**‘Tic Tac Toe’ Game**

(AI Fundamentals)

Prof: Dr. Partha Mukherjee

Summer, 2022

Shape

Description automatically generated

**An artificial intelligence system that implements Epsilon-Greedy Algorithm and Role Based Learning System to develop a 5x5 Tic-Tac-Toe game with the objective to win.**

Team:

Abhay Chamu Haridas ([ffa5177@psu.edu](mailto:ffa5177@psu.edu))

Basem Barbary ([bpb5322@psu.edu](mailto:bpb5322@psu.edu))

Lakshmi Kausalya Kavuri ([lkk5341@psu.edu](mailto:lkk5341@psu.edu))

Project Contact Person:

Abhay Chamu Haridas (ffa5177@psu.edu)

**Table of Contents**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl No.** | **Topic** | | **Page Number** |
| 1 | Introduction | | 3 |
| 1.1 |  | AI Project Topic | 3 |
| 1.2 |  | Description | 3 |
| 1.3 |  | Playing Grid | 4 |
| 1.4 |  | How to play | 4 |
| 1.5 |  | Expected Results | 5 |
| 2 | Considerations | | 5 |
| 2.1 |  | Picking an Algorithm | 5 |
| 2.2 |  | State Space Consideration | 6 |
| 2.3 |  | Speed Consideration | 7 |
| 3 | Artificial Intelligence Implementation | | 7 |
| 3.1 |  | Description of AI Model | 7 |
| 3.2 |  | Workflow: Setting up Environment | 8 |
| 3.3 |  | Workflow: Playing Game in Environment | 10 |
| 4 | Results | | 14 |
| 5 | Experimentation Reflection | | 16 |
| 5.1 |  | Experimentation | 16 |
| 5.2 |  | Runtime | 16 |
| 5.3 |  | Limitations | 16 |
| 5.4 |  | Improvements | 17 |
| 5.5 |  | Final Thoughts | 17 |
| 6 | References | | 18 |
| 7 | Appendix | | 19 |

**1**

**INTRODUCTION**

**1.1 AI Project Topic**

To build an unbeatable 5x5 Tic Tac Toe game using python and the libraries at its disposal such that the game can be played against a self-learning Artificial Intelligence system. Rule Based learning and the epsilon greedy algorithm are implemented.

**1.2 Description**

The project, ‘Tic Tac Toe’ game, is a two-player game. Currently there are many versions to it, like the 4 x 4 game and 5 x 5 game. This game’s original, 3 x 3 version is one of the most popular games dated back to 1300 BC in Ancient Egypt. There have also been found many other games which were quite similar to the now ‘TTT’. One of them is called the ‘terni lapilli’, from Rome, which is one of the variations of how the present version of TTT is played. This game was very popular in the first place due to its simplicity. It has 2 symbols to play with: the X and the O. Usually, the simple version of this game is a 3 x 3 board game, in which the possibility of playing the game is 9!, considering that the game does not end when there are 3 X’s or O’s in a row. But for a 5 x 5 game, the possibilities to lay the game is 125!, which makes it impossible for brute force searchThe goal of the TTT is for one of the players to get three of same symbols in a row, which can be horizontal, vertical, or diagonal, on the 3 x 3 grid.

As already mentioned above there are many types of ways to play the tic tac toe game. One of them is, we will be given an infinite grid and the players will have to place either 5 X’s or 5 O's in a row, either horizontally or vertically or diagonally, to win. The other way to play is, we will be given 5 x 5 grid and the players will have to place at least 4XS or 4 OHS in a row. And the other type maybe the players will be given a 5 x 5 grid and they will have to place 5 X’s or 5 O’s in a row. Our project is mostly focused on the third type mentioned above, which is, the players will be given a 5 x 5 grid and they will have to place 5 X’s or 5 O’s in a row. First let us understand how to play the roles and basic rules of the game before we go further into its development.

