COMP250: Artificial Intelligence

5: Game Tree Search

FALMOUTH
UNIVERSITY

Minimax search

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 – example

`minimax.pptx`

Minimax search pseudocode

procedure MINIMAX(state)

Minimax search pseudocode

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

Minimax search pseudocode

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

Minimax search pseudocode

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

Minimax search pseudocode

```
procedure MINIMAX(state)
    if state is terminal then
        return value of state
    else if state.currentPlayer = 1 then
        bestValue = −∞
```

Minimax search pseudocode

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

Minimax search pseudocode

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

Minimax search pseudocode

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

Minimax search pseudocode

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

Minimax search pseudocode

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

Minimax search pseudocode

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

Stopping early

for each possible nextState **do**
    v = MINIMAX(nextState)
    bestValue = MAX(bestValue, v)

Stopping early

for each possible nextState **do**
    v = MINIMAX(nextState)
    bestValue = MAX(bestValue, v)

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

Stopping early

for each possible nextState **do**
    v = MINIMAX(nextState)
    bestValue = MAX(bestValue, v)

- 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)
    bestValue = MAX(bestValue, v)

- 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 and game theory

Minimax and game theory

- For a **two-player zero-sum** game with **perfect information** and **sequential moves**

Minimax and game theory

- For a **two-player zero-sum** game with **perfect information** and **sequential moves**
- Minimax search will always find a **Nash equilibrium**

Minimax and game theory

- For a **two-player zero-sum** game with **perfect information** and **sequential moves**
- Minimax search will always find a **Nash equilibrium**
- I.e. a minimax player plays **perfectly**

Minimax and game theory

- For a **two-player zero-sum** game with **perfect information** and **sequential moves**
- Minimax search will always find a **Nash equilibrium**
- I.e. a minimax player plays **perfectly**
- **But...**

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

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
  - Connect 4 has $\approx 10^{13}$ 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
  - Connect 4 has $\approx 10^{13}$ states
  - Chess has $\approx 10^{47}$ states

FALMOUTH
UNIVERSITY

Heuristics for search

Depth limiting

Depth limiting

- Standard minimax needs to search all the way to **terminal** (game over) states

Depth limiting

- Standard minimax needs to search all the way to **terminal** (game over) states
- **Depth limiting** is a common technique to apply minimax to larger games

Depth limiting

- Standard minimax needs to search all the way to **terminal** (game over) states
- **Depth limiting** is a common technique to apply minimax to larger games
- Still evaluate terminal states as $+1$ / 0 / -1

Depth limiting

- Standard minimax needs to search all the way to **terminal** (game over) states
- **Depth limiting** is a common technique to apply minimax to larger games
- Still evaluate terminal states as $+1$ / 0 / -1
- For nonterminal states at depth d, apply a heuristic evaluation instead of searching deeper

Depth limiting

- Standard minimax needs to search all the way to **terminal** (game over) states
- **Depth limiting** is a common technique to apply minimax to larger games
- Still evaluate terminal states as $+1$ / 0 / -1
- For nonterminal states at depth d, apply a heuristic evaluation instead of searching deeper
- Evaluation is a number between -1 and $+1$, estimating the probable outcome of the game

1-ply search

1-ply search

- Case $d = 1$

1-ply search

- Case $d = 1$
- For each move, evaluate the state resulting from playing that move

1-ply search

- Case $d = 1$
- For each move, evaluate the state resulting from playing that move
- This is computationally fast

1-ply search

- Case $d = 1$
- For each move, evaluate the state resulting from playing that move
- This is computationally fast
- Often easier to design a "which state is better" heuristic than to directly design a "which move to play" heuristic

1-ply search

- Case $d = 1$
- For each move, evaluate the state resulting from playing that move
- This is computationally fast
- Often easier to design a "which state is better" heuristic than to directly design a "which move to play" heuristic
- This is essentially a **utility-based AI**

Move ordering

Move ordering

- Minimax can **stop early** if it sees a value of $+1$ for maximising player or -1 for minimising player

Move ordering

- Minimax can **stop early** if it sees a value of $+1$ for maximising player or -1 for minimising player
- Modifications to minimax algorithm (e.g. **alpha-beta pruning**) lead to more of this

Move ordering

- Minimax can **stop early** if it sees a value of $+1$ for maximising player or -1 for minimising player
- Modifications to minimax algorithm (e.g. **alpha-beta pruning**) lead to more of this
- Thus ordering moves from **best to worst** means faster search

Move ordering

- Minimax can **stop early** if it sees a value of $+1$ for maximising player or -1 for minimising player
- Modifications to minimax algorithm (e.g. **alpha-beta pruning**) lead to more of this
- Thus ordering moves from **best to worst** means faster search
- How do we know which moves are "best" and "worst"? Use a heuristic!

Designing heuristics

Designing heuristics

- The **playing strength** of depth limited minimax depends heavily on the design of the **heuristic**

Designing heuristics

- The **playing strength** of depth limited minimax depends heavily on the design of the **heuristic**
- Good heuristic design requires **in-depth knowledge** of the tactics and strategy of the game

Designing heuristics

- The **playing strength** of depth limited minimax depends heavily on the design of the **heuristic**
- Good heuristic design requires **in-depth knowledge** of the tactics and strategy of the game
- What if we don't possess such knowledge?

