Inteligencja obliczeniowa w analizie danych cyfrowych

dr inż. Mateusz Baran

Akademia Górniczo-Hutnicza

Lato 2024

Inteligencja obliczeniowa w analizie danych cyfrowych

dr inż. Mateusz Baran

Akademia Górniczo-Hutnicza

Lato 2024

Treści

- Wprowadzenie do sztucznej inteligencji.
- Strategie przeszukiwania, heurystyki.
- Reprezentacja wiedzy, wnioskowanie.
- Procesy decyzyjne i uczenie ze wzmocnieniem.
- Problemy optymalizacji.
- Teoria gier.

Treści
- Wprowadzenie do sztucznej inteligencji.
- Strategie przeszukiwania, heurystyki.
- Reprezentacja wiedzy, wnioskowanie.
- Procesy decyzyjne i uczenie ze wzmocnieniem.
- Problemy optymalizacji.
- Teoria gier.

1. Tematy do pewnego stopnia będą się przeplatać na wykładach.

Wprowadzenie

Polecane książki:

1. S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach. Edycja 3 lub 4.
2. R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, Massachusetts, 2018.
3. D. L. Poole and A. K. Mackworth, Artificial Intelligence: Foundations of Computational Agents, 3rd edition. Cambridge, United Kingdom; New York, 2023.

Wprowadzenie

Polecane książki:
S. Russell and P. Norvig, Artificial Intelligence: A Modern Approach. Edycja 3 lub 4.
R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, Massachusetts, 2018.
D. L. Poole and A. K. Mackworth, Artificial Intelligence: Foundations of Computational Agents, 3rd edition. Cambridge, United Kingdom; New York, 2023.

1. Edycja 3 Russella i Norviga nie opisuje nowoczesnych technik uczenia ze wzmocnieniem, ale reszta treści jest tam bardzo dobrze opisana.

Zasady zaliczeń:

- Ocena końcowa jest równa ocenie z ćwiczeń laboratoryjnych.
- Oceny z ćwiczeń będą wystawiane na podstawie mini-projektów realizowanych w grupach do trzech osób.
- Zadanych zostanie około 6 mini-projektów.
- Do zaliczenia ćwiczeń konieczne jest zaliczenie wszystkich mini-projektów. Ocena z ćwiczeń jest wyliczana na podstawie średniej z punktów z mini-projektów.
- Ocena z mini-projektu będzie wystawiana między innymi w oparciu o:
  - Czy program realizuje funkcjonalność opisaną w zadaniu.
  - Poprawny opis rozwiązania w raporcie.
  - Umiejętność wyjaśnienia (ustnie) działania programu.
- Jeśli zadany mini-projekt nie zostanie przesłany w zadanym terminie konieczne będzie rozwiązanie zadania poprawkowego które może mieć inną treść i zasady oceny niż oryginalne.
- Dalsze szczegóły będą udzielane przez osobę prowadzącą konkretne zajęcia (dr inż. Tomasz Nabagło, mgr inż. Fabian Bogusz).

Zasady zaliczeń:
- Ocena końcowa jest równa ocenie z ćwiczeń laboratoryjnych.
- Oceny z ćwiczeń będą wystawiane na podstawie mini-projektów realizowanych w grupach do trzech osób.
- Zadanych zostanie około 6 mini-projektów.
- Do zaliczenia ćwiczeń konieczne jest zaliczenie wszystkich mini-projektów. Ocena z ćwiczeń jest wyliczana na podstawie średniej z punktów z mini-projektów.
- Ocena z mini-projektu będzie wystawiana między innymi w oparciu o:
  - Czy program realizuje funkcjonalność opisaną w zadaniu.
  - Poprawny opis rozwiązania w raporcie.
  - Umiejętność wyjaśnienia (ustnie) działania programu.
- Jeśli zadany mini-projekt nie zostanie przesłany w zadanym terminie konieczne będzie rozwiązanie zadania poprawkowego które może mieć inną treść i zasady oceny niż oryginalne.
- Dalsze szczegóły będą udzielane przez osobę prowadzącą konkretne zajęcia (dr inż. Tomasz Nabagło, mgr inż. Fabian Bogusz).

