* ***Motivation***:

When exploring recursion and tree in the class in T12 and T13, I realized the power a recursive function. In my CSC 386 Computation Complexity course, I learned about optimization problems and state space. I decided that I would create a game that would intelligent enough to play and beat humans. I started by exploring chess, but due to time constraints and chess being an NP-complete problem, I decide to go with something easier like the Tic Tac Toe game. During my research I learned about Game Trees from the following link <https://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html> that inspired me to go ahead and create this project.

* ***Purpose***:

The purpose of this program is to create a user interactive Tic Tac Toe game, where user can play against computer or another human. The main purpose was to allow users to play against computer, who plays intelligently that it is impossible to beat the computer. One of the purposes was to implement a user interface using a GUI library like Tkinter and as an unexpected outcome I wrote an API that can be reused in the future for prototyping or amateur software development.

* ***Audience***:

The main audience for the programs are gamers who enjoy playing strategy games, like Tic Tac Toe, Connect 4, Go, Chess etc. It could also be used as a teaching tools for computer science students who interested in learning more about algorithms, power of recursive functions, and runtime tree. Another audience could be amateur developers who are interested in developing prototypes of interfaces for their applications. They will be able to use the Tkinter Frame controlling API, I developed as an unexpected outcome of this project.

* ***Instructions:***

To run the application you have to run the Main.py file in the folder in a python 2.7 environment. It uses internal libraries like Tkinter, Numpy that usually comes with python 2.7 installation.

* ***Design***:

The Application has four modules and one driver file.

1. **Boards.py:** contains the TicTacToe() class for the Tic Tac Toe game.
2. **AIntel.py**: his module contains the AI() class for the Tic Tac Toe game and needs the Board class to work.
3. **TicTacToeApp.py**: contains the TicApp() class that is used to create the Graphic User Interface for Tic Tac Toe
4. **Pages**.py: contains the page classes that are essentially frames to change the window controlled by the TicApp Class in the module TicTacToeApp.py

|  |  |  |
| --- | --- | --- |
| **Class name:** | TicTacToe | |
| **Class Description:** | A Tic-Tac-Toe game board created using numpy array. The game board is easily adjustable to create an NxN matrix, where N is the size that has to be passed in to the class. The default size is 3x3. The class contains methods to set and get values at each position in the board. It also has methods, for checking, if game is over, if there is a winner, who is the winner. | |
| **Class Responsibilities (data and/or methods):** | | **Class Collaborations (other classes):** |
| **Methods:**   * **\_\_init\_\_(self, size=3):** the initializer where we input the size of the game board. * **available\_moves(self):** Returns a list of tuples where each tuple is a location of an empty spot on the board. e.g. [(a,b), (c,d)] a is the row and b is the column position in self.board where there is an empty position. * **game\_over(self):** Returns True if the game is over. Game is over when either it is a draw(i.e. all the blocks on board are filled up), or there is a winner(i.e. X-wins or O-wins). * **X\_won(self):** Returns a True bool value if X\_VAL wins the game. * **O\_won(self):** Returns a True bool value if O\_VAL wins the game * **tied(self):** Returns a True bool value if the game is tied. * **winner(self):** Returns the winner of the game if there is a winner, else returns None type. * **check\_winner(self, player):** Returns a True boolean value if the passed player is winner, else False. The function checks all the rows, columns and diagonal of the matrix to see if player is all in one row, one column or all in one diagonals e.g In this array [[X, O, X], [X, X, X], [X, O, O]] if player is equal to X\_VAL it returns True. If player = O\_VAL it returns False.   **Args:**  **player:** Player has to be either self.X\_VAL or self.O\_VAL, otherwise function wil not work as expected.   * **make\_move(self, pos, value):** Sets the given position on self.board numpy array with the given value   **Args:**  **pos:** an integer or a tuple that represents the index of self.board  **value:** any data type that has to be set at that position   * **get\_value(self, x, y):** Returns the value of a given position in self.board.   **Args:**  x = row index of the position in self.board  y = column index of the position in self.board   * **get\_opponent(self, player):**   """Switches and returns the given player in the game board  **Args:**  **player:** player should be equal to either self.X\_VAL or self.O\_VAL  **Data:**   * **size:** Stores the size of the tic tac toe board * **X\_VAL:** This sets how we want to represent the x-value in the game board(numpy array). * **O\_VAL:** This sets how we want to represent the o-value in the game board. * **NO\_VAL:** This sets how we want to represent an empty position on the game board. * **winning\_combos:** This stores the winning positions in the board if the game is won. * **board:** A 2-D Numpy array which is the database to record the value of each position of the game board. It is initialized with every position having self.NO\_VAL. | | ● Numpy |

