JAIST Repository
https://dspace.jaist.ac.jp/

Title	A Platform for Turn-Based Strategy Games, with a Comparison of Monte-Carlo Algorithms
Author(s)	Fujiki, Tsubasa; Ikeda, Kokolo; Viennot, Simon
Citation	2015 IEEE Conference on Computational Intelligence and Games (CIG): 407-414
Issue Date	2015
Type	Conference Paper
Text version	author
URL	http://hdl.handle.net/10119/12992
Rights	This is the author's version of the work. Copyright (C) 2015 IEEE. 2015 IEEE Conference on Computational Intelligence and Games (CIG), 2015, 407-414. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.
Description	

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

A Platform for Turn-Based Strategy Games, with a Comparison of Monte-Carlo Algorithms

Tsubasa FUJIKI
Japan Advanced Institute of Science and Technology
Ishikawa, Japan
Email: s1310062@jaist.ac.jp

Kokolo IKEDA
Japan Advanced Institute of Science and Technology
Ishikawa, Japan
Email: kokolo@jaist.ac.jp

Simon VIENNOT
Japan Advanced Institute of Science and Technology
Ishikawa, Japan
Email: sviennot@jaist.ac.jp

***Abstract*—A lot of research has been done on classical games such as Chess or Shogi, but not so much on more recent games such as turn-based strategy games, where the players can move multiple pieces at each turn. In this paper, we analyze the game components found in most strategy games, and propose a set of simple rules that could be used as a standard game for research on turn-based strategy games. We have implemented these rules in an open platform, and in the second part of the paper we compare four different Monte-Carlo search algorithms with this platform. Especially, we show the importance of distinguishing and handling differently tactical moves and attacking moves.**

I. Introduction

A lot of research in the artificial intelligence field has already been done on games such as Chess, Shogi or Go, leading to the development of many search algorithms and machine learning methods. Thanks to these algorithmic developments as well as hardware improvements, it was possible for DeepBlue, a program from IBM, to beat the world chess champion in 1997. In Computer Shogi, the programs also reached the professional level, with the breakthrough of the Bonanza method consisting in the machine learning of an evaluation function. This level of play is sufficient for almost all human players. Considering these successes, it is natural to consider now games more difficult than Chess or Shogi, where programs are still weaker than humans. In this paper, we consider strategy games where it is possible to move multiple pieces in one turn, and we propose a platform to facilitate future research on these games. Such Turn-Based Strategy (TBS) games, like the "Daisenryaku" series, have a lot of success, and many variants have been developed. However, because of the lack of a standard platform for doing benchmarks, research on this topic is limited, and the computer players of most commercial games are still weak compared to humans. In this paper, we start by presenting in Section II some well-known and representative games. Then, in Section III, we list the rules and design components found in most TBS games, and in Section IV we propose a set of rules that could be used as a benchmark representative of TBS games. We detail in Section V the search algorithms that we have tested on this platform, and finally we show their relative performance in Section VI.

II. Existing Turn-Based Strategy Games

Daisenryaku is a famous Turn-Based Strategy game series from System Soft, which achieved a commercial success, especially in Japan with more than 800,000 copies sold since the first title in 1985. As in Chess and Shogi, the players move in turn the pieces (called units) placed on a board (called the map). The originality of Daisenryaku is that all the units of a player can be moved in the same turn. Many games related to Daisenryaku have been developed through the years, so in this section, we list the different variants and their specificities. Then, we discuss which kind of rules should be retained for a research platform on TBS games.

A. Arimaa

Arimaa is a two-player board game similar to chess in many aspects, especially in the fact that it is a zero-sum perfect information game, but it has the particularity of authorizing the moves of up to 4 pieces at each turn [1]. Because of this rule, the number of legal moves is very high, which is a problem for a direct usage of the classical search algorithms. Arimaa can be used as a good benchmark for testing algorithms designed for games with multiple moves in a single turn [6]. However, the popularity of Arimaa is somewhat limited, which makes it difficult to obtain the game records of strong players or to evaluate the programs against humans.

B. Simulation Role Playing Games (SRPG)

Simulation Role Playing Game (SRPG) is a genre of games containing components from both strategic games and Role Playing Games (RPG). A representative title in the many games of this genre is Final Fantasy Tactics, with almost 2 millions copies sold in Japan alone. SRPG often contain many small elements specific to the game, which is somewhat an unneeded difficulty for research about algorithms.

C. Real Time Strategy Games

Real Time Strategy (RTS) games, like StarCraft or Age of Empires, are a relatively new genre of games, made possible by the improvement of the standard computer hardware. In RTS games, there is no concept of turn. It is replaced by a continuous time, and the players act freely at any time, simultaneously. There is also usually no concept of cell on the map where the pieces are moved. RTS games are particularly played in Europe and America, and a lot of research has already been done on these games. Especially, some open platforms are available for algorithm development and test [2] [3]. However, because of the real-time aspect and the usually

complex rules, RTS is a genre already quite far from Chess and Shogi.

