پروژه درس هوش مصنوعی

Hexy

## مقدمه

این گزارش، تحلیل و بررسی کد پایتونی پیاده‌سازی بازی هگز را ارائه می‌دهد. بازی هگز یک بازی استراتژیک دو نفره است که بر روی یک شبکه شش‌ضلعی انجام می‌شود. هدف هر بازیکن این است که با قرار دادن مهره‌های خود، یک مسیر پیوسته از یک لبه به لبه مقابل ایجاد کند.

## هدف کد

این کد یک نسخه ساده از بازی تخته‌ای Hex را پیاده‌سازی می‌کند. بازی Hex یک بازی دو نفره است که در آن بازیکنان تلاش می‌کنند با قرار دادن مهره‌ها در خانه‌های خالی تخته، مسیری بین دو سمت مقابل ایجاد کنند. این برنامه قابلیت بازی بین کاربر و کامپیوتر را فراهم می‌کند و از الگوریتم Minimaxبرای تصمیم‌گیری هوشمندانه کامپیوتر استفاده می‌کند.

## شرح خطوط کد

### ۱ .کتابخانه‌ها و تعریف کلاس

```
import math
```

کتابخانه mathبرای استفاده از مقادیر مانند math.infبه‌منظور نشان دادن مثبت یا منفی بی‌نهایت در الگوریتم Minimax استفاده می‌شود.

```
class HexGame:
    def __init__(self, size=11):
        self.size = size
        self.board = [["." for _ in range(size)] for _ in range(size)]
        self.current_player = "X"
```

کلاس HexGameتعریف می‌شود.

**متغیرها:**

- size:اندازه تخته بازی (به‌صورت پیش‌فرض ۱۱x۱۱ است).
- board:تخته بازی به‌صورت یک لیست دوبعدی که خانه‌های آن با "."(خالی) مقداردهی اولیه شده‌اند.
- current_player:بازیکنی که نوبت او است (شروع با "X")

**۲ .نمایش تخته بازی**

```
def display_board(self):
    for i in range(self.size):
        print(" " * i + " ".join(self.board[i]))
    print()
```

این متد تخته بازی را نمایش می‌دهد.

**جزئیات:**

- هر ردیف تخته با مقداری فاصله مناسب (برای نمایش مورب تخته شش‌ضلعی) چاپ می‌شود.
- فاصله‌ها با " " * i "ایجاد می‌شوند.

۳ .بررسی حرکت معتبر

```
def is_valid_move(self, x, y):
    return 0 <= x < self.size and 0 <= y < self.size and self.board[x][y] == "."
```

- بررسی می‌کند که آیا حرکت پیشنهادی بازیکن معتبر است یا خیر.

- **جزئیات:**
  - حرکت باید در محدوده تخته باشد (0 <= x, y < size) و خانه انتخابی باید خالی (".") باشد.

۴ .انجام حرکت

```
def make_move(self, x, y):
    if self.is_valid_move(x, y):
        self.board[x][y] = self.current_player
        self.current_player = "O" if self.current_player == "X" else "X"
        return True
    return False
```

حرکت بازیکن را روی تخته اعمال می‌کند.

**جزئیات:**

- اگر حرکت معتبر باشد، خانه تخته به نشان بازیکن فعلی تغییر می‌کند.
- نوبت بازیکن به بازیکن بعدی تغییر می‌کند "X") به "O"و بالعکس.(

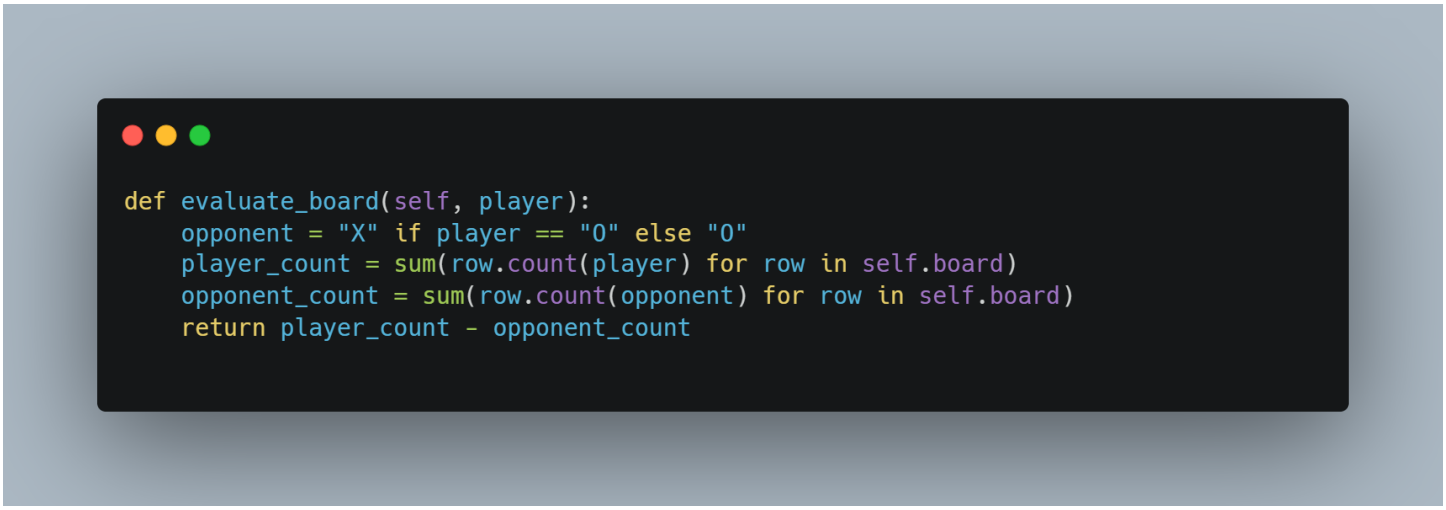
۵ .بررسی برنده

```python
def check_winner(self):
    visited = set()

    def dfs(x, y, player, target_edge):
        if (x, y) in visited:
            return False

        if (player == "X" and y == target_edge) or (player == "O" and x ==
target_edge):
            return True

        visited.add((x, y))
        directions = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, 1), (1, -1)]

        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < self.size and 0 <= ny < self.size and self.board[nx][ny]
== player:
                if dfs(nx, ny, player, target_edge):
                    return True

        return False

    for i in range(self.size):
        if self.board[i][0] == "X" and dfs(i, 0, "X", self.size - 1):
            return "X"
        if self.board[0][i] == "O" and dfs(0, i, "O", self.size - 1):
            return "O"

    return None
```