|  |  |  |
| --- | --- | --- |
| **Class name:** | **AI(**TicTacToe**)** | |
| **Class Description:** | Artificial Intelligence Class for playing the Tic Tac Toe game against human player. The class allows the computer to become first or second player in the Tic Tac Toe game. The intelligence of the class is based on an alpha beta algorithm, implemented using a recursive function self.aplhabeta that takes advantage of the runtime tree. The class inherits properties from the TicTacToe class in the Boards module. | |
| **Class Responsibilities (data and/or methods):** | | **Class Collaborations (other classes):** |
| **Methods:**   * **\_\_init\_\_(self,** player='O', board=TicTacToe(3)**):** the initializer where we initialize with a player value assigned for computer and a game board TicTacToe class. * **get\_opponent(self, player):** Returns the opponent of the player passed in the function.   **Args:**  **player:** Player should be \_aiPlayer or \_humanPlayer to work as expected**.**   * **alphabeta(self, player, alpha, beta):** Returns if computer is winning, losing, or drawing in the current state of the game board.   It is a recursive function that takes advantage of the runtime tree. It places X\_VAL and O\_VALalternatively in the game board temporarily until the board is filled to search all possiblestates of the board could be possible. It assigns -1 if computer looses, +1 if computer wins, and 0 if it draws, when the board is filled and backtracks to determine a value of the initial board state.  **Args:**  **player:** This tells who is the next player in the game. The value has to be \_aiPlayer or \_humanPlayer for this function to work as expected.  **alpha:** The initial value has to be worst negative value a player can have, anything less than -1  **beta:** The initial value has to be best positive value a player can have, anything greater than +1  **Returns:** It returns an integer. -1 if to the board if the initial board state causes the computer to loose, +1 if the computer can win, and 0 if the computer can draw.  **Algorithm for this function:**  alpha-beta(player,board,alpha,beta)  if(game over in current board position)  return winner  children = all legal moves for player from this board  if(max's turn)  for each child  score = alpha-beta(other player,child,alpha,beta)  if score > alpha then alpha = score (we have found a better best move)  if alpha >= beta then return alpha (cut off)  return alpha (this is our best move)  else (min's turn)  for each child  score = alpha-beta(other player,child,alpha,beta)  if score < beta then beta = score (opponent has found a better worse move)  if alpha >= beta then return beta (cut off)  return beta (this is the opponent's best move)   * **get\_move(self, player):** Returns the best move computer can play for the current board state.   This functions temporarily places O\_VAL at every available position and calls the recursive function to see if it would win or lose the computer chose that temporary position. And then it stores the best moves in a list and return a random choice from the list.  **Args:**  **Player:** It has to be the value of the computer player, either X\_VAL or O\_VAL depending on who goes first in the game.  **Data:**   * **\_aiPlayer:** This represents if the computer will be X\_VAL or O\_VAL in thegame. * **\_humanPlayer:** This represents the human player's value in the game (opposite of aiPlayer). * **\_board:** This is a copy of the TicTacToe board class that is passed into the object, while initializing the class. | | ● Inherits from TicTacToe from Class in Boards.py module |

|  |  |  |
| --- | --- | --- |
| **Class name:** | **TicApp(**Frame**)** | |
| **Class Description:** | Tic Tac Toe app that controls frames crated using Tkinter library. This API inherits from Tkinter Frame class and allows to easily create and control pages in the GUI. The TicApp has a dictionary that stores the pages that are to be implemented separately as classes and added to the frames dictionary.  All the pages must be stored in the module Pages.py | |
| **Class Responsibilities (data and/or methods):** | | **Class Collaborations (other classes):** |
| **Methods:**   * **\_\_init\_\_(self, root, \*args, \*\*kwargs):**   Frame.\_\_init\_\_(self, root, \*args, \*\*kwargs)  **Parameters:**  **root:** A Tkinter object that has to passed into the class while creating an instance  **\*args:** allows to pass any argument into the class without knowing its name  **\*\*kwards:** allows to use any passed argument without knowing its name   * **show\_frame(self, c):** Show a frame for the given class   **Data:**   * **self.app:** It is a copy of the passed Tk() object * **self.frames:** A dictionary that stores each page, the key is the object itself and the value it maps to the instance of the same object within a fixed container (Frame). | | ● Inherits from Frame class in Tkinter module. |

