [6], [7]. An entirely different form of evaluation is required when there is no static evaluation function that can grasp the intricacies of the board, such as the game of Go [8]. Monte-Carlo evaluations are then used [9]; a number of random games are played and their outcome is used as evaluation score.

The third type of research deals with the challenge of a large number of discrete possible moves that renders the search space to be almost continuous. There are neither classic tree search [10] nor stochastic tree search methods [11], [12] that can efficiently traverse a continuous search space [13]. Researchers applied the idea of making plans, as it was done in *Chess* [14], [15], [16] and more recently in *Arimaa* [17]. The idea of plans to narrow down the number of possibilities will be used in the move generator of a modern board game, our test bed, namely Axis and Allies: The number of moves considered is first reduced, then a search technique is applied to play strongly.

David Lupien St-Pierre and Mark H.M. Winands are with the Games and AI group, Department of Knowledge Engineering, Faculty of Humanities and Sciences,Maastricht University, The Netherlands (e-mail: {david.st-pierre, m.winands}@maastrichtuniversity.nl). David A. Watt is member of the Department of Computing Science, Faculty of Information and Mathematical Sciences, Glasgow, Scotland (e-mail: daw@dcs.gla.ac.uk).

This paper focuses specifically on move generation in the context of a set of atomic moves which cannot be efficiently generated or traversed. The goal is to develop a model that consists of a set of tactics. Once combined together, these tactics generate more complex maneuvers. Moreover, the model includes a technique that allows the removal of the chance nodes during the combat resolution.

The outline of the paper is as follows. First, Section II presents a brief description of the game Axis and Allies. Second, Section III discusses its search space. Subsequently, Section IV covers related work. Next, Section V describes the model. Section VI explains the experiments and gives the results. Finally, Section VII presents the conclusion and Section VIII gives an outlook on future research.

II. DESCRIPTION OF THE GAME

Axis and Allies (*AAA*) is a 50 year old *WWII* game, initially invented by Larry Harris Jr. It is a modern multiplayer board game with a high level of complexity, which can be described as a turn-based, perfect information, non-deterministic, strategy game that provides several challenges for an AI engine. In terms of complexity, Axis and Allies can be roughly ranked between the game of Risk and Civilization.

Figure 1 shows a part of the standard board. According to Bell's games categorization [18], this game falls into the wargame genre.

Over time, five versions of the main game have been released, the most recent in 2008. Originally published by Milton Bradley, it is now owned by Hasbro under Avalon

Fig. 1. Part of the standard board

Hill: Wizard of the Coast. Due to a high degree of popularity among the fan community, an Java-based open-source release — TRIPLEA — has been created and enables the design of various scenarios [19]. This environment is used as test bed in this paper.

There are three ways to win the standard game: total, major or minor victories. The most popular one is the total victory. The purpose of the total victory is to secure all cities, but most of the time it scales down to capture every opponent's capital. Minor and major represent a certain number of cities one must control to secure a victory. The last two victory conditions are popular in tournaments because it shortens the length of the game. Although there are multiple ways to win, an evaluation function that grasps the actual state of the game is possible [20].

AAA bears both similarities and differences to Risk [21], [22]. Structurally and economically, AAA deploys some similar traits. Every province has an economic value; a player controls provinces and the sum of their respective values equals his income per turn; players take turns to battle for the control of provinces. However, AAA is much more complex in several ways; specifically, it has more phases, more decisions to take per phase and more combinations of possible moves.

Three types of province exist: (1) land, (2) water and (3) inaccessible. Land and water provinces are only accessible to certain classes of units whereas the inaccessible type cannot be crossed by any unit.

Each unit has four characteristics: (1) cost, (2) attack points, (3) defense points and (4) movement points. The cost is the amount of money one must spend in order to buy the unit. The attack and defense points are within a range between 1 and 5 and used during the resolution of a combat. Basically, for every unit involved in combat a die is rolled and there is a hit if the result is equal or lower than the specification of the units. Obviously, the attacker uses the attack point and the defender uses the defense point. The movement point is the number of provinces a unit can cross during 1 turn.

Figure 2 summarizes the specification for the 12 units in the game. Unit cost is displayed to the right of each icon; below, each value triplet denotes $attack/defense/move$ attributes. For instance, the infantry unit has a cost of 3, an attack of 1, a defense of 2 and a move of 1.