III. Design Components and Cluster Analysis of TBS Games

Many different specific rules are found in Turn-Based Strategy (TBS) games, but not all of them have the same degree of importance. In this section, we start by listing the different rules that we have observed in most TBS games. Then, we try to group the different games in categories.

A. Design Components of TBS Games

We list here the concepts and components found in most standard TBS games. The originality of a TBS game is often determined by the components that are used or not in the game. We have ordered the components roughly in function of their proximity or not to the game of Chess.

F1 Map
: As in Chess and Shogi, a board map with square cells is the most common, but sometimes hexagonal cells or no cell at all are used.

F2 Number of players
: Not always limited to 2 players.

F3 Pieces (units) characteristic
: Many different units are used. In SRPG, one unit corresponds usually to one character, with its own characteristics.

F4 Order of moves
: One unit in turn, several units in turn, all units in turn, or real-time.

F5 Victory condition
: Destruction of all opponent pieces, capture of a specific piece, control of a specific cell...

F6 Piece relative strength
: As in the Rock-Paper-Scissor game, pieces are often strong or weak depending on the opponent piece.

F7 Hit Points (HP)
: Number representing the remaining power of the piece, with the piece destruction when it reaches 0. This concept is important and needed for many other more advanced concepts.

F8 Attack system
: In Chess, opponent pieces are captured just by moving on their cell, but the most common system is to attack and decrease the HP of a target piece from a nearby cell. Long-range attacks are also common.

F9 Counter-attack
: Possibility for a piece under attack to counter-attack, either immediately or during the next turn.

F10 Map landform
: The cells of the map are often different, such as forests, swamps or fortifications. It affects the moves and the attacks of the pieces. Usually, the landform is fixed at the beginning of the game, but the construction of bridges or fortifications is sometimes possible. Multiple layers of landform are also possible.

F11 Piece movements
: Pieces often have a movement capacity, and they pay a given amount of movement cost for each cell that they go through. Units can often go through units of the same team, but not through opponent units. It can also be impossible to go through an area controled by the opponent (Zone of Control).

F12 Player Asymmetry
: The initial placement of the pieces and the landform are not necessarily the same for the players.

F13 Occupation
: Some cells of the map, like a town, a factory or an airport can be occupied by some infantry units. The player controling them usually obtains some benefits.

F14 Production
: System of production of new units, for example from a factory, by using the revenues of towns.

F15 Experience, Level
: System of promotion of the units when they gain experience after a number of attacks.

F16 Remaining bullets
: System to limit the number of times a particular attack can be executed.

F17 Supply, Replenishment
: Possibility of restoring the HP or the number of remaining bullet of a unit, for example in a town.

F18 Enemy search
: In some games, only the units and cells controled by the player are visible to him, creating the need to search the enemies. This component has an important impact on the algorithms because it introduces imperfect information.

F19 Randomness
: The damages of the attacks can contain an amount of randomness. This affects algorithms widely because it introduces probabilities in the state transitions of the game.

F20 Commanding Officer
: Unit with particular capacities, that affects all the units nearby.

F21 Gathering, Dispersion
: Multiple units gathering on the same cell to group their HP and bullet numbers, or the reverse system.

F22 Internal management
: Use of the town revenues to increase the productivity or to develop new units. Such component introduces long-term goals in the game.

F23 Tactical formation, Surrounding effect, Support effect
: Systems that change the efficiency of the attacks during a battle depending on the relative placement of the units.

F24 Strategic bombardment
: Possibility of damaging or destroying opponent facilities, like towns or factories.

F25 Weather, Climate, Time
: Dynamic changes of the game not controled by the players, like a loss of visibility during the night, or a change of a flat land cell into a swamp cell when it rains.

B. Cluster Analysis of TBS Games

In the last two decades, many strategy games have been proposed and commercialized. It is not so easy to trace the relations between them. Some games evolved into other ones, and sometimes different games merged to lead to a new genre. In this section, we analyze the main components of 17 existing representative games, and we group these games in categories, in order to show their relative proximity. We plot on Figure 1 the result of a cluster analysis done with R, projected on 2 dimensions. The features used for the analysis are the existence or not of some components: pieces characteristics (F3), multiple pieces move in one turn (F4), real-time aspect (F4), relative strength of pieces (F6), hit-points (F7), landform (F10), multiple layers of landform (F10), zone of control (F11), occupation concept (F13), production system (F14), variable level of the units (F15), enemy search (F16), internal management (F22).