**1.3 Playing Grid**

This game can be played in a 3x3 grid, as shown below. Or it can also be played in a 4x4 grid or a 5 x 5 grid pictured below:

Since this game is typically played by two players, it can be played by two humans or two AI's or maybe for human entertainment, it can be played with a human and an AI, which is exactly what our project is focusing on.

At the start of every game, the user will be given a choice of selecting both the players. If both the players are selected as humans, then each of them will be given a chance to place their X or O. If the option selected is one human, then automatically the first player will be human and the second player is the AI i.e., computer, which will again learn as the player keeps playing and tries to perfect the game.

So, now let's learn how to play the game for our basic understanding. And since our project focuses on one player and one AI, we will be considering the first player to be human or the user while the second player is the AI.

**1.4 How to Play**

Tic-Tac-Toe is a very simple game that only allows for two people to play. A player can choose between playing the first turn or letting the AI take the first turn. Typically, the game uses ‘X’ and ‘O’ for symbols to represent each player. If the user decides to let the AI take the first turn, the first move is made at random and the user is prompted to choose their next move. A player marks any of the 25 squares with their assigned symbol (may be ‘X’ or ‘O’. A player marks any of the squares In the grid with their choice of symbol they selected, typically with an aim to place 5 of their symbols in a row, either horizontally or vertically or diagonally with two intensions [[1](https://www.youtube.com/watch?v=USEjXNCTvcc)]:

a) Create a straight line before the opponent, to win the game.

b) Restrict the opponent from creating a straight line.

**1.5 Expected Results**

The tic-tac-toe game will be given the task to ultimately win and complete a straight line. This is accomplished with the use of the greedy epsilon algorithm which will decide the action the agent will take. The greedy epsilon (E) is in other words a percentage of chance that the AI (Agent) will play a random move. The agent will take random actions at first, but once it better understands the environment, the agent is expected to play to win. In a case where neither the user nor the AI could place three of their symbols in a row, the game results with a tie.

In summary, we are making an agent play the game of tic tac toe thousands of times and recording what happens in every state of the game. We then assign a value to each state and correct this value using the value update function. When the agent is trying to win, he will always move to the state with the highest value. So, logically there are only three possible outputs/results, which are: space the user wins, the opponent i.e., the AI wins or there are no spots left in which results with a tie. With the use of Rule Based strategy the AI will be able to win the game based on rules identified using this learning strategy.

**2**

**Considerations**

**2.1 Picking an Algorithm**

John von Neumann pioneered within the field of two-player zero-sum games the proof of the min-max algorithm published in 1928. It is considered the founding blocks of game theory for which the existence of a saddle-point solution to 2-person, zero sum games was recognized by the proof and sparked a revolution for this theory (Ben-El-Mechaiekh & Dimand, 2010). There are many types of algorithms that we can use in the AI that plays the tic tac toe game. While the Rule-based strategy is simply the player following the given set of mandatory rules provided for the game, the Minmax search algorithm is the typical backtracking algorithm used for decision theory and game theory, statistics, philosophy etc. and the Alpha-Beta Pruning algorithm is an optimization algorithm for the minimax algorithm. While the Rule-based strategy is simply the player following the given set of mandatory rules provided for the game, the Minmax search algorithm is the typical backtracking algorithm used for decision theory and game theory, statistics, philosophy etc. and the Alpha-Beta Pruning algorithm is an optimization algorithm for the minimax algorithm. The minimax algorithm is essentially a back and forth between the two players. The player who is currently playing, will be placing his symbol, either X or O, in such a way that, that move will result the player with the maximum score possible. While in return, the scores of the rest of the available moves are determined by the player who will be playing the next move, such that his move will result in the minimum score for the current player that is if player 1 makes the first move for the maximum score possible then, play it 2 will make a move in such a way that it will minimize this score of the player 1. In this project however, we are using the Rule Based Learning System based on which the AI will be taking rule based actions. Our AI will be taking random actions at first, but once it starts getting the hang of the game, it will start understanding the game and start playing in a way that has more likelihood to win. So, what we do is, set a variable epsilon, E, that represents the AI’s chance of taking a random action.