Three classes of unit exist: (1) land, (2) aerial and (3) water. There are five land units: infantry, artillery, tank, anti-aerial gun and factory. These units must remain on land unless they are carried by a specific water unit. The infantry unit is the cheapest to buy. It has a weak attack and a moderate defense. It can only move to an adjacent province during a turn. The artillery unit is similar in terms of specifications. Moreover, if an infantry unit is supported by an artillery unit during an attack, the attack of the former increases to 2. The tank constitutes the third land unit. It is a well rounded unit. The anti-aerial gun does not possess attack or defense points. Every time an enemy aerial unit crosses a

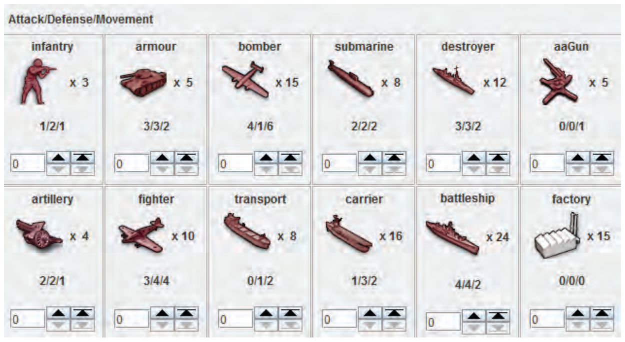


Fig. 2. Units specification

province with an anti-aerial gun, the player can throw a die and if the die equals 1 it kills the aerial unit. The factory allows the possibility to spawn units in the province.

There are two aerial units: plane and bomber. These units can move through both land and water provinces, but always have to land on a province that was in the possession of the player at the beginning of the turn. The plane can move across 4 provinces. It is a powerful unit in defense with the ability to reach strategic positions rapidly. The bomber unit is expensive, but is the ultimate attack unit on land provinces. It can perform two types of attack: normal or economic. Normal is the same type of attack as for the other units whereas economic attack is the ability to bomb the enemy factory. In doing so, the opponent loses money based on the result of the roll of a die.

There are five water units: submarine, destroyer, transport, carrier and battleship. Every unit in this class can move across two provinces. Naturally, they have to remain on water provinces. When a submarine attacks an opposing fleet that does not have a destroyer, it can cause casualties before the start of the combat by attacking first. The destroyer prevents the submarine from launching their devastating surprise attack. The carrier has the ability to carry up to two planes. The battleship is the ultimate water unit. It needs to be hit twice before it is destroyed.

A turn is split into five phases. The first phase consists of gambling an investment into *Research and Development*. It consists of using dice to decide whether there is a technological advance. A technological advance improves the specification of a unit.

During the second phase, the player uses his remaining money to buy units. This phase is rather straightforward. There are two constraints; the amount of money a player possess and the maximum number of units one can place on the board per turn. It is only possible to place new units on a province where there is an industrial complex — factory. The number of units that can be placed on a province is bounded by the value of the province. The units are bought at this phase, but are only placed on the board during the final phase, at the end of a turn.

The third phase is called the 'combat move'. This differs to a great extent from Risk. All the attacks must be declared before any of them is resolved. Every attack has a probabilistic outcome. Therefore, the crucial point here is to send enough units to achieve a goal, but also to divide them in such way that as many goals as possible are achieved. After the combat moves are declared, there is the combat resolution. At this point, there is a decision to take after every dice roll: retreat or not. Only the attacker can retreat, the defender cannot. If the situation becomes clearly disadvantageous, it is better to retreat than lose units of high economical value.

Experienced human players often use a strategy called *strafing*. This strategy allows the attacker to use the most powerful units in attack and to retreat after all the cheap units are dead to inflict a maximum damage to the opponent without losing any valuable units.

The fourth phase, 'non combat move', follows the 'combat resolution'. In this phase, all units that were not directly involved in a combat can be moved and all aerial units used in combat must land. This phase is separated from the combat move to bring fresh forces to the front and to allow the possibility that, if a combat resolution has gone tremendously wrong, at least a defensive move can be attempted to protect the front.

The fifth and final phase is called deployment — reinforcement. In this phase, the units bought at the beginning of the turn can be placed. This is to simulate the time to build them. It has one main characteristic different from Risk. At the beginning of the turn, a player cannot use his newly bought units to launch a surprise attack like in Risk.

III. Branching Factor

The search space for *AAA* is large. To compute the number of possible moves, the branching factor, one must multiply every possible combinations of every type of unit for every province. Equation 1 gives the formula to compute such a number.

$$\prod_{k\in K}\prod_{p\in P}\frac{(r_p-1+u_{kp})!}{(r_p-1)!u_{kp}!} \qquad (1)$$

Where K is the number of unit types, P is the number of provinces on a map, u_{kp} is the number of units of type k on the province p and r_p is the number of provinces the unit u_{kp} can reach, including the starting province.

Equation 1 computes the number of all possible moves, but identical boards from different interactions still have to be removed. For instance, if two adjacent provinces send one infantry unit to each other, the resulting board is the same. Equation 2 only shows the formula to remove the interaction between a pair of provinces. It gets increasingly complex to remove identical boards with interactions for triplets or more and the magnitude of possible moves does not change significantly. Therefore, only the interactions from pairs of provinces is shown.

$$\sum_{p_x=0}^{P_x}\sum_{i=1}^{u_{kp}}B\prod_{k\in K}\prod_{p_y\in P_y}\frac{(r_{p_xp_y}-2+u_{kp_xp_y}-i)!}{(r_{p_xp_y}-2)!(u_{kp_xp_y}-i)!} \qquad (2)$$

Where P_x is the set of all pair of adjacent provinces and P_y is every specific province within the pair. B is the rest of the board given by Equation 1, where P is replaced with $P-P_y$.

