Assignment Lecture 2: MiniMax Algorithm

Håkon Måløy

September 24, 2021

Adversarial Search

Setting with two or more agents with (often) conflicting goals
$\Rightarrow$ Games

Adversarial Search

Adversarial Search

Two players are included in the game.

- Two-Player

Adversarial Search

Players take turns making moves.

- Two-Player
- Turn-taking

Adversarial Search

- Two-Player
- Turn-taking
- Zero-sum games.

The score at the end is equal and opposite for the agents.

Adversarial Search

- Two-Player
- Turn-taking
- Zero-sum games.

The score at the end is equal and opposite for the agents.

- The total sum of the game is zero.

Adversarial Search

- Two-Player
- Turn-taking
- Zero-sum games.

The score at the end is equal and opposite for the agents.

- The total sum of the game is zero.
- If one player gains, the other player looooses.

Adversarial Search

- Two-Player
- Turn-taking
- Zero-sum games.
- Games with perfect information.

The game is fully observable for all players.

Adversarial Search

- Two-Player
- Turn-taking
- Zero-sum games.
- Games with perfect information.

The game is fully observable for all players.

- Cost and Utility functions.

Adversarial Search

- Two-Player
- Turn-taking
- Zero-sum games.
- Games with perfect information.

The game is fully observable for all players.

- Cost and Utility functions.
- Game history.

Adversarial Search

- Two-Player
- Turn-taking
- Zero-sum games.
- Games with perfect information.

The game is fully observable for all players.

- Cost and Utility functions.
- Game history.
- All moves are visible.

Adversarial Search

- Two-Player
- Turn-taking
- Zero-sum games.
- Games with perfect information.
- Deterministic games.

The outcome of making a move is the same every time.

Example games

- Chess

Example games

- Chess
- Tic Tac Toe

Example games

- Chess
- Tic Tac Toe
- Go

Search Trees in Games

- 2 Players: MAX, MIN.

Search Trees in Games

- 2 Players: MAX, MIN.
- Moves in turns, starting with MAX.

Search Trees in Games

- 2 Players: MAX, MIN.
- Moves in turns, starting with MAX.
- Each level is the opposite player as the one above and below it.

Search Trees in Games

- 2 Players: MAX, MIN.
- Moves in turns, starting with MAX.
- Each level is the opposite player as the one above and below it.
- Leaf nodes are terminal states where the game has finished.

Search Trees in Games

An optimal strategy leads to outcomes at least as good as any other strategy given an infallible opponent.

Search Trees in Games

An optimal strategy leads to outcomes at least as good as any other strategy given an infallible opponent.

Optimal Strategy

Given a search tree, the optimal strategy can be determined from the minimax value of each node.

Search Trees in Games

An optimal strategy leads to outcomes at least as good as any other strategy given an infallible opponent.

Optimal Strategy

Given a search tree, the optimal strategy can be determined from the minimax value of each node.

MiniMax value

The utility for MAX of being in the corresponding state (given optimal play from both players).

The MiniMax Value

The MiniMax Value

Given a choice, MAX will move towards the maximum value and MIN will move towards the minimum value.

The MiniMax Value

Given a choice, MAX will move towards the maximum value and MIN will move towards the minimum value.

$$Minimax(s) = \begin{cases} UTILITY(s) & \text{if TERMINAL-TEST}(s). \\ max_{a \in Actions(s)} MINIMAX(RESULT(s, a)) & \text{if PLAYER}(s) = \text{MAX}. \\ min_{a \in Actions(s)} MINIMAX(RESULT(s, a)) & \text{if PLAYER}(s) = \text{MIN}. \end{cases} \tag{1}$$

MiniMax Algorithm

The minimax algorithm computes the minimax decision from the current state.

MiniMax Algorithm

The minimax algorithm computes the minimax decision from the current state.

- Recursively computes the minimax value of succesor states.

MiniMax Algorithm

The minimax algorithm computes the minimax decision from the current state.

- Recursively computes the minimax value of succesor states.
- When a leaf node is reached, the minimax values are backed up through the tree.

MiniMax Algorithm

The minimax algorithm computes the minimax decision from the current state.

- Recursively computes the minimax value of succesor states.
- When a leaf node is reached, the minimax values are backed up through the tree.
- Complete depth-first exploration of the search tree.

