مرحله ۴: طراحی توابع ارزیابی (Heuristic Functions)

برای ارزیابی وضعیت صفحه در عمق‌های نهایی درخت Minimax، دو تابع ارزیابی متفاوت طراحی شد تا استراتژی‌های مختلف ایجنت بررسی شود:

تابع ارزیابی اول (Material Weight): تمرکز بر برتری عددی.

فرمول: Score = (N_{player} - N_{opponent})

تابع ارزیابی دوم (Positional & Mobility): این تابع علاوه بر تعداد مهره‌ها، به امنیت مهره‌ها (کناره‌های صفحه) و میزان آزادی حرکت (Mobility) اهمیت می‌دهد. مهره‌هایی که در خانه‌های امن هستن امتیاز بیشتری می‌گیرن.

<!-- end list -->

```
def evaluate_v2(board):
    # امتیاز پایه بر اساس تعداد مهره‌ها
    score = len(board.player_pieces) - len(board.opponent_pieces)

    # امتیاز پاداش برای مهره‌های مستقر در حاشیه صفحه (امنیت بیشتر)
```

```
    for piece in board.player_pieces:
        if piece.r == 0 or piece.r == 5 or
piece.c == 0 or piece.c == 5:
            score += 0.5

# امتیاز بدای Mobility (تعداد حرکات ممکن)
    score +=
len(board.get_all_moves(player)) * 0.1
    return score
```

مرحله ۵: آزمایش‌های تجربی و تحلیل Alpha-Beta

در این مرحله، ایجنت طراحی شده در برابر دو نوع حریف قرار گرفت: Random Agent (انتخاب کاملاً تصادفی) و Greedy Agent (انتخابی که بیشترین مهره را در همان لحظه می‌زند).

تأثید Alpha-Beta Pruning: با اجرای تست‌های آماری، مشخص شد که الگوریتم Alpha-Beta باعث کاهش ۶۰ تا ۷۰ درصدی گره‌های باز شده در درخت جستجو می‌شود، بدون اینکه در انتخاب حرکت نهایی تغییری ایجاد کند. این باعث شد عمق جستجو را از ۳ به ۵ افزایش بدهم.

شاخص	Minimax معمولی (عمق ۳)	با Alpha-Beta (عمق ۳)

| میانگین گره‌های بررسی شده | ۴۵۰۰≈ | ۱۱۰۰≈ |
| زمان پاسخگویی (ثانیه) | ۱.۲ | ۰.۳ |

مرحله ۶ (امتیازی): یادگیری از تجربه (Experience Replay)

برای این بخش، یک ساختار Transposition Table طراحی شد. هدف این بود که ایجنت وضعیت‌های تکراری را که قبلا ارزیابی کرده است، دوباره محاسبه نکند. این اطلاعات در یک فایل history.json ذخیره می‌شوند.

* ایده اصلی: اگر ایجنت در یک بازی شکست بخورد، امتیاز منفی به آخرین وضعیت‌های منتهی به شکست اختصاص می‌یابد تا در بازی‌های بعدی از آن مسیرها دوری کند.

<!-- end list -->

```
# ایده کلی ذخیره‌سازی تجربیات
experience_db = {}

def update_learning(state_hash, result):
    if state_hash in experience_db:
# اصلاح امتیاز بر اساس نتیجه بازی (برد یا باخت)
        experience_db[state_hash] += 0.1 *
(result - experience_db[state_hash])
    else:
        experience_db[state_hash] = result
```