On a map such as Figure 3, Equation 1 gives a number $\approx$ 2.6 $\times$ 10^{16}. When identical boards from the pairwise interaction are removed using Equation 2, the number of possible moves drops to $\approx 9.3\times10^{15}$. It can be compared with the game Diplomacy which has $\approx 2.2\times10^{15}$ unique opening moves [23]. The branching factor for *AAA* is thus very large. Even worse, this is only an estimation for the initial board. As the game progresses, the number of units changes which means the number of moves can become much larger. Also, this is only the number of possible moves for the board, it neither includes the number of possible combinations during the buying phase nor the stochastic outcome of a battle.

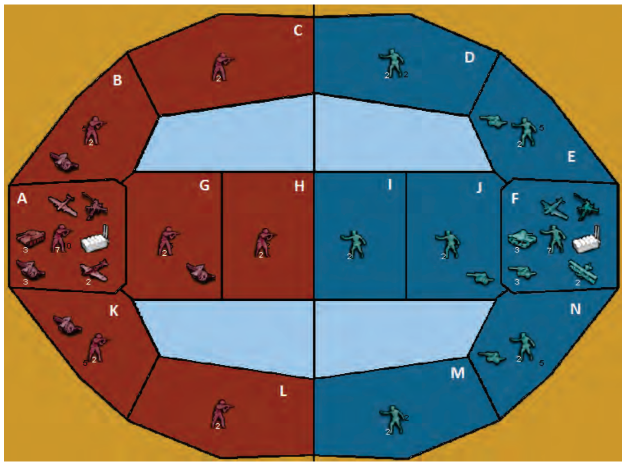


Fig. 3. Minimap

IV. Related Work

AAA is a complex game. There are multiple decisions to make at the five phases. The only AI engine available[1] — called E.Z. FODDER [19] and written by Sean Bridges — is heavily scripted and only provides an interesting challenge for beginners. It plays weakly against more experienced opponents partly because it behaves like a greedy player. For *AAA*, simple brute-force algorithms such as minimax [10] and its $\alpha\beta$ pruning enhancement [2] are not computationally feasible.

Shannon [24] first formalized the idea of a selective move generator, later known as 'forward pruning'; however, this was not further developed due to unconvincing results for a few decades. Forward pruning is attractive, yet it is known to be error prone and dangerous. The selective move generator proposed in this paper is entirely distinct from Shannon's forward pruning. Instead of pruning moves that are not promising, it deliberately avoids the generation of quasi-similar moves.

[1]The TRIPLEA community is currently developing a second engine that will be released soon.

This paper proposes the creation of a set of atomic 'meta moves' or 'tactics' that would generate complex maneuvers in a search based approach. It is an alternative to providing a smaller set of possible moves from domain-specific expert knowledge as it was done in the game *Settlers of Catan* [25] and *Amazons* [26]. A similar concept of our idea has recently been tested in *WARGUS* [27], a Real-Time Strategy game, and in the game of *Kriegspiel* [28].

The purpose of this paper is to work on move generation, not on the search technique. Therefore, $\alpha\beta$ was preferred because the strengths and weaknesses of this algorithm are well tested and known [29]. We think that our idea of move generation can be transferred to other search algorithms such as Monte-Carlo Tree Search [11], [12], [30], [31].

V. Model: dlsAI

In this section we propose a model — *dlsAI* — to play *AAA*. Subsection V-A develops the notion of core tactics in dlsAI. In Subsection V-B, a modification in the combat resolution is presented to abstract a stochastic search to a deterministic one. This modification aims to further reduce the search space. In Subsection V-C, the evaluation function is described.

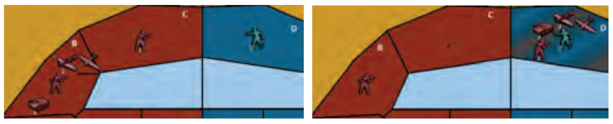
A. Tactics

To reduce the number of moves, the model separates the provinces into three abstract types: (1) front line, (2) reinforcement and (3) supply. A province can be more than one type. The front line provinces juxtapose the opposing armies. For these provinces, tactical decisions regarding combats have to be taken. The reinforcement provinces contain a factory. Specific choices have to be made on where and how to send units to the front. The last type of provinces — supply — are provinces that have units on them but cannot participate in any combat and are not a reinforcement province. They simply send units toward the closest front line.

To elaborate a functional AI, four basic tactics were coded into the move generator. The first one is the attack, which means all units move from a front province to attack the adjacent enemy province. There can be more than one enemy province neighboring the front province, but the selection of which one is the best is beyond the scope of this paper.