However, the value of each is equal to 1 at the beginning of the game. Then, when the agent starts to understand the game, E will start decreasing slowly and overtime, the value of epsilon, E i.e., the chance of the AI taking random actions reduces and thereby increasing the probability of the AI winning. It is done by determining a random number, A, that lies between 0 and 1 before the value of epsilon, E, is even decreased by a small amount and updated. Now, if the value of the random number taken, **A** < **E**, the epsilon value, a random state space is selected which is, a random move is done by the AI. If **A** > **E** then, it implies that the move pertaining to the highest valued state space is selected, which is, the AI does not make a random move. The reward is only received at the last state. So, the value of the last state is set only to 1 (if won) or -1 (if lost). The Agent Environment Interaction can be pictured below.

Diagram

Description automatically generated

**2.2 State Space Consideration:**

State space tree generated is essential to consider. Traditional 3x3 tic-tac-toe is a small game for which an unbeatable algorithm can be developed due to a small state space tree generated and recognizable possibilities (Garg, 2017). If the state space generated is high, the algorithm will take longer due to the increased number of possibilities from 39 to 416 or 525 for example, which is a notable drawback. Sandlund et al. (2014) observed the possibilities within a 4x4 tic-tac-toe numerical scheme. Highlighting the earlier mentioned content, in 3x3 tic-tac-toe, the win of player 2 can be over-turned by player 1, the computer, every time depicting the computer always winning. However, in 4x4 tic-tac-toe, player 2 can force a win. In other words, comparing the amount of computation required to determine the answers to our original questions, it appears that in random play that it is much easier to stop player one from winning than it is to make player two win (Sandlund, et al., 2014). Sandlund et al. (2014) concluded that in 3x3tic-tac-toe, the odds of player 2 winning are nonexistent because it is not so hard to stop player two from winning but apparent how to have player one win.

**2.3 Speed Consideration:**

The vanilla min-max algorithm is not the most optimal in terms of arriving at the goal in the least number of moves. Paul, A.J.’s (2020) study for 3x3 tic-tac toe included using decision trees called T3DT which outperformed the non-randomized methods Min-max (MM), Min-max with alpha beta pruning (ABP), and Min-max with advanced alpha beta (ABA) in terms of speed. It is important to note that the speed of an algorithm to reach its end, is of value. Min-max with alpha beta pruning is proven to be quicker in running time than vanilla min-max.

**3**

**Artificial Intelligence Implementation**

**3.1 Description of AI Model**

**Initial state:** The initial position of the board is an empty 5x5 grid at position (0,0). Epsilon is set to 1 at the initial state and at the start position.

**Player(s**): The game can be played by two players. There are two options for players:

(a) Player 1- User starts the game with the first move in the state space

(b) Player 2- Computer starts the game with the first move in the state space

**Action space:** The model will loop through every spot and, if it's empty, record the state that would come from the player moving in that spot. If the current players value for this state is higher than the top recorded value, set the top value to the value of this state and set action = the spot that will lead to this state. It is important to keep in mind that no user input is a possibility. Or, if the user does not pick an action, the algorithm will return a random action.

* E is ultimately the percentage of chance that the AI (Agent) will play a random move.
* A random number ‘A’ between 0 and 1 is generated and then E is decreased by a small amount.
* If the value of the random number taken, A < E, the epsilon value, a random state space is selected.
* If A > E then, it implies that the move pertaining to the highest valued state space is then selected, which is, the AI does not make a random move, but an educated one.

**Result (s, a):** The AI identifies the space as a result of the action **a** on state **s**, in the environment and continues to update results based on rules until the algorithm reaches the end state.

**Reward/Payoff:** A payoff function generates a numeric value for the outcome of the game for a playerwith the end state. For tic-tac-toe, the outcomes are a win, loss, or draw, the utility values are +1, -1, and 0 respectively.

**3.2 Workflow: Setting up Environment**

The goal of the workflow is to find out all the possible next states the agent can be in, then find which one of those states has the highest value. This is done by looping through the game board (spot\_placeholders) and if a spot is open, we record what the new state would be if the agent went in this free spot. We then look in the agent's state value table and find the value of this state. If this state has the highest value, we move to the spot that results in this state.

