Finding Shortest Move Count in Shotgun King by Best-First Search

Kara Bora Kara Author
TOBB ETU

***Abstract*-These instructions explain how to calculate the shortest move count in Shotgun King using Best-First Search algorithm, and how to determine proper weights of heuristic data.**

I. Introduction

The purpose of this writing is to develop an Artificial Intelligence (AI) that can complete the Shotgun King game in the shortest possible move.

Shotgun King Game Description

Shotgun King is a chess-based game played with chess pieces, In Shotgun King each piece has a health and turn count. The health indicates how much damage a piece can take, and the turn count indicates how often piece will play. Player plays with black king and the objective of the game is to kill the white king. Ref [1] show the gameplay of Shotgun King.

Own Shotgun King Game Design

As can also be seen from Ref [1], in Shotgun King, the pieces' health, turn counts, gun damage and gun range are predetermined. In addition, chess piece amounts are random. In order to facilitate optimal testing and observation of the AI, I have implemented all parameters to be adjustable by the user.

Additionally, to allow testing of the AI in different positions beyond Shotgun King. I have implemented a system that accepts a string in the FEN notation format and starts the game at the desired positions.

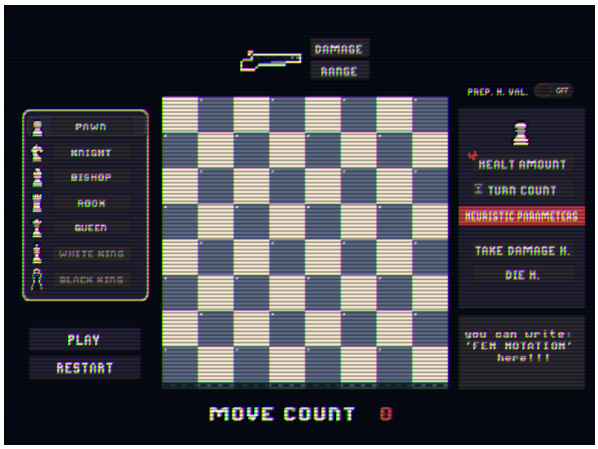

Fig. 1 Own Shotgun King Game Menu

On the left side of Fig. 1, the numbers of the chess pieces can be adjusted.

On the right side of Fig. 1, the health amount and turn count of the chess piece can be adjusted.

On the bottom right of Fig. 1, FEN notation can be written.

II. Game Artifical Intelligence

A. White Piece Artificial Intelligence

Before designing the AI for the black king, we need to consider the best move that the white pieces can play against the black piece. When considering that the goal of the white pieces in the game is also to capture the black king, we can see that the definition of the best move for white pieces is formulated as follows:

Try to capture the black king?
Try to check the black king?
Try to narrow the black king's area?
Evade the black king.

When the AI receives a "yes" answer to one of the above questions, it makes the appropriate move accordingly. Unlike other pieces, the white king has only one move to "evade the black king" in order to protect itself from the black king.

Before considering how white pieces' AI makes the right move, we need more information about the system. We can start by performing a Performance measure, environment, actuators, and sensors (PEAS) analysis of the system.

TABLE I
White Pieces PEAS
Analysis

Agent	White Pieces
Performance Measure	Capture black king
Environment	Chessboard & Other Pieces
Actuator	Moving & Capturing
Sensor	Square information

Environment types: Fully observable, strategist, sequential, discreate, multi-agent.

Since the system is fully observable, the white pieces know the position of the black king, allowing them to examine the current state of the board on the position of other pieces before making a move. This enables the system to make the best move possible based on the board's current state.

B. Black King Artificial Intelligence

The AI approach of the black king is completely different from that of the white piece because its goal is not to find the best move ahead, but rather to find the shortest path to victory until the end of the game. Therefore, the AI approach of the black king is not as simple as selecting the best move one move ahead, which is the case for the white pieces.

The artificial intelligence approach for the black king involves the idea of reaching the goal as quickly as possible, which requires testing all possible options and finding the shortest path to victory. However, considering that the sum of different movement paths for each of the black king's moves, and capture any piece's probabilities is b, the number of possibilities will experience an exponential growth of b to the power of maximum depth n (1). This results in such a large number of possibilities that it cannot be tested.

$$\textit{Worst Case: } O(b^n) \tag{1}$$

The Since Shotgun King is a chess-like game, when examining the chess AI in Ref [2], there are unpredictable possibilities for both black and white player much more than Shotgun King, and the chess AI tries to determine the next best move by looking ahead n moves using alpha-beta pruning. There is no goal of winning in the shortest move in chess AI, but even if there were, calculating it with current computers would take a very long time. Since the value of n should be equal to the number of moves to be played until the end of the game, not setting a specific limit for n makes it impossible to calculate it in short time, even for Stockfish, which can calculate up to 30-50 moves ahead.