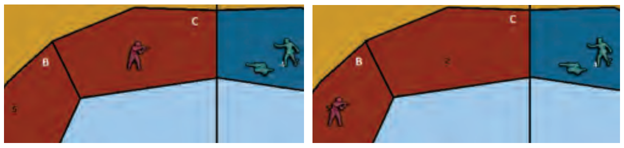
Figure 4 shows the attack tactic used by the left player where province 'B' is a supply province and province 'C' is a front line province. Subfigure 4(a) represents the board before the left player launches an attack. Subfigure 4(b) illustrates 2 infantries from the province 'C' and 2 fighters, 1 bomber and 3 tank units from the province 'B' that are deployed over province 'D'. The infantry on the province 'B' did not move because it cannot participate in the combat.

The second tactic is the retreat. It consists in moving away from the enemy units. Figure 5 shows the retreat tactic used by the left player where province 'B' is a supply province and province 'C' is a front line province. Subfigure 5(a) represents the board before the retreat move. Subfigure 5(b) illustrates the retreat of 1 infantry from province 'C' to province 'B'.



(a) Start (b) Attack move

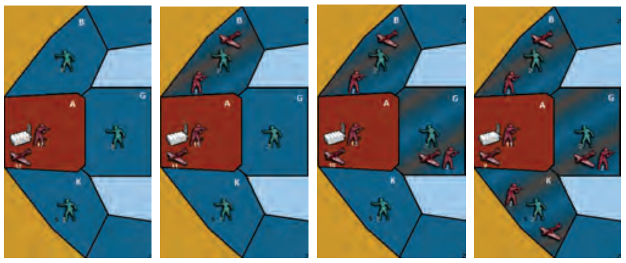
Fig. 4. Attack tactic



(a) Start (b) Retreat

Fig. 5. Retreat tactic

The third one is a split-attack tactic. This tactic only sends enough units to take a province, but not necessarily to hold it. Figure 6 shows the split-attack tactic used by the left player where province 'A' is both front line and reinforcement province. This sequence is separated into 4 different phases. Subfigure 6(a) represents the board before the left player executes the split-attack tactic. Subfigure 6(b) illustrates a part of the left player's army sent to province 'B' — 2 infantries and 1 plane. Subfigure 6(c) presents another part of the left player's army sent to province 'G' — 2 infantries and 1 plane. Subfigure 6(d) shows another part of the left player's army sent to province 'K' — 2 infantries and 1 plane.



(a) Start (b) Split 1 (c) Split 2 (d) Split 3

Fig. 6. Split-attack tactic

The fourth tactic is the pass action. Nothing moves on the selected province.

From these four tactics on the front line, different combinations become possible depending on the number of front lines. dlsAI is able to plan a joint attack where stacks of units from different front lines can combine their forces on a specific province. It also allows the planning of maneuvers such as sending a small force from two or more adjacent stacks to take one or multiple strategic provinces. From depth three and onwards, dlsAI is able to create different traps by strategically retreating a part of its force to entice the opponent.

As mentioned in Section II, units can only be put on a province where there is a factory. From this province, they must be sent to one of the front lines. To keep the number of moves generated as small as possible, a simplification was introduced in the model. The board possesses different fronts; while on supply provinces the units are not included in the move generation unless they can participate in one of the tactics.

Another decision is the way the units are split over different fronts. There are three tactics considered at this phase for the reinforcement provinces. dlsAI can either choose to send every unit to a specific front, split his forces along different fronts or not reinforce any of them. The method that determines the number of front lines is not dependent upon the board state. It is hard coded for a specific map and does not take into consideration the situation of the board.

The number of possible reinforcements moves is given by $reinforcingAFront \times numberOfFronts + SplitReinforcement + NoMove$. Considering Figure 3, there are 5 — $1 \times 3 + 1 + 1$ — possible reinforcement moves. The number of possible moves is simply given by $tactics^{provinces} \times reinforcementMoves$. For a map such as Figure 3 — where there are three fronts and one reinforcement province — there are 320 possible moves — $4^3 \times 5$ — instead of $\approx 9.3 \times 10^{15}$. The reduction is substantial yet the idea is that with those few core tactics, it should be enough to play relatively strong when a sufficient search depth is reached.

B. Combat Resolution

A major simplification can be executed for the combat resolution. Once all invasions are announced, the probabilistic resolution begins using dice. For non-deterministic outcomes, Expectimax [32] is the algorithm of choice. It extends the minimax concept to stochastic outcomes, by adding chance nodes to the search tree. The value of chance node is the weighted sum of the values of the chance events [1]. For the combat resolution phase, it means to weight every possible outcome by their respective probability of occurrence. Different pruning methods were developed such as safe pruning (e.g. Star1 and Star2 [33], [34]) and speculative pruning (e.g. ChanceProbCut [3]) to increase the search performance.

However, keeping track of all possible outcomes from a battle involving 20 units for instance would shatter the reduction presented in the previous subsection. To avoid this drawback in terms of performance, some straightforward substitutions are available to remove the probability in the outcome. One consists of evaluating the amount of damage that a stack of units is likely to inflict. For instance, consider 2 tank units that execute an attack. In the case of an attack, one must throw a die for every unit and the result must be lower or equal to the attack point to have a hit. Tank units have 3 attack points. Hence, the probability that 1 tank will hit is 50%.

