Abstract

OCTI is a two-player strategy board game developed by Donald Green in 200. This project dealt specifically with the OCTI: New Edition. The game centers on a subset of artificial intelligence that deals with computer representations of traditional board games and models used to simulate intelligent game play. By modeling human thought by constructing a game tree of possible outcomes, an accompanying search algorithm traverses the game tree in search of optimal moves, which are scored by heuristic evaluators. I chose OCTI as the subject of this project because it was developed to be a computer-resistant board game. OCTI’s many nuances make it ideal for human creativity and a challenge in computational modeling. This implementation of OCTI uses the minimax search algorithm to develop a game tree and a custom heuristic evaluation function to score game board positions. The minimax algorithm is a fixed depth implementation of 2-ply.

The project sought to programmatically model the OCTI game structure as well as basic strategic play. Its command line interface for game play was constructed using Ruby.

The project’s source code is available in the CPSC 490 project directory.

Deliverables

1. A working Ruby implementation of OCTI built with a command line interface
2. A written report discussing the implementation and possible continuations.

OCTI

The game combines different game features found in other strategy games, making it difficult to computerize. Players are able to continually add pieces to the board—namely prongs, pieces that are added to the main pieces, pods, to dictate whether a pod can move on the board and in which direction each a pod can move. Players are able to capture other pieces, as in chess, by jumping them in a manner that resembles checkers.

The game is played on a 6x7 game board. Each player is given four pods, one for each of his/her bases. Players are also given 12 prongs to be used to build their pod pieces to allow mobility in different directions. Each pod is capable of holding eight prongs, allowing it to move north, northeast, east, southeast, south, southwest, west and northwest
The game is won when a player lands on an opponent’s base. See Figure 2 for example game play.

Figure 1. Octi pod piece and A-H prong locations.

Figure 2. Example of gameplay
The game tree complexity is one of the reasons why developing a program to play the game is difficult. At the start of the game, each player has 32 possible moves, prong insertions in one of the 8 prong positions of one of the 4 pods. After the first move, the average number of children at each node, or branching factor, is about 31. As the game progresses, the branching factor can grow to an average of 101. In comparison, chess has an average branching factor of 35 and Go has a branching factor of 250. OCTI’s branching factor grows due to the possibility of jump moves and their accompanying captures. To develop an idea of how large the game tree is, an estimate of game length is needed along with the branching factor. Research on the 6x7 version of OCTI has concluded an approximation of 25 moves, thus bringing the game tree size to $31^{25}$. The more advance version of the game, OCTI-X, would be even more difficult to model due to the fact that the board is larger, 9x9, and it introduces the option to stack friendly pods. This new move allows a player to combine the mobility of the stacked pods. Any or all stacked pods can move or jump simultaneously in any allowed direction. The game also provides 7 pods and 25 prongs for each player. These changes would increase both the game length and the branching factor. Legal move generation would be much more complex and time intensive, because there are more options at each state and more possible states.

**Search Algorithm**

*Although recent research has raised other possibilities, I decided the first step in development of a program was the minimax algorithm. Minimax is the algorithm of choice for most zero-sum games of perfect information, unless the branching factor is so large, that the algorithm is no longer an efficient choice. Minimax aims to minimize the possible loss from the worst-case scenario, a maximum loss. It can also be viewed as a means of maximizing the minimum gain for an opponent. Furthermore, many other decision heuristics are built off of the minimax model, which makes it an excellent gateway for future expansion. A great deal of research is focusing on ways to winnow a computer’s focus in game tree search, specifically randomly sampling promising branches of the tree and thus ignoring parts of the tree that will not lead to a successful outcome.*

*Minimax examines branches down to a fixed level, at which point it requires a heuristic static evaluation function that attempts to rate how good a position is using purely local features.* I crafted a heuristic evaluation function to score each game board position. The heuristic took into account material and position to calculate a score, placing a high value on movement towards the opponent’s base and maintaining the optimum prong and pod count. Specifically, the evaluation function gave points for each pod on the board, and each prong in the player’s possession. To encourage the strategy of moving pods in groups, points were awarded for pods in the vicinity of other friendly pods. Certain prong insertions were favorable, prongs in the direction of the opponent’s bases. Prongs in opposite directions were also favorable, because this increased mobility and allowed pods to retreat back to their base. Strategically, I felt that if a pod was not in danger of being captured, the computer player should aim to build its pieces. Friendly pods with the
ability to jump over each other also improved the score of the board position. A pod’s distance to the opponent’s base is the most influential metric in the evaluator. The minimax algorithm used the score to determine the benefit or detriment of all possible legal moves.

In the two-ply implementation of the minimax algorithm, move sequences with length two are examined. The chosen move is the worst option for the opponent. If the algorithm was to examine a game of OCTI-X, a two-ply depth search would have a difficult time finding the best option for the max player due to the fact that the game tree would be so large that the time alone would be unreasonably long for a computer game. Also, due to jumps on a larger board and stacks, it is difficult to compute the best move choice since the computer does not have an understanding of the positions generated by these options. This implementation would not be effective in predicting outcomes from such a narrowed perspective.

Conclusion & Further Work

The project succeeded in creating a legal OCTI simulation able to play against and a human opponent. A continuation of the project could seek to make the computer more competitive by allowing it to analyze specific cases that involve possible captures. A quiescence heuristic could be added to the minimax algorithm to expand the computer’s knowledge of proceeding game positions. The quiescence heuristic keeps examining branches as long as a capture is possible. Otherwise, a move to which the opponent can respond with a move that puts it in jeopardy of being captured could be underrated. Further steps might include the implementation of alpha-beta pruning, an expansion of the minimax algorithm that could potentially enhance the speed of the computer’s analysis by reducing the number of positions the computer examines.
1. This image depicts the beginning of the program. Moves are written in OCTI notation, the horizontal location x and the vertical location y of a pod are represented as xy. Prongs are designated with letters A-H (see figure 1). Prong insertions are represented as xy + prong letter. For example, 42 + H, represents a prong inserted in the northwest position of the pod located at (4, 2) square of the board. Moves from a starting position to a new destination are represented as xy(start) – xy(destination). Starting and ending locations are separated by dashes. A jump includes the starting location, any intermediate steps,
and the destination. Captures are represented with an “x” next to the destination coordinates immediately after the capture position.

After the human player added prong H to pod 42, the computer added a prong to position E of pod 26. Building pieces is the first step to every OCTI game.

2. Human player adds prong A to pod 42. Computer moves pod 26 to position 25. Now that the computer has a pod that is mobile, it choses to move its first piece, because a
decreased distance between a player’s pod and an opponent’s base is scored highly in the heuristic evaluator.
4. Human player adds prong A to pod 22. Computer adds prong D to pod 24. The computer avoids a potential capture of its pod by adding a prong instead of moving forward. It chose to add prong A, so the pod could move in the reverse direction, this maneuver is scored highly because if a pod gets close enough to an occupied opponent base with the ability to jump over and capture that pod, the ability to move in the reverse direction will provide a win. Mainly, this improves pod mobility.
7. In turn 8, human player adds prong E to pod 23. Computer jumps pod 14, lands on position 23 and continues jump to position 32, the human player’s base, thus winning the game.
Work Cited
