Project Blokus

CS 4701 Final Report
Spring 2017

Members

Tomasz Chmielewski and Ivan Zaitsev

Game Description

Blokus is an strategy board game that is played by 4 people, where players take turns trying to place one of 21 free polyominoes onto a 20 by 20 grid. Each player starts in their respective corner, and must place pieces that touch the corners of their own tiles, but do not touch edges of their own tiles. The game ends when no players can place any pieces, and the victor is determined by counting the number of tiles placed on the board and giving a +15 bonus to players who placed all their pieces and a +5 bonus if their last piece was the 1x1 piece.

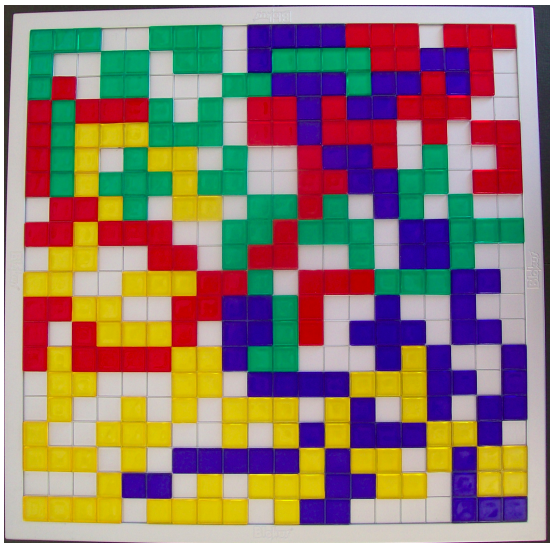

Figure 1. *A finished game where no players can place any more tiles. Notice how no tile of the same color is adjacent to each other, but all tiles of the same color are touching corners.*

Original Goals

Our objective was to create an intelligent agent that can play this board game both against humans and other agents. As Blokus is a perfect information, turn-based game, we wanted to explore various adversarial search algorithms that deal with this decision theory such as minimax. Due to the exponential increase in size of the search tree with each branch having ~100 valid moves, we also wanted to create and test various heuristics devised from available strategy guides and best practices. After this preliminary agent was devised, we envisioned using machine learning concepts and having the agents play against humans and each other to tweak the heuristic constants and possibly generate endgame tables.

Accomplished Goals Overview

We initially began with a simple informed search algorithm that would use a single strategy and a hill-climbing approach to choose the next move. We tested these preliminary AIs against Blokus novices and pros and found that though certain strategies could beat first-time players of Blokus, they failed to win a single game against Tomasz (self-proclaimed Blokus expert).

We then combined all the strategies and and assigned a weight to each. Using an evolutionary algorithm, we simulated 4 AIs playing against each other and eventually arrived at a local maxima of weights. Meanwhile, we also observed various interesting emergent behaviors that arose from combining strategies and varying their weight. Finally, we playtested these final AIs against Tomasz and other volunteers and surprisingly found that they were able to win against Tomasz in 60% of games. With this, we definitely achieved our goal of creating a strong Blokus AI capable of beating humans.

Informed Search

The first step to building our AI was to generate list of valid moves (i.e. valid positions and rotations of tiles) based on the current game board. Generating this move list efficiently proved to be quite difficult, but eventually was accomplished by limiting iteration to only available corners on the map and other optimizations that reduced time-costly loops.

With this valid move list complete, we implemented an agent that choose a random move each turn. As shown in Figure 2, we found the number of possible moves with this AI started off relatively small at the 200 range but quickly ballooned to over 650 as more corners were available to place pieces onto. Surprisingly, the number of possible moves seems to decrease in distinct steps. Though we cannot confidently determine the cause of these stepwise deductions, we hypothesize that they correspond to eliminating n-sized category of pieces, where n corresponds to the number of squares contained within the piece.