Instead of representing the odds as a percentage, dlsAI assumes that it is real damage. Therefore, a tank that attacks does not have 50% chance to make a kill, but causes 0.5 damage. Obviously, every time the damage equals 1 there is one unit removed on the opposite side.

In the following example, if 2 tank units execute an attack (2 dice that hit with 3 or less each), they are likely to make 1 kill on the first turn (50%) and the remaining odds are 25% chance to make 2 kills and 25% chance to make 0 kill. DlsAI considers the attack of 2 tank units as one kill on the first round. This transformation can be seen as if every unit had a hit point. At every iteration during a combat — when the dice are rolled — the respective defense or attack points are deduced from the opponent's hit points. The total number of hit points, attack points and defense points are updated and the process repeats until one side reaches 0 or the attacker retreats. Basically, dlsAI computes the mean and does not include the likelihood. For a stack of units that has 26 points of attack, the outcome should be 4 casualties ($\frac{26}{6}$). The rule for units removal, albeit not optimal, is efficient and simple: The cheapest units are removed first. Figure 7 presents the pseudocode for resolving combat.

```
WHILE attackingUnits $>$ 0 OR defendingUnits $>$ 0
        FOR every attackingUnit
            attackHitPoints+=1;
            attackPoints+= attack value of attackingUnit;
        END FOR
        FOR every defendingUnit
            defendHitPoints+=1;
            defendPoints+= defence value of defendingUnit;
        END FOR
        defendHitPoints-=(attackPoints$/$6);
        attackHitPoints-=(defendPoints$/$6);
        attackingUnits.update(attackHitPoints);
        defendingUnits.update(defendHitPoints);
END WHILE
```

Fig. 7. Pseudocode for combat resolution.

The situation where there is a remainder still has to be addressed. There are two possible solutions for this problem. The first one is to roll a die and look if the number is lower than the remainder. The advantage of this technique is that it allows the use of integers to represent units. The disadvantage is that if you have to simulate the same combat several times, the outcome is most of the time different.

For this reason, a second solution was chosen. It involves the use of doubles instead of integers. If the number of casualties is $4\frac{1}{3}$, then this is exactly the number of units that will be removed. In this scenario, it is possible that a $\frac{1}{2}$ tank unit attacks a $\frac{1}{3}$ infantry unit. The performance of dlsAI is slowed down by the use of doubles, but the non-determinism is factored out of the combat resolution.

C. Evaluation function

The aim of this subsection is to explain how the underlying strategies in *AAA* are represented by the evaluation function. It describes how dlsAI evaluates a board position by applying an evaluation function that uses three main features.

The first one is the economical value of its army compared to the opponent. Every unit has a value — cost — and dlsAI compares the sum of the existing units. It gives an indication on the current board situation of the standing armies and their relative weights.

The second feature is the economic situation. Every province has a value and is under control of one player. The sum of the values of all the provinces that belongs to a player gives its income per turn. dlsAI compares those two incomes. This measure is important to estimate the potential growth.

The third feature is the number of units dlsAI can place per turn compared to the opponent. A new unit can only be placed on a province where a factory is built and the number of units is given by the value of the province. A province with a factory has a high strategic importance because the absolute number of units one can place per turn compared to the opponent can change the purchase strategy.

Those three features give a rough idea of the current board situation. The evaluation function could be improved with more features, but the focus is on the move generation and this evaluation function allows dlsAI to create a set of strategies.

VI. Experiments

This section describes the experiments that were conducted. First, the test domain is defined and its properties are dicussed. Second, the results are presented. dlsAI was tested against E.Z. Fodder and against itself, playing at different depths. Each game used the total victory condition.

A. Test Domain

The model described earlier was tested on the map *mini-map*, depicted in Figure 3. This map was selected for multiple reasons. It has two players and it is balanced in a way that both players start with the same board situation. It uses 7 unit types instead of the 12 standard ones. Basically, there are no water units in this version. There are only 14 provinces. However, as mentioned in Section III this game still has a large branching factor.

Even though both players have a symmetric start, the first player has a significantly higher chance of winning.

B. Results

DlsAI finds the best move in less than a second at depth 1, in more or less 10 second at depth 2 and approximately 3 minutes at depth 3. The first series of experiments compare the performance of dlsAI at a depth of 3 against E.Z. Fodder. The results are shown in Table I. 'Num of games' represents the number of games played. Victory is expressed as a percentage. 'Avg lgt(turn)' means the average length expressed in turns. Similarly, 'Max(turn)' and 'Min(turn)' are the longest and shortest length of a game, respectively expressed in turns, before victory. The suffix following dlsAI represents the depth at which it plays for the specific experiment.

Table I shows that when dlsAI plays first, after 100 games, it has maintained a score of 99 victories. When it plays second, it realizes 98 victories out of 100. These results demonstrate that on this specific map, dlsAI:D3 is better than E.Z. Fodder.

