Towards an Evenly Match Opponent AI in Turn-based Strategy Games

Kittisak Potisartra
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Payathai Road, Patumwan
Bangkok 10330, Thailand
Tel: +66859303372
ffseries_t@hotmail.com

Vishnu Kotrajaras
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Payathai Road, Patumwan
Bangkok 10330, Thailand
Tel: +66890212323
Vishnu@cp.eng.chula.ac.th, ajarntoe@gmail.com

Abstract

Research in turn-based strategy (TBS) games mostly involves classic games, such as Chess, and how such games could be beaten by a computer controlled artificial intelligence. Guaranteeing that opponents will be beaten, however, is not the focus of commercial Turn-based Strategy games. For commercial games, if human players do not win, they quit the game. This can result in horrific future sales. Therefore, keeping players engage in the game is much more important. This paper presents an artificial player that learns to adjust its skills to match a player it is playing against. A Final Fantasy Tactics-like game is used in our experiment. We introduce evaluation functions for calculating the score from each unit's action. By evaluating a human player's score, our artificial player can estimate his skill and play at the same level throughout the game.

Keywords

Turn-based Strategy (TBS) games, evaluation functions, Artificial Intelligence

1. Introduction

Opponents Artificial Intelligence (AI) in computer games varied greatly. Many researchers presented AI that was guaranteed to beat its opponent. This was the case for turn-based strategy games such as Go and Chess. However, AI that always beat human players was frustrating to play against. For a commercial game, this could easily cause players to quit the game, damaging the game's reputation and consequently its sale figures. An AI that drew players to the game must be an AI that put reasonable challenges on each player. In other words, an AI should be as good as a player it was competing against.

Spronck and Herik [1] proposed an AI for Computer RolePlaying Games (CRPGs) that used three different enhancements to dynamic scripting to play "even games" with human players in a tutoring system. Their enhancement techniques included high-fitness penalization, weight clipping, and top culling. The result showed that dynamic scripting with top culling was the best for difficulty scaling. In this research, however, we focused on AI for turn-based strategy (TBS) games. The AI for TBS games offered many challenges for researchers, such as planning and decision making under

uncertain situations. Players were able to spend as much time as they wanted to consider their moves and actions. No time constraint affected players' decisions. Bergsma and Spronck [2, 3] proposed ADAPTA architecture for implementing the learning AIs for TBS games. They chose one subtask and learned new AIs using an evolutionary algorithm. Their methodology was able to generate tactics that defeat all single opponents.

The challenge of our research, however, was to create an adaptive AI that could compete evenly with human players. To accomplish this, we introduced an evaluation function for managing our AI when playing against players.

The outline of this paper is as follows. In Section 2 we provide a definition of this research's TBS game. Section 3 explains the evaluation function of this game. Section 4 covers our difficulty adjustment mechanism. In Section 5 we explain the experimental setup. The results are discussed in Section 6. Section 7 provides conclusions.

2. Game Definition

For our research, we developed our own TBS game using information and actions based on Final Fantasy Tactics series. The game map was simplified to two-dimensional, tile-based map, as shown in Figure 1. The tiles were split into 2 types. Players could move over the first type, not the second type. Movement in the game could occur either be vertically or horizontally. Diagonal movement was not permitted.

There were 4 character types in our game. They were warrior, archer, black mage and white mage. Each type had its own set of parameters, which included (1) The starting amount of health points and magic points, (2) The number of moves the character could make per turn, (3) The range of physical attack and magic attack, (4) The attack, defense, magic attack and magic resistance value, and (5) The character's speed.