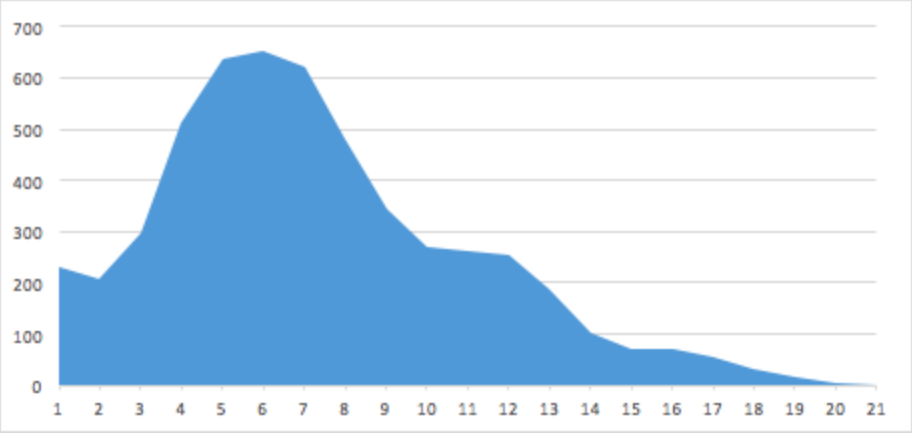

Figure 2. *Maximum observed number of possible moves per turn.*

Nonetheless, with a maximum observed branching factor of 650, we determined that any minimax algorithm would be too time costly for our game as it would

take a depth of five just to reach the next move of the agent in the search tree. Even for the endgame, where players have branching factors of around 60 a minimax algorithm would still prove to be too costly given that there are 4 players to cycle through before returning to your turn.

Defining Heuristics

With our original idea for minimax rejected, the majority of this project was focused on defining strategies (functions) and weights to create a best-playing heuristic. These strategies were determined based on our own knowledge of playing Blokus and online strategy guides. The details of each are listed below.

Biggest Piece (BP)

This strategy prioritizes moves that utilize pieces while contain many squares. Since score is directly proportional to number of tiles placed, this function directly corresponds to the winning. Furthermore, it indirectly also increases score by ensuring the smallest 1x1 piece is placed last for the extra +5 bonus.

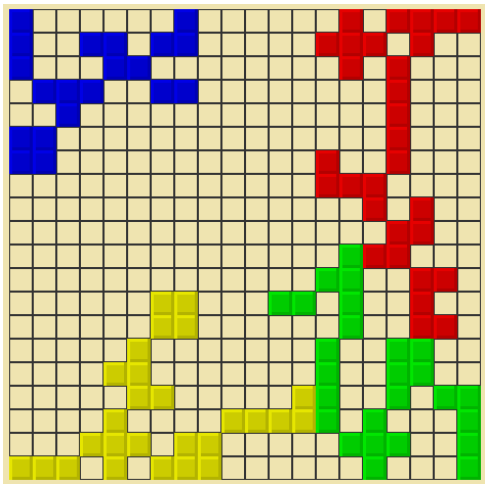

Figure 3: *The Red AI is using BP strategy, evidenced by its use of 5-sized pieces, compared to the random selection of the other AIs.*

Closest to Middle (CM)

A common theme in online strategy guides for Blokus is to immediately place your pieces towards the center to “capture” as much area as possible. This would give the player more corner options and space for future moves.

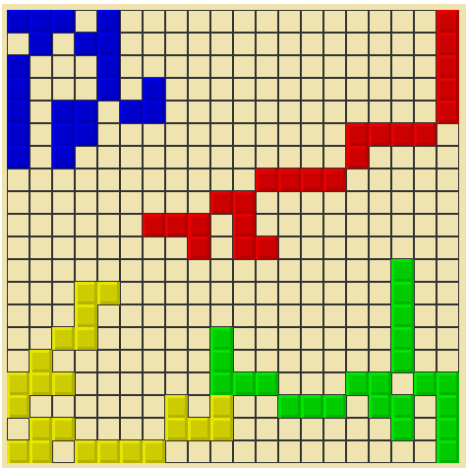

Figure 4: *The Red AI is using CM strategy, evidenced by its path to the middle.*

Adds Most Corners (AMC)

Players with more corners generally have more options as to where to place their pieces. Therefore, this strategy prioritizes moves that add many new corners onto the map, and disincentivizes occupying many corners with one piece.

