Computer Educable Noughts and Crosses Engine (CENACE)

Author: Aaban Saad Preyanta
NSU ID: 2233389642
Bachelor of Computer Science and Engineering
Department of Electrical and Computer Engineering
North South University, Dhaka, Bangladesh

What is CENACE?

CENACE: Computer Educable Noughts and Crosses Engine is an artificially intelligent system that can play noughts and crosses (tic-tac-toe) against a human opponent.

How does it work?

CENACE learns to play noughts and crosses (tic-tac-toe) by playing it repeatedly against another player, fine-tuning its technique each time, until after having played a certain number of games it becomes nearly perfect and its opponent can only draw or lose against it. Similar to how a kid learns, the process of learning is called reinforcement learning, which involves being "punished" for losing and "rewarded" for drawing or winning.

History and origin

The name CENACE originated from the 1961 MENACE* reinforcement learning model by Donald Michie. I have programmed CENACE as a computer simulation of Michie's MENACE. Instead of matchboxes, CENACE uses folders and text files to keep track of its moves.

*MENACE: The Matchbox Educable Noughts and Crosses Engine was a mechanical computer made from 304 matchboxes designed and built by artificial intelligence researcher Donald Michie in 1961. It was designed to play human opponents in games of noughts and crosses (tic-tac-toe) by returning a move for any given state of play and to refine its strategy through reinforcement learning. (Wikipedia)

Interface of CENACE

After CENACE has loaded the user will see a menu like this:

1. Auto Train CENACE
2. Play
3. Rules
4. About
5. Exit

→

User can press the corresponding number to go to that menu.

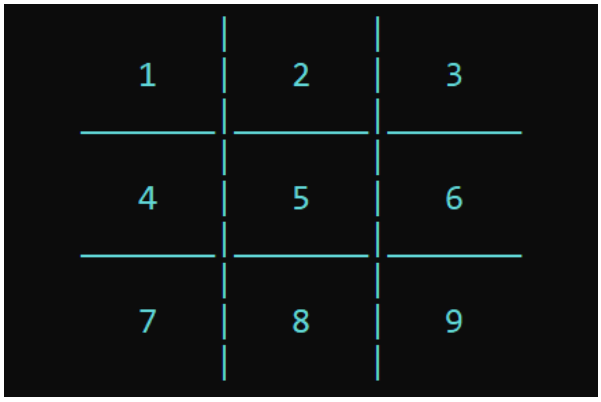
1. Auto Train CENACE
CENACE learns to play by playing hundreds and thousands of matches against the opponent and this takes A LOT OF TIME!
To fast forward this process, in this menu, CENACE can play against its opponent but, this time the opponent is also a computer. No, CENACE will not play against itself, instead, it will play against an already existing tic-tac-toe algorithm. In this case, I have hardcoded an algorithm that I created.

2. Play
This is the main part where you can play against CENACE.
CENACE will get better and better as you play more.

3. Rules
You can see the rules here.

RULES:

1. CENACE will always go first.
2. Move positions are from 1 to 9 on the game board.



3. +3 for win, +1 for draw, -1 for lose.

4. About
Shows a short description of CENACE.

5. Exit
To exit CENACE

About the source code of CENACE

CENACE is written in C language.

Functions used in the code are:

int main()
void CENACE_intro()
void menu()
void choose_player()
void inputmove()
void updateLearning_data()
void updateGraphing_data()
int checkwin()
void scoreUpdate()
void newboard()
void printboard()
int space_in_board()

int main():

This is the main function. It calls other functions and coordinates everything.
It creates the "Learning_data" and the "Graphing_data" directory or folder when CENACE runs for the first time, marks the moves on the game board, and also shows messages and warnings to the user.

Learning_data: This folder will be automatically created and updated after each match and will store the data that CENACE learned.

Graphing_data: This folder will be automatically created and updated after each match and will store the data for drawing a graph of CENACE's score.

void CENACE_intro()

This function is for the loading animation.

void menu()

This function prints the main menu and takes menu input from the user.

void choose_player():

This function is for selecting player signs.
Currently 'O' is fixed for CENACE and 'X' is fixed for its opponent.

And CENACE will always play first.
Maybe I'll add this feature in a future version of CENACE.

void inputmove():

This is the MOST IMPORTANT function of the program.
It handles all the critical tasks and keeps CENACE working.

While in "Play" mode (2), it takes input from the user against CENACE

While in "Auto Train CENACE" mode (1), it takes input from the existing algorithm that I created to play against CENACE.
This is a simple algorithm that only knows how to win and how to block CENACE from winning. The rest of the time it plays randomly.

This function also takes input from CENACE itself. And this is the most critical part.
Every time before playing its move, CENACE will look at the game board. the game board is actually a string of length 10 (Including the null character).
for example this one "XO XO   ". The blank areas on the game board are occupied with spaces. CENACE will take this string, and will make a copy of this and replace the spaces with a '-' sign. So the copied string is "XO--XO---". Now CENACE will try to create a folder named "XO--XO---" inside the "Learning_data" folder. If the folder doesn't exist then it will be created and inside "Learning_data\XO--XO---" some text files will be created with the names of all blank move positions. Move positions are from 1 to 9 (1 will be added to the string's actual element positions). So, in this case, 5 text files will be created as shown below:
3.txt
4.txt
7.txt
8.txt
9.txt
The text files represent the move positions that are available for CENACE. In these text files, there will be written a priority point. Initially, the points will be 2 for every text file. The points will update after each match. CENACE will choose its move based on the points. The higher the point is greater the chance that CENACE will choose that move.
Now let's have a look at how it works. Assume that the text files are created and there is written "2" inside all the text files. There are a total of 5 text files. CENACE will create an array of size 10 (2+2+2+2+2). there will be 2 copies of every available move (Names of the text files). The array will look like this:
{3, 3, 4, 4, 7, 7, 8, 8, 9, 9}
Now, CENACE will choose randomly from this array.
Let's say it picked up 8, so, it will place its move at the 8th position.
The file path for the given move will be "Learning_data\XO--XO---\8.txt". This path will be stored in a 2D character array. Similarly, all the other paths during the match will be saved. So that at the end of a game all the moves that CENACE gave can be tracked.