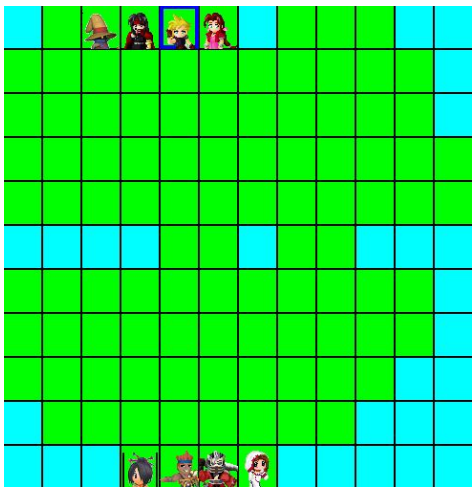


Figure 1. A screenshot of the TBS game used in this research

Each type of character was able to perform different actions. For example, a warrior could only attack its opponent from close range, but the attack was more powerful than other types of characters. An archer could perform a long range physical attack. But the attack strength was less powerful compared to a warrior's. A mage had very low defense, but it could perform magic that affect several characters at once. A black mage could spend magic points to carry out a magic attack, while a white mage could restore any character's health point.

At each character's turn, its player chose any action he wanted the character to perform, such as move, physical attack, magic attack, or wait. Move and attacks could be carried out in the same turn, but move must come first. A single tile could accommodate only one character. Characters were able to pass through a tile that was occupied by characters in the same team. They were unable to pass through a tile occupied by an opponent.

3. The Evaluation Function

Many researchers used evaluation function to maintain game AI. Bakkes [4] used the number of enemy units scaled by visibility range. Strattman [5] used an evaluation function for finding the best waypoint in first person shooting games. For our approach, we developed our own evaluation function. Our function calculated a score for each character's turn. The score was calculated using equation (1).

$$S(x,y) = \sum_{c \in C_E} \beta\big(EN, DS, \delta(c,x,y)\big) + \sum_{c \in C_A} \gamma\big(EN, \delta(c,x,y)\big) \quad (1)$$

Where:

C_A represented the set of all allies.

C_E represented the set of all enemies.

$\beta(EN,DS,d)$ calculated a partial score using:

- Enemies' environment information, *EN*
- The damage score *DS*, and
- Distance *d* from enemy.

$\gamma(EN,d)$ calculated a partial score using:

- Allies' environment information *EN*
- Distance *d* from character.

$\delta(c,x,y)$ calculated the distance from character *c* to tile (x,y).

In summary, this equation calculated the score for each tile (x,y) by using environment information and distance from other characters in the same map. We used this function to predict the score of a character if it moved to a tile.

4. Difficulty Adjustment

Once a move was made, our evaluation function was able to determine how good it was. By analyzing the score from a human player's previous move(s), our system was able to adjust the AI to make a similar-level move. In its turn, before our AI made its decision, it calculated how even the game was at that stage using equation (2).

$$E(x,y) = S_{AI}(x,y) - \frac{\sum_{t \in T} S_{OP}}{N} \quad (2)$$

Where:

E(x,y) was the evenness score for each tile (x,y).

$S_{AI}(x,y)$ was the AI score, if it moved to tile (x,y).

S_{OP} was the score of a human opponent character that made a move after our AI's previous turn. We need the sum because it was possible for two or more characters of the same side to move in succession.

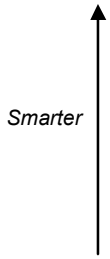
T was the set of the human player's turn.

N was the number of members in *T*.

Positive *E* meant that the AI was at a better position than its opponent, and vice versa. If *E* was zero, it meant the two sides performed equally at the current stage of the game. Figure 2 shows different levels of evenness, where $L_0, L_1, \ldots, L_n$ limited the edge of each level. When our AI was made to choose its action, it chose from available actions in an appropriate level. If the player's score changed during play, the AI responded accordingly.

5. Experiment

For our experiment, we played our AI against different scripted AIs. We split characters into two teams and made them to fight each other. Each team had the same configuration of characters: warrior, archer, black mage, and white mage.



