گزارش پروژه نهایی درس هوش مصنوعی (دکتر سلیمی بدر)

گروه **AI Rebels** : نیلا چناری | مهدی رضایی

در این فاز علاوه بر الگوریتم مینیماکس با عمق محدود و هیوریستیک، الگوریتمی با الهام از روش های جست و جو غیرکلاسیک، برای یادگیری و بهبود ایجنت قبلی پیاده سازی کرده ایم.

پیاده سازی و رویکرد کلی

- در این فاز برای استفاده از کارت های همراه، بخشی به MiniMax باید اضافه کنیم تا در صورت وجود امکان انتخاب کارت، کارت همراه و کارت های همراه آن را نیز به خوبی انتخاب کند.

- **ارزیابی وضعیت بازی با تنظیم وزن‌ها**

با تحلیل وزن های اختصاص داده شده به خاندان های مختلف، کار خود را پیش می بریم.

  - در صورت پیروزی وزن های خاندان ها ثابت مانده و تا 10 قدم سعی می کند حرکت حریف را محدود کند.
  - در صورت باخت، وزن‌های مربوط خاندان هایی که کمتر از نصف کارت آنها را داریم افزایش پیدا می کنن (+10) و وزن های خاندان هایی که بیشتر از نصف آن ها را داریم کاهش پیدا می کنند(-1). در این صورت سعی می کند با اندکی فراموشی، سبک بازی خود را پیدا کند

- **الهام از الگوریتم های Gradient decent، Genetic، Simulated Anealng**
  - لازم به ذکر است که منظور رویکردی شبیه این الگوریتم است و پیاده سازی این الگورتیم برای بازی ما مناسب نیست. زیرا گرادیان دیسنت نیازمند تابعی مشتق پذیر است ولی تابع بازی ما به این صورت نیست.
  - پس از مدل ژنتیک کمک گرفته و با وزن های رندم شروع می کنیم. در پایان هر بازی که انجام می شود، با توجه به نتیج بازی وزن ها را کم یا زیاد می کند.

- با توجه به اینکه بعد از گذشتن چند مرحله ارزش عدد ثابتی که به وزن ها اضافه یا کم می شود، کاهش پیدا می کند، الگوریتم بعد از یک بازه زمانی یک سبک بازی را به خاطر سپرده و شروع به بهبود دادن آن می کند.
- این روش کمک میکند تا مدل اهمیت کارت‌های خاندان‌های مختلف، میزان محدود کردن حریف و ارزش کارت های کمکی را در پیروزی بهتر درک کند.

پیاده سازی در کد

- تابع **minimax_right** برای کارت های راست

با گرفتن استیت بازی، برای تمام کارت های موجود و انتخاب های پیش روی آنها، با فرض انجام شدن، تابع minimax را صدا میزند تا بهترین گزینه پیش رو را انتخاب کند.

```
best_move = None
if maxplayer:
    best_val = -float("inf")
    for move in next_move:

        if move == 'Jon' or move == 'Sandor' or move == 'Jaqen':...

        elif move == 'Gendry':...

        elif move == 'Ramsay':...


        elif move == 'Melisandre':...

        else:
```

تابع همچین ساختاری دارد و در آخرین انتخاب، مینیمکس اصلی بازی را صدا میزند.

```
elif move == 'Gendry':
    print("Gendry is controlling")

    cards_copy = copy.deepcopy(cards)
    player1_copy = copy.deepcopy(player1)
    player2_copy = copy.deepcopy(player2)
    companion_cards_copy = copy.deepcopy(companion_cards)
    new_move = ["Gendry"]
    choose_companion, Maxplayer, player1_copy, player2_copy = apply(new_move, companion_cards
    val, _ = minimax(cards_copy, Maxplayer, alpha, beta, player1_copy, player2_copy, start_ti

    if val < best_val:
        best_val = val
        best_move = new_move
```

- **تابع apply**

همچنین برای اعمال حرکت ها برای جلوگیری از پیچیدگی پیاده سازی برای اعمال حرکت از تابع کمکی apply استفاده شده تا وضعیت را با توجه به کارت های انتخابی اعمال کند.

```
def apply(move, companion_cards, turn, player1, player2, cards):
    choose_companion = True  # Reset the flag
```

- **تابع و منطق هیوریستیک**

از آنجایی که در فاز یک هیوریستیک ما تلاش به گرفتن بیشترین قبیله ها در کمترین زمان ممکن بود، و در این فاز استفاده از کارت های همراه مستلزم گرفتن پرچم هستند، تغییر زیادی در آن ایجاد نمیکنیم.

اما ورودی choose_companion را به آن اضافه کردیم، تا اگر حرکتی منجر به انتخاب یک کارت همراه شود امتیاز مثبت برایش لحاظ شود.

```
if choose_companion:
    score += weight[0]
```

ترین کردن و پیدا کردن وزن ها

- پس از ترین کردن عامل ۸۰ درصد بازی هایی که با مدل ترین نشده خود داشت را برنده میشود.

- **main**

وزن هایی رندم جنریت میکنیم، و ۱۰۰ بار تابع evalutae_fitness را صدا زده تا بهترین حالت را پیدا کنیم

```
if __name__ == "__main__":

    weight = generate_random_weights()
    count = 0
    for i in range(100):
        weight,count = evaluate_fitness(weight,count)

    print(weight)
```

- **تابعی کمکی برای وزن های رندم**

```
def generate_random_weights():
    # Generate random weights for the evaluation function
    w = [random.randint( a: 0,  b: 100) for _ in range(9)]
    w[0] = sum(w[2:])//7
    w[1] = 1
    return w
```

- **تابع evaluate_fittness**

این تابع بخش تصمیم گیری شایسته بودن را دارد. در صورت برنده شدن در چند دست متوالی زمین را عوض می کند در غیر این صورت در همان زمین سعی می کند بهترین بازی خود را ارائه دهد.

