Zastosowania sztucznej inteligencji

Tworzenie botów do gier

Twoim zadaniem jest stworzenie inteligentnego bota, który może grać w określoną grę komputerową i rywalizować z ludzkimi graczami lub innymi botami. Bot ma wykazywać strategię, umiejętności i reakcje charakterystyczne dla gracza.

Kroki do wykonania zadania:

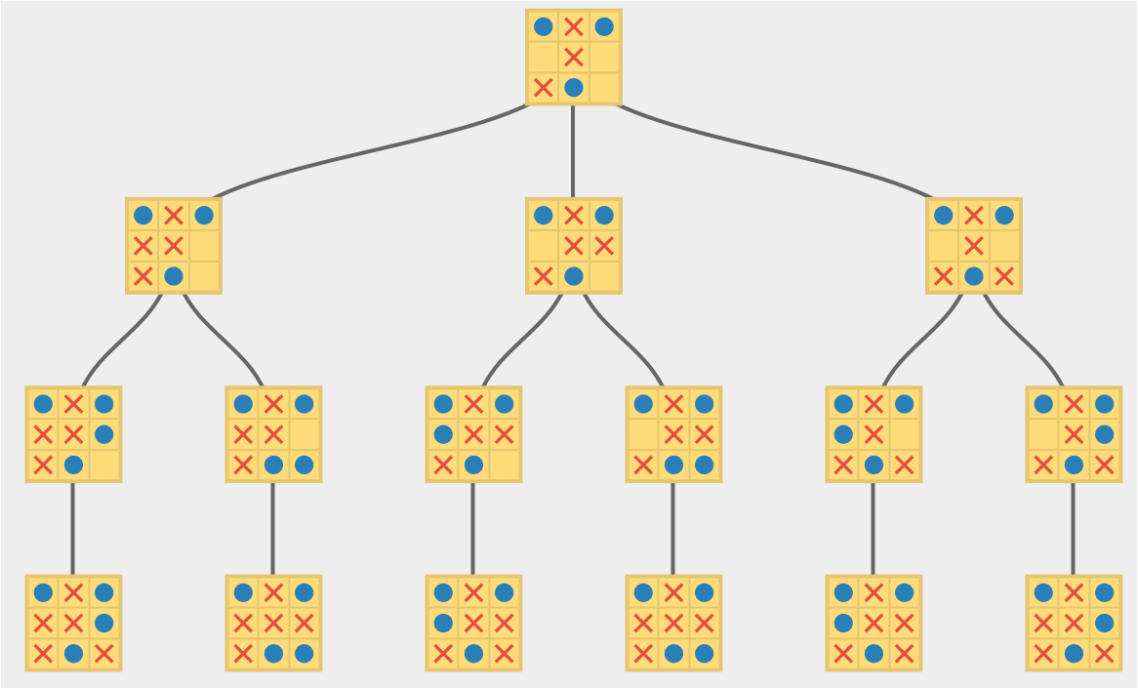
1. Wybór Gry: Wybierz konkretną grę komputerową, w którą bot ma grać. Może to być gra strategiczna, strzelanka, gra planszowa, gra sportowa lub inny rodzaj gry, który cię interesuje.
2. Zrozumienie Zasad Gry: Dokładnie zrozum zasady i mechanikę gry, w którą bot ma grać. To jest kluczowe dla skutecznego programowania bota.
3. Budowa Bota:Stwórz bot-a do gry, który będzie działać w środowisku gry. Wybierz odpowiedni język programowania i narzędzia, które umożliwią interakcję z grą.
4. Programowanie Zachowań: Programuj zachowania bota w zgodzie z zasadami gry. Upewnij się, że bot reaguje na różne sytuacje w grze i podejmuje decyzje na podstawie swojego stanu w grze.
5. Testowanie: Przetestuj bota w środowisku gry, aby ocenić jego umiejętności i efektywność w rywalizacji z innymi graczami lub botami. Analizuj wyniki i doskonal bot-a.
6. Dostosowanie i Doskonalenie: Monitoruj działania bota w grze i dostosuj jego zachowanie, aby osiągnąć lepsze wyniki. Możesz także dostosować strategię bota w zależności od reakcji innych graczy.

7. Rozwinięcie Umiejętności Bota: Jeśli masz dostęp do narzędzi do uczenia maszynowego, możesz rozważyć szkolenie bota w celu poprawy jego umiejętności i adaptacji do różnych stylów gry.
8. Rywalizacja lub Współpraca: W zależności od gry, możesz dostosować bota do rywalizacji z innymi graczami lub stworzyć go w celu współpracy z ludzkimi graczami.

To zadanie pozwoli ci na rozwinięcie bota, który może być wyzwaniem dla innych graczy i zapewni ciekawą rozrywkę w świecie gier komputerowych. Pamiętaj, że proces tworzenia bota do gier może być skomplikowany i wymagać nie tylko programowania, ale także zrozumienia strategii gry.

