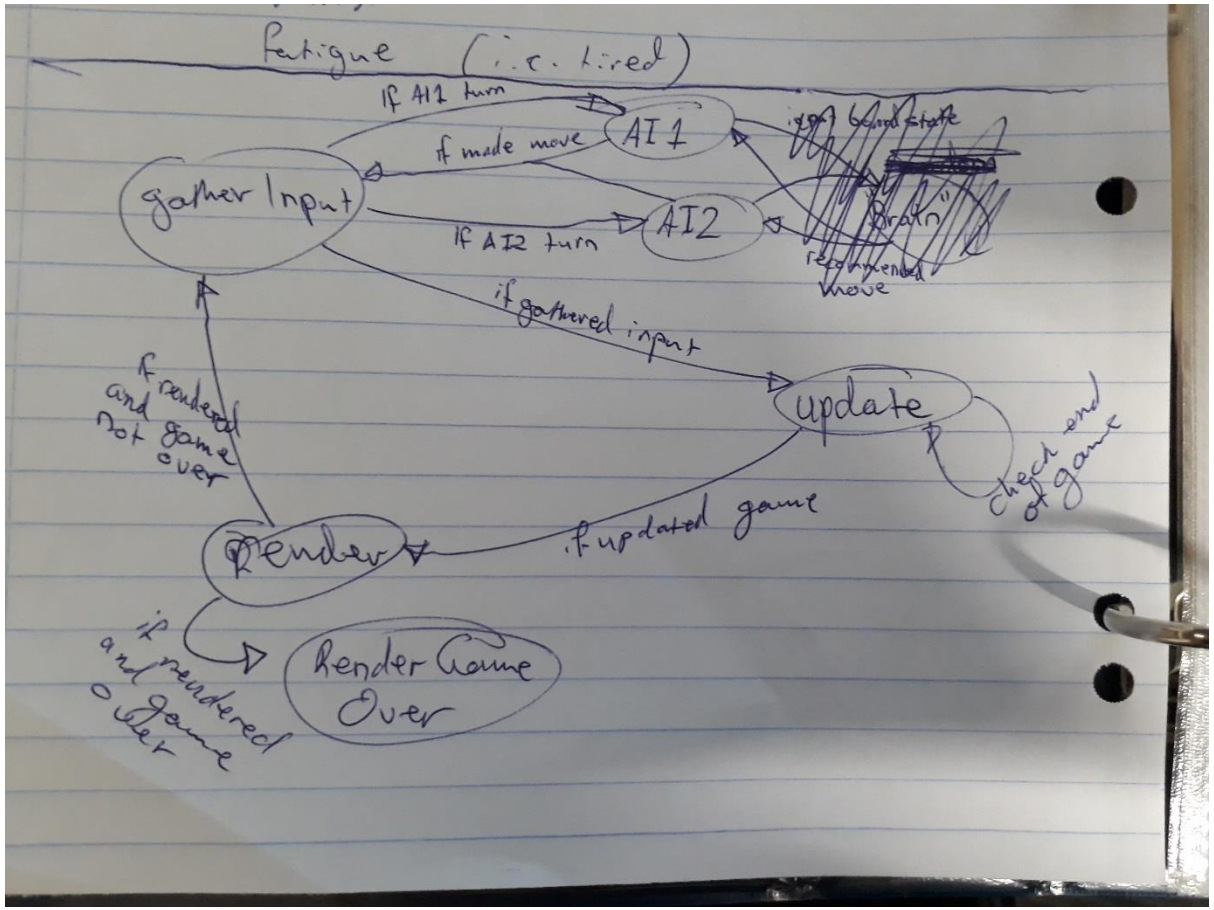
# Tic Tac Toe Task Code

## Game Loop Diagram



## TicTacToe.py Code

```
import TicTacToeGame as gameModule

while True:
    game = gameModule.TicTacToeGame()

    replay = game.GameLoop()

    if replay is not True:
        break
```