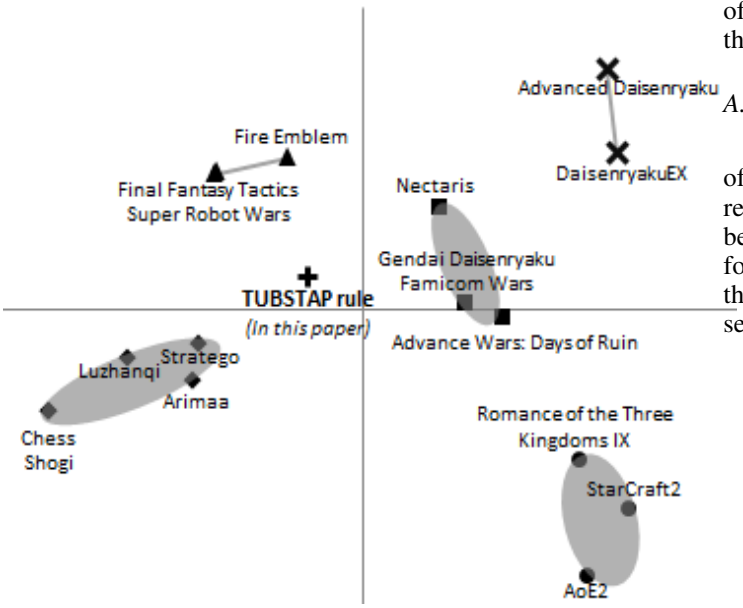

Fig. 1. Cluster Analysis from the main design components

The clusters found in Figure 1 can be explained as follows.

- In Arimaa, multiple moves in one turn are possible, and Stratego or Luzhanqi are games with imperfect information and piece relative strength. These characterics make these games more modern than Chess or Shogi, but they are still directly related.
- Near the center, we find games like Daisenryaku or Famicom Wars, which can be considered as close to a standard of TBS games.
- In SRPG like Fire Emblem, some components are simplified (like Production F14) which make these games in a way closer to Chess, but it is a different cluster because of the addition of personal characters to the pieces.
- Advanced Daisenryaku or DaisenryakuEX contain more complex components like Enemy Search (F18) or multiple layers of landforms.
- Romance of the Three Kingdoms or StarCraft also contain more complex component like Internal Management (F22) and a real-time aspect.

We have also ploted on Figure 1 the position of the rules proposed in Section IV for our platform (called TUBSTAP). These rules are close to Arimaa and Chess, while still retaining the main aspects of standard TBS like Daisenryaku. We consider this position of the rules as a good target for general research on TBS games, without becoming too specific to a particular game.

IV. Rules of a TBS for Research Purpose

In the previous section, we have presented the components found in many strategy games, and the results of the cluster analysis show the relation of these games to classical games such as Chess and Shogi. In this section, our goal is to define a set of basic rules common to many strategy games. This set of rules will be our base to develop a research platform for these games.

A. Motivation of the Design

An important point for research on games is the availability of an open platform, so that the research results can be reproduced and compared with others. We list below what can be considered as the main requirements of a research platform for games. R3 and R6 are not mandatory, but we tried to keep them in mind when designing the platform presented in this section.

- R1 The rules are clearly and precisely defined.
- R2 The rules contain the main components common to most games, but compact enough without minor components very specific to a particular game.
- R3 Extension of the rules to a richer set of components is possible.
- R4 The game is fun for real players, so that a human evaluation of the computer performance and the gathering of game records is possible.
- R5 The game is sufficiently close to existing games, so that players can start to play easily.
- R6 It is possible to compare the algorithms with the commercial ones of existing games.
- R7 Relatively short games are possible, for a fast evaluation.
- R8 A platform containing the rules and a graphical interface is available, with the possibility of doing battle experiments to test the algorithms. It should be easy to change the main algorithms.
- R9 A server is available to perform human vs human, human vs computer and computer vs computer games.

An important point is to find a good balance between R2 and R4. The rules should be left as compact and simple as possible, but it should still be interesting to play for human players. In respect to R5 and R6, we have used the game “Famicom Wars DS2 Advance Wars Days Of Ruin”

(FWDS2) from Nintendo as a starting point, from which we have removed the unneeded components. Especially, this game is commercialized with the possibility of selecting options and defining user maps, so it is possible to reproduce inside this commercial game the rules that we propose.

If we describe FWDS2 with the components listed in III-A, we can note that it is close to Chess. The map is made of square cells (F1), it is a two-player game (F2), multiple kind of pieces are available (F3). But it is also quite different from Chess, since it is possible to move multiple pieces in one turn (F4), and in fact, it contains all the components from F5 to F21. In light of the cluster analysis done in III-B, we have grouped the components of FWDS2 into 4 groups as follows.

