Khulna University of Engineering & Technology

Department of Computer Science and Engineering

CSE 4110: Artificial Intelligence Laboratory

Project Name : **Four consecutive key in grid's cell**

Presented By:

Name: Anik Mahmud

Roll : 1707107

Date : 03/07/2022

Contents

Abstract

A simple command-line version of the game "Four consecutive key in grid's cell."A game for 2 players on a 7x6 grid game board.Players take turns dropping their game piece onto the grid by picking a column number 1-7 (inclusive).The game pieces fall to the lowest position available on the grid for the column they selected.The first player to reach 4 pieces aligned in a row, wins. Rows can be any direction: horizontal, vertical, diagonal up, diagonal down.Choosing a column off the game board grid will prompt the player to re-pick.

1 Introduction

This is a centuries-old game even played by Captain James Cook with his officers on his long voyages. Milton Bradley (now owned by Hasbro) published a version of this game called "Four consecutive key in grid's cell" in 1974. It is also called "Four-in-a-Row" and "Plot Four." Two players play this game on an upright board with six rows and seven empty holes. Each player has an equal number of pieces (21) initially to drop one at a time from the top of the board. Then, they will take turns to play and whoever makes a straight line either vertically, horizontally, or diagonally wins. With the constant evolution of technology, it comes as no surprise that what seemed like a brilliant source of entertainment back then is slowly fading into the background with a shift towards digital games. The emergence of computer and mobile games, offering a variety of gaming genres have taken over the need for physical board games such as Monopoly, UNO,Four consecutive key in grid's cell, Scrabbles, etc. Though some games managed to stay relevant in this day and age, namely the ones with multiple players option, others fade to grey. Classic physical games such as "Four consecutive key in grid's cell"and Chess are slowly losing their place in the society, hence it is necessary to computerize the core elements of the games to recreate them in a virtual environment. As such, this would serve as an effort to preserve the continuity of these games in the future and hopefully resuscitate the fun and joy it once bought to the society "Four consecutive key in grid's cell"is a classic two-player board game, each represented by yellow and red pieces respectively on a board matrix of seven rows and six columns. Players are free to place their piece alternately at any available positions, constricted to the six columns by dropping it into their desired.

"Four consecutive key in grid's cell"is a two player ion game on a 6x7 board. The goal of the game is to strategically insert a disk in one of the seven columns giving you a higher chance to "Four consecutive key in grid's cell" cells by row, column, or diagonal. Players alternate turns."Four consecutive key in grid's cell"is a solved game. With perfect play, the first player always wins.

2 Problem Statement

The game is categorized as a zero-sum game. Therefore, the minimax algorithm, which is a decision rule used in AI, can be applied. The project goal is to investigate how a decision tree is applied using the minimax algorithm in this game by Artificial Intelligence.

2.1 Import the necessary Libraries for the "Four consecutive key in grid's cell"Game

There are some library to implement our Artificial Intelligence project.Basically all these libraries are **Python** Libraries and bellow we will discuss these libraries shortly.

2.2 NumPy module

NumPy stands for Numerical Python. NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. It is an open source project and you can use it freely. NumPy is a Python library that provides a simple yet powerful data structure: the n-dimensional array.

If you don't have NumPy already pre-installed on your system, type the following command in your window's cmd: **pip install numpy**
When you call the statement import numpy as np , you are shortening the phrase "numpy" to "np" to make your code easier to read. It also helps to avoid namespace issues.

2.3 Pygame module

Pygame is a free and open-source cross-platform library for the development of multimedia applications like video games using Python.

It uses the Simple DirectMedia Layer library and several other popular libraries to abstract the most common functions, making writing these programs a more intuitive task.

If you don't have Pygame already pre-installed on your system, type the following command in your window's cmd: **pip install numpy**

2.4 Python sys module

The python sys module provides functions and variables which are used to manipulate different parts of the Python Runtime Environment. It lets us access system-specific parameters and functions. import sys. First, we have to import the sys module in our program before running any functions. sys.modules.

2.5 Python math module

Some of the most popular mathematical functions are defined in the math module. These include trigonometric functions, representation functions, logarithmic functions, angle conversion functions, etc. In addition, two mathematical constants are also defined in this module.

