horizontale Linie

**Beuth University of Applied Science**

Advance Software – Game Project

Prof. Dr. Edlich Stefan

Tic Tac Toy – Game Project

**Basem Dabbour**

# Introduction about Tic Tac Toy :

Tic Tac Toe is a game against a simple artificial intelligence. An **artificial intelligence** (or **AI**) is a computer program that can intelligently respond to the player’s moves. This game doesn’t introduce any complicated new concepts. The artificial intelligence that plays Tic Tac Toe is really just a few lines of code.*.*

Back when we were a kids , two childrens used to play Tic Tac toy with paper and pencil when one of them is “X“ and the other is “O“ and if one of the players get three of the their marks on the bord in row or column or one of the two diagonals , they WIN

And when the bord fills up with neither player winning the game break even.

## C:\Users\Basem 3\Desktop\AS-GAME\1.PNGSimple run for Tic Tac Toy Game :

Figure - Simple run for TTT Game

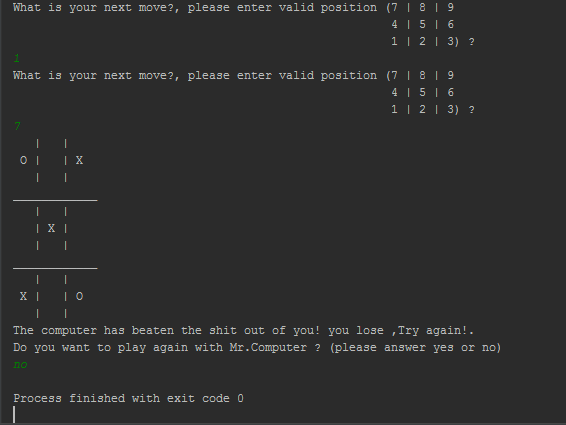


Figure - Simple run of TTT Game 2

## The Tic Tac Toy source code :

Please copy the below code to your shell and run it :

# Tic Tac Toe  
  
import random  
  
  
def Gameboard(board):  
 # This function prints out the board that it was passed.  
  
 # "board" is a list of 10 strings representing the board (ignore index 0)  
 print(' | |')  
 print(' ' + board[ 7 ] + ' | ' + board[ 8 ] + ' | ' + board[ 9 ])  
 print(' | |')  
 #print('-----------')  
 print('\_\_\_\_\_\_\_\_\_\_\_\_')  
 print(' | |')  
 print(' ' + board[ 4 ] + ' | ' + board[ 5 ] + ' | ' + board[ 6 ])  
 print(' | |')  
 # print('-----------')  
 print('\_\_\_\_\_\_\_\_\_\_\_\_')  
 print(' | |')  
 print(' ' + board[ 1 ] + ' | ' + board[ 2 ] + ' | ' + board[ 3 ])  
 print(' | |')  
  
  
def inputPlayerletter():  
 # Lts the player type which letter they want to be.  
 # Returns a list with the player’s letter as the first item, and the computer's letter as the second.  
 letter = ''  
 while not (letter == 'X' or letter == 'O'):  
 print('Kindly choose what letter do you want to be X or O?')  
 letter = input().upper()  
  
 # the first element in the list is the player’s letter, the second is the computer's letter.  
 if letter == 'X':  
 return [ 'X', 'O' ]  
 else:  
 return [ 'O', 'X' ]  
  
  
def whoGoesFirst():  
 # Randomly choose the player who goes first.  
 if random.randint(0, 1) == 0:  
 return 'computer'  
 else:  
 return 'player'  
  
  
def playAgain():  
 # This function returns True if the player wants to play again, otherwise it returns False.  
 print('Do you want to play again with Mr.Computer ? (please answer yes or no)')  
 return input().lower().startswith('y')  
  
  
def MakeAMove(board, letter, move):  
 board[ move ] = letter  
  
  
def isWinner(b, L):  
 # Given a board and a player’s letter, this function returns True if that player has won.  
 # We use b instead of board and L instead of letter so we don’t have to type as much.  
 return ((b[ 7 ] == L and b[ 8 ] == L and b[ 9 ] == L) or # across the top  
 (b[ 4 ] == L and b[ 5 ] == L and b[ 6 ] == L) or # across the middle  
 (b[ 1 ] == L and b[ 2 ] == L and b[ 3 ] == L) or # across the bottom  
  
 (b[ 7 ] == L and b[ 4 ] == L and b[ 1 ] == L) or # down the Lft side  
 (b[ 8 ] == L and b[ 5 ] == L and b[ 2 ] == L) or # down the middle  
 (b[ 9 ] == L and b[ 6 ] == L and b[ 3 ] == L) or # down the right side  
  