## TicTacToeGame.py Code

```
import AI1 as ai1Module
import AI2 as ai2Module

class TicTacToeGame:
    def __init__(self):
        self.ai1 = ai1Module.AI1()
        self.ai2 = ai2Module.AI2()

        self.input = 0
        self.currentAI = 0
        self.boardSpaces = [" ", " ", " ", " ", " ", " ", " ", " ", " "]
        self.acceptableBoardSpaces = [1, 2, 3, 4, 5, 6, 7, 8, 9]

        # self.finished = False
        self.gameOver = False
        self.winner = None

    def GameLoop(self):
        result = None

        while not self.gameOver:
            self.input = self.GatherInput(self.boardSpaces, self.acceptableBoardSpaces)

            self.Update(self.currentAI, self.input, self.boardSpaces, self.acceptableBoardSpaces)

            self.Render(self.currentAI, self.boardSpaces)
```

```
        result = input("Play again? (Y/N): ")
        if result in ["Y", "y", "Yes", "yes"]:
            return True
        else:
            return False

    def GatherInput(self, boardSpaces, acceptableSpaces):
        # increment which AI's turn it is, and let it have its turn
        if self.currentAI == 1:
            self.currentAI = 2
            return self.ai2.MakeMove(boardSpaces, acceptableSpaces)
        else:
            self.currentAI = 1
            return self.ai1.MakeMove(boardSpaces, acceptableSpaces)

    def Update(self, ai, input, boardSpaces, acceptableSpaces):
        # update game state based on which AI's turn it is and what move they made
        boardSpaces[input - 1] = self.GetAIMark(ai)
        acceptableSpaces.remove(input)

        # check if an AI has won the game
        self.CheckGameWon(boardSpaces, ai)

        # check if all spaces on the board have been filled
        if len(acceptableSpaces) == 0:
            self.gameOver = True

    def GetAIMark(self, ai):
        if ai == 1:
            return "X"
        else:
            return "O"

    def CheckGameWon(self, boardSpaces, ai):
        #local variables
        aiSpaces = []

        # check which spaces have been occupied by which player
        for i in range (len(boardSpaces)):
```

```
            if boardSpaces[i] == self.GetAIMark(ai):
                aiSpaces.append(str(i + 1))

        if len(aiSpaces) > 2 and self.FoundWinningCombination(aiSpaces):
            self.gameOver = True
            if ai == 1:
                self.winner = "AI1"
            else:
                self.winner = "AI2"

    # check for winning combinations
    def FoundWinningCombination(self, aiSpaces):

        if self.CheckCombination(aiSpaces, ["1", "2", "3"]):
                return True
        elif self.CheckCombination(aiSpaces, ["4", "5", "6"]):
                return True
        elif self.CheckCombination(aiSpaces, ["7", "8", "9"]):
                return True
        elif self.CheckCombination(aiSpaces, ["1", "4", "7"]):
                return True
        elif self.CheckCombination(aiSpaces, ["2", "5", "8"]):
                return True
        elif self.CheckCombination(aiSpaces, ["3", "6", "9"]):
                return True
        elif self.CheckCombination(aiSpaces, ["1", "5", "9"]):
                return True
        elif self.CheckCombination(aiSpaces, ["3", "5", "7"]):
                return True
        else:
            return False

    # check a specified winning combination
    def CheckCombination(self, aiSpaces, set):
        for i in range(len(set)):
            if (set[i] not in aiSpaces):
                return False

        return True
```

```
    # render game state to the terminal
    def Render(self, ai, boardSpaces):
        '''
            Display the space numbers on the game board to the screen:
             1 | 2 | 2
            -----------
             4 | 5 | 6
            -----------
             7 | 8 | 9
        '''

        # display the results of the AI's move
        print("AI no. " + str(ai) + " placed an " + self.GetAIMark(ai) + " in space number " + str(self.input))

        # Display the current game board to screen
        print('    %s | %s | %s' % tuple(boardSpaces[:3]))
        print('   -----------')
        print('    %s | %s | %s' % tuple(boardSpaces[3:6]))
        print('   -----------')
        print('    %s | %s | %s' % tuple(boardSpaces[6:]))

        # Check if the game is over
        if self.gameOver:
            print("Game Over!")
            if self.winner != None:
                print("The winner is " + self.winner)
        else:
            input("Press enter to continue")
```