- G1 Components found in most games and essential: Multiple moves in one turn (F4), Relative strength of pieces (F6), Hit points (F7), Short and long-range attacks (F8), Movement capacity (F11).
- G2 Components found in many games, not essential but still important: Different victory conditions (F5), Counter-attack (F9), Landform (F10), Assymetry (F12).
- G3 Components found in many games, interesting as extensions but not needed in a basic set of rules: Occupation (F13), Production (F14), Bullet count (F16), Supply and replenishment (F17), Enemy search (F18), Randomness (F19).
- G4 Components found only in some games, that do not need to be considered: Experience level (F15), Commanding Officer (F20), Gathering (F21).

As in FWDS2, we have not included in our platform components like Zone of Control (F11), Real-time aspect (F4), Dispersion (F21), Internal management (F22) or Tactical formation (F23). For the sake of simplicity, we have also not included the components of the group G3, like Occupation and Production, even if they are found in Daisenryaku and many other strategy games. The group G2 is not essential for a basic set of rules, but we chose to include it in our platform, because it is an important part of what makes the game interesting for human players.

B. Description of the Rules

We describe in this section the concrete rules that we have retained for our platform. As shown in the cluster analysis of Section III-B, these rules are in a way an intermediate between games like Chess and Arimaa, and games like Daisenryaku. Based on the design motivation described in Section IV-A, we have included components from F1 to F11 only, and no components from F12 to F25.

- F1 Board. Like in Chess, a two-dimensional board of square cells is used. The dimensions are not fixed.
- F2 Players. Two players. The units of the player to play first are red, and the units of the opponent are blue.
- F3 Pieces. 6 different pieces (units) are used: Fighter (denoted by F), Attack Aircraft (A), Tank (P), Anti-aircraft Tank (R), Infantry (I), Artillery (U).
- Map. The dimensions of the map, the landform, and the initial disposition of units is not fixed. Some maps are prepared, possibly assymetric, and the players can choose any of them to play. Figure 2 shows an example of map, with the landform and initial units.

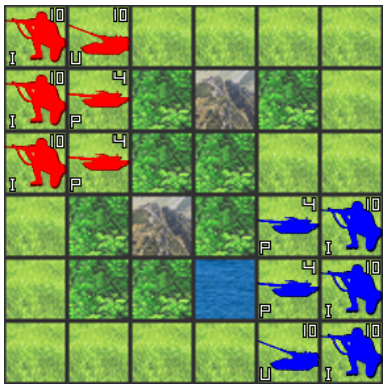

Fig. 2. Screenshot of TUBSTAP

- F4 Order of moves. In a turn, the first player can move all its units one time in any order. When he has finished, the second player can do the same, and when both players have finished, the next turn starts. The 4 possible actions for a unit during one move are "movement only" "movement and close-range attack" "close or long-range attack" or "nothing".
- F5 Victory condition. A player wins when all the opponent units are destroyed. A limit is fixed on the number of turns, and when the limit is reached, the winner (or no winner) is decided by other conditions.
- F6 Units relative strength. We show in Table I the coefficient efficiency of an attack between two types of units. F and R are strong against air units, and A, T, U are strong against land units.
- F7 Hit-Points. Each unit has between 1 and 10 hit points. The number of HP decreases when receiving damages from an attack, and the unit is removed from the board when its HP reaches 0.
- F8 Attack, F9 Counter-attack. Except for U, all units can only attack units on a neighbor cell, either before or after moving. An attacked unit that is not destroyed (value of at least 1 for HP) can counter-attack. Only before moving, U can attack any opponent unit at a Manhattan distance of 2 or 3, and the attacked unit cannot counter-attack.
- F10 Landform. There are currently 5 landforms: mountain, sea, forest, plain, road. This list could be easily extended. The landform affects the movement cost and the defense capacity of the units. Extensions are possible.
- Attack effect. The damages on the HP that result of an attack depend on the relative strength of the units (Table I) and on the protective effect of the landform (Table II), with the equation below. We have

Defense / Attack	A	F	R	I	P	U
A	0	0	85	115	105	105
F	65	55	0	0	0	0
R	70	70	45	105	15	50
I	0	0	3	55	5	10
P	0	0	75	75	55	70
U	0	0	65	90	60	75

TABLE I. UNIT RELATIVE STRENGTH

Landform / Defending unit	Mountain	Forest	Plain	Road	Sea
A,F	0	0	0	0	0
R,I,P,U	0.4	0.3	0.1	0	0

TABLE II. PROTECTIVE EFFECT

not introduced any randomness (F19).

$$\text{effect} = \frac{(\text{relative strength}) \times (\text{attack HP})}{10 + (\text{protective effect}) \times (\text{defense HP})}$$

- F11 Movements. Each unit can move vertically or horizontally, by consuming some of its movement capacity. The movement cost depends on the landform as shown in Table III. There is no need for a unit to use all its movement capacity at each turn. Units can go through cells containing units of the same team, but cannot go through cells contaning opponent units.