 (b[ 7 ] == L and b[ 5 ] == L and b[ 3 ] == L) or # diagonal  
 (b[ 9 ] == L and b[ 5 ] == L and b[ 1 ] == L)) # diagonal  
  
  
def getboardCopy(board):  
 # Make a duplicate of the board list and return it the duplicate.  
 dupeboard = [ ]  
  
 for i in board:  
 dupeboard.append(i)  
  
 return dupeboard  
  
  
def FreeSpace(board, move):  
 # Return true if the passed move is free on the passed board.  
 return board[ move ] == ' '  
  
  
def getPlayerMove(board):  
 # Lt the player type in their move.  
 move = ' '  
 while move not in '1 2 3 4 5 6 7 8 9'.split() or not FreeSpace(board, int(move)):  
 print('What is your next move?, please enter valid position (7 | 8 | 9 ',"\n"  
 ' 4 | 5 | 6 ' "\n"  
 ' 1 | 2 | 3) ?')  
 move = input()  
 return int(move)  
  
  
def chooseRandomMoveFromList(board, movesList):  
 # Returns a valid move from the passed list on the passed board.  
 # Returns None if there is no valid move.  
 possibLMoves = [ ]  
 for i in movesList:  
 if FreeSpace(board, i):  
 possibLMoves.append(i)  
  
 if len(possibLMoves) != 0:  
 return random.choice(possibLMoves)  
 else:  
 return None  
  
  
def getComputerMove(board, computerletter):  
 # Given a board and the computer's letter, determine where to move and return that move.  
 if computerletter == 'X':  
 playerletter = 'O'  
 else:  
 playerletter = 'X'  
  
 # Here is our algorithm for our Tic Tac Toe AI:  
 # First, check if we can win in the next move  
 for i in range(1, 10):  
 copy = getboardCopy(board)  
 if FreeSpace(copy, i):  
 MakeAMove(copy, computerletter, i)  
 if isWinner(copy, computerletter):  
 return i  
  
 # Check if the player could win on their next move, and block them.  
 for i in range(1, 10):  
 copy = getboardCopy(board)  
 if FreeSpace(copy, i):  
 MakeAMove(copy, playerletter, i)  
 if isWinner(copy, playerletter):  
 return i  
  
 # Try to take one of the corners, if they are free.  
 move = chooseRandomMoveFromList(board, [ 1, 3, 7, 9 ])  
 if move != None:  
 return move  
  
 # Try to take the center, if it is free.  
 if FreeSpace(board, 5):  
 return 5  
  
 # Move on one of the sides.  
 return chooseRandomMoveFromList(board, [ 2, 4, 6, 8 ])  
  
  
def isboardFull(board):  
 # Return True if every space on the board has been taken. Otherwise return False.  
 for i in range(1, 10):  
 if FreeSpace(board, i):  
 return False  
 return True  
  
  
print('Welcome to AI-Game advance software project ,its Tic Tac Toe!!')  
  
while True:  
 # Reset the board  
 TheBoard = [ ' ' ] \* 10  
 playerletter, computerletter = inputPlayerletter()  
 turn = whoGoesFirst()  
 print('The ' + turn + ' will go first as player1 to make first move .')  
 gameIsPlaying = True  
  
 while gameIsPlaying:  
 if turn == 'player':  
 # Player’s turn.  
 Gameboard(TheBoard)  
 move = getPlayerMove(TheBoard)  
 MakeAMove(TheBoard, playerletter, move)  
  
 if isWinner(TheBoard, playerletter):  
 Gameboard(TheBoard)  
 print('very impressive !! You have won the game, Smart Ass!!')  
 gameIsPlaying = False  
 else:  
 if isboardFull(TheBoard):  
 Gameboard(TheBoard)  
 print('The game is a tie!, no one won !')  
 break  
 else:  
 turn = 'computer'  
  
 else:  
 # Computer’s turn.  
 move = getComputerMove(TheBoard, computerletter)  
 MakeAMove(TheBoard, computerletter, move)  
  
 if isWinner(TheBoard, computerletter):  
 Gameboard(TheBoard)  
 print('The computer has beaten the shit out of you! you lose ,Try again!.')  
 gameIsPlaying = False  
 else:  
 if isboardFull(TheBoard):  
 Gameboard(TheBoard)  
 print('The game is a tie!')  
 break  
 else:  
 turn = 'player'  
  
 if not playAgain():  
 break

## The desigen of the game program :

Here is how the Flowchart for Tic Tac Toy look like, in this game the player (you) will choose between X and O and who take the first turn will be randomly choosen by using random module in python   
“ import random ” :