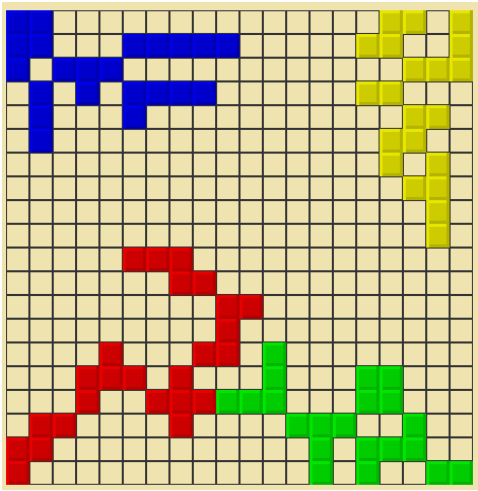

Figure 5: *The Red AI is using AMC strategy, evidenced by its use of pieces with the most corners.*

Blocks Most Corners (BMC)

Likewise, players have an incentive to prevent others from amassing too many corners to deny them options and stop their growth. This strategy's value is therefore proportional to the number of enemy corners occupied by the placement of a piece.

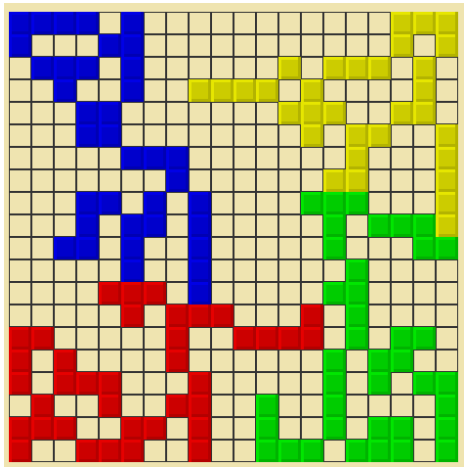

Figure 6: *The Red AI is using BMC strategy, evidenced by how the T-shape and L-shape pieces block corners Blue could use.*

Endgame Possible Moves (EPM)

Similar to *Adds Most Corners*, this strategy is based on the assumption that more options will result in a higher score. However, since calculating the total number of possible moves involves generating each possible move, this function proved to be prohibitively expensive for early to mid game. Therefore, we set this strategy to only activate on Turn 12 and return zero for all previous turns.

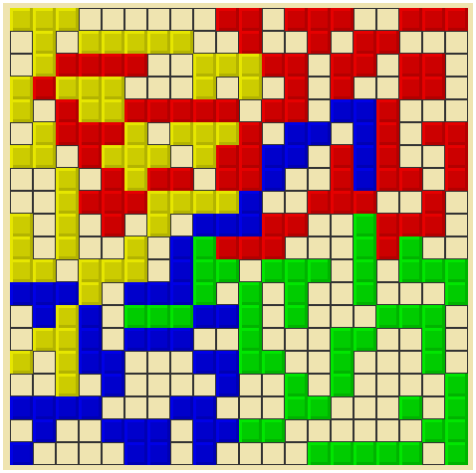

Figure 7: *The Red AI is using EPM strategy, evidenced by the placement of the 1x1 piece in the top left corner, which allowed for the placement of the 1x4 piece.*

Hill-Climbing Analysis

For each strategy above, we defined a hill-climbing algorithm that would simply choose the highest priority piece as the current move. Tomasz, our local Blokus expert, then played against three preliminary AIs that each utilized one of the strategies to evaluate their strength. We also invited some of our friends to participate and finally pitted these AIs against each other to see their results.