Level	Range of *E*
3	$L_2 < E$
2	$L_1 < E < L_2$
1	$L_0 < E < L_1$
0	$-L_0 < E < L_0$
-1	$-L_1 < E < -L_0$
-2	$-L_2 < E < -L_1$
-3	$-L_3 < E < -L_2$

Figure 2. Evenness Range

There were 4 scripted AIs we tested against. The AIs used the following strategies:

- The first script used the strategy that always moved towards the nearest opponent character, and attacked whenever possible (Rush).
- The second script used the strategy that always moved towards one of the opponent character, and kept attacking until the character was defeated, before targeting the next opponent character (Unit Offence).
- The third script used the strategy that kept the characters close to each other, and attacked only when provoked (Defense).
- The forth strategy attempted to balance both offense and defense (Balance).

6. Results

In Figure 3, the histograms show the percentage of total health point (Hp) that scripted AI and adapted AI had after each battle for each strategy (except Defense). A positive value indicated that the scripted AI won the battle. A negative value indicated otherwise. For a battle to be even, the corresponding health point left in that battle has to be close to zero. Each team had 4 characters. Therefore one character could be roughly represented by 25% of the total health point.

Our AI played evenly against Rush AI and Balance AI. After each battle finished, if a scripted AI won the battle, it often had only one character left. If our AI won the battle, it often had only one character left also. The total health point percentage mostly dropped below 25%. Most of the time, only one character (or two characters; each with small health point) remained. For the Unit Offence AI, however, our adapted AI won most of the battles quite comfortably. We investigated this and discovered that characters controlled by the script usually moved in a group. Our AI played similarly in response. This caused our AI's character that was chosen by the Unit Offence to be surrounded by allies. This resulted in the Unit Offence AI having to move characters around a lot in order to be able to reach and attack its chosen target. Most of the moves ended up not attacking. Our evaluation function did not scale down well in this case. For the defense AI, its characters waited to be attacked first. Our AI tried to play an even match by choosing not to move also, causing the battle to stall. Therefore we could not obtain the battle results.

It could be seen that our adapted AI played evenly against two of the four common strategies. For Defense strategy, although the battles stalled, it was reasonable to state that our AI was doing the right thing to maintain an even match. In actual use in a commercial game environment, we could disable the even play temporary to allow initial attacks to take place. The only weakness of our AI came up when playing against the Unit Offence Strategy. Improving our evaluation function further by implementing a better scale down and considering more past and future moves would fix this weakness.