Zastosowania

Sztuczna inteligencja ma wiele zastosowań, między innymi:

- sterowanie robotami,
- rozpoznawanie mowy i obrazów,
- planowanie i harmonogramowanie,
- gry,
- detekcja spamu,
- przetwarzanie języka naturalnego,
- diagnostyka wspomagana komputerowo.

Zastosowania

Sztuczna inteligencja ma wiele zastosowań, między innymi:
- sterowanie robotami,
- rozpoznawanie mowy i obrazów,
- planowanie i harmonogramowanie,
- gry,
- detekcja spamu,
- przetwarzanie języka naturalnego,
- diagnostyka wspomagana komputerowo.

1. Dziedzina dynamicznie rozwija się dzięki rosnącym zasobom obliczeniowym i zbiorom danych na podstawie których można tworzyć modele.

Części składowe problemów AI:

- Kryterium oceny (*performance measure*): jakie wyniki są pożądane?
- Środowisko (*environment*): z czym AI wchodzi w interakcje?
- Czujniki (*sensors*): jak AI pozyskuje informacje?
- Elementy wykonawcze (*actuators*): jak AI wchodzi w interakcje ze środowiskiem?

Na agenta składają się architektura (czujniki i elementy wykonawcze) oraz zarządzający nimi program.

Części składowe problemów AI:
- Kryterium oceny (*performance measure*): jakie wyniki są pożądane?
- Środowisko (*environment*): z czym AI wchodzi w interakcje?
- Czujniki (*sensors*): jak AI pozyskuje informacje?
- Elementy wykonawcze (*actuators*): jak AI wchodzi w interakcje ze środowiskiem?

Na agenta składają się architektura (czujniki i elementy wykonawcze) oraz zarządzający nimi program.

1. Agent wchodzi w interakcje ze środowiskiem. Wybiera akcje które wykona na podstawie kryterium oceny i odbieranych bodźców.
2. Wiele agentów działa w wirtualnych środowiskach(nie mają fizycznych sensorów czy elementów wykonawczych). Czasami nazywane są agentami programowymi (*software agents*).
3. W niektórych dziedzinach często ciężko jest osiągnąć wyniki lepsze niż wprawny człowiek (warto tu odnotować niedawny sukces AlphaGO).
4. Wymienine elementy problemu AI muszą zostać określone przed przystąpieniem do pracy nad algorytmem.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Figure 2.5 Examples of agent types and their PEAS descriptions.

Rysunek: Źródło: Russell and Norvig 2009

Agent Type | Performance Measure | Environment | Actuators | Sensors
Medical diagnosis system | Healthy patient, reduced costs | Patient, hospital, staff | Display of questions, tests, diagnoses, treatments, referrals | Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system | Correct image categorization | Downlink from orbiting satellite | Display of scene categorization | Color pixel arrays
Part-picking robot | Percentage of parts in correct bins | Conveyor belt with parts; bins | Jointed arm and hand | Camera, joint angle sensors
Refinery controller | Purity, yield, safety | Refinery, operators | Valves, pumps, heaters, displays | Temperature, pressure, chemical sensors
Interactive English tutor | Student's score on test | Set of students, testing agency | Display of exercises, suggestions, corrections | Keyboard entry
Figure 2.5 Examples of agent types and their PEAS descriptions.
Rysunek: Źródło: Russell and Norvig 2009

1. Porównanie kryteriów oceny, rodzajów środowisk, elementów wykonawczych i sensorów w różnych zastosowaniach.

W ogólności mamy jeszcze osobę definiującą problem AI która może nie być pewna jak wykonać swoje zadanie i próbuje zdefiniować problem na podstawie posiadanych informacji (np. korzystając z odwrotnego uczenia ze wzmocnieniem lub inżynierii kryterium oceny).