FALMOUTH
UNIVERSITY

Monte Carlo evaluation

Monte Carlo methods

Monte Carlo methods

- In computing, a **Monte Carlo method** is an algorithm based on **averaging over random samples**

Monte Carlo methods

- In computing, a **Monte Carlo method** is an algorithm based on **averaging over random samples**
- The **average** over a large number of samples is a good approximation of the **expected value**

Monte Carlo methods

- In computing, a **Monte Carlo method** is an algorithm based on **averaging over random samples**
- The **average** over a large number of samples is a good approximation of the **expected value**
- Used for **quickly approximating** quantities over **large domains**

Monte Carlo methods

- In computing, a **Monte Carlo method** is an algorithm based on **averaging over random samples**
- The **average** over a large number of samples is a good approximation of the **expected value**
- Used for **quickly approximating** quantities over **large domains**
- Generally designed to **converge in the limit**

Monte Carlo methods

- In computing, a **Monte Carlo method** is an algorithm based on **averaging over random samples**
- The **average** over a large number of samples is a good approximation of the **expected value**
- Used for **quickly approximating** quantities over **large domains**
- Generally designed to **converge in the limit**
  - An **infinite** number of samples would give an **exact** answer

Monte Carlo methods

- In computing, a **Monte Carlo method** is an algorithm based on **averaging over random samples**
- The **average** over a large number of samples is a good approximation of the **expected value**
- Used for **quickly approximating** quantities over **large domains**
- Generally designed to **converge in the limit**
  - An **infinite** number of samples would give an **exact** answer
  - As the **number of samples** increases, the **accuracy** of the answer improves

Monte Carlo methods

- In computing, a **Monte Carlo method** is an algorithm based on **averaging over random samples**
- The **average** over a large number of samples is a good approximation of the **expected value**
- Used for **quickly approximating** quantities over **large domains**
- Generally designed to **converge in the limit**
  - An **infinite** number of samples would give an **exact** answer
  - As the **number of samples** increases, the **accuracy** of the answer improves
- Applications in physics, engineering, finance, weather forecasting, graphics, ...

Aside: “randomness” in computing

Aside: "randomness" in computing

- Digital computers are **deterministic**, so there's no such thing as true randomness

Aside: "randomness" in computing

- Digital computers are **deterministic**, so there's no such thing as true randomness
  - Cryptographically secure systems use an external source of randomness e.g. atmospheric noise, radioactive decay

Aside: "randomness" in computing

- Digital computers are **deterministic**, so there's no such thing as true randomness
  - Cryptographically secure systems use an external source of randomness e.g. atmospheric noise, radioactive decay
- What we actually have are **pseudo-random number generators (PRNGs)**

Aside: "randomness" in computing

- Digital computers are **deterministic**, so there's no such thing as true randomness
  - Cryptographically secure systems use an external source of randomness e.g. atmospheric noise, radioactive decay
- What we actually have are **pseudo-random number generators (PRNGs)**
- A PRNG is an algorithm which gives an **unpredictable** sequence of numbers based on a **seed**

Aside: "randomness" in computing

- Digital computers are **deterministic**, so there's no such thing as true randomness
  - Cryptographically secure systems use an external source of randomness e.g. atmospheric noise, radioactive decay
- What we actually have are **pseudo-random number generators (PRNGs)**
- A PRNG is an algorithm which gives an **unpredictable** sequence of numbers based on a **seed**
- Sequence is **uniformly distributed**, i.e. all numbers have equal probability

Aside: "randomness" in computing

- Digital computers are **deterministic**, so there's no such thing as true randomness
  - Cryptographically secure systems use an external source of randomness e.g. atmospheric noise, radioactive decay
- What we actually have are **pseudo-random number generators (PRNGs)**
- A PRNG is an algorithm which gives an **unpredictable** sequence of numbers based on a **seed**
- Sequence is **uniformly distributed**, i.e. all numbers have equal probability
- Seed is generally based on some source of **entropy**, e.g. system clock, mouse input, electronic noise

Monte Carlo evaluation in games

Monte Carlo evaluation in games

- Based on **random rollouts**

Monte Carlo evaluation in games

- Based on **random rollouts**

while *s* is not terminal **do**
    let *m* be a random legal move from *s*
    update *s* by playing *m*

Monte Carlo evaluation in games

- Based on **random rollouts**

while *s* is not terminal **do**
    let *m* be a random legal move from *s*
    update *s* by playing *m*

- The **value** of a rollout is the **value** of the terminal state it reaches (i.e. 1 for a win, −1 for a loss, 0 for a draw)

Monte Carlo evaluation in games

- Based on **random rollouts**

while *s* is not terminal **do**
    let *m* be a random legal move from *s*
    update *s* by playing *m*

- The **value** of a rollout is the **value** of the terminal state it reaches (i.e. 1 for a win, −1 for a loss, 0 for a draw)
- Averaging gives the **expected value** of the initial state

Monte Carlo evaluation in games

- Based on **random rollouts**

while *s* is not terminal **do**
    let *m* be a random legal move from *s*
    update *s* by playing *m*

- The **value** of a rollout is the **value** of the terminal state it reaches (i.e. 1 for a win, −1 for a loss, 0 for a draw)
- Averaging gives the **expected value** of the initial state
- Higher expected value = more chance of winning

Monte Carlo search

Monte Carlo search

- **Flat Monte Carlo search**: 1-ply search with Monte Carlo evaluation

Monte Carlo search

- **Flat Monte Carlo search**: 1-ply search with Monte Carlo evaluation
- How about minimax with $d > 1$ and Monte Carlo evaluation?

Monte Carlo search

- **Flat Monte Carlo search**: 1-ply search with Monte Carlo evaluation
- How about minimax with $d > 1$ and Monte Carlo evaluation?
  - Minimax assumes the evaluation is **deterministic**, but Monte Carlo is not

Monte Carlo search

- **Flat Monte Carlo search**: 1-ply search with Monte Carlo evaluation
- How about minimax with $d > 1$ and Monte Carlo evaluation?
  - Minimax assumes the evaluation is **deterministic**, but Monte Carlo is not
  - Not commonly used, mainly because there's something better...