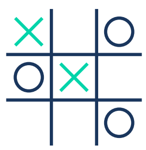
Effat University • Computer Science Department

CS3081: Artificial Intelligence

Spring 2026

Assignment 1: Tic-Tac-Toe

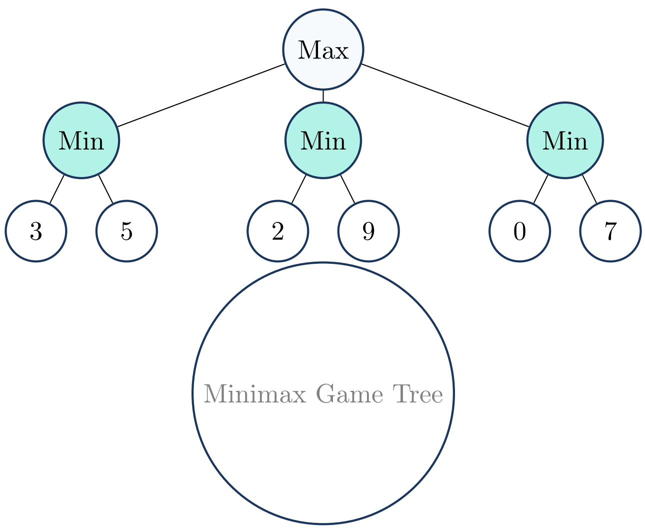
Using Minimax to Build an Unbeatable AI



Instructor: Dr. Naila Marir
Total Points: 10 points
Submission: GitHub Repository Link

1 Overview

In this assignment, you will implement an AI that plays **Tic-Tac-Toe optimally** using the **Minimax algorithm**. When completed, your AI will be unbeatable — the best a human opponent can achieve is a tie!



2 Learning Objectives

Upon completing this assignment, you will be able to:

- Implement game state representation and manipulation
- Apply the **Minimax algorithm** for adversarial search
- Understand concepts of **terminal states** and **utility functions**
- Build an AI agent that plays optimally in a two-player game
- (Bonus) Implement **Alpha-Beta Pruning** for efficiency

3 Getting Started

3.1 Download the Distribution Code

Download the starter code from the course page and unzip it. You should have the following files:

File	Description
`tictactoe.py`	Your code goes here (functions to implement)
`runner.py`	Graphical interface (do not modify)
`requirements.txt`	Required packages

3.2 Install Dependencies

Open a terminal in the project directory and run:

```
pip install -r requirements.txt
```

This will install `pygame`, which is required for the graphical interface.

3.3 Run the Game

Once you've completed all required functions, you can play against your AI:

```
python runner.py
```

4 Understanding the Code

4.1 File Structure

- `tictactoe.py` — Contains all the logic for playing the game and making optimal moves. **This is the file you will edit.**
- `runner.py` — Contains the graphical interface code using pygame. Do not modify this file.

4.2 Provided Variables

In `tictactoe.py`, three variables are already defined:

```
X = "X"
O = "O"
EMPTY = None
```

These represent the possible values for each cell on the board.

4.3 Board Representation

The board is represented as a **list of three lists**, where each inner list represents a row:

```python
# Example board state:
board = [
    [X, O, EMPTY],
    [O, X, EMPTY],
    [EMPTY, EMPTY, EMPTY]
]
```

X	O	(0,2)
O	X	(1,2)
(2,0)	(2,1)	(2,2)

Board indices: (row, column)

5 Specification

You must implement the following **seven functions** in `tictactoe.py`:

5.1 `player(board)`

Function Signature

```python
def player(board):
```

Input: A board state
Output: Returns which player's turn it is (`X` or `O`)
Rules:

- In the initial game state, `X` gets the first move
- Players alternate turns after each move
- Any return value is acceptable if the board is terminal (game over)

Example:

```python
board = [[X, O, EMPTY], [EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY]]
player(board)  # Returns X (2 moves made: 1 X, 1 O, so X's turn)
```

5.2 `actions(board)`

Function Signature

```python
def actions(board):
```

Input: A board state
Output: A `set` of all possible actions (moves) on the board
Rules:

- Each action is a tuple `(i, j)` where:
  - `i` = row (0, 1, or 2)
  - `j` = column (0, 1, or 2)
- Possible moves are cells that are `EMPTY`
- Any return value is acceptable if the board is terminal

Example:

```
board = [[X, O, EMPTY], [EMPTY, X, EMPTY], [EMPTY, EMPTY, O]]
actions(board)  # Returns {(0,2), (1,0), (1,2), (2,0), (2,1)}
```

5.3 `result(board, action)`

Function Signature

```
def result(board, action):
```

Input: A board state and an action tuple `(i, j)`
Output: A **new board state** after the action is taken

△ **Important**

Do not modify the original board! The Minimax algorithm needs to explore many board states. You must create a **deep copy** of the board before making changes.

```
import copy
new_board = copy.deepcopy(board)
```

Rules:

- If the action is invalid, raise an exception
- The returned board should reflect the move by the current player