It can be noticed that compared to chess, there are very few possibilities left when only the black king's moves need to be tested for the AI of the Shotgun King game. In this regard, a more result-oriented approach such as the best-first search algorithm would be more appropriate than alpha-beta pruning for testing these possibilities. Furthermore, the fact that the moves of the white pieces occur in fixed manner contradicts the two-player logic of alpha-beta pruning. In our case, the black king should be able to choose the best move towards its goal according to the changing environment.

When using Best-First search algorithm with an informative heuristic function, finding the solution in the shortest possible move (?) can be computed (2) time complexity, where b is the number of all possible moves for the black king, and n is the length of the shortest path. However, this complexity is only valid for worst-case scenarios, and real-world problems often have more optimized. Therefore, after the best-first search algorithm, the possible options become observable and can be analyzed.

$$\textit{Worst Case: } O(b^n * \log b^n) \tag{2}$$

III. Calculating Heuristic Parameters

Creating a heuristic function that is informative is crucially important, and it should be constructed in best possible way.

In Ref [2], when creating the alpha-beta pruning heuristic function for chess artificial intelligence, the values of the chess piece were used. The same approach was taken for creating the heuristic function for black king's AI, where the damage dealt, and the piece captures were determined based on the piece's point value. The white king's damage dealt, and capture points are much higher compared to other pieces because the objective is to capture the white king.

TABLE II
White Pieces
Heuristic Values

White Pieces	Take Damage Heuristic	Captured Heuristic
Pawn	1	3
Knight	2	4
Bishop	2	4
Rook	3	5
Queen	4	6
White King	15	1000

Furthermore, one of the main conditions for capture the white king is to approach it. Therefore, an addition has been made to the heuristic function that will reward the AI for getting closer to the white king. Value of heuristic function after a move by the black king. (4)

$$h(n) \mathrel{+}= k * (8 - \textit{distanceBySquare (white king, black king)}) \tag{3}$$

In addition, in order to reach the result in the shortest move, each move should have a negative effect on the heuristic function.

How does the search algorithm work in the Shotgun King game that I have implemented? Before each move, the black king generates all possible moves it can make and takes a screenshot of the resulting board state. These screenshots become the nodes of tree, with each node storing information such as the position of all pieces, references to the pieces themselves, the depth of the tree, and the total heuristic value up to that point. These nodes are then placed into a priority queue and stored in descending order based on their heuristic values. When the black king makes a move, the first element of the priority queue is popped, and the chessboard is updated to match the state of popped node. The process is repeated until the game is won.

IV. Shortcomings of Best-First Search

A. *Optimality Problem*

Up until this point in the writing, the Best-First search algorithm has been described as an efficient algorithm that can quickly find the shortest path to winning in Shotgun King using an informative heuristic function. However, upon further research and consideration Ref [3], we can see that the Best-First search algorithm may not always provide the shortest possible solution. While it can generate outputs that are very close to the correct solution according to the given heuristic function, it may not always provide the best possible result when we expect the optimal solution. Therefore, using the Best-First search algorithm alone is not enough to find the shortest path to winning Shotgun King, and we need to try all possible scenarios. As mentioned earlier in the writing, there are too many possibilities to test without any pruning. Therefore, using the Best-First search algorithm to generate the first solution and pruning other possibilities based on the generated results may lead us to the most accurate outcome.

As a result, the shortest path to win the game can be found by testing the game through a manageable number of possibilities by pruning with best-first search algorithm.

B. *Completeness Problem*

Best-first algorithm is not complete. Therefore, there is a possibility of infinite loops when reaching the solution. Furthermore, after each move, the previously played moves are removed from the priority queue and added to the list of played moves. Before each move, it is checked whether the move has already been played or not to prevent an infinite loop.