بررسی می‌کند آیا یکی از بازیکنان برنده شده است یا خیر.

- **جزئیات:**
  - از الگوریتم جستجوی عمق اول (DFS) برای یافتن مسیر متصل از لبه شروع تا لبه هدف استفاده می‌شود.
  - بازیکن "X"باید از چپ به راست و بازیکن "O"باید از بالا به پایین مسیر متصل ایجاد کند.
  - متغیر visitedاز بازدید دوباره خانه‌ها جلوگیری می‌کند.

۶ .ارزیابی وضعیت تخته



```python
def evaluate_board(self, player):
    opponent = "X" if player == "O" else "O"
    player_count = sum(row.count(player) for row in self.board)
    opponent_count = sum(row.count(opponent) for row in self.board)
    return player_count - opponent_count
```

**هدف :**وضعیت تخته را از نظر تعداد مهره‌های بازیکن و حریف ارزیابی می‌کند.

**جزئیات:**

- تعداد مهره‌های بازیکن و حریف شمرده شده و اختلاف آن‌ها بازگردانده می‌شود.

## ۷ .الگوریتمMinimax

```python
def minimax(self, depth, is_maximizing, alpha, beta):
    winner = self.check_winner()
    if winner == "O":
        return 1000
    elif winner == "X":
        return -1000
    elif all(cell != "." for row in self.board for cell in row):
        return 0

    if depth == 0:
        return self.evaluate_board("O")

    if is_maximizing:
        max_eval = -math.inf
        for x in range(self.size):
            for y in range(self.size):
                if self.is_valid_move(x, y):
                    self.board[x][y] = "O"
                    eval = self.minimax(depth - 1, False, alpha, beta)
                    self.board[x][y] = "."
                    max_eval = max(max_eval, eval)
                    alpha = max(alpha, eval)
                    if beta <= alpha:
                        break
        return max_eval
    else:
        min_eval = math.inf
        for x in range(self.size):
            for y in range(self.size):
                if self.is_valid_move(x, y):
                    self.board[x][y] = "X"
                    eval = self.minimax(depth - 1, True, alpha, beta)
                    self.board[x][y] = "."
                    min_eval = min(min_eval, eval)
                    beta = min(beta, eval)
                    if beta <= alpha:
                        break
        return min_eval
```

حرکت بهینه برای کامپیوتر را با استفاده از Minimax به همراه برش آلفا-بتا پیدا می‌کند.

- **جزئیات:**
  - نودهای برنده/بازنده با مقادیر ثابت 1000) یا (۱۰۰۰-ارزیابی می‌شوند.

- در سطوح پایین‌تر، ارزش تخته با evaluate_boardمحاسبه می‌شود.
- is_maximizing:اگر کامپیوتر در حال حرکت باشد، تلاش می‌کند حداکثر امتیاز را به دست آورد.

۸ .پیدا کردن بهترین حرکت

```
def find_best_move(self):
    best_value = -math.inf
    best_move = None
    for x in range(self.size):
        for y in range(self.size):
            if self.is_valid_move(x, y):
                self.board[x][y] = "O"
                move_value = self.minimax(3, False, -math.inf, math.inf)
                self.board[x][y] = "."
                if move_value > best_value:
                    best_value = move_value
                    best_move = (x, y)
    return best_move
```

بهترین حرکت ممکن را برای کامپیوتر پیدا می‌کند.

- **جزئیات:**
  - تمامی حرکات ممکن بررسی شده و بهترین حرکت انتخاب می‌شود.

۹ .حلقه اصلی بازی

```python
def play(self):
    print("Welcome to Hex!")
    self.display_board()

    while True:
        if self.current_player == "X":
            print(f"Player {self.current_player}'s turn")
            try:
                x, y = map(int, input("Enter your move (row and column):
").split())
                if self.make_move(x, y):
                    self.display_board()
                    winner = self.check_winner()
                    if winner:
                        print(f"Player {winner} wins!")
                        break
                else:
                    print("Invalid move. Try again.")
            except ValueError:
                print("Invalid input. Please enter two integers.")
        else:
            print("Computer's turn")
            move = self.find_best_move()
            if move:
                self.make_move(*move)
                print(f"Computer chose: {move[0]} {move[1]}")
                self.display_board()
                winner = self.check_winner()
                if winner:
                    print(f"Player {winner} wins!")
                    break
```

حلقه اصلی بازی که تعامل بازیکن و کامپیوتر را مدیریت می‌کند.

**جزئیات:**

- بازیکن با وارد کردن مختصات حرکت می‌کند.
- کامپیوتر از find_best_moveبرای تصمیم‌گیری استفاده می‌کند.

کد ها در گیت هاب به لینک زیر موجود هستند

https://github.com/vahidseyyedi/Hex-Game

سید وحید سیدی پاییز ۱۴۰۳ درس هوش مصنوعی استاد آقایی