MiniMax Algorithm

MiniMax Algorithm

```
function MINIMAX-SEARCH(game, state)
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move
```

MiniMax Algorithm

```
function MINIMAX-SEARCH(game, state)
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move
```

```
function MAX-VALUE(game, state)
    if game.TERMINAL-STATE(state) then
        return game.UTILITY(state, player), null
    v, move ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
        if v2 > v then
            v, move ← v2, a
    return v, move
```

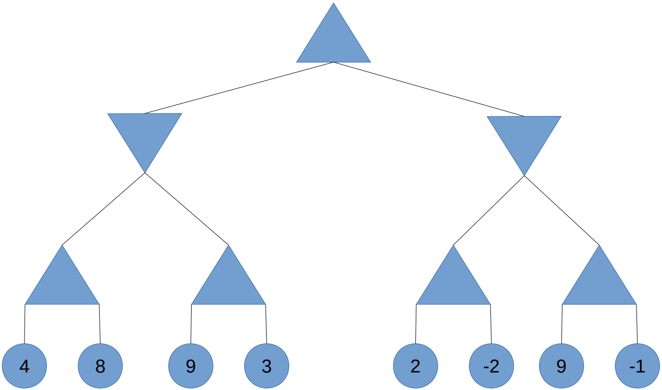
MiniMax Algorithm

```
function MINIMAX-SEARCH(game, state)
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state)
    return move
```

```
function MAX-VALUE(game, state)
    if game.TERMINAL-STATE(state) then
        return game.UTILITY(state, player), null
    v, move ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a))
        if v2 > v then
            v, move ← v2, a
    return v, move
```

```
function MIN-VALUE(game, state)
    if game.TERMINAL-STATE(state) then
        return game.UTILITY(state, player), null
    v, move ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a))
        if v2 < v then
            v, move ← v2, a
    return v, move
```

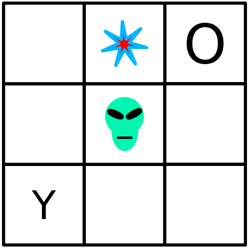
MiniMax Algorithm - Example



MiniMax Algorithm - Exam Question

When you arrive at star system G you take out your speech to look it over one last time, but when you do an unknown life form grabs your speech and runs off with it. You chase after the life form. Fortunately, one of the organizers saw what happened and helps you. Eventually, you are able to corner the life form in a room as shown in the Figure below. The organizer knows that this particular type of alien loves "bluuurgh", a spiky type of food. You now play the following turn-based game:

1. The life form moves either Left, Right, Up of Down.
2. You move either Up or Right
3. The organizer moves either Down or Left.
4. Everyone must make a move.



If the alien life form enters the square with the bluuurgh in it, it gets a reward of +10. If either you or the organizer enters the same square as the alien life form you catch it and get the speech, giving the alien life form a reward of -20. The game ends when you've either caught the alien life form, or everyone has made one move.

Construct a minimax tree of depth one for the situation described. The alien life form is the maximizer and you and the organizer are both minimizers.

MiniMax Problems

The number of nodes the minimax algorithm has to explore is exponential in the depth of the game tree ($O(b^m)$).

MiniMax Problems

The number of nodes the minimax algorithm has to explore is exponential in the depth of the game tree ($O(b^m)$).

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  $\Rightarrow$ exact solution completely infeasible

MiniMax Problems

The number of nodes the minimax algorithm has to explore is exponential in the depth of the game tree ($O(b^m)$).

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  $\Rightarrow$ exact solution completely infeasible
- But do we need to explore every path?

MiniMax Problems

The number of nodes the minimax algorithm has to explore is exponential in the depth of the game tree ($O(b^m)$).

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  $\Rightarrow$ exact solution completely infeasible
- But do we need to explore every path?

Trick

MiniMax Problems

The number of nodes the minimax algorithm has to explore is exponential in the depth of the game tree ($O(b^m)$).

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  $\Rightarrow$ exact solution completely infeasible
- But do we need to explore every path?

Trick

We can compute the correct minimax decision without looking at every node.

MiniMax Problems

The number of nodes the minimax algorithm has to explore is exponential in the depth of the game tree ($O(b^m)$).