---

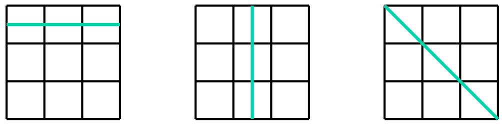
5.4 `winner(board)`

Function Signature

```
def winner(board):
```

Input: A board state
Output: The winner (`X`, `O`, or `None`)
Rules:

- Return `X` if X has three in a row (horizontal, vertical, or diagonal)
- Return `O` if O has three in a row
- Return `None` if there is no winner yet (game in progress or tie)



Horizontal Vertical Diagonal

5.5 `terminal(board)`

Function Signature

```
def terminal(board):
```

Input: A board state
Output: `True` if the game is over, `False` otherwise
Rules:

- Return `True` if someone has won
- Return `True` if all cells are filled (tie)
- Return `False` if the game is still in progress

5.6 `utility(board)`

Function Signature

```
def utility(board):
```

Input: A **terminal** board state
Output: The utility value of the board
Rules:

- Return `1` if X has won
- Return `-1` if O has won
- Return `0` if the game is a tie

Info

You may assume `utility()` will only be called on a board where `terminal(board)` is `True`.

5.7 `minimax(board)`

Function Signature

```
def minimax(board):
```

Input: A board state
Output: The optimal action `(i, j)` for the current player
Rules:

- Return the optimal move as a tuple `(i, j)`
- If multiple moves are equally optimal, any of them is acceptable
- If the board is terminal, return `None`

Algorithm Overview:

- **X** is the **maximizing** player (wants highest score)
- **O** is the **minimizing** player (wants lowest score)
- Recursively evaluate all possible game states
- Choose the action that leads to the best outcome

6 Hints

Hint

1. **Testing individual functions:** You can test your functions in a separate Python file:

```python
from tictactoe import initial_state, player, actions
board = initial_state()
print(player(board))  # Should print X
```

2. **Deep copy:** Use `copy.deepcopy()` in the `result()` function to avoid modifying the original board.
3. **Helper functions:** You may add additional helper functions (e.g., `max_value()`, `min_value()`) to implement Minimax.
4. **Alpha-Beta Pruning:** This optimization is **optional** but can make your AI run faster. It's worth bonus points!
5. **Counting moves:** To determine whose turn it is, count the number of X's and O's on the board.

7 Grading Rubric

Function	Points	Criteria
`player(board)`	10	Correctly returns whose turn it is
`actions(board)`	10	Returns all valid empty cells as tuples
`result(board, action)`	15	Returns new board without modifying original
`winner(board)`	15	Correctly identifies winner (rows, cols, diagonals)
`terminal(board)`	10	Correctly identifies game over states
`utility(board)`	10	Returns correct utility values (1, -1, 0)
`minimax(board)`	25	Returns optimal move using Minimax
Code Quality	5	Clean, readable, well-commented code
Total	**100**	
Bonus: Alpha-Beta	+10	Implement Alpha-Beta Pruning

8 Testing Your Code

8.1 Manual Testing

Test each function individually before running the full game:

```
# test_functions.py
from tictactoe import *

# Test player()
board1 = initial_state()
assert player(board1) == X, "X should go first"

board2 = [[X, EMPTY, EMPTY], [EMPTY, EMPTY, EMPTY], [EMPTY, EMPTY,
    EMPTY]]
assert player(board2) == O, "O should go after X"

# Test actions()
board3 = [[X, O, X], [O, X, O], [EMPTY, EMPTY, EMPTY]]
assert actions(board3) == {(2,0), (2,1), (2,2)}

# Test winner()
board_x_wins = [[X, X, X], [O, O, EMPTY], [EMPTY, EMPTY, EMPTY]]
assert winner(board_x_wins) == X

# Test terminal()
assert terminal(board_x_wins) == True
assert terminal(initial_state()) == False

print("All tests passed!")
```

8.2 Playing Against Your AI

Once all functions are implemented:

```
python runner.py
```

⚠ Important

Since Tic-Tac-Toe is a tie given optimal play by both sides, you should **never be able to beat the AI**. If you can beat it, there's a bug in your Minimax implementation!

9 What to Submit

Submit a **GitHub repository link** containing:

File	Description
`tictactoe.py`	Your completed implementation
`runner.py`	Original file (unmodified)
`requirements.txt`	Original file
`README.md`	Explanation of your approach (see below)

9.1 README.md Requirements

Your README file should include:

1. **Your Name** and Student ID
2. **Brief Description** of your implementation approach
3. **Challenges** you faced and how you solved them
4. **Bonus:** If you implemented Alpha-Beta Pruning, explain how it works
5. **Screenshots** of your AI playing (optional but encouraged)

10 Academic Integrity

⚠ Important

- This is an **individual assignment**. You must write your own code.
- You may discuss concepts with classmates, but do not share code.
- You may use the Python standard library, but no external AI libraries.
- Plagiarism detection tools will be used to check submissions.
- Violations will result in a zero grade and disciplinary action.

11 Resources

- **Lecture Slides:** Adversarial Search and Minimax Algorithm
- **Textbook:** Russell & Norvig, Chapter 5 (Adversarial Search)

Good luck and have fun building your AI!

Dr. Naila Marir

CS3081 • Artificial Intelligence • Spring 2026