If you don't have math already pre-installed on your system, type the following command in your window's cmd: **pip install maths**

2.6 Implementing "Four consecutive key in grid's cell"Game in Python

2.6.1 Step 01

Import the NumPy package as np. Then we will create a python function named **create_board()**.

np.zeros() function is used to create a matrix full of zeroes. This function in Python can be used when you initialize the weights during the first iteration in TensorFlow and other statistic tasks. ((6,7)) are the dimensions. 6 rows and 7 columns. Then we just **return** that board.We will begin writing the main game loop now. We're going to create a loop as **while not game _over**. A while not loop repeatedly executes the body of the loop until the condition for loop termination is met. Our loop is going to run as long as this game _over variable is false. We will initialize the game _over as False. The only time its going to switch to true is if someone gets 4 circles in a row.To increase the turn by 1 we will use **turn += 1**. To make it switch between player 1 and 2 alternatively, we use **turn = turn % 2**.

2.6.2 Step 02

In the step 02, we make a few alterations and updates to the previous code.

We want the **selection variable** to actually drop a piece on the board. For that, the first thing we will do is create another three functions called **def drop_piece(), def is_valid_location( ), def get_next_open_row()**.

How these functions are going to Work together is as follows, the player will make a selection. The **(0-6)** in the code represents the column where they want to drop their piece. Hence we update the name of **selection variable** to **column variable (col)**.

Now we will take this col in the current board that we have and pass it as parameters in all the three functions with board.

We will initialize global variables called **ROW_COUNT** and **COLUMN_COUNT**. In Python, a variable declared outside of the function or in **global** scope is known as a **global variable**. This means that a global variable can be accessed inside or outside of the function.

The **np.flip()** function reverses the order of array elements along the specified axis, preserving the shape of the array.

2.6.3 Step 03: Complete Code Walkthrough

In step 03, we will create a game with a GUI and not just the one with the matrices. The above code along with the new modifications that we do will make the game look like an actual board game.

First we will import all the necessary libraries.

Next we will define the colours **blue, black, red and yellow** as global static variables. These values are going to be **rgb** values.

We will initialize global variables called **ROW_COUNT** and **COLUMN_COUNT.** Number of rows are 6 and number of columns are 7.

Then we create 5 functions named **create_board(), drop_piece(), is_valid_location(), get_next_open_row() and print_board()**.

Then we create a function called **winning_move()** and we check for horizontal locations to win, vertical locations to win, positively and negatively sloped diagonals to win.

In **horizontal and vertical locations,** we create a nested for loop for the rows and columns and check an if condition statement to see if the piece has been dropped to that location on the board. If the if condition is satisfied, it will return **TRUE**. We will repeat the same procedure for vertical locations, positively and negatively sloped diagonals as well.

In the function def **draw_board()**, pygame.draw is a module for drawing shapes.

pygame.draw.rect is used to draw a rectangle. Now we will define the triangle. Define the height and width and the position.

So the position is going to be, c*SQUARESIZE and the position of the y-axis is going to be **r*SQUARESIZE+SQUARESIZE**.

Height and width are going to be the other two parameters and thats just going to be **SQUARESIZE, SQUARESIZE**. Repeat the same procedure for circles as well.

3 Winning Strategy

3.1 Place in the middle column

If the player can play first, it is better to place it in the middle column. Since the board has seven columns, placing the discs in the middle allows connection to go up vertically, diagonally, and horizontally. In total, there are five possible ways.

3.2 Make traps for the opponent

One typical way of not losing is to try to block the opponent's paths toward winning. For example, preventing the opponent from getting a connection of three by placing the disc next to the line in advance to block it. This strategy also prevents the opponent from setting a trap on the player.

3.3 Make a "7."

A 7 trap is a name for a strategic move where one positions his disks in a configuration that resembles a 7.

With three horizontal disks connected to two diagonal disks branching off from the rightmost horizontal disk. The 7 can be configured in any way, including right way, backward, upside down, or even upside down and backward. This disk formation is a good strategy because it gives players multiple directions to make a Four consecutive key in grid's cell.

4 Logic and Mathematics behind Four consecutive key in grid's cell

4.1 Decision Tree

A Decision tree is a tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. Decision trees can be applied in different studies, including business strategic plans, mathematics studies, and others. In addition, since the decision tree shows all the possible choices, it can be used in logic games like "Four consecutive key in grid's cell"to be served as a look-up table.