	Mountain	Forest	Plain	Road	Sea
A,F	1	1	1	1	1
R,P,U	∞	2	1	1	∞
I	2	1	1	1	∞

TABLE III. MOVEMENT COST

C. TUBSTAP platform implementation

We have implemented the rules proposed in this paper in a platform called TUBSTAP. It can be accessed and used freely [13]. We plan to improve or extend the rules when needed, with the long-term goal of obtaining a platform shared by researchers to compare algorithms and reproduce the results. The project page with source code in C# and binaries is available on the web site of our laborary [13]. We have not implemented yet a server, but plan to do so in the future.

D. Large number of possible actions

The rules of TUBSTAP are inspired from FWDS2, without some advanced components, but even with the current basic set of rules, the game already presents new difficulties compared to classical games like Shogi. For example, a direct application of the $\alpha\beta$ algorithm is difficult. Because of the multiple moves of units in a single turn, the total number of possible actions available to a player is very large. Even with only 6 units that can make 10 possible moves each, there are 720 millions actions available if we consider all possible order of the moves (different order of the moves can lead or not to a similar game state).

In games where multiple piece moves are possible in the same turn, we can distinguish mainly two different ways to implement a game tree search. It is possible to either create a node for each piece move, and consider that a turn is made of successive actions as in Figure 3, or either create a node

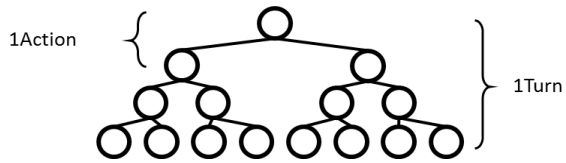

Fig. 3. Game tree with a node for each piece move

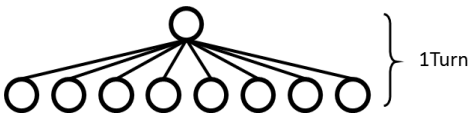

Fig. 4. Game tree with a node for each global action in a turn

for each global action as in Figure 4. The internal structure of the tree is different but the total number of nodes at the bottom of both figures is the same, and corresponds to the end of a possible action. Because of the large number of possible actions, it is usually not possible to search all of them. In the following sections, we present the methods that we have already tested on our platform to handle this difficulty.

V. SEARCH ALGORITHMS

In this section, we present some methods that seem promising to handle the problem of the large number of available actions in games with multiple pieces moves. In the description of the algorithms, we use the following notations.

s: state of the game (board and pieces).

a: one piece action.

$s'(s,a)$: state when the action a has been done from s. It contains information about which pieces can still be moved or not.

$p = \{a_1, a_2, ...\}$: list of all the pieces actions done in one turn.

$P(s)$: set of all possible list of actions p that can be done in one turn from the state s. $P'(s)$ is some subset of $P(s)$.

$s'(s,p)$: state when the list of actions p has been done from s. The opponent is to move next.

$g(s)$: state evaluation function.

A. UCT

UCT is one of the most used Monte-Carlo tree search algorithm, where the formula for guiding the search is UCB1, as presented in Equation 1.

$$UCB1 = \overline{x_j} + \sqrt{\frac{2\log n}{n_j}} \quad (1)$$

$\overline{x_j}$: winning ratio of node j
n_j: number of simulations done at node j
n: total number of simulations of the parent node

UCT was applied successfully to many games, in particular to the game of Go, and it is also promising for turn-based

strategy games [4] [5]. To use fully the potential of the UCB1 formula, we have implemented UCT with a tree as in Figure 3, with one node for each piece action.

However, because of the large number of actions in the TUBSTAP game, some branch pruning needs to be added to UCT. In the following sections, we compare different branch pruning methods to reduce the number of searched actions. In order to compare mainly the pruning aspect of these methods, the same UCB1 formula is used in all algorithms.

Progressive Widening

Progressive Widening (PW) is an advanced method for the UCT algorithm, that consists in searching only a limited set of promising actions [7]. The number of searched actions is increased progressively with the number n of simulations. In this research, we have used PW in the UCT algorithm to search in priority attack actions. Concretely, we have used Equations 2 and 3 to decide the number of searched actions in function of the number n of simulations, as proposed in [9] for the game of Go. The attack actions are added progressively to the search, in a random order, and when all attack actions have already been added, then we add movement actions.

$$Candidates = \frac{\log \frac{n}{40}}{1.4} + 2 \quad (n < 3000) \quad (2)$$

$$Candidates = \frac{\log \frac{n+2000}{45}}{1.2} - 11 \quad (n > 3000) \quad (3)$$

n: number of simulations

Simulation details

In TUBSTAP, the available actions can be divided mainly in two categories, attack actions (that can include or not a movement) and movement only actions. Usually, for a given piece, the number of available attack actions is fairly small compared to the number of movement actions.