Strategy	% games won vs. Tomasz	% games won vs. others
Biggest Piece	0%	40%
Closest to Middle	0%	10%
Adds Most Corners	0%	40%
Blocks Most Corners	0%	0%
Endgame Possible Moves	0%	10%
Random Moves	0%	0%

Table 1. *Three AIs of single strategy were played against Tomasz and other volunteers. AI win rate was recorded with 5 sample games against Tomasz and 10 sample games against other volunteers.*

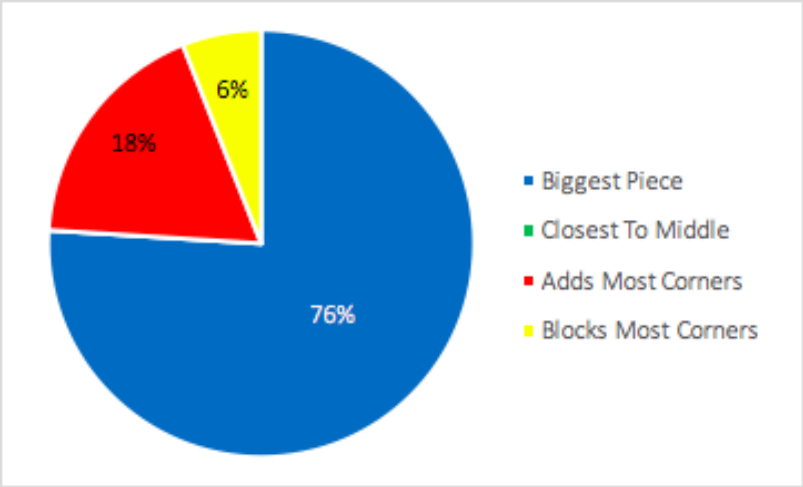

Figure 8. *BP, CM, AMC, and BMC AIs were played against each other for 50 games. The percentage of games won is displayed above.*

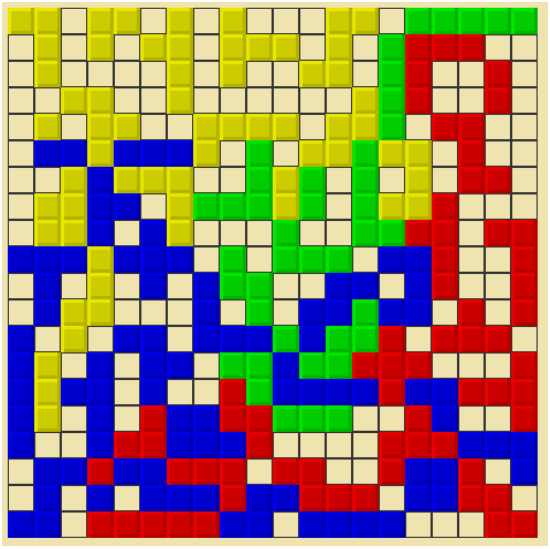

Figure 9. *End game state where Blue AI (BP) won with a max score against Red (AMC), Green (CM), and Yellow (BMC) .*

It was clear from these results that a mere hill-climbing approach to our AI that focuses on a single strategy would not be sufficient. Nonetheless, from the results of the AI vs. AI testing we found that the *Biggest Piece* strategy performed significantly better than other strategies. However, since it did not completely dominate the other strategies, we decided to implement a weighted heuristic.

Evolutionary Algorithm

We defined each of the previous strategies as a “gene” in our algorithm and initially gave each AI a random vector of weights, with each weight corresponding to a strategy. After simulating a full game between four AI’s, the AI with the greatest score was declared the winner. The winner’s weights were “mutated” by sampling from a Gaussian Distribution with the mean equal to the weights and a variance of 0.1. These new sampled weights were assigned to two of the four AI’s, to act as the offspring of the original winner. The other two AI’s are once again assigned random weights, to serve as evolutionary pressure. This update process is repeated after each simulation, leading to emergence of a few dominant strategy weighings.

Evolutionary Observations

We implemented the algorithm by adding genes in one at a time. The first two genes were *Biggest First* and *Closest to Middle*. Surprisingly we observed that both AIs with large BF weights and AIs with large CM weights behaved similarly in the first few moves as they both placed their largest pieces first. This aligns with our reasoning that larger pieces will enable players to reach the center first. The two strategies seemed to balance each other as BF’s failure to capture space was counteracted by CM’s wasteful placement of small pieces to maintain its central position, and this observation was supported by the gene’s constantly fluctuating weights.

The next gene added was *Adds Most Corners*. After simulating the evolutionary algorithm, we found although CM’s weight was drastically reduced compared to both BF and AMC, the AI behaved in a similar fashion to CM in the first few moves as it placed pieces towards the center. We concluded that by prioritizing adding corners, the AI indirectly prefers the central board as the empty space allows for more corners than pieces placed along the edge.

