Shallow blue: finalni prototip min-max algoritma

Autori: Sara Pruševi, Milena Mijucić, Mihajlo Bencun

Elektronski Fakultet u Nišu

Funkcija *minmax* je algoritam koji se koristi za određivanje najboljeg poteza u igrama sa dva igrača. Osnovni princip je minimizacija gubitaka za maksimiziranje dobitaka. U svom radu, funkcija koristi rekurziju da istraži sve moguće ishode do određene dubine (*depth*).

```python
def minmax(depth, maximizingPlayer, alpha=float('-inf'), beta=float('inf')):
    if depth == 0 or is_terminal_node():
        return evaluate_game_state()

    if maximizingPlayer:
        maxEval = float('-inf')
        for move in get_valid_moves(appState, appState.get_opponent(appState.currentPlayer).color):
            prevState = apply_move(move)
            eval = minmax(depth - 1, False, alpha, beta)
            maxEval = max(maxEval, eval)
            undo_move(prevState)
            alpha = max(alpha, eval)
            if beta <= alpha:
                break
        return maxEval
    else:
        minEval = float('inf')
        for move in get_valid_moves(appState, appState.currentPlayer.color):
            prevState = apply_move(move)
            eval = minmax(depth - 1, True, alpha, beta)
            minEval = min(minEval, eval)
            undo_move(prevState)
            beta = min(beta, eval)
            if beta <= alpha:
                break
        return minEval
```

Slika 1

Za igrača koji maksimizira rezultat (AI), funkcija traži potez koji će dovesti do najveće vrednosti rezultata. Za igrača koji minimizira rezultat (protivnik), traži potez koji će AI-u doneti najmanju vrednost rezultata. Ako je *maximizingPlayer* istinit, funkcija traži potez koji maksimizuje skor (Slika 1). Za svaki mogući potez, ona simulira potez (*apply_move*), poziva sebe rekurzivno sa smanjenom dubinom, i potom vraća igru na prethodno stanje (*undo_move*). Vrednost *alpha* se ažurira svakim pronađenim boljim skorom, a pretraga se prekida ako *alpha* postane veća ili jednaka *beta*. U suprotnom ako *maximizingPlayer* nije istinit, funkcija traži potez koji minimizira skor.

Pomoćne funkcije koje koristi minmax (Slika 2) su:

- **is_terminal_node()**: Proverava da li je igra završena.
- **apply_move()**: Primjenjuje potez na trenutno stanje igre (*appState*) i čuva prethodno stanje.
- **undo_move()**: Vraća igru na prethodno stanje.

```
def is_terminal_node():
        # proveri da li je igra zavrsena
        if appState.finished == True:
            return True
        return False

def apply_move(move):  # samo privremeno nek ode potez u appState
    global appState
    previous_state = appState.copy_state()
    appState.set_state(move[0], move[1], move[2])
    return previous_state
def undo_move(previous_state):
        global appState
        appState = previous_state
```

Slika 2

U funkciji **evaluate_game_state()** (Slika 3) smo implementirali heuristike na osnovu pravila igre. Naime koristili smo 4 strategije:

1. **Kontrola AI-a**: Prebrojavanje koliko stekova AI kontrolise. To je osnovna heuristika koja daje bodove za svaki stek gde je poslednji marker AI-ov.
2. **Kontrola Centra**: Daje dodatne bodove za stekove koji su bliži centru table. To je urađeno kako bi se AI podstakao da zauzme i drži pozicije koje su strateški važne. Bodovi se računaju tako što se kvadrira razlika između stvarne pozicije i centra, promovišući tako stekove bliže centru.
3. **Potencijalni Potezi**: Ocena broja mogućih poteza koje AI može da napravi. To omogućava AI-u da razmotri svoju mobilnost na tabli.
4. **Mobilnost Stekova**: Daje bodove na osnovu broja opcija za kretanje koje AI ima. To je kvadrat broja opcija za kretanje, naglašavajući važnost održavanja fleksibilnosti u pokretima stekova.

