Reversi - Gra Sieciowa (Programowanie Współbieżne)

Autor: Marta Piotrowska

Przedmiot: Programowanie Współbieżne

Temat: Zadanie 11 - Gra Reversi w architekturze klient-serwer.

---

1. Treść zadania

Projekt realizuje grę strategiczną **Reversi** dla dwóch osób rozgrywaną na planszy o wymiarach 8x8.

Zasady gry:

- **Początek:** Na środku planszy znajdują się dwa białe i dwa czarne pionki w układzie diagonalnym.
- **Przebieg:** Gracze na zmianę stawiają pionki swojego koloru (Czarny lub Biały). Ruch jest dozwolony tylko wtedy, gdy otacza (w poziomie, pionie lub po skosie) co najmniej jeden pionek przeciwnika.
- **Przejmowanie:** Otoczone pionki przeciwnika zmieniają kolor na kolor gracza wykonującego ruch.
- **Brak ruchu:** Jeśli gracz nie może wykonać ruchu, traci kolejkę.
- **Koniec gry:** Gra kończy się, gdy plansza zostanie zapełniona lub żaden z graczy nie może wykonać ruchu. Wygrywa osoba z większą liczbą pionków na planszy.

---

2. Instrukcja Uruchomienia

Projekt został napisany w języku **Python**. Do obsługi interfejsu graficznego wykorzystano bibliotekę **Pygame**, a do komunikacji sieciowej standardową bibliotekę **socket**.

Wymagania

- Python 3.x
- Biblioteka Pygame

Instalacja zależności

Przed uruchomieniem należy upewnić się, czy zainstalowana jest biblioteka pygame. Jeśli nie, można to zrobić następującą komendą:

```
pip install pygame
```

Uruchamianie gry

Gra działa w architekturze klient-serwer. Należy zatem uruchomić procesy w następującej kolejności:

1. **Uruchomienie serwera:**
   ```
   python server.py
   ```

   Serwer nasłuchuje na localhost (127.0.0.1) na porcie 5001.

2. **Uruchomienie pierwszego klienta (Gracz 1 - Czarny):**
   ```
   python main.py
   ```

3. **Uruchomienie drugiego klienta (Gracz 2 - Biały):**
   ```
   python main.py
   ```

---

3. Dokumentacja Techniczna

3.1. Wykorzystane technologie i biblioteki

- **pygame**: Odpowiada za renderowanie okna, rysowanie planszy, pionków oraz obsługę zdarzeń myszy (kliknięcia).
- **socket**: Zapewnia komunikację TCP/IP między klientami a serwerem.
- **threading**: Używany zarówno po stronie serwera (obsługa wielu klientów jednocześnie), jak i klienta (nasłuchiwanie wiadomości od serwera bez blokowania interfejsu graficznego).

3.2. Schemat Komunikacji (Protokół)

Komunikacja odbywa się za pomocą gniazd sieciowych (TCP). Serwer jest "źródłem prawdy" – przechowuje stan gry i waliduje ruchy.

Format wiadomości:

Komunikaty są przesyłane jako ciągi tekstowe kodowane w UTF-8.

1. **Nawiązanie połączenia:**
    - **Serwer -> Klient:** *WELCOME* <player_id>
        - Przydziela ID: 1 (Czarny) lub 2 (Biały).
    - Jeśli połączy się 3. gracz, otrzyma komunikat *FULL* i zostanie rozłączony.
2. **Synchronizacja stanu planszy:**
    - **Serwer -> Wszyscy Klienci:** *BOARD* <board_string> <current_turn>
        - <board_string>: Ciąg 64 znaków reprezentujący planszę (np. 00000012...). 0=puste, 1=czarny, 2=biały.
        - <current_turn>: ID gracza, który ma teraz wykonać ruch (lub 0 jeśli koniec gry).
3. **Wykonywanie ruchu:**
    - **Klient -> Serwer:** *MOVE* <row> <col>
        - Wysyłane po kliknięciu myszką w pole planszy.
    - Serwer sprawdza poprawność ruchu w logic.py. Jeśli ruch jest poprawny, aktualizuje planszę i rozsyła nowy stan BOARD do obu graczy.

3.3. Logika Gry (logic.py)

Plik logic.py zawiera klasę ReversiLogic, która działa całkowicie niezależnie od grafiki.

- **Reprezentacja:** Plansza to dwuwymiarowa lista 8x8.
- **Walidacja (get_valid_moves):** Algorytm sprawdza 8 kierunków (poziomo, pionowo, skosy) od wybranego pola. Ruch jest ważny tylko wtedy, jeśli w danym kierunku znajduje się ciąg pionków przeciwnika domknięty pionkiem gracza.
- **Zmiana tury (handle_turn_change):** System automatycznie sprawdza, czy następny gracz ma dostępne ruchy. Jeśli nie, tura wraca do obecnego gracza (tzw. "pass"). Jeśli żaden gracz nie ma ruchu, ustawiany jest stan turn = 0 (Koniec gry).

3.4. Generowanie Planszy i Interfejs (board.py i main.py)

Proces rysowania i aktualizacji przebiega następująco:

1. **Odbiór danych:** Klient w osobnym wątku (NetworkClient) odbiera komunikat *BOARD* z ciągiem 64 znaków.
2. **Parsowanie:** Metoda board.update_from_string() w pliku board.py iteruje po tym ciągu i wypełnia lokalną macierz stanu.
3. **Renderowanie (draw):**
    - Pętla gry w main.py wywołuje board.draw().
    - Rysowane jest zielone tło i czarna siatka.
    - Dla każdego pola macierzy, jeśli wartość wynosi 1, rysowane jest czarne koło; jeśli 2 – białe.
    - Interfejs wyświetla również etykiety kolumn (A-H) i wierszy (1-8).

---

4. Obsługa sytuacji błędnych

Program został zabezpieczony przed typowymi problemami:

1. **Próba ruchu poza kolejnością:**
   - Klient lokalnie sprawdza czy my_id == turn. Jeśli gracz kliknie w planszę podczas tury przeciwnika, w konsoli pojawi się komunikat "Czekaj na swoją kolej!", a żądanie nie zostanie wysłane.
   - Serwer dodatkowo weryfikuje turę przed akceptacją ruchu.
2. **Nieprawidłowy ruch:**
   - Jeśli gracz kliknie w pole, które nie spełnia zasad Reversi, serwer odrzuci ruch, a stan gry nie ulegnie zmianie.
3. **Zerwanie połączenia:**
   - Serwer usuwa rozłączonego gracza z listy klientów.
   - Klient w przypadku błędu połączenia bezpiecznie zamyka aplikację.
4. **Brak serwera:**
   - Próba uruchomienia klienta bez serwera zakończy się komunikatem błędu i wyjściem z programu.