W ogólności mamy jeszcze osobę definiującą problem AI która może nie być pewna jak wykonać swoje zadanie i próbuje zdefiniować problem na podstawie posiadanych informacji (np. korzystając z odwrotnego uczenia ze wzmocnieniem lub inżynierii kryterium oceny).

As a general rule, it is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave.

Even when the obvious pitfalls are avoided, some knotty problems remain. For example, the notion of "clean floor" in the preceding paragraph is based on average cleanliness over time. Yet the same average cleanliness can be achieved by two different agents, one of which does a mediocre job all the time while the other cleans energetically but takes long breaks. Which is preferable might seem to be a fine point of janitorial science, but in fact it is a deep philosophical question with far-reaching implications. Which is better—a reckless life of highs and lows, or a safe but humdrum existence? Which is better—an economy where everyone lives in moderate poverty, or one in which some live in plenty while others are very poor? We leave these questions as an exercise for the diligent reader.

Źródło: Russell and Norvig 2020

As a general rule, it is better to design performance measures according to what one actually wants to be achieved in the environment, rather than according to how one thinks the agent should behave.

Even when the obvious pitfalls are avoided, some knotty problems remain. For example, the notion of "clean floor" in the preceding paragraph is based on average cleanliness over time. Yet the same average cleanliness can be achieved by two different agents, one of which does a mediocre job all the time while the other cleans energetically but takes long breaks. Which is preferable might seem to be a fine point of janitorial science, but in fact it is a deep philosophical question with far-reaching implications. Which is better—a reckless life of highs and lows, or a safe but humdrum existence? Which is better—an economy where everyone lives in moderate poverty, or one in which some live in plenty while others are very poor? We leave these questions as an exercise for the diligent reader.

Źródło: Russell and Norvig 2020

For each possible percept sequence, a ***rational agent*** *should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

Źródło: Russell and Norvig 2020

Problem-solving agents *use* ***atomic representations*** *(...) — that is, states of the world are considered as wholes, with no internal structure visible to the problem-solving algorithms. Goal-based agents that use more advanced* ***factored*** *or* ***structured*** *representations are usually called* ***planning agents*** *(...).*

(...) ***Uninformed search algorithms*** *— algorithms that are given no information about the problem other than its definition. Although some of these algorithms can solve any solvable problem, none of them can do so efficiently.* ***Informed search algorithms****, on the other hand, can do quite well given some guidance on where to look for solutions.*

Źródło: Russell and Norvig 2009

For each possible percept sequence, a ***rational agent*** *should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has.*

Źródło: Russell and Norvig 2020

Problem-solving agents *use* ***atomic representations*** *(...) — that is, states of the world are considered as wholes, with no internal structure visible to the problem-solving algorithms. Goal-based agents that use more advanced* ***factored*** *or* ***structured*** *representations are usually called* ***planning agents*** *(...).*

(...) ***Uninformed search algorithms*** *— algorithms that are given no information about the problem other than its definition. Although some of these algorithms can solve any solvable problem, none of them can do so efficiently.* ***Informed search algorithms****, on the other hand, can do quite well given some guidance on where to look for solutions.*

Źródło: Russell and Norvig 2009

1. Factored representations use vectors of attribute values (for example Booleans, numbers, elements of discrete sets). Structured representations include objects with relationships and (optionally) their own internal structure.

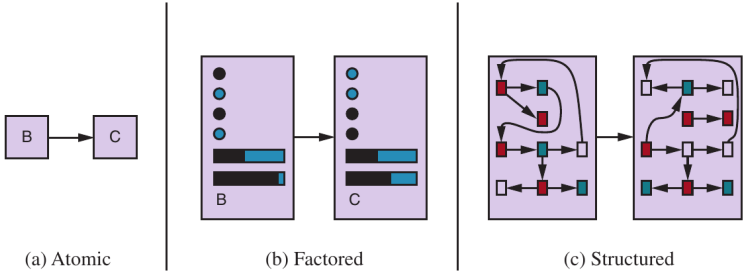
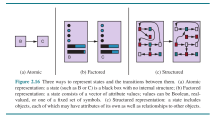


Figure 2.16 Three ways to represent states and the transitions between them. (a) Atomic representation: a state (such as B or C) is a black box with no internal structure; (b) Factored representation: a state consists of a vector of attribute values; values can be Boolean, real-valued, or one of a fixed set of symbols. (c) Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.