Ove heuristike zajedno omogućavaju AI-u da proceni stanje igre kombinujući različite strateške faktore, što rezultira složenijim i potencijalno snažnijim odlukama u odnosu na jednostavnije metode ocenjivanja.

```python
def evaluate_game_state():
    global appState
    ai_control = 0
    potential_moves = 0
    center_control = 0
    stack_mobility = 0
    ai_color = list(filter(lambda x: x.type == PlayerType.Computer, appState.players))[0].color
    center = appState.matrixSize // 2

    for row in range(appState.matrixSize):
        for col in range(appState.matrixSize):
            field = appState.matrix.matrix[row][col]
            if field.stack:
                # prebroj stekove koje kontrolise AI
                if field.stack[-1].color == ai_color:
                    ai_control += 1
                    # dodatni poeni za kontrolisanje stekova blizu centra
                    center_distance = max(abs(center - row), abs(center - col))
                    center_control += (center - center_distance) ** 2
                #Racuna moguce poteze za AI
                if field.stack[0].color == ai_color:
                    move_options = len(possibleDestinations((row, col)))
                    potential_moves += move_options
                    #dodatni poeni za stekove sa vise opcija
                    stack_mobility += move_options ** 2

    # skoring funkcija je linearna kombinacija svih faktora
    return ai_control + potential_moves + center_control + stack_mobility
```

