COMP250: Artificial Intelligence

2: Designing AI behaviours

Noughts and Crosses

Clone the following repository:

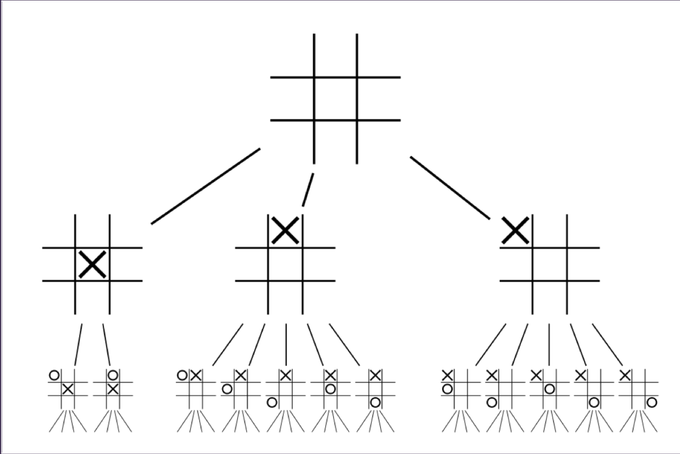
`https://github.com/Falmouth-Games-Academy/bsc-live-coding`

Open `COMP250/02_oxo` in PyCharm and run `oxo.py`

FALMOUTH UNIVERSITY

Minimax search

Game trees



Minimax

Minimax

- Terminal game states have a **value**

Minimax

- Terminal game states have a **value**
  - E.g. $+1$ for a win, -1 for a loss, 0 for a draw

Minimax

- Terminal game states have a **value**
  - E.g. $+1$ for a win, -1 for a loss, 0 for a draw
- I want to **maximise** the value

Minimax

- Terminal game states have a **value**
  - E.g. $+1$ for a win, -1 for a loss, 0 for a draw
- I want to **maximise** the value
- My opponent wants to **minimise** the value

Minimax

- Terminal game states have a **value**
  - E.g. $+1$ for a win, -1 for a loss, 0 for a draw
- I want to **maximise** the value
- My opponent wants to **minimise** the value
- Therefore I want to **maximise** the **minimum** value my opponent can achieve

Minimax

- Terminal game states have a **value**
  - E.g. $+1$ for a win, -1 for a loss, 0 for a draw
- I want to **maximise** the value
- My opponent wants to **minimise** the value
- Therefore I want to **maximise** the **minimum** value my opponent can achieve
- This is generally only true for **two-player zero-sum** games

Minimax search

Minimax search

- Recursively defines a **value** for non-terminal game states

Minimax search

- Recursively defines a **value** for non-terminal game states
- Consider each possible "next state", i.e. each possible move

Minimax search

- Recursively defines a **value** for non-terminal game states
- Consider each possible “next state”, i.e. each possible move
- If it’s my turn, the value is the **maximum** value over next states

Minimax search

- Recursively defines a **value** for non-terminal game states
- Consider each possible "next state", i.e. each possible move
- If it's my turn, the value is the **maximum** value over next states
- If it's my opponent's turn, the value is the **minimum** value over next states

Minimax search pseudocode

procedure MINIMAX(state, currentPlayer)

Minimax search pseudocode

procedure MINIMAX(state, currentPlayer)
    if state is terminal **then**

Minimax search pseudocode

```
procedure MINIMAX(state, currentPlayer)
    if state is terminal then
        return value of state
```

Minimax search pseudocode

```
procedure MINIMAX(state, currentPlayer)
    if state is terminal then
        return value of state
    else if currentPlayer = 1 then
```

Minimax search pseudocode

procedure MINIMAX(state, currentPlayer)
    if state is terminal **then**
        return value of state
    else if currentPlayer $= 1$ **then**
        bestValue $= -\infty$

Minimax search pseudocode

procedure MINIMAX(state, currentPlayer)
    if state is terminal **then**
        return value of state
    else if currentPlayer $= 1$ **then**
        bestValue $= -\infty$
        for each possible nextState **do**

Minimax search pseudocode

procedure MINIMAX(state, currentPlayer)
    if state is terminal **then**
        return value of state
    else if currentPlayer $= 1$ **then**
        bestValue $= -\infty$
        for each possible nextState **do**
            v = MINIMAX(nextState, 3− currentPlayer)

