Blokus Duo Game on FPGA

Ali Jahanshahi
Department of Electrical and Computer Engineering
University of Tehran
Tehran, Iran
ali.jahanshahi@ut.ac.ir

MohammadKazem Taram
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
taram@ce.sharif.edu

Nariman Eskandari
Department of Electrical and Computer Engineering
Shahid Beheshti University
Tehran, Iran
eskandari_nariman@robocyrus.ir

***Abstract*—There are a number of Artificial Intelligence (AI) algorithms for implementation of "Blokus Duo" game. We needed an implementation on FPGA, and moreover, the design had to respond under a given time constraint. In this paper we examine some of these algorithms and propose a heuristic algorithm to solve the problem by considering intelligence, time constraint and FPGA implementation limitations.**

Keywords—Blokus Duo game; FPGA; monte-carlo tree search; min-max tree search;

I. INTRODUCTION

The first National Digital Systems Design contest of Iran was held in conjunction with Computer Architecture and Digital Systems (CADS) conference from July to October 2013. The FPGA Challenge part of the contest was a Blokus Duo game on a 12x12 board. This paper presents implementation details of the design realized by the authors who managed to secure the first place in the contest among the teams who had successfully passed initial tests and were granted entry to the final round of the game on October 29th in Sharif University of Technology.

There are several versions of Blokus duo game. The version of this game used in this paper is palyed on a 12 x 12 square board. Two players play this game and each one has 17 different shaped tiles as illustrated in Fig. 1. Tiles can be rotated in 8 possible way. All rotations of tile 'P' are illustrated in Fig. 2. Players alternativly put one of their tiles on the board according to two primary rules:

1. Newly placed tile must have at least one corner-to-corner contact with a tile of the same color.
2. Newly placed tile must not have edge-to-edge contact with any tile of the same color.But there is no limitation for two opposite color tiles for putting on board.

Fig. 3 illustrates some allowed and prohibited moves.

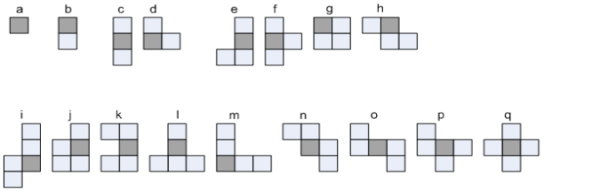

Fig. 1. Existing tiles

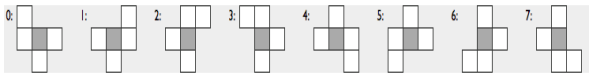

Fig. 2. Different rotations of a tile

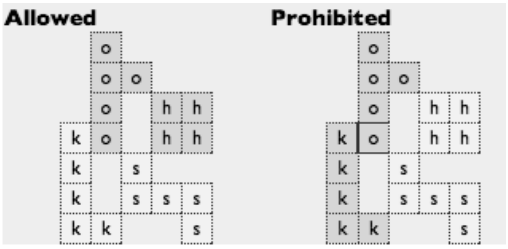

Fig. 3. Allowed and prohibited moves

There are some other rules for starting and finshing the game:

- On the First move players must cover either (4,4) or (9,9) on the board.
- When it is not possible for a player to place a tile on the board, it must *pass*.
- The game continues until both players pass, one player plays all its tiles, or one player makes an

978-1-4799-0565-2/13/$31.00 ©2013 IEEE

invalid move.

There are also some rules for calculating scores of the players:

- Basic score is given by minus total number of squares of unplaced tiles.
- If a player played all 17 tiles, the player score will be 15.
- If the last tile he puts on the board is 'a', the bounce increases to 20.
- invalid move results in immediate loss of game.

II. ATTEMPTED ALGORITHMS

Our goal is to find an algorithm to implement this game on an FPGA so that the FPGA plays the game. At first we explain the algorithms by which this game can be implemented on FPGA. There are some timing constraints for players to choose and place a tile on the board. Implemented algorithm should make a move in upto 10 seconds.

So we first investigated some alorithms and approaches to maintain these constraints.

A. Min-max with alpha-beta pruning

Two-player deterministic games with perfect information have been under study in Artificial Intelligence (AI) research for many years and significant results have been achieved. One well-known framework to deal with such problems is the alpha-beta framework. However, this framework works well only under two conditions:

1. A suitable evaluation function exists
2. The game doesn't have a high branching factor.

Because of high branching factor of Blokus duo game we must evaluate the game at very early stages, so the result is not acceptable.