TABLE I
DLSAI:D3 VS E.Z. FODDER

	dlsAI:D3 (First player)	dlsAI:D3 (Second Player)
Num of games	100	100
Victory(%)	99	98
Avg lgt(turn)	15.21	16.77
Max(turn)	22	23
Min(turn)	9	8

The average length is similar whether dlsAI:D3 plays first — 15.21 — or second — 16.77. The Max(turn) and Min(turn) indicators shows that in a hundred games, the length did not vary much.

The defeats were scrutinized to understand what happened and in the three cases, the same pattern was found. In fact, dlsAI:D3 took the capital of the enemy and was clearly winning. However, from this point on, it does not protect adequately its own capital because the evaluation function is returning a certain win. The opponent, having a stack of units closer to the capital of dlsAI:D3, continues its attack and takes the capital. From there, dlsAI:D3 is helpless and it ends in a defeat. Clearly, this is a weakness in the evaluation function.

The main interest in this experiment was to evaluate if dlsAI:D3 could execute more complex maneuvers than the four core tactics. At depth 2 and more, it retreats when facing a stronger army. However, at depth 3, it will sometimes retreat even when there is an equilibrium in strength to lure the enemy out. This highly efficient maneuver can be described as a trap. Other more subtle maneuvers are used more often such as a joint attack or a combined split attack.

In the next series of experiments we evaluate whether the search depth matters for dlsAI. Table II shows the result of dlsAI at depth 2 vs dlsAI at depth 1. In this case, dlsAI:D1 is a greedy player. It does not use the retreat tactics because it does not consider the possibility of being attacked.

TABLE II
DLSAI:D2 VS DLSAI:D1

	dlsAI:D2 (First player)	dlsAI:D2 (Second Player)
Num of games	100	100
Victory(%)	98	89
Avg lgt(turn)	17.41	16.37
Max(turn)	62	40
Min(turn)	5	8

When dlsAI:D2 plays first, 98% of the time it achieves victory compared to 89% when it plays second. The difference in winning percentage is significant at a confidence interval of 95%. This result allows us to infer that dlsAI:D2 is statistically better than dlsAI:D1.

There is also a significant difference in the winning percentage if dlsAI:D2 played first or second. It seems that the map favors the first player.

The average length of the game when dlsAI:D2 plays first — 17.41 — or second — 16.37 — is comparable to Table I. This is a relatively short game. The Max(turn) value indicates that even if dlsAI:D2 is losing for a good part of the game, a comeback against a weaker opponent is possible. Again, it is partly explained by the use of the retreat tactic. dlsAI:D2 keeps its strength and waits for an opportunity. The Min(turn) measure shows that dlsAI:D2 can win rapidly against a greedy player.

Table III presents a test with dlsAI at depth 3 versus dlsAI at depth 2. It provides some insight on the gain in strength per depth. The victory percentage, albeit less impressive than noted in Table II, nevertheless demonstrates the dominance of dlsAI:D3 over dlsAI:D2. When playing first, dlsAI:D3 achieves 81.64% of victory whereas when playing second the victory percentage drops to only 50%. Also, the length of the game is significantly longer than the precedent table.

TABLE III
DLSAI:D3 VS DLSAI:D2

	dlsAI:D3 (First player)	dlsAI:D3 (Second Player)
Num of games	50	50
Victory(%)	81.64	50
Avg lgt(turn)	40.12	31.05
Max(turn)	161	45
Min(turn)	13	22

Table IV represents the results of dlsAI playing each time against itself both at depth 3. The results show that the map is advantageous for the first player. A perfectly balanced map if the data was normally distributed would give 50% of victory for both side. A Student's test revealed that at a confidence interval of 95%, we may infer that there is an advantage for the first player. In other words, to play first gives a significant advantage.

TABLE IV
DLSAI:D3 VS DLSAI:D3

	dlsAI(First player)	dlsAI(Second Player)
Num of games	50	50
Victory(%)	78.85	21.15
Avg lgt(turn)	32.46	40.27
Max(turn)	75	75
Min(turn)	11	21

The average length of a game is much higher than in Table I and Table II. It does not directly translate into a stronger opponent, but it is a good hint in this direction. This value shows the impact of a strategic retreat. The 'Min turn' value again shows that the first player can expect to win quicker than the second.

Based on the experiments presented in Table II, III, and IV, Table V shows the impact of the outcome of a probabilistic combat resolution in the evaluation function. This value represents the highest score dlsAI allocated to a board position for where the game was lost. The maximum value the evaluation function can return is 10,000 and the cheapest unit (i.e. infantry) is worth 15 points. The impact of search depth and play order on the behavior of the evaluation function gives an important insight on its robustness.

TABLE V
IMPACT OF THE PROBABILISTIC ASPECT IN *AAA*

depth / Max(value)	First player	Second Player
D2 vs D1	467	152
D3 vs D2	1257	429
D3 vs D3	483	464

In a game that involves chance such as *AAA*, one battle can change the entire equilibrium. It is expected that dlsAI will make miscalculations from the removal of the chance nodes in the combat resolution. When dlsAI:D2 plays first against a greedy player, the result is around 500. The maximum error in the evaluation when it plays second is significantly low compared to the other values. In this case, it is simply because there were only two games that dlsAI:D2 managed to lose therefore preventing it from having a higher value.