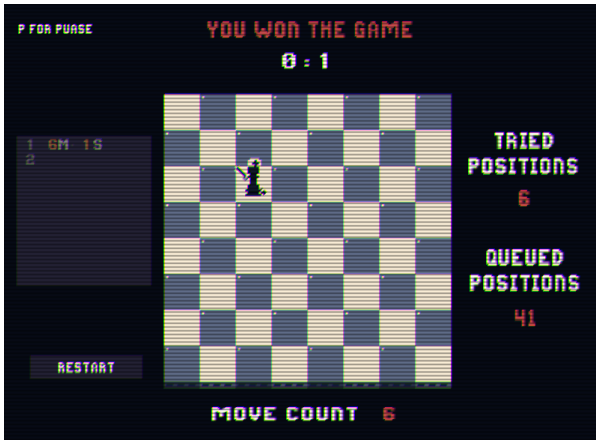

Fig. 2 End of the black king vs white king position with predetermined heuristic parameters

On the left side of Fig. 2, a board is available showing the number of moves that a successful best-first search algorithm took to reach the solution and the amount of time it took to reach that solution since the start of the game. As seen at the end of the game, the board displays “6M 1S”, indicating that the white king was captured in 6 moves and this process took 1 second to compute.

On the bottom side of Fig 2, “Move Count” represents the depth of the move played in the tree.

On the right side of Fig. 2, there are two different outcomes, where “Tried Positions” indicates the total number of moves played, while “Queued Positions” shows the total number of all possible moves that have been added to the priority queue until that point.

V. Testing Heuristic Function

As we mentioned before, the heuristic function plays a crucial role in providing the best result in the shortest time for the best-first search algorithm. We can examine the importance of this by looking at a few examples.

Looking at the parameters for the black king in the game menu, we can see three different heuristic parameters: initial heuristic value, which represents the heuristic value that the black king will have at the beginning of the game, decrease heuristic value, which indicates how much the heuristic value will decrease for each move made by the black king, and heuristic multiplier, which represents the multiplication factor for the heuristic value gained from a move. The constant value of k in the third equation is represented by this parameter.

On the top right of Fig. 1, There is a toggle for predetermined heuristic values.

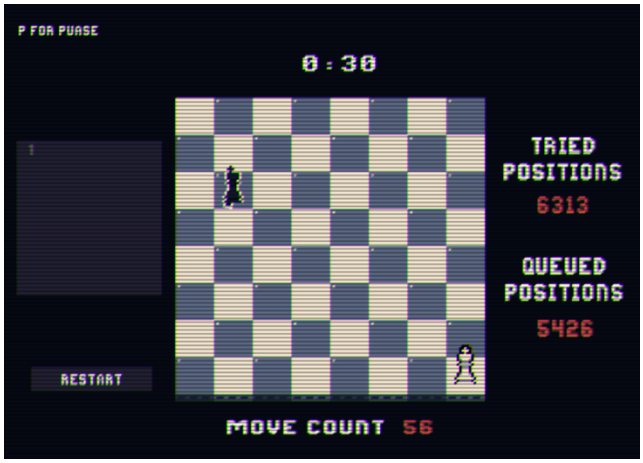

Fig. 3 30th second of the black king vs white king position without any heuristic parameters

As seen from Fig. 3, when all heuristic parameters are set to 0, the heuristic value of all possible moves becomes equal. Therefore, the black king plays random moves among them. Even though the tree is traversed up to the 56th depth, it fails to find a solution, and the black king continues to make empty moves if the search is continued.

A. The Effect of the Decrease Heuristic Parameter

We knew that the decrease heuristic parameter is the value that is subtracted from the heuristic function every time the black king makes a move. But what is its effect on the AI? When the decrease heuristic value is increased linearly, the total heuristic value will decrease at the end if each move, so the AI will prefer to finish previous moves first before going deeper. This will turn the search AI into a Breadth-First Search (BFS) algorithm.

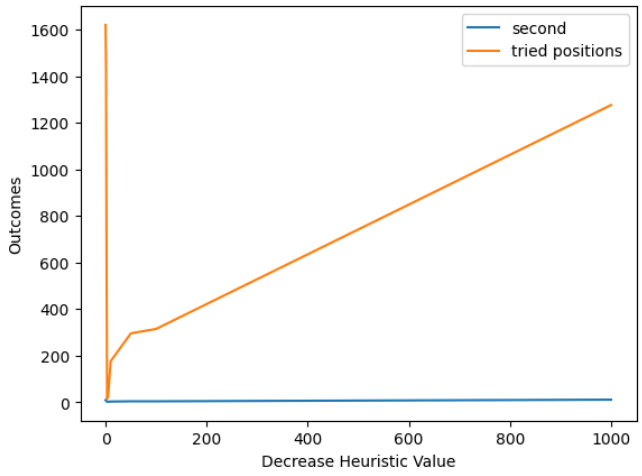

Fig. 4 The effects of linear increase in the decrease heuristic value

As seen from Fig. 4, there is an optimal value for the decrease heuristic parameter. When the decrease heuristic value is close to 0, there is no decrease in the total heuristic value after each move of the AI, causing It to follow a single path instead of trying other paths. On the other hand, when we linearly increase the decrease heuristic value above its optimal value, the number of tried positions also increase correspondingly, and the algorithm begins to show the characteristics of BFS by trying all unnecessary positions.