Rysunek: Źródło: Russell and Norvig 2020



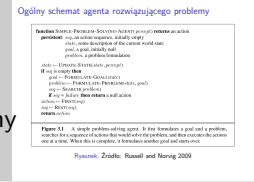
Ogólny schemat agenta rozwiązującego problemy

function SIMPLE-PROBLEM-SOLVING-AGENT(*percept*) **returns** an action
  persistent: *seq*, an action sequence, initially empty
    state, some description of the current world state
    goal, a goal, initially null
    problem, a problem formulation

  state ← UPDATE-STATE(*state*, *percept*)
  if *seq* is empty **then**
    goal ← FORMULATE-GOAL(*state*)
    problem ← FORMULATE-PROBLEM(*state*, *goal*)
    seq ← SEARCH(*problem*)
    if *seq* = *failure* **then return** a null action
  action ← FIRST(*seq*)
  seq ← REST(*seq*)
  return *action*

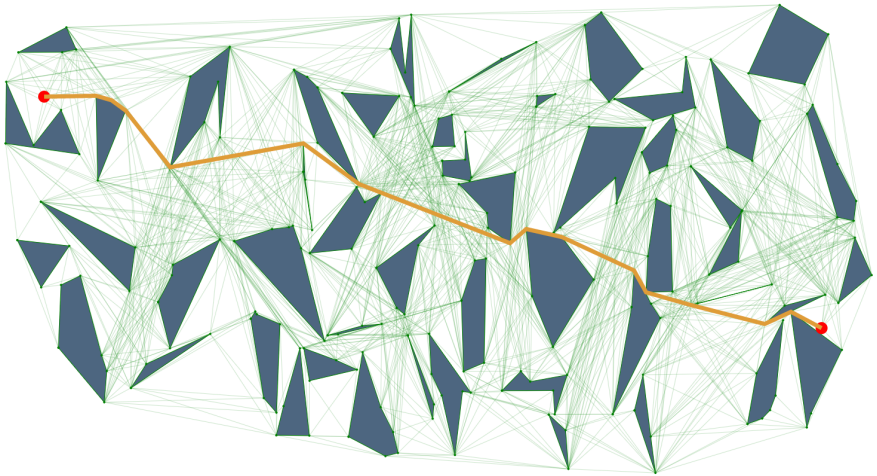
Figure 3.1 A simple problem-solving agent. It first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.

Rysunek: Źródło: Russell and Norvig 2009

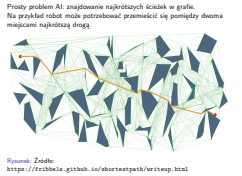


1. Here, generic agent program is presented. It returns a single action based on a percept.
2. It maintains an internal state of the agent (*state*), updated with each percept.
3. It repeatedly formulates a goal and a problem. Then a sequence of actions that solve the problem is searched and performed.

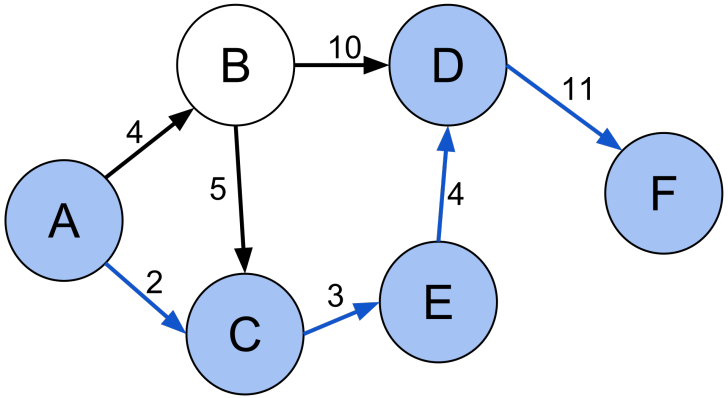
Prosty problem AI: znajdowanie najkrótszych ścieżek w grafie.
Na przykład robot może potrzebować przemieścić się pomiędzy dwoma miejscami najkrótszą drogą.