The pseudocode for this implementation to assign rules for the AI is as follow:

**Rule 1:** Check whose turn it is.

**Rule 2:** Find all the next possible states that the player can be in.

**Rule 3:** Perform ε greedy to either take a random action or be greedy(take the best action).

**Rule 4:** If greedy, loop through the next possible states and find the state with the highest value.

**Rule 5:** Try to perform this action, but if the agent doesn’t have any of these states in its table then do a random action.

Initially we create the 5x5 grid with spot placeholders, 25 entries in total.



The count of the grid starts at top left with count 0 and moving to the right then to the next row until the last spot,24. Epsilon is initialized to 1 as seem below:

Text

Description automatically generated

The workflow proceeds with having to prepare for user input to pick between Player 1 and Player 2. Entering the marker “1” will allow for the user to play the first turn meaning the user will start with the first move. Entering the marker “2” will allow the user to play the second turn meaning the computer will start with the first move.

Text

Description automatically generated

To register the user’s input, the action is loop through every spot and, if it's empty, record the state that would come from the player moving in that spot and the next possible state is up for determination. We want the agent to take random actions at first, but once it starts getting the hang of things, it should play to win. So, what we do is set a variable E that represents the agent's chance of taking a random action, and slowly decrease E over time. To determine the position, a random number (A) between 0 and 1 is generated and then E is decreased by a small amount. If the current players value for this state is higher than the top recorded value, set the top value to the value of this state and set action = the spot that will lead to this state. It is important to keep in mind that no user input is a possibility. Therefore, or, if the user does not pick an action, the algorithm will return a random action as portrayed in the code below:

Text

Description automatically generated

Next a definition for storing the moves on the grid as the game progresses is written to the grid. This will keep “X” and “O” stored in the grid. If the spot is empty, it will return the empty position, visa-versa. If a spot is already occupied, the user will need to input a different position that holds a spot place holder value of 0. All possible winning combinations that satisfy the winning criteria are written in based on neighboring spot placeholders displayed in Appendix (1).

**3.3 Workflow: Playing Game in Environment**

After setting up the environment and state space, the workflow proceeds with identifying the space as a result of the action, a, on state, in the environment. Next, both scenarios of Human starting the game and AI starting the game are written with spot placeholders of 1 and 2 respectively. After the first move, the first move is written in to be stored as “state\_history” to update the board with the choice made.

The algorithm shown below then checks if win conditions are satisfied after every move. If reward conditions = 1, the reward condition is satisfied, and the respective player wins the game. This is done by satisfying the overall goal to get 5 same designated symbols in a row that can be horizontally, vertically, or diagonally, on a 5 x 5 grid and win the game. Also, if this is true, neither player 1 or player 2 turn of play because the game has ended. If reward conditions = -1 and the respective player loses the game and neither player1 or player2 turn of play because the game has ended. If the reward condition =0, the game ends at a draw.

Text

Description automatically generated

If the win condition is false, the values will update, and the next move will be made according to state history seen by the code below:

Text

Description automatically generated

The message for the start of the game is necessary to inform the user upon input. The computer is either to play as player 1 or player 2 based on the selection made by the user. The game then prints “Loading Game…” and “Please wait” while setting up the gameboard.

Text

Description automatically generated

Printing the gameboard involves setting a single spot on the board to print ‘\_’ for 0 entry on the board, to print ‘X’ for player 1 entry and to print ‘O’ for player 2 two entry in a designated spot on the board and applied to each spot on the board for when the board prints.