B. Monte carlo tree search

One of the most important components required in AI to achieve good results is defining a proper evaluation function. The task of such functions is to estimate the state of the game in a non-final state. However, to define a good evaluation function we need heuristics based on specific knowledge in the game domain. In simpler domains, where AI has already achieved significant results, it is easier to define the functionbut in more complex environments it is unlikely to find such function. Recent researches including Monte-Carlo based techniques are employing other approaches. They have already been applied successfully to many games, including POKER [7] and SCRABBLE [10]. For example Monte-Carlo Tree Search (MCTS) that was first proposed in 2006, is implemented in top-rated GO programs. MCTS programs could defeat professional GO players on a 9x9 board. The idea is not specific to GO and can be used simply in any other boardgames. To achieve acceptable result we need to repeat the pseudo-random simulation for large enough iterations, but on a low frequency FPGA this can violate the time constraint of the game.

III. PROPOSED HEURISTIC ALGORITHM

We investigated two well known AI algorithms that are common in the field of computer games but they do not maintain timing constraints. So we decided to propose a heuristic algorithm. In the proposed heuristic algorithm, at each step of game, we find all possible moves then rank each move according to a number of strategies described below. After ranking the moves, we finally select one of the highest score moves randomly.

This algorithm proposed three heuristic strategies for scoring each move:

1. **Developing our game:** by putting bigger tiles on the board we try to place more squares early in the game. Because of the game scoring rule which state that "the player who puts more squares on the board is winner.", our first scoring strategy is selecting bigger tiles. According to the game rules, we are allowed to put a tile on board if it has a corner-to-corner contact with exiting same color tiles. This rule shows the importance of free corners for each player. In fact, the number of next possible moves depends on the number of free corners. Thus, we detect moves that increase free corners and give them a higher rank.
2. **Disrupting the other player game:** As said in previous strategy, free corners are important for players. So we can destroy free corners of the other player by putting our tiles there. Fig. 4 and Fig. 5 illustrate a situation in which the player with black tiles wants to choose a move between two possible moves. According to this strategy, the move which destroys more corners of the white player is selected as the next move. So it chooses the move of Fig. 5.

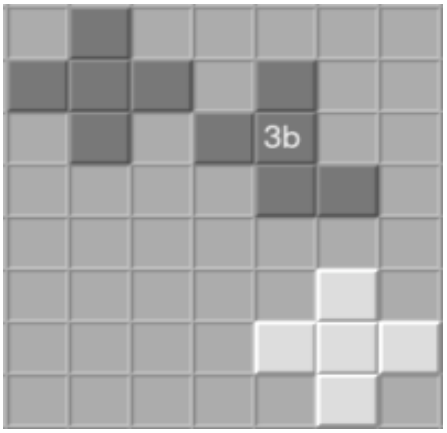

Fig. 4. The move does not destroy any corner

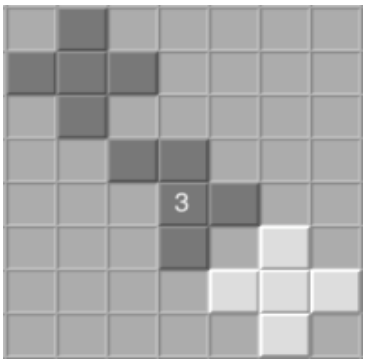

Fig. 5. The move destroys two corners

3. **Survive:** The last and the most important sterategy is to survive. There are situations in which one player is trapping the other player in a small region of board and wants to limit the other player moves. In these situations the player who is trapped by the other player, must find a way to escape from trap. We detect places on the board that can help us escape from the other player's trap. We call these places "strategic places." Fig. 6 and Fig. 7 illustrate two situations in which the white player is creating a wall in order to trap the black player. Here we also have some possible moves. Fig. 6 illustrates a move by which the black player did not consider a strategic place for later escaping from the trap. But figure 7 illustrates a move in which the strategic place is considered and we can pass to other side of the white player as in Fig. 8.

IV. DYNAMIC STRATEGIES

Another heuristic used in the proposed algorithm is that the weights of each ranking strategy, i.e. Progressing our game, Disrupting the other player game and survive, dynamically changes during the game. We called this heuristic "**dynamic strategies.**" In this strategy, we partition the game to three phases: beginning , middle and end of the game. At each phase we have diffrent weights for the ranking strategies. We detect each phase by the number of moves we have played. For example at the beginning of the game, strategic points are less important than the other strategies, so they have little priority and weight, whereas the other strategies are more important at that time. TABLE I. shows priority of our strategies during game.