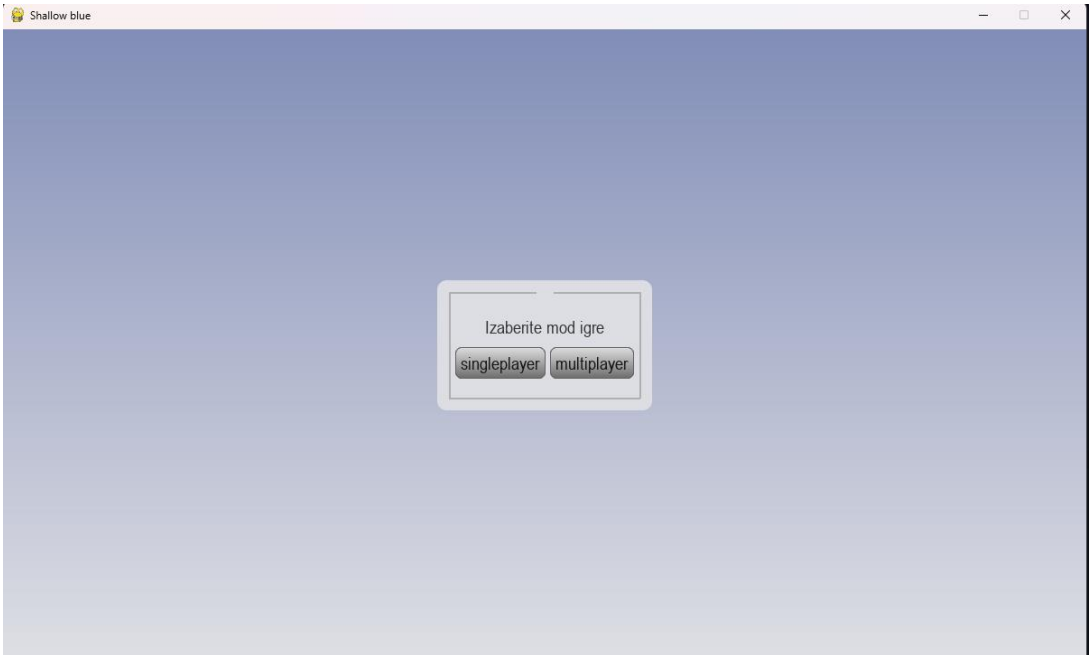
Slika 3

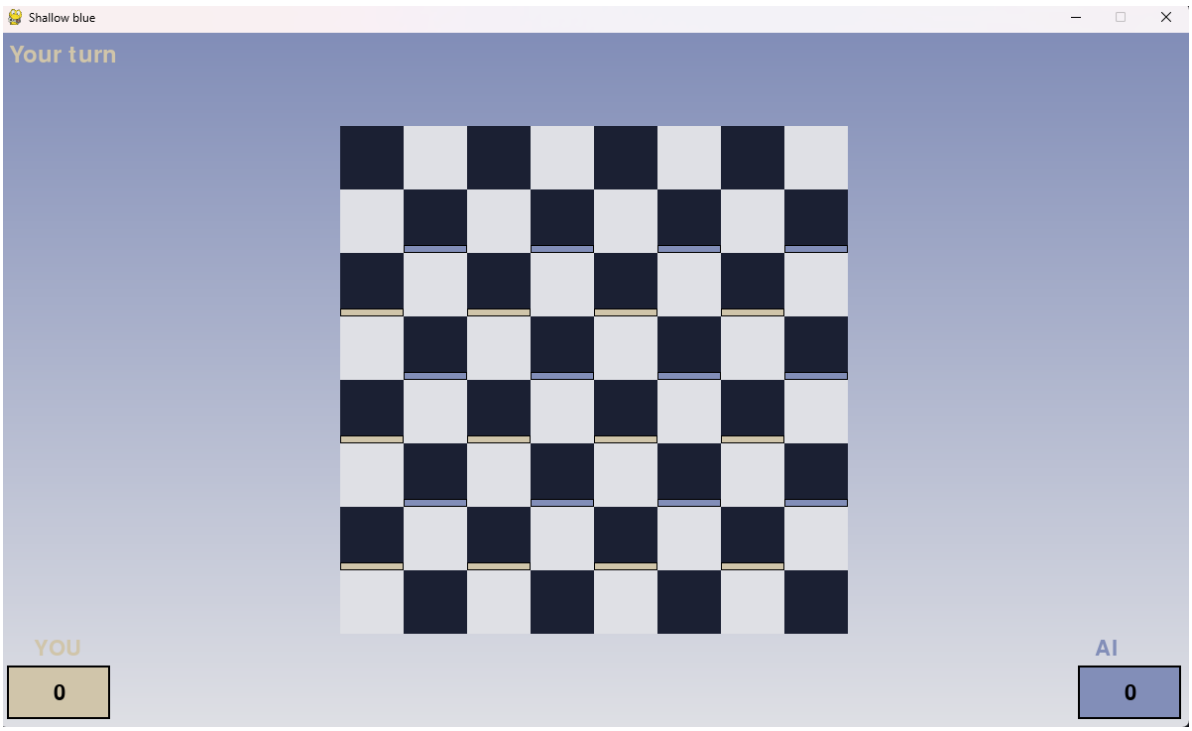
Konačno, funkcija **aiMove()** ()koristi *minmax* metodu da izračuna najbolji mogući potez za AI (slika 4). Nakon što se pređe kroz sve validne poteze, izabere se onaj sa najboljom ocenom i primeni na igru.

```python
def aiMove():
    print("AI move")
    best_score = float('-inf')
    best_move = None
    moves = get_valid_moves(appState, appState.currentPlayer.color)
    for move in moves:
        previous_state = apply_move(move)
        score = minmax(depth=1, maximizingPlayer=False)
        undo_move(previous_state)

        if score > best_score:
            best_score = score
            best_move = move
            print(move)

        tp.refresh_waiting_bar()

    if best_move:
        appState.currentMove = [best_move[0], best_move[1], best_move[2]]
        performMove()

    return best_move
```

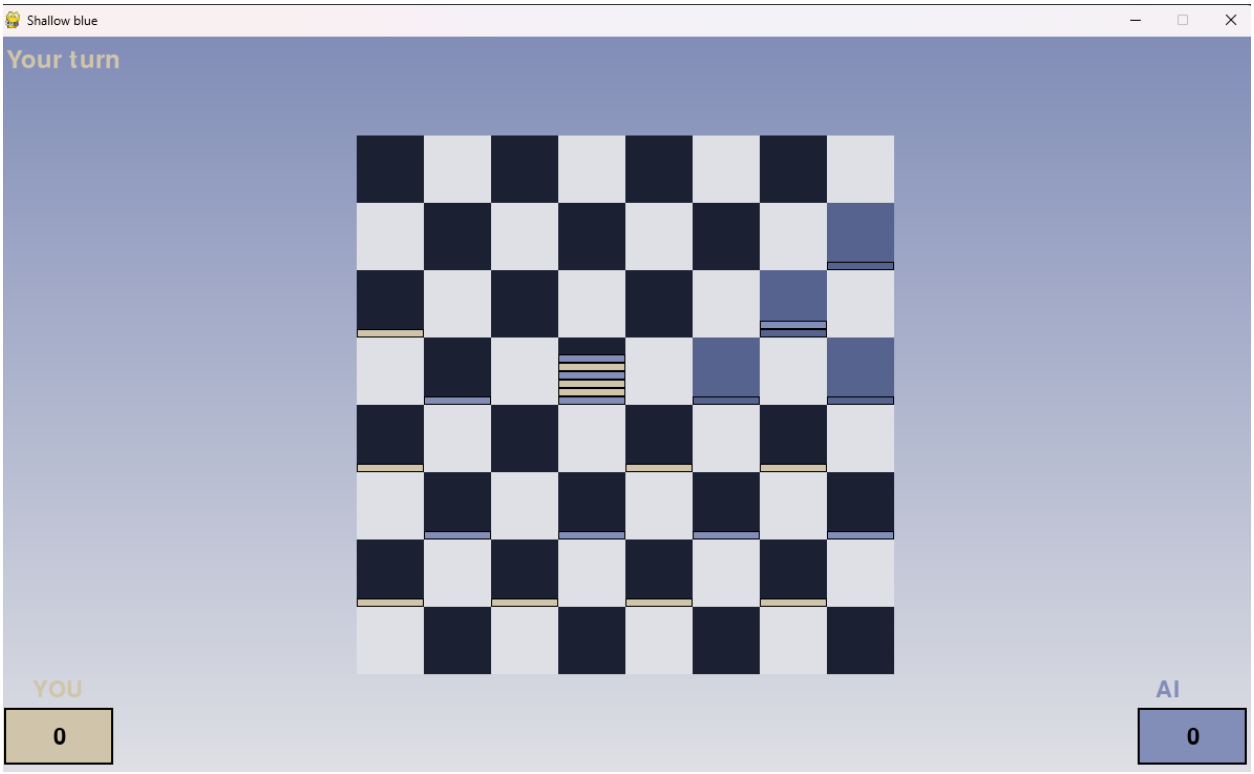
Slika 4

Izgled aplikacije je sledeći:

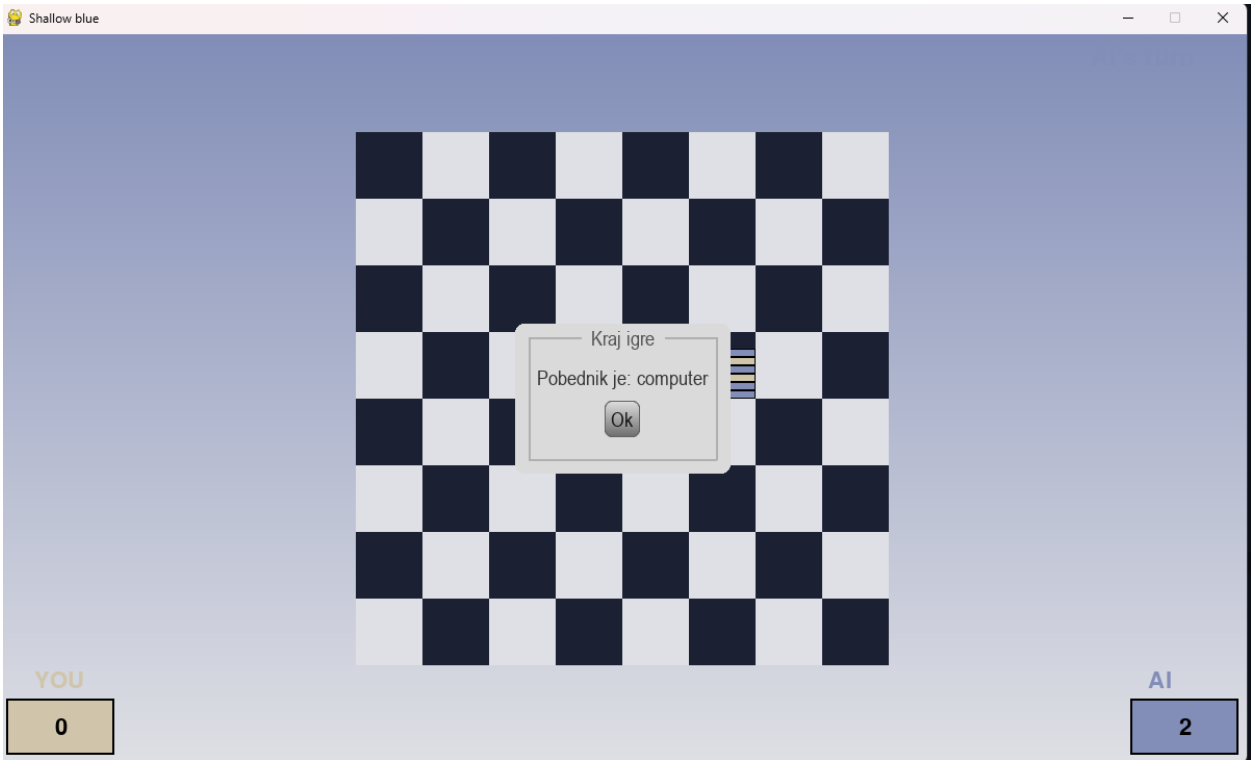




Inicijalno stanje



Selektovanje mogućih poteza



Nažalost (ili pak na našu srecu) mi nismo uspeli da pobedimo Shallow blue u ovoj igri. Nakon temeljnog razvoja i usavršavanja AI-a kroz Minimax algoritam sa alfa-beta orezivanjem, naša veštačka inteligencija sada predstavlja ozbiljan izazov čak i za iskusne igrače. Sa implementiranim strategijama kao što su kontrola centra i mobilnost stekova, AI je postao izuzetno teško pobediti. Ipak, u duhu igre i neprestanog poboljšanja, izazivamo vas da testirate svoje strategije protiv našeg AI-a Shallow blue (koji nije bas toliko shallow) i podelite svoja iskustva. Možda ipak vi otkrijete ključ za prevagu u ovoj uzbudljivoj igri uma. Srećno!