در 100 مرحله ای که آموزش می بیند سعی، نزدیک به 7 تا 8 زمین مختلف را کامل پیروز می شود.

بخش ابتدایی تابع:

```
1 usage   new *
def evaluate_fitness(chromosome,count):
    global Board
    board = copy.deepcopy(Board)
    fe = []
    winner, cards, player1, player2 = main(parser.parse_args(),board,chromosome)

    if winner == 1:
        # count = 0
        if chromosome[1] >= 10:
            print("New Board")
            Board = make_board()
        else:
            chromosome[0] = min(2*chromosome[1],10)
    else:
        chromosome[1] = max(chromosome[1]-1,1)
```

بازی را انجام داده و داده ها را در فایلی یادداشت میکنیم و ورودی و خروجی ها به همچین مدلی ذخیره خواهند شد.

```
for house, weight in house_weights.items():
    if len(player1.cards[house]) > weight / 2:
        score += house_weights_[house]
        if winner == 2:
            chromosome[10 - weight] -= 1
    elif len(player2.cards[house]) > weight / 2:
        score -= house_weights_[house]
        if winner == 2:
            chromosome[10-weight] += 10
    elif len(player1.cards[house]) == len(player2.cards[house]) == weight / 2:
        point = (player2_banners[house] - player1_banners[house])
        score -= point * house_weights_[house]
        if winner == 2:
            if point < 0:
                chromosome[10-weight] -=1
            else:
                chromosome[10-weight] += 10
    p1.append(len(player1.cards[house]))
    p2.append(len(player2.cards[house]))
fe.extend(p1)
fe.extend(p2)
fe.extend(chromosome)
fe.append(winner)
fe.append(score)
# File path to save the array
file_path = 'feN3.csv'
```

```
0,1,0,1,0,0,0,1,0,1,0,1,1,1,3,5,3,4,3,0,1,4,2,3,1,4,3,1,181,9,179,1
1,0,0,1,1,0,1,0,1,1,0,0,1,0,4,4,1,3,4,1,2,4,4,5,2,1,2,0,181,10,179,
1,0,0,1,1,0,1,0,1,1,0,0,1,0,4,4,1,3,4,1,2,4,4,5,2,1,2,0,181,10,179,
0,0,1,1,0,1,0,1,1,0,0,1,0,1,3,0,2,5,1,3,0,7,3,2,0,1,0,2,185,9,189,1
0,0,1,1,0,1,0,1,1,0,0,1,0,1,3,0,2,5,1,3,0,7,3,2,0,1,0,2,188,8,199,1
0,0,1,1,0,1,0,1,1,0,0,1,0,1,3,0,2,5,1,3,0,7,3,2,0,1,0,2,190,7,209,1
0,0,1,1,0,1,0,1,1,0,0,1,0,1,3,0,2,5,1,3,0,7,3,2,0,1,0,2,193,6,219,1
0,0,1,1,0,1,0,1,1,0,0,1,0,1,3,0,2,5,1,3,0,7,3,2,0,1,0,2,196,5,229,1
0,0,1,0,1,1,1,1,1,0,1,0,0,0,3,3,3,2,3,3,2,5,4,3,2,2,0,2,196,10,229,
0,0,1,1,0,1,0,1,1,0,0,1,0,1,3,0,2,5,1,3,0,7,3,2,0,1,0,2,198,9,239,1
0,0,1,0,1,1,1,1,1,0,1,0,0,0,3,3,3,2,3,3,2,5,4,3,2,2,0,2,198,10,239,
0,0,1,1,0,1,0,1,1,0,0,1,0,1,3,0,2,5,1,3,0,7,3,2,0,1,0,2,201,9,249,1
0,0,1,0,1,1,1,1,1,0,1,0,0,0,3,3,3,2,3,3,2,5,4,3,2,2,0,2,201,10,249,
0,0,1,0,1,1,1,1,1,0,1,0,0,0,3,3,3,2,3,3,2,5,4,3,2,2,0,2,201,10,249,
1,1,0,0,0,0,0,0,0,1,1,1,1,1,5,6,1,1,2,1,0,2,1,4,4,3,2,2,203,9,248,1
1,1,0,0,0,0,0,0,0,1,1,1,1,1,5,6,1,1,2,1,0,2,1,4,4,3,2,2,210,8,247,1
0,1,0,0,1,0,1,1,0,1,1,0,1,0,4,5,2,2,3,1,2,4,2,4,3,1,2,0,217,7,257,1
0,1,0,0,1,0,1,1,0,1,1,0,1,0,4,5,2,2,3,1,2,4,2,4,3,1,2,0,222,6,267,1
0,1,0,0,1,0,1,1,0,1,1,0,1,0,4,5,2,2,3,1,2,4,2,4,3,1,2,0,228,5,277,1
0,1,0,0,0,1,0,1,0,1,1,1,0,1,4,4,3,2,2,2,1,4,3,4,3,2,1,1,233,4,287,1
0,1,0,0,1,0,1,1,0,1,1,0,1,0,4,5,2,2,3,1,2,4,2,4,3,1,2,0,240,3,297,1
```