4.2 Decision tree in "Four consecutive key in grid's cell"

When the game begins, the first player gets to choose one column among seven to place the colored disc. There are 7 columns in total, so there are 7 branches of a decision tree each time. After the first player makes a move, the second player could choose one column out of seven, continuing from the first player's choice of the decision tree.
Notice that the decision tree continues with some special cases. First, if both players choose the same column 6 times in total, that column is no longer available for either player. It means that their branches of choice are reduced by one. Second, when both players make all choices (42 in this case) and there are still no 4 discs in a row, the game ends as a draw, and the decision tree stops. Finally, if any player makes 4 in a row, the decision tree stops, and the game ends.

5 Minimax algorithm

Minimax algorithm is a recursive algorithm which is used in decision-making and game theory especially in AI game. It provides optimal moves for the player, assuming that the opponent is also playing optimally. For example, considering two opponents: Max and Min playing. Max will try to maximize the value, while Min will choose whatever value is the minimum. The algorithm performs a depth-first search (DFS) which means it will explore the complete game tree as deep as possible, all the way down to the leaf nodes. The algorithm is shown below with an illustrative example.
Initially, the algorithm generates the entire game tree and produces the utility values for the terminal states by applying the utility function. For example, in the below tree diagram, let us take A as the tree's initial state. Suppose maximizer takes the first turn, which has a worst-case initial value that equals negative infinity. Then,

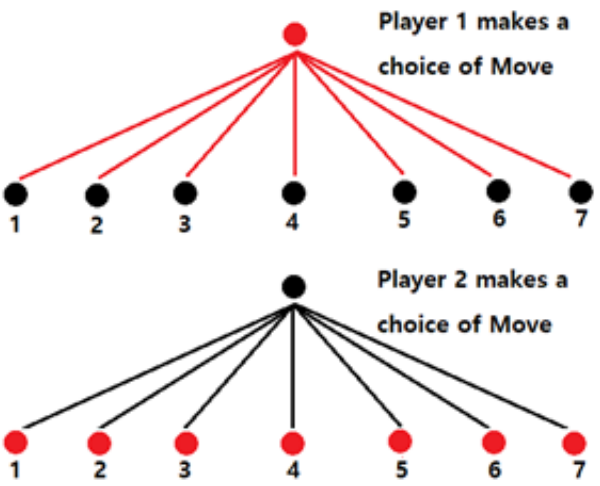

Figure 1: Possible moves for each iteration of the "Four consecutive key in grid's cell" game shown in the decision tree

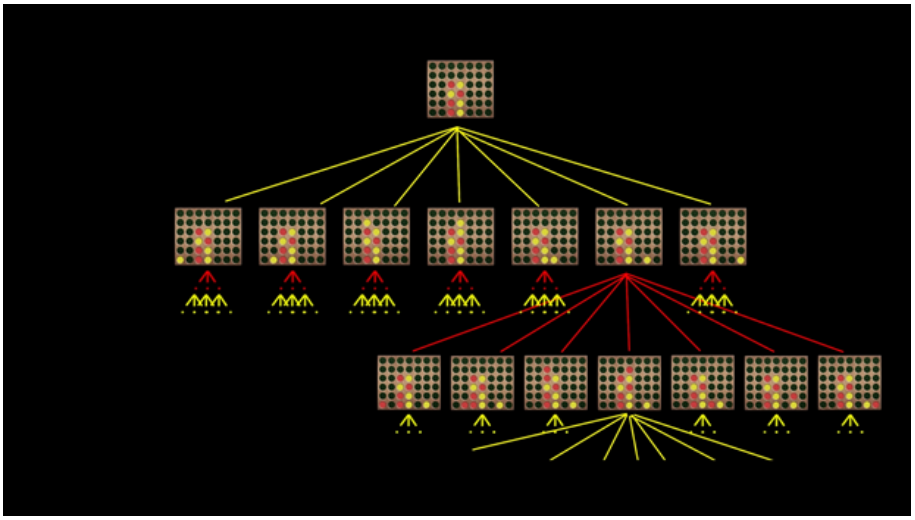

Figure 2: Decision tree of "Four consecutive key in grid's cell" possible moves

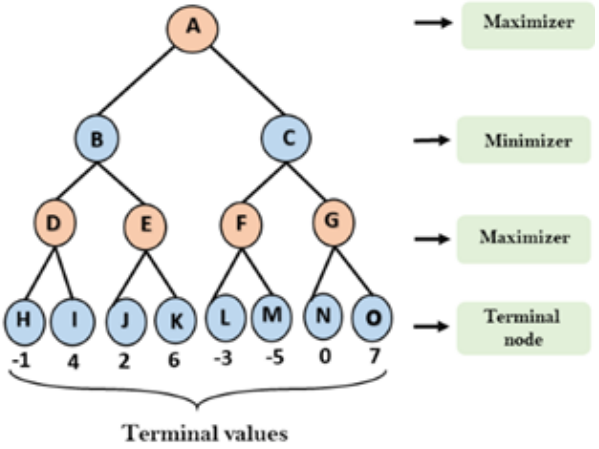

Figure 3: Minimax algorithm in tree format — the initial step

the minimizer will take the next turn, which has a worst-case initial value that equals positive infinity. First, we consider the Maximizer with initial value = -∞. Each terminal node will be compared with the value of the maximizer and finally store the maximum value in each maximizer node. Take the third row (Maximizer) from the top, for instance.