|  |  |  |
| --- | --- | --- |
| **Module name:** | **Pages.py** | |
| **Modules Description:** | Contains the page classes that are essentially frames to change the window controlled by the TicApp Class in the module TicTacToeApp.py | |
| **Class Responsibilities (data and/or methods):** | | **Class Collaborations (other classes):** |
| **Classes:**   * **StartPage(Frame):** Contains everything to be shown on start page. * **BoardSize(Frame):** Contains everything to be shown on choosing board page * **Instructions(Frame):** Contains everything to show on Instructions page * **Ai3X3(Frame):** Contains everything to show on a 3X3 single player game board page * **Multi3X3(Frame):** Contains everything to show on a 3X3 multi player game board page * **Ai4X4(Frame):** Contains everything to show on a 4X4 single player game board page * **Multi4X4(Frame):** Contains everything to show on a 4X4 multi player game board page | | ● Inherits from Frame class in Tkinter module.   * TicTacToe Class * Ai Class * Frame * Tk * Button * Label |

* ***Enhancement***: First I implemented the back end of the program, and made sure each of the classes and function work, then I enhance the program by implementing the graphical user interface using the tkinter library.
* ***Functionality***:
  + ***Single Player Mode***
  + ***Multi Player Mode***
  + ***3x3 Tic Tac Toe game board***
  + ***4x4 Tic Tac Toe game board***
* ***Files***:

1. Boards.py
2. AIntel.py
3. TicTacToeApp.py
4. Pages.py
5. Test.py
6. Main\_development.py: This file was created for development purposes
7. Tic-Tac-Toe-Game.ico
8. Agarwali-CSC236-FP.docx

* ***Utilized Data Structures:*** I have used multiple data structures in this project. My main recursive function self.alphabeta in AIntel.py module is implemented using a recursive function that takes advantage of the runtime stack.  I also used a 2-D Numpy array to record the data on the game board, which is actually implemented using hash tables, that increases making searches faster. I also used dictionaries several times, like to store buttons on the Board GUI and to store all the frames/pages shown in the tkinter window.
* ***Big O Analysis:*** The Big-O analysis of the main part of the code, the self.alphabeta function in AI() class in AIntel.py module has a Big O can be calculated with an (average or constant) branching factor of b, and a search depth of d plies, the maximum number of leaf node positions evaluated (when the move ordering is pessimal) is O(b\*b\*...\*b) = O(b^d)
* ***Resources***: No external software used. The following internal libraries have been used: Numpy, Tkinter. I was inspired from this link <https://www.ocf.berkeley.edu/~yosenl/extras/alphabeta/alphabeta.html> to understand how I can implement this design. I also to referred to StackOverflow several times, the lines have been appropriately cited in my modules.
* ***Challenges***: The biggest challenge for me was to implement the self.aplhabeta recursive function and designing the user interface using Tkinter.
* ***Testing***: All the possible functions that can be tested were implemented as test suite in test.py function.
* ***Errors***: The program takes a lot of computation time when user player is single player mode on a 4x4 board. I am planning to create an heuristic approach for the 4x4 single payer game board.
* ***Measures and Assessment***: My initial target was only to implement the artificial intelligence on a 3x3 game board using a recursive function. But this project has been a great learning curve for me. Whenever, I reached my goal I increased my boundary of measure and assessments to allow me to do more. I think have successfully implemented the targets I had started with.
* ***Summary***: As I mentioned before this project was a great learning curve for me. I started by thinking by how humans think when they try to beat their opponent. I wanted to implement the same strategy in a code that computer could understand. I drew a lot of diagrams in the big chalkboards of Draper to understand and create and algorithm for the artificial intelligence of the computer. Once I had a clear understanding I wrote the design plan. I implemented the classes, and tested my functions as I developed. Finally, I implemented the graphics using tkinter library.
* ***Comments***: This final project was great opportunity for me to learn and create new things that I am proud of. If someone wanted to use my tkinter page controlling API it will be available soon on my [github](https://github.com/agarwali/). I am planning to write a better documentation on that.