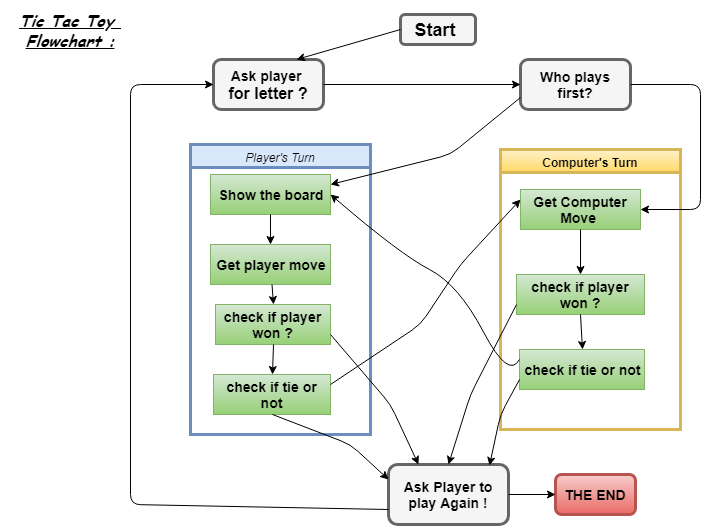


Figure - TTT Flowchart

Note : the board is numbered as keyboard number pad as per the following sample :

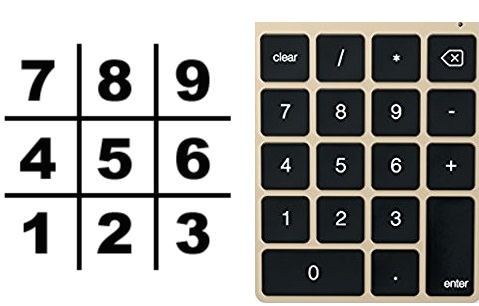


Figure - Board numbering

# UML Diagrams :

* **.Class Diagram :**

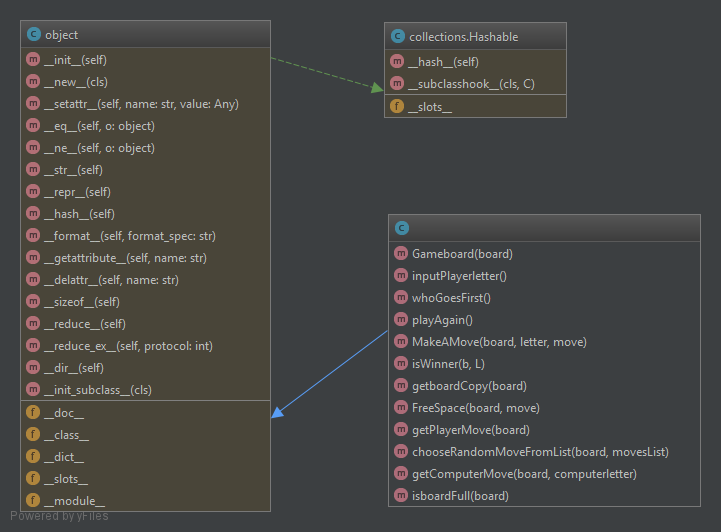
****

Figure - Class Diagram

* **Use Case Diagram :**

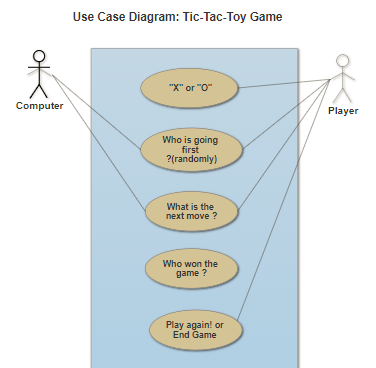
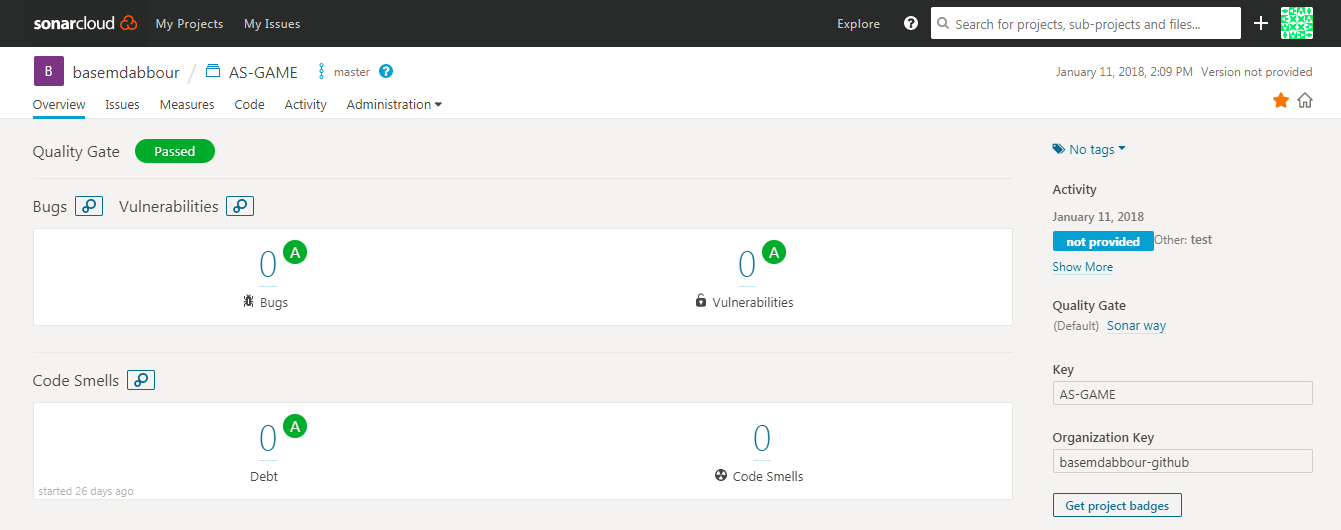