For this reason, if the Monte-Carlo simulations are purely random, it is frequent that the victory of one player is not reached inside the simulation before the given fixed maximal number of turns is reached. Such simulation ends in a draw. A similar problem would also arise in Chess with random simulations. To solve this problem, attack actions are chosen whenever available, in all the simulations. The same simulation policy is used for all the algorithms.

Other improvements of the simulations could be considered, for example variations of an epsilon-greedy policy as in [8], but our focus in this paper is mainly the search part of the UCT algorithm.

B. Attack Action Search (AAS)

Murayama et al. proposed a method [10] to focus the search on attack actions. The details are as follows.

1) First, from the current state s, all the lists of actions of length at most l containing only attack actions are listed in a set $P'(s)$. If there are no attack actions, some lists with movement actions are added to the set.

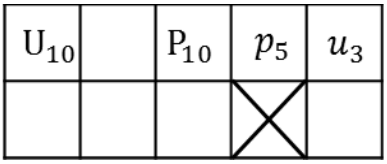

Fig. 5. Example of state where the attack action search works well. (Capital letter to play first)

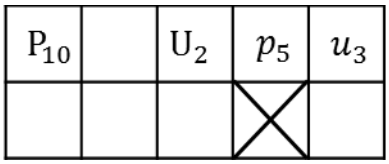

Fig. 6. Example of state where the attack action search does not work well. (Capital letter to play first)

2) For each action list in $P'(s)$, the resulting state $s'(s,p)$ is evaluated with an evaluation function g, and the action list p with the highest evaluation is chosen.

In the first step, the length of the action list is limited to l to avoid particular cases where the total number of action lists is large. There are some states where all pieces can attack, and when all the possible ordering of attacks are taken into account, the total number of action lists become quite large. The length limitation l prevents such case.

Figure 5 shows an example where this attack action search works well. The teams are distinguished with small or capital letters, and the player with capital letters plays first. The numbers show the remaining HP of each unit, and the cross shows a cell where it is forbidden to move. If the piece U (artillery) does not move, it can attack at a distance of 2 or 3, and the piece P (tank) can attack even after moving, but only the neighbor cell. In this case, if P moves first, it will lose the opportunity to attack u during the turn. The best option is to attack u with U first, and then attack u again with P. AAS is able to analyze such case correctly, and find the best action within the attack actions.

On the contrary, Figure 6 shows an example where the attack action search does not work. U cannot attack the neighbor cell, and attacking u is not urgent. Also, the position of U prevents P from attacking, so the best action is to make a retreat move with U. However, this action is not found by AAS since it is a movement action. The attack action search is also not efficient on the starting position of Figure 2, since no attack is possible.

C. DLMC

We present now the Depth-Limited Monte-Carlo (DLMC) method, already applied to games like Amazon [11] [12]. It can handle well the position of Figure 2. It works as follows.

1) A set $P'(s)$ is created with m samples selected randomly from the possible list of actions p.
2) For each action list p in $P'(s)$, n simulations limited to d turns are done, and the state that is reached is evaluated by the state evaluation function g(s). The evaluation of p is the average of the simulation results.

3) The action list p^* with the highest evaluation in $P'(s)$ is chosen.

This DLMC method uses a tree as in Figure 4, where each node is a total action (list of actions of all the pieces). For each total action, a simulation is done, but only for a limited number of d turns, after which the state is evaluated with an evaluation function. The advantage of this method is that the simulations are faster. Since we are using total actions instead of the actions of each piece, the state is easier to evaluate.

Since there is potentially a large number of total actions, we search only m random actions. This creates the risk of missing promising actions, especially attack actions, but it works well to find good movement actions or defensive actions. The proportion of movement actions in all the possible actions is higher than that of attack actions, so most of the m random action samples are movement actions. Because of this, DLMC works well on Figure 2 and Figure 6, but on the contrary, it is weak in states like Figure 5.