Rysunek: Źródło:
https://fribbels.github.io/shortestpath/writeup.html

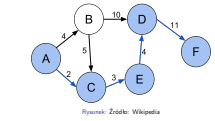


1. A sample problem of robot navigation on a planar surface with obstacles. Only shortest straight paths between corners are considered to minimize the distance. The orange line is the final shortest path.



Rysunek: Źródło: Wikipedia

Jak znaleźć najkrótszą ścieżkę pomiędzy węzłami A i F?



1. In this case the nodes may represent cities or parts of a room (some parts are blocked by obstacles like on the previous slide). Numbers on edges represent the distance (or time needed to travel between two given nodes).

Części składowe tego problemu (z punktu widzenia AI):

- **Stan początkowy**: agent znajduje się w węźle A.
- Możliwe **działania** (akcje): przemieszczenie się do innego węzła połączonego krawędzią. Na przykład będąc w węźle A możliwe działania to GO(B) i GO(C).
- **Rozwiązanie**: ciąg akcji prowadzących do stanu docelowego.
- **Zbiór stanów docelowych** (opisuje w jakich stanach aktor ma się znaleźć po wykonaniu ciągu akcji).
- **Koszt akcji**: reprezentowany wagami krawędzi.

Trzy pierwsze elementy określają **przestrzeń stanów** (*state space*).
Rozwiązanie: ciąg akcji prowadzących do stanu docelowego.
Rozwiązanie optymalne: rozwiązanie o najmniejszym koszcie liczonym jako suma kosztów poszczególnych kroków na ścieżce.

Części składowe tego problemu (z punktu widzenia AI):
- **Stan początkowy**: agent znajduje się w węźle A.
- Możliwe **działania** (akcje): przemieszczenie się do innego węzła połączonego krawędzią. Na przykład będąc w węźle A możliwe działania to GO(B) i GO(C).
- **Rozwiązanie**: ciąg akcji prowadzących do stanu docelowego.
- **Zbiór stanów docelowych** (opisuje w jakich stanach aktor ma się znaleźć po wykonaniu ciągu akcji).
- **Koszt akcji**: reprezentowany wagami krawędzi.

Trzy pierwsze elementy określają **przestrzeń stanów** (*state space*).
Rozwiązanie: ciąg akcji prowadzących do stanu docelowego.
Rozwiązanie optymalne: rozwiązanie o najmniejszym koszcie liczonym jako suma kosztów poszczególnych kroków na ścieżce.

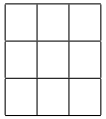
1. The state space is the set of all states reachable from the initial state.
2. Note: we assume that all step costs are nonnegative.

- Określamy problem: jeśli to możliwe, to znajdź sekwencję **akcji** prowadzących do węzła docelowego najniższym możliwym kosztem.
- Następnie poszukujemy **rozwiązania** naszego problemu (może być ich wiele).
- Poprawne rozwiązanie może nie istnieć, co jest określane jako *porażka*. W takim przypadku wykonywana jest akcja zerowa.
- Ostatecznie wykonujemy znalezioną sekwencję akcji. W problemie najkrótszej ścieżki możliwe jest określenie całego ciągu akcji na samym początku (po otrzymaniu początkowego bodźca).

- Określamy problem: jeśli to możliwe, to znajdź sekwencję **akcji** prowadzących do węzła docelowego najniższym możliwym kosztem.
- Następnie poszukujemy **rozwiązania** naszego problemu (może być ich wiele).
- Poprawne rozwiązanie może nie istnieć, co jest określane jako *porażka*. W takim przypadku wykonywana jest akcja zerowa.
- Ostatecznie wykonujemy znalezioną sekwencję akcji. W problemie najkrótszej ścieżki możliwe jest określenie całego ciągu akcji na samym początku (po otrzymaniu początkowego bodźca).