When dlsAI:D3 plays against dlsAI:D2 as the first player, this value is surprisingly high — 1257. However, at depth 2, the fact is that dlsAI will use the retreat tactics therefore minimizing the mistakes. At depth 3, it is able to plan a trap but not to evaluate what will happen after. For instance, dlsAI:D3 sometimes launches an attack knowing that the rest of the stack will be taken at the next turn by the opponent. However, by a clever reinforcement, it will be able to take it back. The problem is that it does not evaluate the chance that the opponent can take it back again (depth 4), therefore resulting in a loss of units. This may explain such a large difference in Table V.

When dlsAI plays against itself at depth 3, the value is fairly consistent whichever side it plays. Also, it seems a good estimation to say once it gives a score over 500, it should be an almost certain win.

VII. CONCLUSION

In this paper, instead of considering the entire search space, we focused on a few core tactics that can create complex maneuvers depending on the depth of the search. $\alpha\beta$ search was applied in *AAA*. To further reduce the search space, we also proposed a technique to remove the stochasticity during the combat resolution phase.

We showed that our model — dlsAI — outperformed the actual scripted AI engine — E.Z. FODDER — on *minimap*. From the experiments presented in Table II and III, we can infer that deeper searches generated sufficient number of complex maneuvers which in turn increase the number of victories significantly.

In *AAA*, it is currently computationally too expensive to achieve a search depth of 4 for 2 players. It is expected that in order to play at a strong level, a tree-search based algorithm has to reach at least twice the number of players. Therefore, for the map *minimap*, as there are two players, the search depth must be four. For a 5-player map, such as the standard board, a 10-ply search would be needed.

VIII. Future Work

The first direction is to improve the evaluation function. Although dlsAI can win quickly, it still has some problems finishing a game. Sometimes, dlsAI could deal a critical blow but does not execute the maneuver because its own losses are larger than the opponent's. A good chess equivalent would be to sacrifice a queen for a bishop in the end game when you have an overwhelming piece advantage. The current evaluation function fails to grasp this dimension of the game. Furthermore, the defeats recorded against E.Z. Fodder indicate that either other features are needed, or the weights require fine tuning to improve the performance. Preliminary experiments were conducted to weight each feature; however, better weights might be found through machine-learning techniques.

The second direction is an implementation of a Monte-Carlo technique such as MCTS / UCT [11], [12], [30], [31]. It is expected that the notion of planning still has to be applied in order to make MCTS work because there are too many moves to be able to execute a trade-off between exploration and exploitation [30].

A third direction is to achieve drastic computational gain by finding a way to separate the board into subgames. This has been done in the game of Amazons using Temperature Discovery Search [35]. For the game AAA, the problem is mainly related to aerial units because they have such a long range of action. Those units can easily move from one front to another and therefore preventing the board to be separated into subgames.

A fourth direction is to use machine-learning technique as it was done in the game *Settlers of Catan* [36] to approach this problem.

Acknowledgment

The authors want to thank Maarten Schadd and Philippe Uyttendaele for their suggestions and the reviewers for their comments.

References

[1] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. New Jersey, USA: Pearson Education inc., 2003.

[2] D. E. Knuth and R. W. Moore, "An Analysis of Alpha-Beta Pruning," *Artificial Intelligence*, vol. 6, no. 4, pp. 293–326, 1975.

[3] M. P. D. Schadd, M. H. M. Winands, and J. W. H. M. Uiterwijk, "ChanceProbCut: Forward Pruning in Chance Nodes," in *IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, P. L. Lanzi, Ed., 2009, pp. 178–185.

[4] J. J. Smith and D. S. Nau, "An Analysis of Forward Pruning," in *AAAI-94*, 1994, pp. 138–1391.

[5] J. Schaeffer, "Experiments in Search and Knowledge," Ph.D. dissertation, Departement of Computing Science, University of Waterloo, Waterloo, Canada, 1986.

[6] H. Berliner, G. Goetsch, M. S. Campbell, and C. Ebeling, "Measuring the Performance Potential of Chess Programs," *Artificial Intelligence*, vol. 43, no. 1, pp. 7–20, 1990.

[7] A. Junghanns and J. Schaeffer, "Search versus Knowledge in Game-Playing Programs Revisited," in *IJCAI-97*, 1997, pp. 692–697.

[8] M. Müller, "Computer Go," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 145–179, 2002.

[9] B. Brügmann, "Monte Carlo Go," Physics Department, Syracuse University, Tech. Rep., 1993.

[10] J. von Neumann, "Zur Theorie der Gesellschaftsspiele," *Mathematische Annalen 100*, pp. 295–320, 1928, translated by Bargmann (1959).

[11] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *Proceedings of the EMCL 2006*, ser. LNCS, J. Fürnkranz, T. Scheffer, and M. Spiliopoulou, Eds., vol. 4212. Berlin: Springer-Verlag, Heidelberg, Germany, 2006, pp. 282–293.