## AI1.py Code

```
import random

class AI1:
    def MakeMove(self, boardSpaces, acceptableSpaces):
        # check for spaces that could complete a 3 in a row; such spaces can only appear for AI1 if 4 spaces are occupied (i.e. if
both AIs have placed 2 marks)
        if (len(acceptableSpaces) <= 5): # if there is enough spaces filled that it is theoretically possible to win
            result = self.CheckWinningMove(acceptableSpaces, boardSpaces)
```

```
            if result is not None:
                return result

        return self.CheckNonWinningMove(acceptableSpaces)

    def CheckWinningMove(self, acceptableSpaces, boardSpaces):
        result = None
        possibleAtk3s = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [1, 4, 7], [2, 5, 8], [3, 6, 9], [1, 5, 9], [3, 5, 7]]
        atk3Spaces = []

        for i in range(len(acceptableSpaces)): #for each playable space
            if (acceptableSpaces[i] not in atk3Spaces): # if it is not already designated as a winning space
                for j in range(len(possibleAtk3s)): #for each theoretical set of winning combinations
                    if (acceptableSpaces[i] in possibleAtk3s[j]): # if the playable space is in that set
                        if (self.CheckIfSetIsWinnable(acceptableSpaces[i], possibleAtk3s[j], boardSpaces)):
                            atk3Spaces.append(acceptableSpaces[i])

            if len(atk3Spaces) > 0:
                if len(atk3Spaces) is 1:
                    result = atk3Spaces[0]
                else:
                    result = atk3Spaces[random.randrange(0, len(atk3Spaces) - 1)]

        return result

    def CheckIfSetIsWinnable(self, emptySpace, winnableSet, board):
        for i in range(len(winnableSet)):                                                   # for each space in the set
            if (winnableSet[i] is not emptySpace and board[winnableSet[i] - 1] is not "X"):     # if the space specified is not an
empty space AND it does not have an "X" in it
                return False                                                                # the space is taken by the opponent and
is not usable

        return True                                                                         # if all spaces besides emptySpace are
Xs, method returns True, saying this is a winnable set

    def CheckNonWinningMove(self, acceptableSpaces):
        #pick a random space from acceptableSpaces
        if len(acceptableSpaces) is 9:
            result = acceptableSpaces[4]
        elif len(acceptableSpaces) > 1:
```

```
            result = acceptableSpaces[random.randrange(0, len(acceptableSpaces) - 1)]
        else:
            result = acceptableSpaces[0]

        return result
```

## Ai2.py Code

```
import random

class AI1:
    def MakeMove(self, boardSpaces, acceptableSpaces):
        # check for spaces that could complete a 3 in a row; such spaces can only appear for AI1 if 4 spaces are occupied (i.e. if
both AIs have placed 2 marks)
        if (len(acceptableSpaces) <= 5): # if there is enough spaces filled that it is theoretically possible to win
            result = self.CheckWinningMove(acceptableSpaces, boardSpaces)

            if result is not None:
                return result

        return self.CheckNonWinningMove(acceptableSpaces)

    def CheckWinningMove(self, acceptableSpaces, boardSpaces):
        result = None
        possibleAtk3s = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [1, 4, 7], [2, 5, 8], [3, 6, 9], [1, 5, 9], [3, 5, 7]]
        atk3Spaces = []

        for i in range(len(acceptableSpaces)): #for each playable space
            if (acceptableSpaces[i] not in atk3Spaces): # if it is not already designated as a winning space
                for j in range(len(possibleAtk3s)): #for each theoretical set of winning combinations
                    if (acceptableSpaces[i] in possibleAtk3s[j]): # if the playable space is in that set
                        if (self.CheckIfSetIsWinnable(acceptableSpaces[i], possibleAtk3s[j], boardSpaces)):
                            atk3Spaces.append(acceptableSpaces[i])

            if len(atk3Spaces) > 0:
                if len(atk3Spaces) is 1:
                    result = atk3Spaces[0]
                else:
                    result = atk3Spaces[random.randrange(0, len(atk3Spaces) - 1)]
```

```
        return result

    def CheckIfSetIsWinnable(self, emptySpace, winnableSet, board):
        for i in range(len(winnableSet)):                                   # for each space in the set
            if (winnableSet[i] is not emptySpace and board[winnableSet[i] - 1] is not "X"):     # if the space specified is not an
empty space AND it does not have an "X" in it
                return False                                                # the space is taken by the opponent and
is not usable

        return True                                                         # if all spaces besides emptySpace are
Xs, method returns True, saying this is a winnable set

    def CheckNonWinningMove(self, acceptableSpaces):
        #pick a random space from acceptableSpaces
        if len(acceptableSpaces) is 9:
            result = acceptableSpaces[4]
        elif len(acceptableSpaces) > 1:
            result = acceptableSpaces[random.randrange(0, len(acceptableSpaces) - 1)]
        else:
            result = acceptableSpaces[0]

        return result
```