1. For node D max(-1, -∞) $\rightarrow$ max(-1,4) = 4
2. For Node E max(2, -∞) $\rightarrow$ max(2, 6) = 6
3. For Node F max(-3, -∞) $\rightarrow$ max(-3,-5) = -3
4. For node G max(0, -∞) $\rightarrow$ max(0, 7) = 7

Next, we compare the values from each node with the value of the minimizer, which is +∞.

1. For node B= min(4, 6) = 4
2. For node C= min(-3, 7) = -3

Finally, the maximizer will then again choose the maximum value between node B and node C, which is 4 in this case. Hence, we get the optimal path of play: A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ I.

1. For node A max(4, -3)= 4

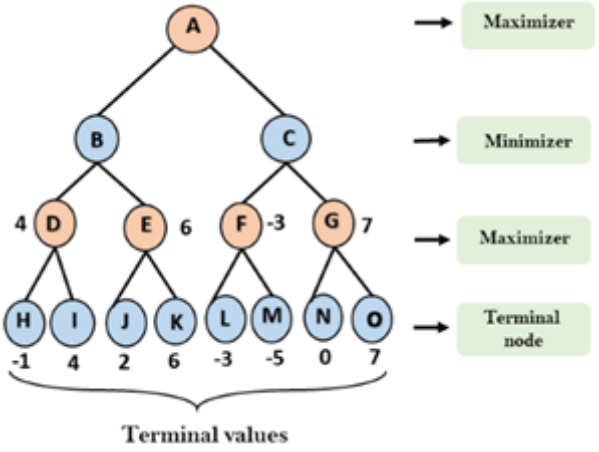

Figure 4: Minimax algorithm in tree format — second step

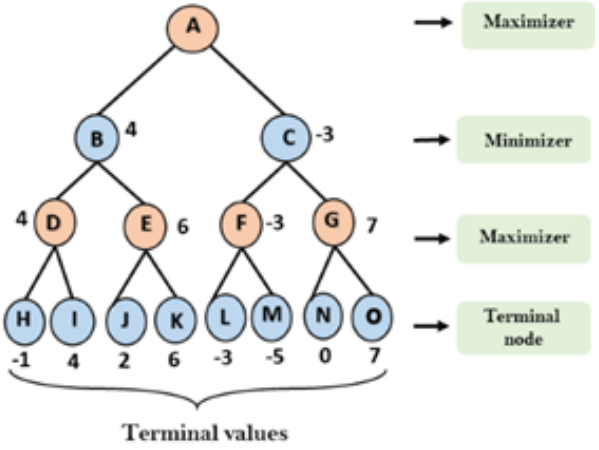

Figure 5: Minimax algorithm in tree format — third step

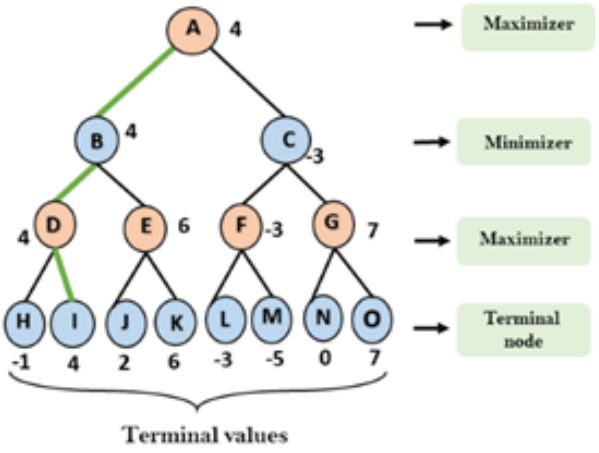

Figure 6: Minimax algorithm in tree format — the final step

5.1 Minimax with alpha-beta pruning

If we apply alpha-beta pruning to a standard minimax algorithm, it returns the same move as the standard one, but it removes (prunes) all the nodes that are possibly not affecting the final decision.