Conversely, most strategy guides recommend players to block their enemies corners with their pieces. However, after adding this BMC gene and running the evolutionary simulation we noticed that the BMC would typically be muted. But after manually creating an AI with a high BMC and low weights for other genes, we found that this combination would dominate the previous best contenders, usually placing first or second score-wise. This adversarial AI also lowered the average score for game from 90-100 to the 70-80 range. From this polarity we concluded that blocking corners must be an all-in strategy where the AI either fully commits to blocking opponents or it commits to improving its own board position by increasing its own corner count.

The final gene added was *Endgame Possible Moves*. We found that the presence of this gene was crucial for a well-performing AI strategy, evidenced by the fact that AIs that weighted EPM highly would consistently beat AIs that did not. We had initially hypothesized that small pieces placed in the endgame would unimportant relative to the pentominoes in the early and mid game. However, our intuition was wrong as the presence of this gene would frequently cause the AI to win by margins of less than 5.

After simulating the evolutionary algorithm, we found a local maxima of weights and play-tested 3 AIs with these weights against Tomasz and other volunteers. The final results are detailed below.

BF	CM	AMC	BMC	EPM
0.224	0.058	0.420	0.137	0.138

Table 2. *The final weights obtained at a local maxima after simulating the evolutionary algorithm 100 times.*

Win rate vs. Tomasz	Win rate vs. Others
53%	87%

Table 3. *Three AIs with using the weights from Table 2 played against Tomasz (15 times) and other novice volunteers (30 times).*

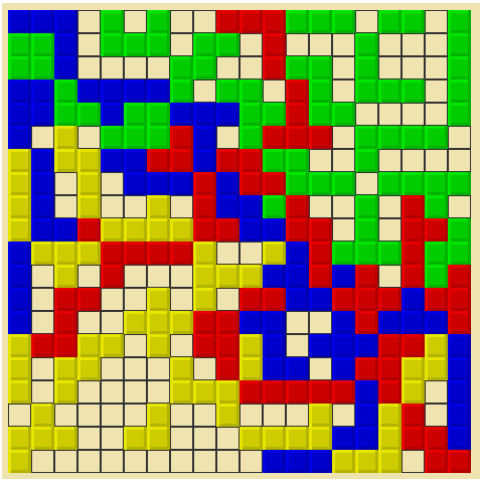

Figure 10. *End game state with Tomasz (Blue) against three final AI's. Final scores: Blue - 71, Red - 72, Green - 73, Yellow - 66*

Conclusion

Overall, we believe we succeeded in creating a sufficiently strong AI to pose a significant challenge on an experienced player. Initially, we chose a few strategies which experts and strategy guides suggested, and assigned them one at a time to the AI's. As expected, these AI's performed better than a Random AI, and even managed to beat several novice players. However, when pitted against an experienced player such as Tomasz, these simple AI's fell short.

With evolutionary programming, we managed to not only combine strategies, but also evolutionary weigh their relative importance for determining which move to pick; this created a significantly more complex and robust AI. After many simulations, we found a few AI weight vectors which kept dominating the game. By using these weights to create new AI's, we managed to beat almost all novice players and some experienced players, including Tomasz.

Further Work

Given more time and resources, we would implement a few crucial improvements for this AI project. One common problem that came up was how time (number of turns taken in the game) should affect the weights. For example, initially it is a good strategy to get to the middle/across the board (CM), but later it is better to invade other AI's corners and spread out. Ideally, we would implement time by creating a weight vector consisting of the weights of our strategies at every single time step; a more optimal way would be to divide the game into "early", "mid", and "late" stages. By doing so, the AI's overall strategy could change as the game progresses.

Another problem we encountered was the computational complexity of finding moves. Given a 20x20 grid, 21 pieces, and 4 players even a single level minimax tree takes too long to compute. With more computational power, we could perhaps consider more complex AI strategy, or at least Endgame Possible Moves (EPM) to no longer be limited to just the endgame.