1. The failure in this problem may only happen when the goal node is not reachable from the initial node.

Drugi przykład: kółko i krzyżyk.
Załóżmy, że wykonujemy pierwszy ruch na planszy:



- Plansza reprezentuje **środowisko**. Stan jest opisywany zajętością poszczególnych pól planszy.
- Bodziec początkowy sygnalizuje rozpoczęcie gry.
- Początkowo ciąg akcji do podjęcia jest pusty.
- Naszym **celem** jest postawienie trzech znaków X w linii zanim przeciwnik postawi w linii trzy znaki O.

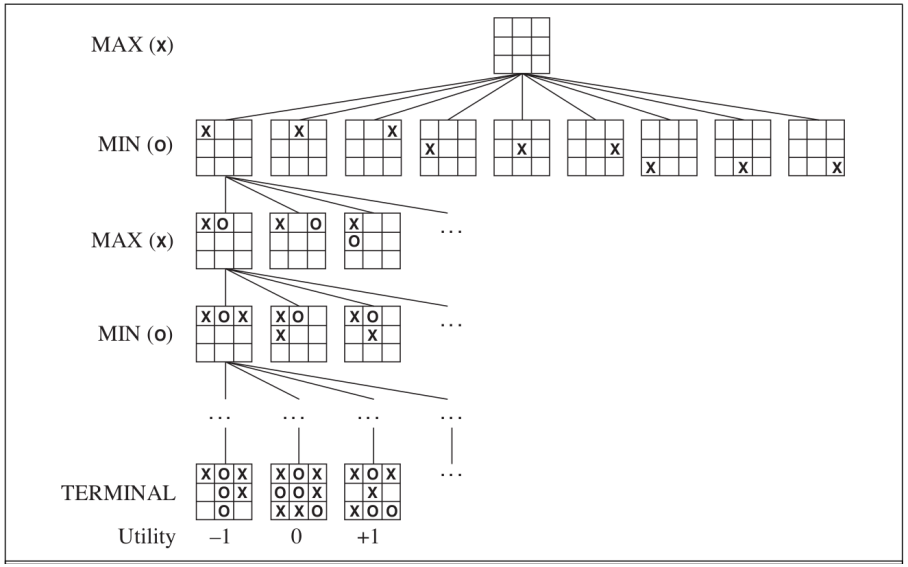
Drugi przykład: kółko i krzyżyk.
Załóżmy, że wykonujemy pierwszy ruch na planszy:

- Plansza reprezentuje **środowisko**. Stan jest opisywany zajętością poszczególnych pól planszy.
- Bodziec początkowy sygnalizuje rozpoczęcie gry.
- Początkowo ciąg akcji do podjęcia jest pusty.
- Naszym **celem** jest postawienie trzech znaków X w linii zanim przeciwnik postawi w linii trzy znaki O.

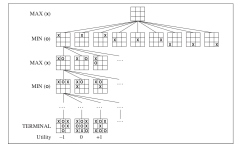
- Określamy nasz problem:
  - Jeśli to możliwe, znajdź ruch (**akcję**) która gwarantuje utworzenie linii trzech znaków X niezależnie od ruchów przeciwnika.
  - Jeśli to nie jest możliwe, to znajdź ruch który uniemożliwia przeciwnikowi utworzenie linii złożonej z trzech znaków O niezależnie od ruchów jakie wykona.
  - Powyższe punkty można też zawrzeć określając **funkcję użyteczności** dla poszczególnych stanów.
- Następnie szukamy rozwiązania (może być wiele poprawnych rozwiązań, ale interesuje nas znalezienie jednego).
- Może się zdarzyć, że nie ma poprawnego rozwiązania, co określamy jako *porażkę*. W takim przypadku podejmujemy akcję zerową (*null action*).
- Ostatecznie wykonujemy akcję która została znaleziona. W grze kółko i krzyżyk obliczamy nową akcję po każdym bodźcu (odpowiadającym ruchowi przeciwnika), więc `Rest(seq)` jest zawsze puste.