Let us understand the intuition behind this first and then we will formalize the algorithm. Suppose, we have the following game tree: In this case,
Minimax Decision = MAX{MIN{3,5,10}, MIN{2,a,b}, MIN{2,7,3}
= MAX{3,c,2}
= 3
Here one important thing is that,How could we calculate the maximum with a missing value? Here is the trick. MIN{2,a,b} would certainly be less than or equal to 2, i.e., c<=2 and hence MAX{3,c,2} has to be 3.

The question now is do we really need to calculate c? Of course not.

We could have reached a conclusion without looking at those nodes. And this is where alpha-beta pruning comes into the picture.

6 Visualization

Here Alpha-beta pruning is a used which is a modified version of the minimax algorithm to reduce the time complexity.It's generally follow cut of search to reduce the complexity of time. Alpha-beta pruning can be applied at any depth of a tree, and

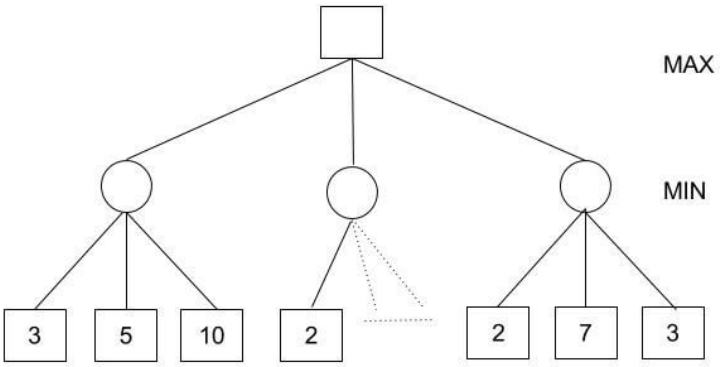

Figure 7: Figure shows alpha beta pruning method

sometimes it not only prune the tree leaves but also entire sub-tree.
It can be of two types:
Worst ordering: In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is O(b^m).

Ideal ordering: The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is O($b^{m/2}$).

Here in each node of decision tree,decision is taken which state it will follow and which state it will discard.Decision tree's depth is a very important factor here.If we increase the decision tree's depth the AI can able to analyze more numbers of state and take decision more wisely.So the AI's capability will increase,if the depth of tree increased.If we take decision tree with less number of depth,then AI's capability will be decreased,as it can able to look less amount of nodes.If we increase the depth,then it will expand exponentially more branches,so the time complexity will increase and AI's capability will also increase.As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half.Here are some wining state with taking different depth levels.

7 Conclusions & Discussions

We came to the conclusion that AI wins by being logical and having a ton of information after counting how many times it has defeated human players in this game. The AI player in this research employs a minimax algorithm to check for optimal moves in advance in order to outperform human players by being aware of every conceivable action in advance. It's interesting to note that the AI player may perform worse when the number of depths at the minimax function is adjusted from high (6, for example) to low (2, for example). However, it has been demonstrated that the method and technique used in this project work and produce spectacular outcomes.
Each node in the decision tree decides which state it will keep in consideration and which state it will disregard. The depth of the decision tree is crucial in this case. The decision tree's depth can be increased so that AI can assess more states and make more informed decisions. As a result, if the depth of the tree expanded, AI capability would also improve. The capabilities of AI will be reduced if we use a decision tree with fewer depths because it can only look at a smaller number of nodes. The depth will expand exponentially as we increase the number of branches, which will raise the time complexity and the power of AI. The number of game states that the minimax search method must investigate is exponentially correlated with the depth of the tree, as we have seen.

8 References

1. V. K. BC, N. Jashank, and M. S. Nadiger, "Alpha-Beta Pruning–A streamline approach for perceptive game playing," International Research Journal of Modernization in Engineering Technology and Science, 2(6), pp. 1306–1318, June 2020.

2. A. M. Sarhan, A. Shaout, and M. Shock, "Real-time matching of four cells game using artificial intelligence," Journal of Computer Science, 5(4), pp. 283-289, 2009.

3. E. Alderton, E. Wopat, and J. Koffman, "Reinforcement learning for "Four consecutive key in grid's cell"," January 2019.

4. J. Hernandez, K. Daza, and H. Florez, "Alpha-Beta vs Scout algorithms for the Othello game," In CEUR Workshops Proceedings, vol. 2846, pp.2019.