Text

Description automatically generated

Finally, the test of the computer is applied to determine if the game has ended according to the computers action. If it is the computers turn, the spot placeholder of the action is set to = 1. If check for winner returns as true, the game will prompt the user with a message, ‘You Lose!’, and the game will reset all the spot placeholders to 0, empty cells. If it is the users turn, the action is inputted by the user after given the prompt ‘Your turn to play Mark at:’. If the input = 0 for the spot placeholder for this action, the users turn is skipped and the computer will make the next move. If the input = 2 for the spot placeholder for this action, the symbol will be printed in that spot and the computer will take the next turn. If when checking for the winner is true, the game presents the message ‘You Win!’ and the user wins the game. The board is returned to original spot placeholder value of 0 if this condition is satisfied. If checking for the winner is equal to ‘WinWin’, the game presents the message ‘Draw Game’ and the spot placeholders are reset and the game ends. If the game has ended it is implied that player 1 and player 2 do not have a turn. If these conditions are satisfied, the user is given the message ‘Would you like to play first or second? Input 1 or 2:’ to start the game. If the user responds with ‘1’, the computer will be recognized as player 2 and take the second turn. If the user responds with ‘2’, the computer will be recognized as player 1 and will take the first turn as seen below:

Text

Description automatically generated

Since there is no assigned memory for the game, what we do is keep a list of all the states visited in an episode. At the end of the episode, we need to call a function that applies the value update function to every state in this list for each agent. Notice that we only receive a reward at the last state of the episode. Therefore, we can just set the value of the last state to the reward. The pseudocode to my implementation is the following:

1. Set the value of the last state equal to either 1 or -1 (reward if game won or punishment if game lost).
2. Append any new, previously undiscovered states to the player's memory.
3. Apply the update function starting from the second to last state and moving towards the first state.

**4**

**Results**

The algorithm written for 5x5 Tic-Tac-Toe originated as a 3x3 then was updated to a 4x4 matrix and finally the 4x4 matrix was updated to a 5x5 matrix. The images below show the outputs upon starting the game and selecting for the computer to take the first turn. A notable point is the spot chosen by the AI upon the first turn when all spots are available.

Text

Description automatically generated

**Image 1:** 3x3 Tic-Tac-Toe constructed. A fork can then be implemented by the computer because of its choice of positioning.

Text

Description automatically generated

**Image 2:** 4x4 Tic-Tac-Toe constructed. The AI choses the best position with the goal to win.

Text

Description automatically generated

**Image 3:** 5x5 Tic-Tac-Toe constructed. The AI choses the best position with the goal to win.

The epsilon greedy algorithm is how we will decide which action our agent will take.

We want the agent to take random actions at first, but once it starts getting the hang of things we want it to play to win. So what we do is set a variable ε that represents the agent's chance of taking a random action, and slowly decrease ε over time. The pseudocode is as follows:

1. Initialize ε = 1
2. Generate a random number between 0 and 1
3. If this random number is < ε, explore the state space (take a random action).
4. If this random number is > ε, perform the action that leads to the state with the highest value.
5. Decrease ε by a small amount

**5**

**Experimentation Reflection**

**5.1 Experimentation**

The game was run back to back for 20 times. The computer won 10/20 games. The more games the AI played, the more the AI learns and is able to have better chance at winning. These results can be improved however by building a system that is smarter perhaps using the Min-Max Algorithm because of the information highlighted in the research done at the early stages of developing the project. From experimenting, it is obvious that the AI ignores the chance of the user winning and does not react defensively to stop a win from the user occurring. This is understandable because our intentions were to build a system that is trained to go after a win and not to stop a win.

**5.2 Runtime**

The runtime of the 5x5 Tic-Tac-Toe algorithm developed is very long. Comparing the load time between 3x3, 4x4, and 5x5, 3x3 of course has the fastest runtime. With more added spots, the increase in runtime can be seen and can become quite an issue when dimensions are very high.