- Określamy nasz problem:
  - Jeśli to możliwe, znajdź ruch (**akcję**) która gwarantuje utworzenie linii trzech znaków X niezależnie od ruchów przeciwnika.
  - Jeśli to nie jest możliwe, to znajdź ruch który uniemożliwia przeciwnikowi utworzenie linii złożonej z trzech znaków O niezależnie od ruchów jakie wykona.
  - Powyższe punkty można też zawrzeć określając **funkcję użyteczności** dla poszczególnych stanów.
- Następnie szukamy rozwiązania (może być wiele poprawnych rozwiązań, ale interesuje nas znalezienie jednego).
- Może się zdarzyć, że nie ma poprawnego rozwiązania, co określamy jako *porażkę*. W takim przypadku podejmujemy akcję zerową (*null action*).
- Ostatecznie wykonujemy akcję która została znaleziona. W grze kółko i krzyżyk obliczamy nową akcję po każdym bodźcu (odpowiadającym ruchowi przeciwnika), więc `Rest(seq)` jest zawsze puste.

1. W tym problemie nie mamy wprost zdefiniowanego kosztu pojedynczego kroku, a jedynie użyteczność całej ścieżki od korzenia do węzła końcowego (+1 dla stanu wygrywającego, -1 dla przegrywającego i 0 dla remisowego).
2. Co istotne, w przeciwieństwie do poprzedniego problemu, pojedynczy agent nie kontroluje całego procesu zmiany stanu. Akcje przeciwnika są wykonywane niezależnie.



Rysunek: Źródło: Russell and Norvig 2009



Rysunek: Źródło: Russell and Norvig 2009

1. This diagram illustrates a search procedure for the tic-tac-toe game. All possible moves are simulated
2. The **search tree** contains the initial state in the root. Children of each node correspond to states achievable from the state corresponding to said node. Each edge corresponds to a possible action.
3. Terminal states are leafs of the tree where no further action is possible (the game ends).
4. The MAX and MIN labels correspond to the minimax strategy for zero-sum two player games (it will be discussed later).
5. At the bottom the utility function of three example terminal states is specified, one for each possible outcome (win, loss, draw).

Przykład definicji problemu w EasyAI

```python
from easyAI import TwoPlayerGame, Human_Player, AI_Player, Negamax

class Board:
    def __init__(self):
        # start with en empty board
        self.state = [[' ', ' ', ' '], [' ', ' ', ' '], [' ', ' ', ' ']]

    def show(self):
        print("Current board: ")
        for i in range(3):
            print(self.state[i])
```

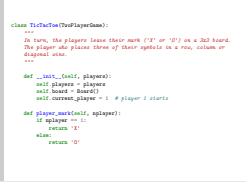

Przykład definicji problemu w EasyAI

1. The first line imports the base class for two-player games (TwoPlayerGame), a player controlled by a human (Human_Player) and an AI player (AI_Player).
2. The class Board represents the game board. Its initial state is represented by a list of lists. Successive lists represent rows. Each element of an inner list corresponds to a cell on the board. An empty cell is represented by a string ' ', while X and O marks will be represented by, respectively, string 'X' and 'O'.

```python
class TicTacToe(TwoPlayerGame):
    """
    In turn, the players leave their mark ('X' or 'O') on a 3x3 board.
    The player who places three of their symbols in a row, column or
    diagonal wins.
    """

    def __init__(self, players):
        self.players = players
        self.board = Board()
        self.current_player = 1  # player 1 starts

    def player_mark(self, nplayer):
        if nplayer == 1:
            return 'X'
        else:
            return 'O'
```

1. Next, there is the class TicTacToe that represents the game of tic-tac-toe. Its constructor saves the players that will play, sets the board to an empty board and marks that player number 1 will start the game.
2. The method player_mark tells whether a player (numbered either 1 or 2, the convention of the EasyAI library) plays with X marks or O marks.
3. Further methods are on the next slide.