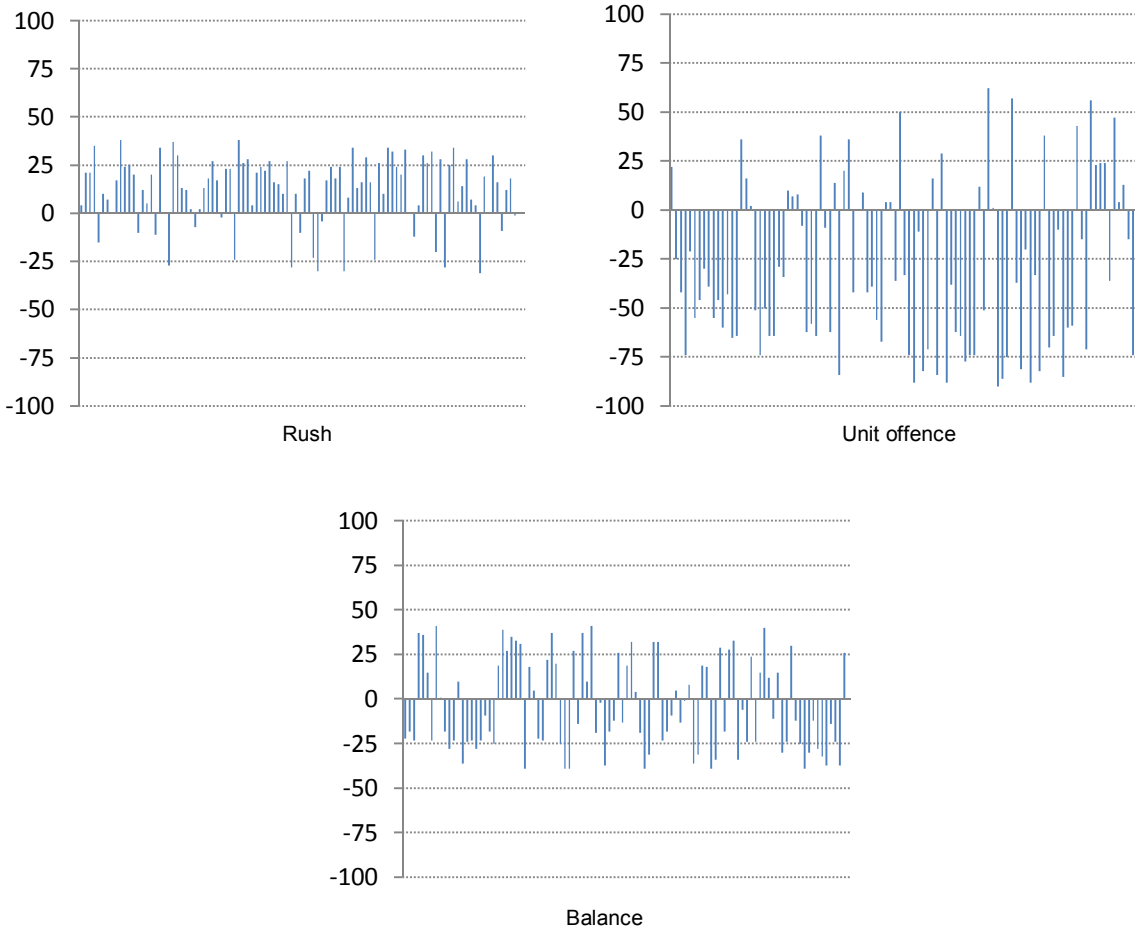


Figure 3. Histogram of the percentage of heath point (HP) of each team AI (Rush, Unit offence, and Balance) after playing against our adapted AI in 100 battles

7. Conclusions and Future Work

The goal of this research was to propose a prototype AI for turn-based strategy games that could play as well as the player it was playing against. To accomplish this, we used an evaluation function to calculate the score of each tile on the map by using data from each team. The score from each tile was then used with opponent characters' scores to calculate the evenness score of each tile. Our AI adjusted its move for a matching evenness level. The result showed that our AI was able to play even matches against most common strategies.

For future work, we intend to enhance our evaluation function to include a better adaptation when an opponent failed to perform attacks. Our evaluation function also needs to look further into past and future moves. A balance adjustment will need to be made because we do not want to look too far into the past or future, since it wastes processing time for commercial games. Furthermore, we intend to add more types of units that can initiate status effects, such as temporary speed-down, magic-disable or poison. Such addition will make our model more like commercial turn-based strategy games available today, thus adding values to our research.

References

[1] Pieter Spronck and Jaap van den Herik. A Tutoring System for Commercial Games. *ICEC 2005* (eds. Fumio Kishino, Yoshifumi Kitamura, Hirokazu Kato, and Noriko Nagata), Lecture Notes in Computer Science 3711, pp. 389-400. Springer-Verlag.

[2] Maurice Bergsma, Pieter Spronck. Adaptive Intelligence for Turn-based Strategy Game. *Proceedings of the BNAIC 2008, the twentieth Belgian-Dutch Artificial Intelligence Conference.pp.* 17-24. University of Twente, Twente, The Netherlands.

[3] Maurice Bergsma, Pieter Spronck. Adaptive Spatial Reasoning for Turn-based Strategy Games. *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference, pp.* 161-166. AAAI Press, Menlo Park, CA.

[4] Sander Bakkes, Pieter Spronck, Jaap van den Herik. Phase-dependent Evaluation in RTS Games. *Proceedings of the 19th Belgian-Dutch Conference on Artificial Intelligence, 2007*, pp. 3-10.

[5] Remco Straatman, William van der Sterren, Arjen Beij. Killzone's AI: dynamic procedural combat tactics. *Game Developers Conference (GDC) 2005*.