**5.3 Limitations**

The runtime of the 5x5 is disappointing and not desirable. The 3x3 runtime is very quick but with the added spots in the matrix, the system takes longer to be processed because of the increase in possible outcomes. Another limitation is the AI only playing offense and not playing defensively. When experimenting the AI’s oblivious to the users chance of winning in the next move and it is ignored by the AI’s next move. The AI should be able to see at the very least the chance of the user winning

**5.4 Improvements**

* Since the project we have worked on has only 5 by 5 grid and the user or the AI have to form 5 of the chosen symbols in a row, either horizontally or vertically or diagonally, there can be further improvement by changing a couple of rules like: updating the game to infinite grid or by limiting the number of X and O to a limited number.
* Another improvement can be said as adding more rules for the AI:
  + If there is an opportunity to have two possible ways of winning (fork) in the next move, the AI should take that move.
  + the AI could place it’s symbol in a spot which allows for more possible opportunities of winning.
  + If the AI detects a winning move from the user, to try and block it.
* Deep Reinforcement Learning
* Min-Max Algorithm can be used to in order to build a close to unbeatable system.

**5.5 Final Thoughts**

Overall, in our reflective opinion, building this AI system was very eye opening to the fact that it is very important to spend time choosing a model and considering runtime as an obstacle depending on the size of the system being built and its system demands. It was indeed very educational to learn and implement the algorithms and very interesting to analyze results and make key determinations from experimentations which.

**REFERENCES**

[1] How to play Tic Tac Toe <https://www.youtube.com/watch?v=USEjXNCTvcc>

[2] Sandlund, B., Staley & Butler, S. (2014). [Numerical Tic-tac-toe on the 4×4 board](https://link.springer.com/chapter/10.1007/978-3-319-08783-2_46). *Lecture Notes in Computer Science*, 537–546. https://doi.org/10.1007/978-3-319-08783-2\_46

[3] *3.1 The Agent-Environment Interface*, http://www.incompleteideas.net/book/ebook/node28.html.

[4] How to [play](https://www.youtube.com/watch?v=qnHpJto5Xfk&ab_channel=MeMeGameplay) 5 x 5 TTT

[5] AM Zain et al 2020 [IOP Conf. Ser.: Mater. Sci. Eng](https://iopscience.iop.org/article/10.1088/1757-899X/864/1/012090/pdf). **864** 012090

[6] BEN-EL-MECHAIEKH, H. I. C. H. E. M., & DIMAND, R. O. B. E. R. T. W. (2010). [Von Neumann, Ville, and the minimax theorem](https://www.worldscientific.com/doi/abs/10.1142/S0219198910002556). *International Game Theory Review*, *12*(02), 115–137.

[7] Dalffa, Mohaned Abu, Bassem S. Abu-Nasser and Samy S. Abu-Naser. “[Tic-Tac-Toe Learning Using Artificial Neural Networks.](https://philpapers.org/rec/DALTLU)” (2019).

[8] Ferraco, Alex, Yuchen Wang and Christopher Todd. “[5 x 5 x 5 Tic-Tac-Toe AI Implemented by Genetic Algorithm.](http://www.personal.psu.edu/users/y/v/yvw5013/stuff/Implementation-of-a-tic-tac-toe-player-using-genetic-algorithm.pdf)” (2011).

[9] Garg, R. (2017). Game of tic-tac-toe: Simulation using Min-Max Algorithm. [*International Journal of Advanced Research in Computer Science*](https://doi.org/10.26483/ijarcs.v8i7.4409), *8*(7), 1074–1077.

[10] Paul, A. J. (2020). Randomized fast no‐loss expert system to play tic‐tac‐toe like a human. [*Cognitive Computation and Systems*](https://doi.org/10.1049/ccs.2020.0018), *2*(4), 231–241.

[11] Rajab, Abubakarsidiq Makame & Hassan, Nadhira & Rajab, Abdul-Rahimsidiq. (2020). [Tic-Tac-Toe Learning Using Artificial Neural Networks](http://ijeais.org/wp-content/uploads/2019/02/IJEAIS190202.pdf). Vol. 3. : 1-6.