Figure - Use Case Diagram

## SonarQube – Metrics :

With SonarQube we can check how much the code is clean from bugs and vulnrability

By creating new project with token name either existing one or generate new linked to new project to test and analyse any project :

# Bag of features

Since the standard Bag of Words (BOW) approach is very slow in feature extraction we are attempting the Bag of Features approach which is also already implemented in the fastText Library.

The step-by-step approach would consist of the following steps: First, the snowball method is used for word stemming. Second we calculate the distribution above all the comments in the training data, where we use Term Frequency (TF- IDF). In concrete this means to give a higher importance to words occurring less in the submitted comments.

The occurring problem is the similarity of comments based on how many matching words there are in a comment.

In the following section we are introducing a possible solution.

# Word embedding

Word embeddings encodes words that are semantically similar with similar vectors. For this we solely rely on fastText for text classification.

Using BOF with fastText results in a comment representation based on the varying number of vectors depending on the number of words in a comment.

The difficulty is to retrieve a single vector for each comment, not for every word. Vector representation over a whole sentence is very inaccurate. Therefore we apply a simple but efficient solution by calculating the mean of the word encodings.

## Model deployment

Our model deployment consist of a combination of field experiments, manual inspection of the most prominent comments. It is based on experiments of the company. A feature extraction combined with the best working machine learning algorithm. The best algorithmes are described by a high true positive Rate (TPR) and a low False Positive Rate (FPR).

## Training data

For the best outcome it would be ideal if the model would be trained with a combination of manually labeled training data, as well as classified tweets and comments in combination with classified words, where the model which is trained by classified words is used to train the model for hate speech detection, in order to keep the structure of the Neuronal Network as well as keeping the problem of the format of the input (input has to be the same format) manageable.

The detection will be running once a day, with both classified comments and classified words. The results are reviewed manually and used to retrain our model.

Since the content of hate speech can change rapidly due to current events, like a terrorist attack in France for example, the faster the iteration the better the model.

## Success indicators

We offer an online user interface with hate speech flagged comments. Success can be measured in resources like reduced moderation time and thus reduced costs. Success can be seen immediately in form of a clean comment section or discussion board and less conflict with the law.

# Planning

The next 6 month consist of meet ups’ with potential customers and people who work on a similar project. We will also do deep research and collect data sets to develop a model that has reliable detection.

## Risks

Our model needs to be frequently updated to stay reliable. As a solution we offer our gold subscription for free in exchange for constant feedback.

Another risk would be running out of money, which we avoid by being very scalable. Expanding means just setting up more servers.

## Advisors

According to CJ Adams, a product manager at Jigsaw, the Google social incubator that built a hate speech detector which is currently tested by big Publications as the *New York Times* they are interested in working with small developers since we all have a shared interest and benefit from healthy online discussions.

In our collective opinion it is most reasonable to join an existing group of researches for collaboration instead of fighting against competitors. We would appreciate to form Think-Tanks with other researchers and groups of interest, in order to advice each other.

A platform for interaction could be the Association of Internet Researchers with their annual conference.

Help could also be found at local or international meetup’s like NLP meetup. Eventually we have an experienced investor that could advise us.

## Market size

The german market is not only our keymarket. Hate Speech detection is pulling great global attention. Facebook for example is employing thousands of people to detect hate speech.

## Competitors

Facebook got into this market not even as user but also as developer of hate speech detection releasing fastText. Google has their hands on this topic already working with the *New York Times*, *the Guardian*, *Financial Times* as well as *BBC*. Within Germany *NoHate*, *HAWK* and *conversar.io* are the leading companies.

Our product has an open API and an interesting subscription model, which includes customized solution or our software on premise. Also our usability is more convenient.