Here ends the task of "void inputmove()" function.
After a match is over all the files that CENACE used will be updated in the void updateLearning_data() function.

void updateLearning_data()

In the previous section, I explained how CENACE plays its moves.
If you notice you will see that 8 was the winning move for CENACE in that example. CENACE won the game and the match finished. We also have all the file paths of CENACE's moves from the beginning of the game. Now, Every text file in those paths will be opened one after one. and 3 points will be added to the existing point. For example, Learning_data\XO--XO---\8.txt had 2 points previously and now it will be 5. So, next time there will be a higher chance that CENACE will choose 8 if it finds the same game board again.
The reword policy is:
→ +3 for winning a game
→ +1 for drawing a game
→ -1 for losing a game.
The maximum point for a position is 300 and the minimum point is 1.

void updateGraphing_data()

This function updates the graphing data after each match inside the Graphing_data folder. There are two text files, Last_Score.txt and Score_List.txt inside that folder. The Score_List.txt stores the score history of CENACE. This data can be used to draw a graph which will show the results of CENACE's games against the opponent. Last_Score.txt stores the last updated score. After each match, Last_score.txt will be updated and appended to the Score_List.txt.
The scoring policy is as before:
→ +3 for winning a game
→ +1 for drawing a game
→ -1 for losing a game.

int checkwin()

This function checks for a winning condition either for the human opponent or for CENACE and returns 1 if found. Or, returns -1 if it is a draw.

void scoreUpdate()

At the end of each match, this function updates the score that is shown on top of the game screen while playing.

void newboard()

This function clears the game board and sets it back to the default for starting a new game.

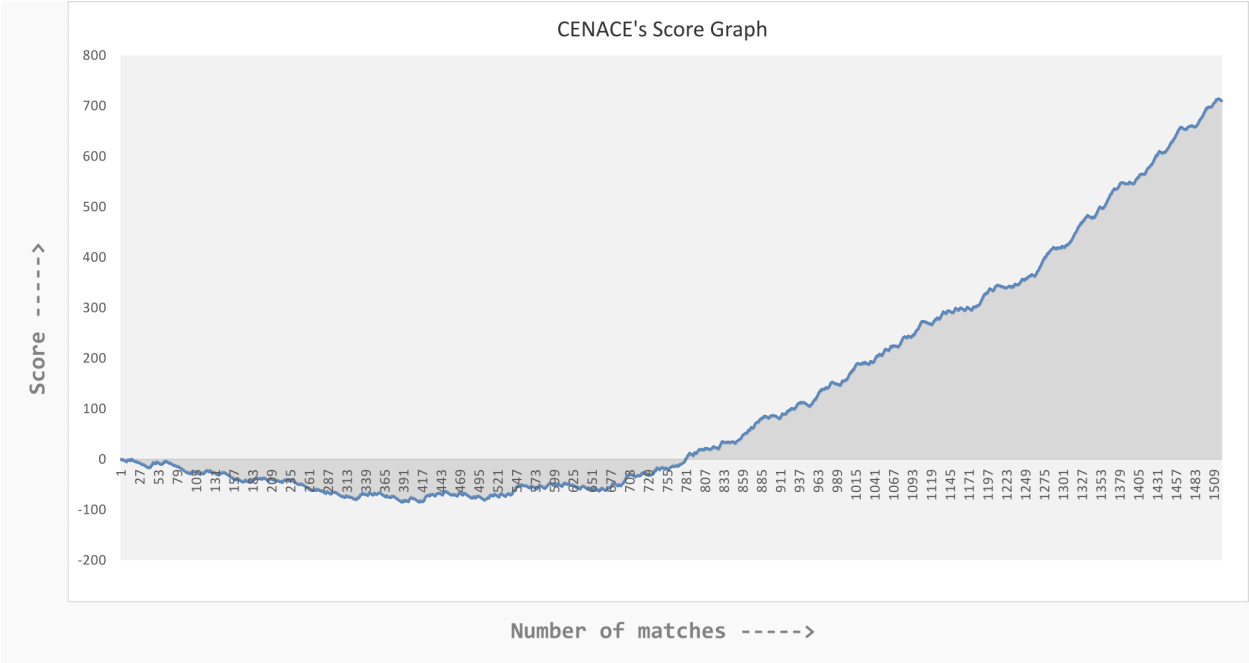
void printboard()

This function prints the current game board and shows the current score above the game board.

int space_in_board()

This function returns the number of blank spaces in the current game board. This information is needed inside the "void inputmove()" function.

Graph



The graph shows the results of CENACE's games against its opponent. +3 for winning, +1 for drawing and -1 for losing.

https://github.com/Aaban-Saad/Project-CENACE