TABLE I. WHEIGHT OF EACH STERATEGY DURING THE GAME

Phases	**Strategies**			
	Disrupt other player	***Increase chance of your next move***	***Bigger tiles***	***Strategic points***
Start of the game	High	High	Very High	Normal
Middle of the game	High	Normal	High	Very High
End of the game	Low	Low	Normal	Very High

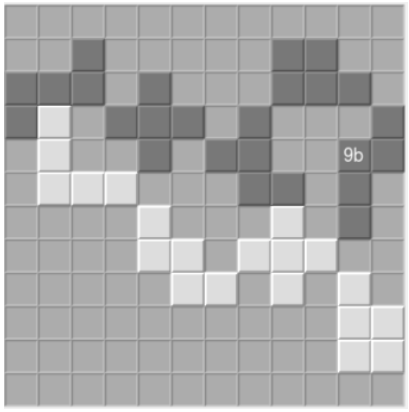

Fig. 6. Move without considering strategic place

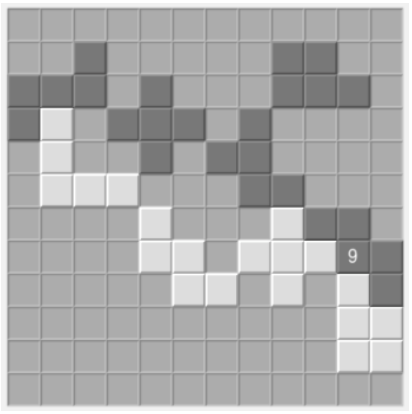

Fig. 7. Move with considering strategic place

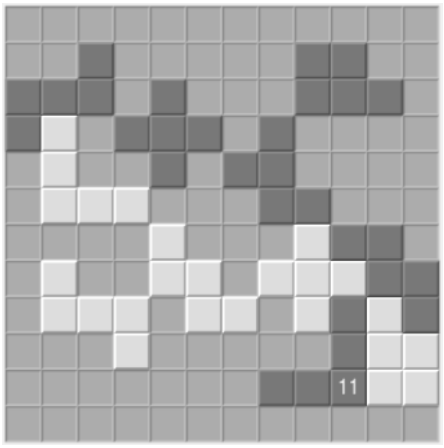

Fig. 8. Passing the white player wall from strategic place

SUMMARY AND CONCLUSION

In this paper we described the heuristic strategies we employed and implemented on FPGA in our design submitted to the FPGA Challenge game of the first National Digital Systems Design contest of Iran. We developed three

heuristics to rank possible choices at each move of our player, and also employed a dynamic weighting strategy to change the significance of each ranking heuristic based on the beginning, middle, and

Our design won the first place in a round robin contest among the 6 contestant teams who had managed to get to the final round of the game.

REFERENCES

[1] Chaslot, Guillaume M. JB, Mark HM Winands, H. JAAP VAN DEN HERIK, Jos WHM Uiterwijk, and Bruno Bouzy. "Progressive strategies for Monte-Carlo tree search." New Mathematics and Natural Computation 4, no. 03 (2008): 343-357.

[2] Winands, Mark HM, Yngvi Björnsson, and Jahn-Takeshi Saito. "Monte-carlo tree search solver." In Computers and Games, pp. 25-36. Springer Berlin Heidelberg, 2008.

[3] Browne, Cameron B., Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. "A survey of monte carlo tree search methods." Computational Intelligence and AI in Games, IEEE Transactions on 4, no. 1 (2012): 1-43.

[4] Stockman, George C. "A minimax algorithm better than alpha-beta?." Artificial Intelligence 12.2 (1979): 179-196.

[5] Knuth, Donald E., and Ronald W. Moore. "An analysis of alpha-beta pruning." Artificial intelligence 6, no. 4 (1976): 293-326.

[6] Chaslot, G.-B.; Saito, J.-T.; Bouzy, B.; Uiterwijk, J.; and van den Herik, H. 2006. Monte-Carlo Strategies for Com- puter Go. In Proceedings of the 18th Belgian-Dutch Con- ference on Artificial Intelligence, 83–90.

[7] Kocsis, L., and Szepesva´ri, C. 2006. Bandit Based Monte- Carlo Planning. In Machine Learning: ECML 2006, Lec- ture Notes in Artificial Intelligence 4212, 282–293.

[8] Sheppard, B. 2002. World-championship-caliber scrabble. Artificial Intelligence 134(1):241–275.

[9] Billings, D.; Davidson, A.; Schaeffer, J.; and Szafron, D. 2002. The challenge of poker. AI 134(1):201–240.

[10] Schaeffer, Jonathan. "The history heuristic and alpha-beta search enhancements in practice." Pattern Analysis and Machine Intelligence, IEEE Transactions on 11.11 (1989): 1203-1212.

[11] Chaslot, Guillaume, et al. "Monte-Carlo tree search: a new framework for game AI." AIIDE. 2008.