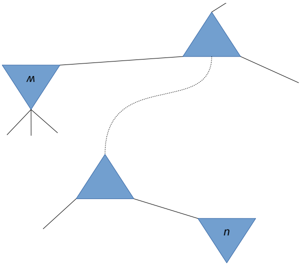
- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  $\Rightarrow$ exact solution completely infeasible
- But do we need to explore every path?

Trick

We can compute the correct minimax decision without looking at every node.

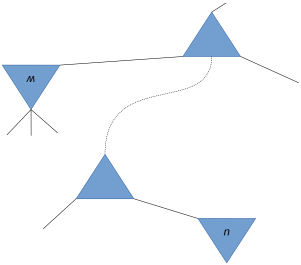
- Prune the tree to avoid inspecting nodes that cannot yield an improvement.

α-β Pruning



α-β Pruning

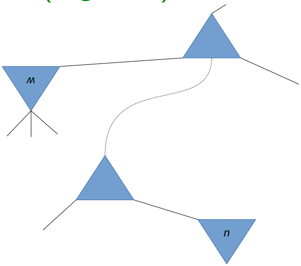
Consider a node *n* somewhere in the game tree such that Player can choose to move to that node.



α-β Pruning

Consider a node *n* somewhere in the game tree such that Player can choose to move to that node.
If Player has a better choice *m* at the parent node of *n* or further up the tree, then *n* will never be reached in play.

Example (Figure 1)



α-β Pruning

Alpha and Beta?

α-β Pruning

Alpha and Beta?

- $\alpha =$ The best value found so far for MAX

α-β Pruning

Alpha and Beta?

- α = The best value found so far for MAX
- β = The best value found so far for MIN

α-β Pruning

Alpha and Beta?

- α = The best value found so far for MAX
- β = The best value found so far for MIN

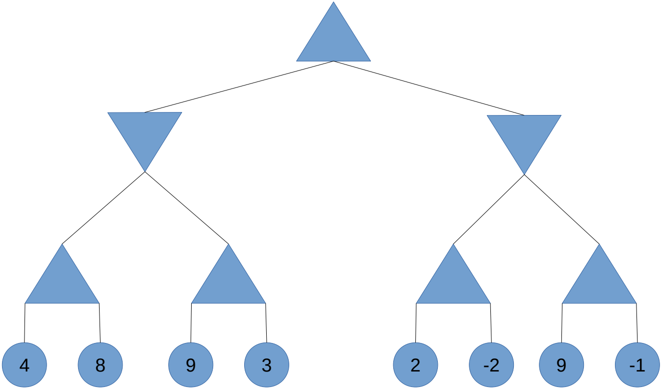
If the algorithm sees that it cannot achieve a better score by going down a branch, the branch is pruned.

α-β Pruning

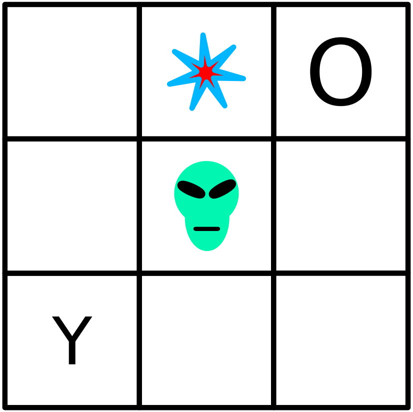
```
function ALPHA-BETA-SEARCH(game, state)
    player ← game.TO-MOVE(state)
    value, move ← MAX-VALUE(game, state, −∞, +∞)
    return move
function MAX-VALUE(game, state, α, β)
    if game.TERMINAL-STATE(state) then
        return game.UTILITY(state, player), null
    v, move ← −∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MIN-VALUE(game, game.RESULT(state, a), α, β)
        if v2 > v then
            v, move ← v2, a
            α ← MAX(α, v)
        if v ≥ β then return v, move
    return v, move
function MIN-VALUE(game, state, α, β)
    if game.TERMINAL-STATE(state) then
        return game.UTILITY(state, player), null
    v, move ← +∞
    for each a in game.ACTIONS(state) do
        v2, a2 ← MAX-VALUE(game, game.RESULT(state, a), α, β)
        if v2 < v then
            v, move ← v2, a
            β ← MIN(β, v)
        if v ≤ α then return v, move
    return v, move
=0
```