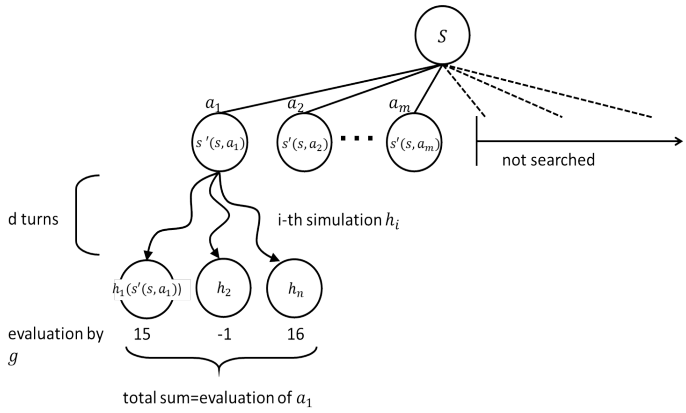

Fig. 7. DLMC method

State Evaluation Function

The state evaluation function $g(s)$ can have an important effect. Since it is called at each simulation, we have chosen a natural function that is easy to compute. It consists in comparing the remaining units of the two players.

$$g(s) \quad = \quad (M - E) * B$$

M: Total sum of HP of the team units (with a coefficient of 0.2 for piece I, and of 0.5 for piece U)
E: Total sum of HP of the opponent units (with the same coefficients)
B: Value of 2 if M or E is 0 (end of the game), 1 otherwise

The coefficient B is used to make a difference between a finished and an unfinished game. If the player wins, the evaluation of the final state will be better than almost any other situation, and if the player loses, it will be worse. Moreover, a coefficient is applied to the piece I, because the attack power of I is limited and this piece is not as useful as the others. A coefficient is also applied to the piece U, because this piece cannot move and attack at the same time, so its usage is more difficult than other pieces.

D. Combination of DLMC and AAS

The Attack Action Search (AAS) algorithm performs well at finding attacking moves, but cannot find defensive or tactical moves in a situation like Figure 2. On the contrary, DLMC is able to find good tactical moves on Figure 2, but is weak at finding the decisive attacking moves on Figure 5. So it is natural to consider a combination of the two algorithms, that can be expected to have the advantages of both algorithms without their weaknesses. The idea is simply to add to the DLMC search the good attack moves found by AAS, as follows.

1) a set $P'(s)$ is created with m random samples of action lists p,
2) The attack action search is performed from the state s and the attack action list p' with the highest evaluation is added to $P'(s)$.
3) The end of DLMC algorithm (steps 2 and 3 in DLMC) is performed with this set $P'(s)$.

The goal of this method is to correct the main weakness of the DLMC algorithm, which frequently misses good attacking moves. As shown on Figure 8, this DLMC+AAS method consists simply in adding the best action list of AAS to the random action lists (called "total action" on the figure) searched by DLMC. As a result, DLMC+AAS is able to handle well both Figure 2 and Figure 5.

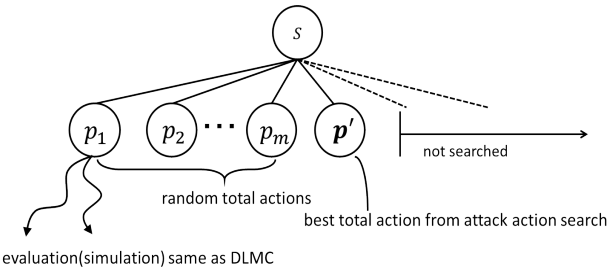

Fig. 8. Combination of DLMC and Attack Action Search

VI. Performance evaluation

A. Battle Experiments

In Section V, we have presented several methods, UCT+PW, DLMC, DLMC+AAS, to handle the very large number of actions available in the game defined in Section IV. Here, we show the battle results obtained when comparing these methods. The conditions of the experiments are as follows.

- The map from Figure 2 is used.
- 1000 games are done in a battle. One player plays first in 500 games, and the other player plays first in 500 games.
- 4 algorithms are compared: UCT, UCT+PW, DLMC, DLMC+AAS.
- A draw is counted as a 1/2 win for both players.

In order to compare the algorithms on a fair base, we have adjusted their parameters as in Table IV so that the time used

for choosing the next action is roughly the same for all the algorithms. The time used for one move was around 3 seconds on a standard computer with 8Gb of memory and a Core i5 3.3Ghz. Also, the number of turns in a simulation is limited to $d = 2$ for DLMC, and the length of the action list is limited to $l = 4$ in the AAS part of DLMC+AAS. In the case of UCT, the number of simulations is fixed, and dispatched between the actions of each piece. For example, if 6000 simulations are available, and if there are 6 pieces, 1000 simulations will be used to search the best action of each piece.

	Number of samples	Number of simulations
DLMC	200	100
DLMC+AAS	180	90
UCT	-	6000
UCT+PW	-	6000

TABLE IV. PARAMETERS

The result of the experiment is shown in Table V. For a given match, we show the winning ratio from the point of view of the algorithm on the given line of the table. As expected, DLMC+AAS is the best algorithm of the four, and is able to win against all the other algorithms. It is not shown in the table, but the number of draws in DLMC+AAS self-play is notably decreased compared to DLMC self-play. It shows that DLMC+AAS correctly plays more aggresive moves, compared to DLMC.