[12] [Randomized fast no-loss expert system to play tic tac toe like a human](https://arxiv.org/abs/2009.11225) by Aditya Jyoti Paul1,2

**APPENDIX**

**1.**

def state\_to\_num(state):

N = state[0]+5\*state[1] + 25\*state[2]+125\*state[3]+625\*state[4]+3125\*state[5]+15625\*state[6]+78125\*state[7]+390625\*state[8]+1953125\*state[9] + 9765625\*state[10]+48828125\*state[11]+244140625\*state[12]+1220703125\*state[13]+6103515625\*state[14]+30517578125\*state[15]+152587890625\*state[16]+762939453125\*state[17]+3814697265625\*state[18]+19073486328125\*state[19]+95367431640625\*state[20]+476837158203125\*state[21]+2384185791015625\*state[22]+11920928955078125\*state[23]+59604644775390625\*state[24]

return N

def num\_to\_state(N):

y = N //(5\*\*24)

x = (N - y\*(5\*\*24)) //(5\*\*23)

w = (N - y\*(5\*\*24) - x\*(5\*\*23)) //(5\*\*22)

v = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22)) //(5\*\*21)

u = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21)) //(5\*\*20)

t = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20)) //(5\*\*19)

s = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19)) //(5\*\*18)

r = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18)) //(5\*\*17)

q = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17)) //(5\*\*16)

p = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16)) //(5\*\*15)

o = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15)) //(5\*\*14)

n = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14)) //(5\*\*13)

m = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13)) //(5\*\*12)

l = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12)) //(5\*\*11)

k = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11)) //(5\*\*10)

j = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10)) //(5\*\*9)

i = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10) - j\*(5\*\*9))//(5\*\*8)

h = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10) - j\*(5\*\*9) - i\*(5\*\*8)) //(5\*\*7)

g = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10) - j\*(5\*\*9) - i\*(5\*\*8) - h\*(5\*\*7)) // (5\*\*6)

f = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10) - j\*(5\*\*9) - i\*(5\*\*8) - h\*(5\*\*7) - g\*(5\*\*6)) // (5\*\*5)

e = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10) - j\*(5\*\*9) - i\*(5\*\*8) - h\*(5\*\*7) - g\*(5\*\*6) - f(5\*\*5)) // (5\*\*4)

d = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10) - j\*(5\*\*9) - i\*(5\*\*8) - h\*(5\*\*7) - g\*(5\*\*6) - f(5\*\*5) - e(5\*\*4)) // (5\*\*3)

c = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10) - j\*(5\*\*9) - i\*(5\*\*8) - h\*(5\*\*7) - g\*(5\*\*6) - f(5\*\*5) - e(5\*\*4) - d(5\*\*3)) // (5\*\*2)

b = (N - y\*(5\*\*24) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - k(5\*\*10) - j\*(5\*\*9) - i\*(5\*\*8) - h\*(5\*\*7) - g\*(5\*\*6) - f(5\*\*5) - e(5\*\*4) - d(5\*\*3) - c(5\*\*2))//(5\*\*1)

a = (N - y\*(5\*\*15) - x\*(5\*\*23) - w\*(5\*\*22) - v\*(5\*\*21) - u\*(5\*\*20) - t\*(5\*\*19) - s\*(5\*\*18) - r\*(5\*\*17) - q\*(5\*\*16) - p\*(5\*\*15) - o\*(5\*\*14) - n\*(5\*\*13) - m\*(5\*\*12) - l\*(5\*\*11) - b(5\*\*10) - j\*(5\*\*9) - i\*(5\*\*8) - h\*(5\*\*7) - g\*(5\*\*6) - f(5\*\*5) - e(5\*\*5) - d(5\*\*3) - c(5\*\*2)- b(5\*\*2))//(5\*\*0)

return([a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y])

# Winning criteria

def check\_for\_winner():

if spot\_placeholders[0] == spot\_placeholders[1] and spot\_placeholders[1] == spot\_placeholders[2] and spot\_placeholders[2] == spot\_placeholders[3] and spot\_placeholders[3] == spot\_placeholders[4] and spot\_placeholders[4]!= 0:

return True

if spot\_placeholders[0] == spot\_placeholders[6] and spot\_placeholders[6] == spot\_placeholders[12] and spot\_placeholders[12] == spot\_placeholders[18] and spot\_placeholders[18] == spot\_placeholders[24] and spot\_placeholders[24] != 0:

return True

if spot\_placeholders[0] == spot\_placeholders[5] and spot\_placeholders[5] == spot\_placeholders[10] and spot\_placeholders[10] == spot\_placeholders[15] and spot\_placeholders[15] == spot\_placeholders[20] and spot\_placeholders[20] != 0:

return True

if spot\_placeholders[1] == spot\_placeholders[6] and spot\_placeholders[6] == spot\_placeholders[11] and spot\_placeholders[11] == spot\_placeholders[16] and spot\_placeholders[16] == spot\_placeholders[21] and spot\_placeholders[21] != 0:

return True

if spot\_placeholders[2] == spot\_placeholders[7] and spot\_placeholders[7] == spot\_placeholders[12] and spot\_placeholders[12] == spot\_placeholders[17] and spot\_placeholders[17] == spot\_placeholders[22] and spot\_placeholders[22] != 0:

return True

if spot\_placeholders[3] == spot\_placeholders[8] and spot\_placeholders[8] == spot\_placeholders[13] and spot\_placeholders[13] == spot\_placeholders[18] and spot\_placeholders[18] == spot\_placeholders[23] and spot\_placeholders[23] != 0:

return True

if spot\_placeholders[4] == spot\_placeholders[9] and spot\_placeholders[9] == spot\_placeholders[14] and spot\_placeholders[14] == spot\_placeholders[19] and spot\_placeholders[19] == spot\_placeholders[24] and spot\_placeholders[24] != 0:

return True

if spot\_placeholders[5] == spot\_placeholders[6] and spot\_placeholders[6] == spot\_placeholders[7] and spot\_placeholders[7] == spot\_placeholders[8] and spot\_placeholders[8] == spot\_placeholders[9] and spot\_placeholders[9] != 0:

return True

if spot\_placeholders[10] == spot\_placeholders[11] and spot\_placeholders[11] == spot\_placeholders[12] and spot\_placeholders[12] == spot\_placeholders[13] and spot\_placeholders[13] == spot\_placeholders[14] and spot\_placeholders[14] != 0:

return True

if spot\_placeholders[15] == spot\_placeholders[16] and spot\_placeholders[16] == spot\_placeholders[17] and spot\_placeholders[17] == spot\_placeholders[18] and spot\_placeholders[18] == spot\_placeholders[19] and spot\_placeholders[19] != 0:

return True

if spot\_placeholders[20] == spot\_placeholders[21] and spot\_placeholders[21] == spot\_placeholders[22] and spot\_placeholders[22] == spot\_placeholders[23] and spot\_placeholders[23] == spot\_placeholders[24] and spot\_placeholders[24] != 0:

return True

if spot\_placeholders[4] == spot\_placeholders[8] and spot\_placeholders[8] == spot\_placeholders[12] and spot\_placeholders[12] == spot\_placeholders[16] and spot\_placeholders[16] == spot\_placeholders[20] and spot\_placeholders[20] != 0:

return True

elif (spot\_placeholders[0] != 0 and spot\_placeholders[1] != 0 and

spot\_placeholders[2] != 0 and spot\_placeholders[3] != 0 and

spot\_placeholders[4] != 0 and spot\_placeholders[5] != 0 and

spot\_placeholders[6] != 0 and spot\_placeholders[7] != 0 and

spot\_placeholders[8] != 0 and spot\_placeholders[9] != 0 and

spot\_placeholders[10] != 0 and spot\_placeholders[11] != 0 and

spot\_placeholders[12] != 0 and spot\_placeholders[13] != 0 and

spot\_placeholders[14] != 0 and spot\_placeholders[15] != 0 and

spot\_placeholders[16] != 0 and spot\_placeholders[17] != 0 and

spot\_placeholders[18] != 0 and spot\_placeholders[19] != 0 and

spot\_placeholders[20] != 0 and spot\_placeholders[21] != 0 and

spot\_placeholders[22] != 0 and spot\_placeholders[23] != 0 and spot\_placeholders[24] != 0 ):

return 'WinWin'

else:

return False