α-β Pruning - Example

Example

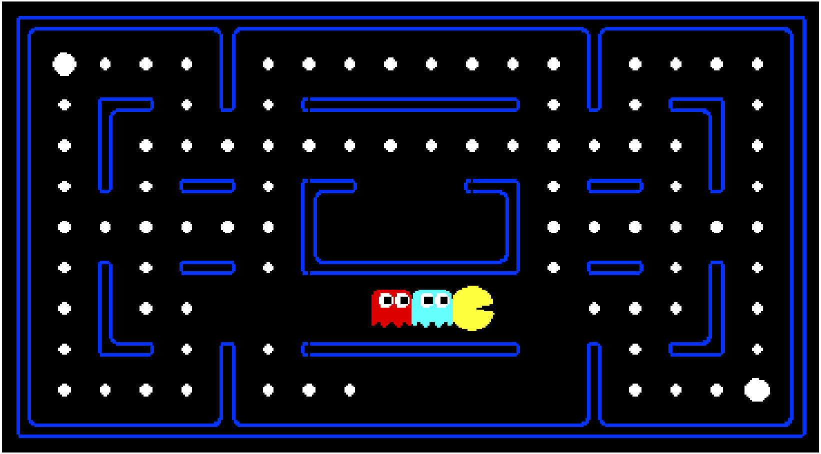


α-β Pruning - Exam Example



Assignment 3

You will be playing Pacman:



Assignment 3

You will follow the “Pac-Man Projects” from UC Berkley.
`http://ai.berkeley.edu/multiagent.html#Q2`

Assignment 3

You will follow the "Pac-Man Projects" from UC Berkley.
`http://ai.berkeley.edu/multiagent.html#Q2`

- The codebase provided includes the pre-coded rules for pacman

Assignment 3

You will follow the "Pac-Man Projects" from UC Berkley.
`http://ai.berkeley.edu/multiagent.html#Q2`

- The codebase provided includes the pre-coded rules for pacman
- It also includes graphics to visualize the game

Assignment 3

You will follow the "Pac-Man Projects" from UC Berkley.
`http://ai.berkeley.edu/multiagent.html#Q2`

- The codebase provided includes the pre-coded rules for pacman
- It also includes graphics to visualize the game
- It includes an autograde.py to verify your implementation

Assignment 3

You will follow the “Pac-Man Projects” from UC Berkley.
`http://ai.berkeley.edu/multiagent.html#Q2`

- The codebase provided includes the pre-coded rules for pacman
- It also includes graphics to visualize the game
- It includes an autograde.py to verify your implementation
- It is written in python 2.7

Assignment 3

Your tasks

You must answer questions 2 and 3 in the Pac-Man Projects.

Assignment 3

Your tasks

You must answer questions 2 and 3 in the Pac-Man Projects.

- You will edit the classes MiniMaxAgent and AlphaBetaAgent in multiAgents.py

Assignment 3

Your tasks

You must answer questions 2 and 3 in the Pac-Man Projects.

- You will edit the classes MiniMaxAgent and AlphaBetaAgent in multiAgents.py
- You will implement both MiniMax and Alpha-Beta Pruning

Assignment 3

Your tasks

You must answer questions 2 and 3 in the Pac-Man Projects.

- You will edit the classes MiniMaxAgent and AlphaBetaAgent in multiAgents.py
- You will implement both MiniMax and Alpha-Beta Pruning
- Test your code by running: python autograder.py -q q2 and python autograder.py -q q3

Assignment 3

Edit code

```python
class MinimaxAgent(MultiAgentSearchAgent):
    """
      Your minimax agent (question 2)
    """

    def getAction(self, gameState):
        """
          Returns the minimax action from the current gameState using self.depth
          and self.evaluationFunction.

          Here are some method calls that might be useful when implementing minimax.

          gameState.getLegalActions(agentIndex):
            Returns a list of legal actions for an agent
            agentIndex=0 means Pacman, ghosts are >= 1

          gameState.generateSuccessor(agentIndex, action):
            Returns the successor game state after an agent takes an action

          gameState.getNumAgents():
            Returns the total number of agents in the game
        """
        "*** YOUR CODE HERE ***"
        util.raiseNotDefined()

class AlphaBetaAgent(MultiAgentSearchAgent):
    """
      Your minimax agent with alpha-beta pruning (question 3)
    """

    def getAction(self, gameState):
        """
          Returns the minimax action using self.depth and self.evaluationFunction
        """
        "*** YOUR CODE HERE ***"
```

Assignment 3

Assignment 3

Read the Berkley provided text carefully:

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:
  - Berkley: One Ply $=$ One pacman move $+$ all the ghosts responses.