TABLE V. WINNING RATE (LINE AGAINST COLUMN)

	UCT	UCT+PW	DLMC	DLMC+AAS
UCT	49.7	-	-	-
UCT+PW	62.5	49.6	-	-
DLMC	41	33.1	53.1	-
DLMC+AAS	60	56.2	65.5	48.25

B. Evaluation of the Opening Performance

As in most turn-based strategy games, there is a very large number of opening moves in a game of TUBSTAP. The choice of the opening move has a big influence on the whole game, so we found it interesting to compare only the first move of the algorithms with the following experiment.

- Only the first move is played by the algorithm that we evaluate.
- All other moves except the first one are played by UCT+PW, which is also used as the opponent.
- We measure the winning ratio after 1000 games.

We have used the map of Figure 2 and UCT+PW is the reference opponent that plays all the moves except the first one. UCT+PW was chosen because we feel that it is usually strong at the end game. The result of the experiment is shown in Table VI. As can be read from the table, when DLMC is used for the first move of the player that opens the game, the winning ratio is greatly improved. It shows the strength of DLMC in the opening. For the first move of the player that plays in second, we can see that attacking moves are already important, as is shown by the better result of DLMC+AAS than DLMC alone.

The UCT+PW line is equivalent to self-play, so it is expected that the average winning ratio should converge to 50%. It is interesting to note that the game seems slightly favorable to the second player. Also, UCT+PW searches in priority attacking moves, so the good result of UCT+PW when playing as the second player confirms that attacking moves are important even at the first move of the second player.

Algorithm	Winratio, first to play	Winratio, second to play	Average winratio
DLMC	71.8	37.9	54.85
DLMC+AAS	72.8	48.5	60.65
UCT	53	47.2	50.1
UCT+PW	45.4	52.3	48.85

TABLE VI. MOVE OPENING

VII. CONCLUSION

Strategy games have evolved from games with relatively simple rules such as Chess or Shogi to games with very detailed and complicated rules, such as StarCraft. In this paper, we have first analyzed the game components found in many well-known games. Then, we have designed the rules of a turn-based strategy game that can be considered as an intermediate between Chess and commercial turn-based strategy games. We have implemented these rules in a platform called TUBSTAP, which could be used in the future by other researchers to investigate general algorithms for turn-based strategy games. Finally, we tested our platform with a comparison of some Monte-Carlo algorithms. Especially, we introduce a depth-limited Monte-Carlo tree search, and we show its efficiency at handling tactical moves. The best performance is obtained when this algorithm is combined with an algorithm that handles correctly attacking moves. Our platform has already reached a level that allowed us to compare successfully different algorithms. In the future, we plan to develop a client-server mechanism, in order to play online against humans and organize program tournaments.

REFERENCES

[1] Arimaa Homepage, *http://arimaa.com/arimaa/*,

[2] BWAPI, *https://code.google.com/p/bwapi/*,

[3] IEEE-CIG Competitions, *http://geneura.ugr.es/cig2012/competitions.html*,

[4] M. Bergsma, P. Spronck, *Adaptive Spatial Reasoning for Turn-based-Strategy Games*, Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, October 22-24, 2008

[5] T. Kato, M. Miwa, Y. Tsuruoka, C. Takashi, *UCT and Its Enhancement for Tactical Decisions in Turn-based Strategy Games*, IPSJ-GPW 2013-11-01pp.138-145, 2013 (in Japanese)

[6] T. Kozelk, *Method of MCTS and the game Arimaa*. Master's Thesis, 2009

[7] R. Coulom, *Computing Elo Ratings of Move Patterns in the Game of Go*. ICGA Journal Vol.30 pp.198-2082007

[8] N. Sturtevant, *An Analysis of UCT in Multi-Player Games*, 6th international conference on Computers and Games, 2008.

[9] H. Yamashita, K. Yoshizoe, H. Matsubara, *Computer Go: Theory and Practice of Monte Carlo Method* (in Japanese), 2012.

[10] K. Murayama, T. Fujiki, K. Ikeda, *Proposal of rules for Turn-based-Strategy-games as an academic platform* . IPSJ-GPW 2013-11-01 ,pp.146-153, 2013 (in Japanese)

[11] J. Kloetzer, H. Iida, B. Bouzy, *The Monte-Carlo Approach in Amazons*, Computer Games Workshop, 2007

[12] J. Kloetzer, *Monte-Carlo Techniques:Applications to the Game of the Amazons*. Japan Advanced Institute of Science and Technology, Doctor 240 Includes bibliographical references pp. 87-92, 2010

[13] TUBSTAP web page: *http://www.jaist.ac.jp/is/labs/ikeda-lab/tbs*