B. The Effect of Heuristic Multiplier

We have seen that the heuristic multiplier is the coefficient of the equation (3) that determines the point the black king will gain as it approaches the white king with each move. When this value is linearly increased, the black king will prefer to approach the white king rather than capturing the white pieces around it. Even if it is valuable to move without capturing the white king, the game will become unwinnable, and the black king will prefer only to move around the white king.

VI. Possible Ways to Further Improve Artifical Intelligence

A. Pre-determining the Minimum Number of Moves

We apply pruning of the millions of possible positions that can be generated during the best-first search, which helps us to set a depth max limit for our search tree. In addition to this, we decided on a lower limit before the game. How does it affect the game when AI stops searching as a result of reaching the lower limit?

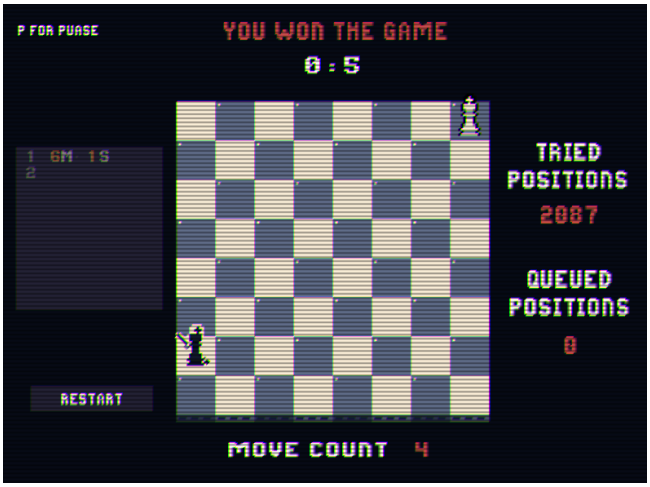

Fig. 5 End of the black king vs white king position with predetermined heuristic parameters without minimum move limit

Through the minimum move limit calculated at the beginning based on the starting positions of the black and white king and the range of the black king's gun range, in Fig. 2, the game can be won within 1 second by trying only 6 positions. However, in Fig. 5, in order to win the game, 2087 positions had to be tried, which took 5 seconds.

We can infer that Fig. 5 has tried all the possibilities from the fact that the queued position is 0.

As a result, setting a minimum move limit has a significant impact on AI's ability to provide results earlier.

B. Not Evaluating Positions Where the Black King is Captured

During the best first search, nodes are also created for the positions where the black king is captured. When the black king is captured and removed from queue, it becomes certain that it has been captured, and the algorithm continues to try other paths. However, what would be effect on the AI if we prevent it from making those moves by pre-determining the positions where the black king is captured and not playing those moves? In addition, we need to consider the costs of extra checks to be made while making this improvement.

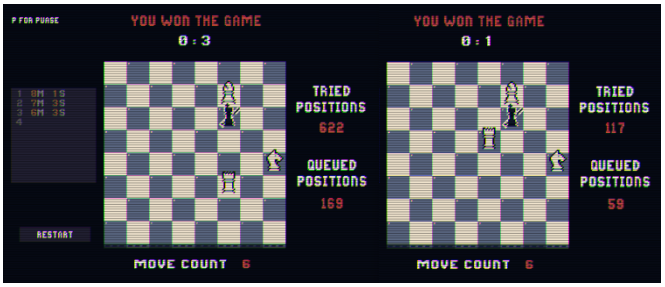

Fig. 6 End of the position

Fig. 7 End of the position with pre-determined black king capture case

By keeping track of which squares on the board are being threatened by white pieces after their moves, and making black king moves accordingly, we are handling capture positions without creating their corresponding nodes. This results in a more efficient AI compared to the previous version, as seen in the comparison between Fig. 6 and Fig. 7, with the new AI working faster and using less space.

REFERENCES

[1] https://www.youtube.com/watch?v=BGLWULPIffI
[2] https://www.youtube.com/watch?v=U4ogK0MIzqk&t=980s
[3] https://www.geeksforgeeks.org/best-first-search-informed-search/
[4] https://en.wikipedia.org/wiki/Best-first_search#:~:text=The%20A*%20search%20algorithm%20is,estimated%20distances%20to%20the%20goal.