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:
  - Berkley: One Ply = One pacman move + all the ghosts responses.
  - Our book: One Ply = One move by one of the agents.

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:
  - Berkley: One Ply = One pacman move + all the ghosts responses.
  - Our book: One Ply = One move by one of the agents.
- Be diligent with who is doing the actions:

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:
  - Berkley: One Ply = One pacman move + all the ghosts responses.
  - Our book: One Ply = One move by one of the agents.
- Be diligent with who is doing the actions:
  - In MIN-VALUE is it the ghosts or Pacman doing the actions?

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:
  - Berkley: One Ply = One pacman move + all the ghosts responses.
  - Our book: One Ply = One move by one of the agents.
- Be diligent with who is doing the actions:
  - In MIN-VALUE is it the ghosts or Pacman doing the actions?
  - Similarly for MAX-VALUE.

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:
  - Berkley: One Ply = One pacman move + all the ghosts responses.
  - Our book: One Ply = One move by one of the agents.
- Be diligent with who is doing the actions:
  - In MIN-VALUE is it the ghosts or Pacman doing the actions?
  - Similarly for MAX-VALUE.
- Be sure to verify how the pruning step should be done (check the autograder.py):

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:
  - Berkley: One Ply = One pacman move + all the ghosts responses.
  - Our book: One Ply = One move by one of the agents.
- Be diligent with who is doing the actions:
  - In MIN-VALUE is it the ghosts or Pacman doing the actions?
  - Similarly for MAX-VALUE.
- Be sure to verify how the pruning step should be done (check the autograder.py):
  - The book: Prune when $v \leq \alpha$ or $v \geq \beta$.

Assignment 3

Read the Berkley provided text carefully:

- Berkley uses Ply differently than our book:
  - Berkley: One Ply = One pacman move + all the ghosts responses.
  - Our book: One Ply = One move by one of the agents.
- Be diligent with who is doing the actions:
  - In MIN-VALUE is it the ghosts or Pacman doing the actions?
  - Similarly for MAX-VALUE.
- Be sure to verify how the pruning step should be done (check the autograder.py):
  - The book: Prune when $v \leq \alpha$ or $v \geq \beta$.
  - autograder.py: Prune when $v < \alpha$ or $v > \beta$.

Assignment 3

Deliverables

Once you have achieved 5/5 points on both q2 and q3 you will deliver:

Assignment 3

Deliverables

Once you have achieved 5/5 points on both q2 and q3 you will deliver:

- Your edited multiAgents.py file

Assignment 3

Deliverables

Once you have achieved 5/5 points on both q2 and q3 you will deliver:

- Your edited multiAgents.py file
- Results: A .txt or .pdf file with your successful output from the autograder.

Assignment 3

Deliverables

Once you have achieved 5/5 points on both q2 and q3 you will deliver:

- Your edited multiAgents.py file
- Results: A .txt or .pdf file with your successful output from the autograder.
  - The file should contain the entire output from running both python autograder.py -q q2 and python autograder.py -q q3.

Assignment 3

Deliverables

Once you have achieved 5/5 points on both q2 and q3 you will deliver:

- Your edited multiAgents.py file
- Results: A .txt or .pdf file with your successful output from the autograder.
  - The file should contain the entire output from running both python autograder.py -q q2 and python autograder.py -q q3.

If you do not achieve 5/5 on q2 and q3 before the deadline, submit how far you've come with both code and results.

Assignment 3

Deliverables

Once you have achieved 5/5 points on both q2 and q3 you will deliver:

- Your edited multiAgents.py file
- Results: A .txt or .pdf file with your successful output from the autograder.
  - The file should contain the entire output from running both python autograder.py -q q2 and python autograder.py -q q3.

If you do not achieve 5/5 on q2 and q3 before the deadline, submit how far you've come with both code and results.

Delivery due: 15.10.2021.