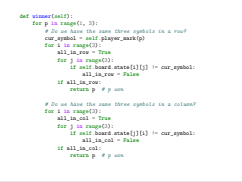
```python
def possible_moves(self):
    moves = []
    for i in range(3):
        for j in range(3):
            if self.board.state[i][j] == ' ':
                moves.append((i, j))
    # print(self.board, moves)
    return moves

def make_move(self, move):
    self.board.state[move[0]][move[1]] = self.player_mark(
                    self.current_player)
```


1. The next method, possible_moves, tells what moves are possible in the current state of the game. It returns a list of possible actions represented by coordinates of the board cell that will be marked. The mark itself is determined from the knowledge which player performs an action.
2. Making a move corresponds to marking a specific cell indicated by the argument move with the mark of the current player.

```python
def winner(self):
    for p in range(1, 3):
        # Do we have the same three symbols in a row?
        cur_symbol = self.player_mark(p)
        for i in range(3):
            all_in_row = True
            for j in range(3):
                if self.board.state[i][j] != cur_symbol:
                    all_in_row = False
            if all_in_row:
                return p  # p won

        # Do we have the same three symbols in a column?
        for i in range(3):
            all_in_col = True
            for j in range(3):
                if self.board.state[j][i] != cur_symbol:
                    all_in_col = False
            if all_in_col:
                return p  # p won
```

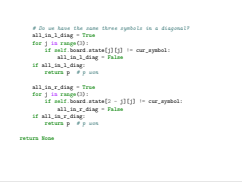



1. The function winner (spanning this one and the next slide) determines wheter the current state of the board indicates win or loss by one of the players. The outer loop ranges over player indices $p \in \{1, 2\}$. Next, cur_symbol is the symbol of player p.
2. The two pair of loops check wheter there is, respectively, a row or coulumn of three symbols cur_symbol. If it's found, p is declared the winner.

```
    # Do we have the same three symbols in a diagonal?
    all_in_l_diag = True
    for j in range(3):
        if self.board.state[j][j] != cur_symbol:
            all_in_l_diag = False
    if all_in_l_diag:
        return p  # p won

    all_in_r_diag = True
    for j in range(3):
        if self.board.state[2 - j][j] != cur_symbol:
            all_in_r_diag = False
    if all_in_r_diag:
        return p  # p won

return None
```

1. The next two loops check for a diagonal formed by three symbols cur_symbol. Again, when it is found, player p i declared the winner.
2. Finally, if neither player formed a line of three symbols, None is returned indicate this situation.

```python
def is_over(self):
    return (self.possible_moves() == []) or (self.winner() != None)

def show(self):
    self.board.show()

def scoring(self):
    won = self.winner()
    if won == self.current_player:
        return 100
    elif won == 3 - self.current_player:
        return -100
    else:
        return 0
```


1. The game is over when no further moves are possible or there is a winner.
2. The game is displayed as the state of the board.
3. The winner gets 1 point, the loser get -1 points and in case of a draw both players receive 0 points. The scoring function returns the score of the current player indicated by self.current_player. The other player has number self.opponent_index.

```python
# Start a match (and store the history of moves when it ends)
ai = Negamax(13) # The AI will think 13 moves in advance
game = TicTacToe( [ Human_Player(), AI_Player(ai) ] )
history = game.play()
```


# Start a match (and store the history of moves when it ends)
ai = Negamax(13) # The AI will think 13 moves in advance
game = TicTacToe([Human_Player(), AI_Player(ai)])
history = game.play()

1. In this code, the Negamax algorithm is used to play the game against a human player. You can enter moves by entreing in the console coordinates of the action, for example "(1, 2)".

Ćwiczenie: zagraj w grę przeciwko wybranemu AI. Zmodyfikuj grę tak, aby odbywała się na planszy rozmiaru 4x3. Czy drugi gracz może w takiej modyfikacji zawsze doprowadzić do remisu?

Ćwiczenie: zagraj w grę przeciwko wybranemu AI. Zmodyfikuj grę tak, aby odbywała się na planszy rozmiaru 4x3. Czy drugi gracz może w takiej modyfikacji zawsze doprowadzić do remisu?