Minimax search pseudocode

```
procedure MINIMAX(state, currentPlayer)
    if state is terminal then
        return value of state
    else if currentPlayer = 1 then
        bestValue = −∞
        for each possible nextState do
            v = MINIMAX(nextState, 3− currentPlayer)
            bestValue = MAX(bestValue, v)
```

Minimax search pseudocode

```
procedure MINIMAX(state, currentPlayer)
    if state is terminal then
        return value of state
    else if currentPlayer = 1 then
        bestValue = −∞
        for each possible nextState do
            v = MINIMAX(nextState, 3− currentPlayer)
            bestValue = MAX(bestValue, v)
        end for
        return bestValue
```

Minimax search pseudocode

```
procedure MINIMAX(state, currentPlayer)
    if state is terminal then
        return value of state
    else if currentPlayer = 1 then
        bestValue = −∞
        for each possible nextState do
            v = MINIMAX(nextState, 3− currentPlayer)
            bestValue = MAX(bestValue, v)
        end for
        return bestValue
    else if currentPlayer = 2 then
```

Minimax search pseudocode

```
procedure MINIMAX(state, currentPlayer)
    if state is terminal then
        return value of state
    else if currentPlayer = 1 then
        bestValue = −∞
        for each possible nextState do
            v = MINIMAX(nextState, 3− currentPlayer)
            bestValue = MAX(bestValue, v)
        end for
        return bestValue
    else if currentPlayer = 2 then
        bestValue = +∞
        for each possible nextState do
            v = MINIMAX(nextState, 3− currentPlayer)
            bestValue = MIN(bestValue, v)
        end for
        return bestValue
```

Minimax search pseudocode

```
procedure MINIMAX(state, currentPlayer)
    if state is terminal then
        return value of state
    else if currentPlayer = 1 then
        bestValue = −∞
        for each possible nextState do
            v = MINIMAX(nextState, 3− currentPlayer)
            bestValue = MAX(bestValue, v)
        end for
        return bestValue
    else if currentPlayer = 2 then
        bestValue = +∞
        for each possible nextState do
            v = MINIMAX(nextState, 3− currentPlayer)
            bestValue = MIN(bestValue, v)
        end for
        return bestValue
    end if
end procedure
```

Stopping early

```
for each possible nextState do
    v = MINIMAX(nextState, 3− currentPlayer)
    bestValue = MAX(bestValue, v)
end for
```

Stopping early

for each possible nextState **do**
    v = MINIMAX(nextState, 3− currentPlayer)
    bestValue = MAX(bestValue, v)
end for

- State values are always between -1 and $+1$

Stopping early

for each possible nextState **do**
    v = MINIMAX(nextState, 3− currentPlayer)
    bestValue = MAX(bestValue, v)
end for

- State values are always between -1 and $+1$
- So if we ever have bestValue $= 1$, we can stop early

Stopping early

for each possible nextState **do**
    v = MINIMAX(nextState, 3− currentPlayer)
    bestValue = MAX(bestValue, v)
end for

- State values are always between -1 and $+1$
- So if we ever have bestValue $= 1$, we can stop early
- Similarly when minimising if bestValue $= -1$

Using minimax search

Using minimax search

- To decide what move to play next...

Using minimax search

- To decide what move to play next...
- Calculate the minimax value for each move

Using minimax search

- To decide what move to play next...
- Calculate the minimax value for each move
- Choose the move with the maximum score

Using minimax search

- To decide what move to play next...
- Calculate the minimax value for each move
- Choose the move with the maximum score
- If there are several with the same score, choose one at random

Minimax for larger games

Minimax for larger games

- The game tree for noughts and crosses has only a few thousand states

Minimax for larger games

- The game tree for noughts and crosses has only a few thousand states
- Most games are too large to search fully, e.g. chess has $\approx 10^{47}$ states

Minimax for larger games

- The game tree for noughts and crosses has only a few thousand states
- Most games are too large to search fully, e.g. chess has $\approx 10^{47}$ states
- Later we will look at **heuristics** and **pruning** to cut down the size of the tree