[12] S. Gelly and Y. Wang, "Exploration exploitation in Go: UCT for Monte-Carlo Go," in *NIPS Workshop on On-line Trading of Exploration and Exploitation*, 2006.

[13] I. Millington, *Artificial Intelligence for Games*. San Francisco, CA: Morgan Kaufmann, 2006.

[14] J. Schaeffer, "Long-range planning in computer chess," in *ACM '83: Proceedings of the 1983 annual conference on Computers : Extending the human resource*. New York, NY, USA: ACM, 1983, pp. 170–179.

[15] D. Wilkins, "Using patterns and plans in chess," *Artificial intelligence*, vol. 14, no. 2, pp. 165–203, 1980.

[16] J. Pitrat, "A chess combination program which uses plans," *Artificial Intelligence*, vol. 8, no. 3, pp. 275–321, 1977.

[17] G. Trippen, "Plans, patterns, and move categories guiding a highly selective search," in *Advances in Computer Games (ACG 2009)*, ser. LNCS, H. J. van den Herik and P. Spronck, Eds., vol. 6048, 2010, pp. 111–122.

[18] R. C. Bell, *Board and Table Games from Many Civilizations*, 2nd ed. New York, USA: Dover Publications inc., 1950.

[19] TRIPLEA, 2010, http://triplea.sourceforge.net/mywiki.

[20] D. L. St-Pierre, "Mind Games: Axis and Allies," Master's thesis, Glasgow University, Glasgow, Scotland, 2009.

[21] S. J. Johansson, "On using Multi-Agent Systems in Playing Board Games," in *Fifth International Joint Conference on Autonomous Agents and MultiAgent Systems*, Hakodate, Japan, 2006, pp. 569–576.

[22] F. Olsson, "A Multi-Agent System for Playing the Board Game Risk," Master's thesis, Blekinge Institute of Technology, Karlskrona, Sweden, 2005.

[23] D. E. Loeb, "Challenge in Multi-player Gaming by Computers," *The Diplomatic PouchZine*, vol. S1995M, 1995.

[24] C. E. Shannon, "Programming a Computer for Playing Chess," *Philosophical Magazine*, vol. 41, no. 4, pp. 256–275, 1950.

[25] I. Szita, G. M. J.-B. Chaslot, and P. Spronck, "Monte-Carlo Tree Search in Settlers of Catan," in *Advances in Computer Games (ACG 2009)*, ser. LNCS, H. J. van den Herik and P. Spronck, Eds., vol. 6048, 2010, pp. 21–32.

[26] R. Lorentz, "Amazons Discover Monte-Carlo," in *Computers and Games (CG 2008)*, ser. LNCS, H. J. van den Herik, X. Xu, Z. Ma, and M. H. M. Winands, Eds., vol. 5131, 2008, pp. 13–24.

[27] R.-K. Balla and A. Fern, "UCT for Tactical Assault Planning in Real-Time Strategy Game," in *IJCAI-09*, 2009, pp. 40–45.

[28] P. Ciancarini and G. P. Favini, "Representing Kriegspiel States with Metapositions," in *IJCAI-07*, 2007, pp. 2450–2455.

[29] A. Junghanns, "Are there Practical Alternatives to Alpha-Beta in Computer Chess?" *ICCA*, vol. 21, no. 1, pp. 14–32, 1998.

[30] G. M. J.-B. Chaslot, M. H. M. Winands, J. W. H. M. Uiterwijk, H. J. van den Herik, and B. Bouzy, "Progressive Strategies for Monte-Carlo Tree Search," *New Mathematics and Natural Computation*, vol. 4, no. 3, pp. 343–357, 2008.

[31] R. Coulom, "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search," in *Computers and Games (CG 2006)*, ser. LNCS, H. J. van den Herik, P. Ciancarini, and H. H. L. M. Donkers, Eds., vol. 4630. Springer-Verlag, Heidelberg, Germany, 2007, pp. 72–83.

[32] D. Michie, "Game-Playing and Game-Learning Automata," in *Advances in Programming and Non-Numerical Computation*, L. Fox, Ed., Pergamon, New York, USA, 1966, pp. 183–200.

[33] B. W. Ballard, "The *-Minimax Search Procedure for Trees Containing Chance Nodes," *Artificial Intelligence*, vol. 21, no. 3, pp. 327–350, 1983.

[34] T. Hauk, M. Buro, and J. Schaeffer, "*-Minimax Performance in Backgammon," in *Computers and Games (CG 2004)*, ser. LNCS, H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu, Eds., vol. 3846. Springer-Verlag, 2006, pp. 51–66.

[35] M. Müller, M. Enzenberger, and J. Schaeffer, "Temperature Discovery Search," in *Nineteenth National Conference on Artificial Intelligence (AAAI 2004)*, San Jose, CA, 2004, pp. 658–663.

[36] M. Pfeiffer, "Reinforcement learning of strategies for Settlers of Catan," in *International Conference on Computer Games: Artificial Intelligence, Design and Education, Reading*, UK, 2004, pp. 384–388.