GRA KÓŁKO i KRZYŻYK

Poniższe ćwiczenie ma na celu przedstawić podstawowe algorytmy grające dla gier przeznaczonych dla dwóch graczy, o sumie zerowej. Suma zerowa oznacza, że zwycięstwo jednego gracza jest równocześnie porażką drugiego (suma nagród i kar jest stała i symetryczna, równa zeru, jak sama nazwa wskazuje). Wszystkie opisane algorytmy będą bazować na jednej reprezentacji rozgrywki — tzw. drzewo gry – spójny cykliczny graf. Wierzchołki drzewa reprezentują możliwe stany gry, gdzie korzeń drzewa jest początkowym stanem gry. Krawędzie natomiast reprezentują ruchy graczy, prowadzące z jednego stanu do drugiego. W przypadku turowych gier dla dwóch graczy, każda gałąź drzewa składa się z krawędzi naprzemiennie reprezentujące ruchy graczy. Poniżej widoczne jest przykładowe drzewo gry dla gry kółko i krzyżyk, rozpoczynającej się w stanie na trzy ruchy przed zapełnieniem planszy.



Oto przykładowe rozwiązanie w języku Python do gry w kółko i krzyżyk dla dwóch graczy. Gra jest implementowana w konsoli i umożliwia dwóm graczom współzawodnictwo na planszy 3x3.

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

def check_winner(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 -
i] == player for i in range(3)):
        return True
    return False

def is_board_full(board):
    return all(cell != ' ' for row in board for cell in row)

def main():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    current_player = 'X'

    print("Witaj w grze w kółko i krzyżyk!")

    while True:
        print_board(board)
        row = int(input(f"Gracz {current_player}, wybierz wiersz (0, 1,
2): "))
        col = int(input(f"Gracz {current_player}, wybierz kolumnę (0, 1,
2): "))

        if board[row][col] == ' ':
            board[row][col] = current_player
```

```
            if check_winner(board, current_player):
                print_board(board)
                print(f"Gracz {current_player} wygrywa!")
                break
            elif is_board_full(board):
                print_board(board)
                print("Remis!")
                break
            current_player = 'O' if current_player == 'X' else 'X'
        else:
            print("To pole jest już zajęte. Wybierz inne.")

if __name__ == "__main__":
    main()
```

To proste rozwiązanie implementuje podstawową grę w kółko i krzyżyk w konsoli. Gracze wybierają wiersz i kolumnę, w których chcą umieścić swój symbol. Gra sprawdza, kto wygrał lub czy jest remis, i wyświetla odpowiednie komunikaty na ekranie.
Możesz uruchomić ten kod w swoim środowisku Python, aby zagrać w kółko i krzyżyk z inną osobą.

Oczywiście, możemy dodać prostego bota do gry w kółko i krzyżyk. W poniższym przykładzie bot będzie wybierał pola na planszy w sposób losowy. Możesz dostosować ten kod, aby bot podejmował bardziej zaawansowane decyzje strategiczne w grze.

```
import random

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

def check_winner(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 -
i] == player for i in range(3)):
        return True
    return False

def is_board_full(board):
    return all(cell != ' ' for row in board for cell in row)

def get_bot_move(board):
    available_moves = [(row, col) for row in range(3) for col in range(3)
if board[row][col] == ' ']
    return random.choice(available_moves)

def main():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    current_player = 'X'

    print("Witaj w grze w kółko i krzyżyk!")

    while True:
```

```
        print_board(board)

        if current_player == 'X':
            row = int(input(f"Gracz {current_player}, wybierz wiersz (0,
1, 2): "))
            col = int(input(f"Gracz {current_player}, wybierz kolumnę (0,
1, 2): "))
        else:
            print(f"Bot ({current_player}) wybiera ruch...")
            row, col = get_bot_move(board)

        if board[row][col] == ' ':
            board[row][col] = current_player
            if check_winner(board, current_player):
                print_board(board)
                print(f"Gracz {current_player} wygrywa!")
                break
            elif is_board_full(board):
                print_board(board)
                print("Remis!")
                break
            current_player = 'O' if current_player == 'X' else 'X'
        else:
            print("To pole jest już zajęte. Wybierz inne.")

if __name__ == "__main__":
    main()
```

Teraz bot (‘O’) będzie losowo wybierał dostępne pola na planszy. Możesz dostosować strategię bota, aby bardziej rywalizował z graczem lub implementować bardziej zaawansowane algorytmy, które pomogą mu wygrywać częściej ;)

Przykładowe zadanie: Stworzenie bota do gry w kółko i krzyżyk na bazie sztucznej inteligencji może być bardziej zaawansowane. Jednym z popularnych podejść jest wykorzystanie algorytmu **Minimax** (w połączeniu z algorytmem **Alpha-Beta Pruning)** do przewidywania najlepszego ruchu bota.

Oto jak możesz to zaimplementować w Pythonie:

```
import math

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

def check_winner(board, player):
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 -
i] == player for i in range(3)):
        return True
    return False

def is_board_full(board):
    return all(cell != ' ' for row in board for cell in row)

def get_empty_cells(board):
    return [(row, col) for row in range(3) for col in range(3) if
board[row][col] == ' ']

def minimax(board, depth, is_maximizing):
    scores = {'X': 1, 'O': -1, 'tie': 0}

    if check_winner(board, 'X'):
        return -1
    if check_winner(board, 'O'):
        return 1
```

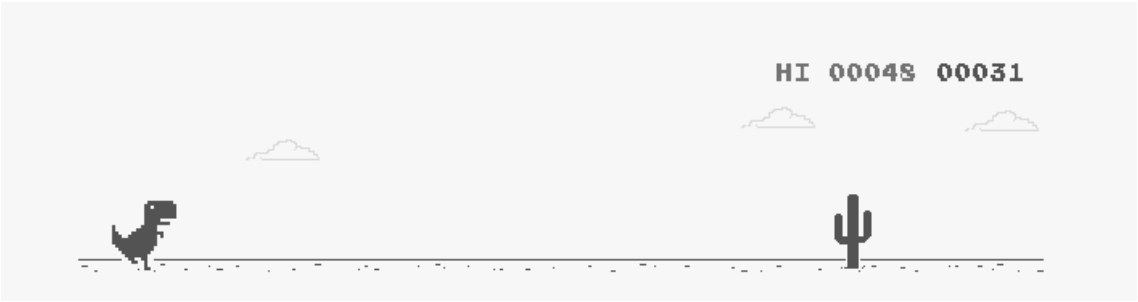
```
    if is_board_full(board):
        return 0

    if is_maximizing:
        best_score = -math.inf
        for row, col in get_empty_cells(board):
            board[row][col] = 'O'
            score = minimax(board, depth + 1, False)
            board[row][col] = ' '
            best_score = max(score, best_score)
        return best_score
    else:
        best_score = math.inf
        for row, col in get_empty_cells(board):
            board[row][col] = 'X'
            score = minimax(board, depth + 1, True)
            board[row][col] = ' '
            best_score = min(score, best_score)
        return best_score

def get_bot_move(board):
    best_score = -math.inf
    best_move = None
    for row, col in get_empty_cells(board):
        board[row][col] = 'O'
        score = minimax(board, 0, False)
        board[row][col] = ' '
        if score > best_score:
            best_score = score
            best_move = (row, col)
    return best_move

def main():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    current_player = 'X'

    print("Witaj w grze w kółko i krzyżyk!")

    while True:
        print_board(board)

        if current_player == 'X':
```

```
            row = int(input(f"Gracz {current_player}, wybierz wiersz (0,
1, 2): "))
            col = int(input(f"Gracz {current_player}, wybierz kolumnę (0,
1, 2): "))
        else:
            print(f"Bot ({current_player}) wybiera ruch...")
            row, col = get_bot_move(board)

        if board[row][col] == ' ':
            board[row][col] = current_player
            if check_winner(board, current_player):
                print_board(board)
                print(f"Gracz {current_player} wygrywa!")
                break
            elif is_board_full(board):
                print_board(board)
                print("Remis!")
                break
            current_player = 'O' if current_player == 'X' else 'X'
        else:
            print("To pole jest już zajęte. Wybierz inne.")

if __name__ == "__main__":
    main()
```

Ten kod implementuje bota, który wykorzystuje algorytm Minimax z Alpha-Beta Pruning do przewidywania najlepszego ruchu. Bot będzie próbował maksymalizować swoje szanse wygranej i minimalizować szanse przeciwnika. Dzięki temu bot będzie w stanie rywalizować na wyższym poziomie niż bot losowy.

Zadanie do przesłania na maila katarzyna.nieszporek@pcz.pl:

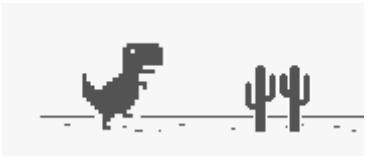
Stworzenie bota do gry **T-Rex Runner** (https://www.trex-game.skipser.com) w Python lub do gry Magic Piano https://www.gamesgames.com/game/magic-piano-tiles lub do innej wybranej gry (wówczas proszę o link do wybranej gry).



Biblioteka pyoutogui umożliwia kontrolowanie myszy i klawiatury co przyda się w tworzeniu naszego bota.

Pyautogui.click(współrzędna_x, współrzędna_y)
Pyautogui.click(współrzędna_x, współrzędna_y, click=liczba_klikniec, interval=dlugosc_pauzy_pomiedzy_kliknieciami, buton="left/right")

Celem oszacowania najlepszego momentu na podskoczenie będzie sprawdzenie, czy kolor piksela będzie szary $(RGB(83,83,83))$ w pobliżu T-Rexa.



Jak znaleźć pixela? *pyautogui.displayMousePosition()*

Inne funkcje kontroli kursora: https://pyautogui.readthedocs.io/en/latest/mouse.html

Funkcje kontroli klawiatury: https://pyautogui.readthedocs.io/en/latest/keyboard.html