Implementacja gry Blackjack w paradygmacie obiektowym (C++).

Autor przeglądu: Łukasz Srebrniak

1. Architektura i dekompozycja systemu

Projekt charakteryzuje się wysokim stopniem modułowości. Zastosowano ścisły podział odpowiedzialności (**Single Responsibility Principle**), co przekłada się na czytelność i łatwość konserwacji kodu. System został podzielony na logiczne warstwy:

- **Warstwa danych (Model):** Klasy Card, Hand, Deck.
- **Warstwa logiki biznesowej:** Klasy Person i jej pochodne (Player, Croupier).
- **Warstwa kontrolera:** Klasa Game, zarządzająca przepływem stanów gry.

2. Wykorzystanie mechanizmów polimorfizmu

Kluczowym elementem projektu jest zastosowanie **polimorfizmu dynamicznego**. Klasa bazowa Person definiuje interfejs za pomocą czysto wirtualnej metody playTurn(), co czyni ją klasą abstrakcyjną.

- Zastosowanie słowa kluczowego **override** w klasach pochodnych gwarantuje poprawność nadpisywania metod i zwiększa bezpieczeństwo kodu na etapie kompilacji.
- Mechanizm ten pozwolił na implementację specyficznej logiki krupiera (dobieranie do 17 punktów) bez modyfikacji głównego silnika gry.

3. Implementacja Wzorców Projektowych

W projekcie zidentyfikowano implementację wzorca **Strategia (Strategy Pattern)** w odniesieniu do ruchów gracza (MoveStrategy).

- Podejście to pozwala na hermetyzację algorytmów decyzyjnych (Hit, Stand, Double) w osobnych klasach.
- Zastosowanie tego wzorca realizuje zasadę **Open/Closed (SOLID)** – system jest otwarty na rozszerzenia (np. dodanie ruchu *Split* czy *Insurance*), ale zamknięty na modyfikacje istniejącego kodu kontrolera.

4. Zarządzanie logiką domenową (Domain Logic)

Na szczególną uwagę zasługuje algorytm obliczania wartości ręki w klasie Hand. Zaimplementowano inteligentną obsługę **Asa**, który dynamicznie zmienia swoją wartość (11 lub 1) w zależności od aktualnej sumy punktów, co zapobiega błędnemu wyliczaniu stanu **BUST**. Dodatkowo system zawiera warstwę analityczną w postaci podpowiedzi strategicznych (Basic Strategy Hints), co podnosi wartość użytkową aplikacji.

5. Bezpieczeństwo i hermetyzacja danych

- Zastosowano odpowiednie specyfikatory dostępu (private, protected), uniemożliwiając nieautoryzowany dostęp do stanu obiektów.
- Zarządzanie stanem widoczności kart krupiera (hide/show) zostało zaimplementowane wewnątrz klasy Card, co jest przykładem poprawnego ukrywania szczegółów implementacji.

Wnioski i Rekomendacja

Kod prezentuje wysoki poziom dojrzałości technologicznej. Architektura jest skalowalna, a zastosowane techniki obiektowe są zgodne z nowoczesnymi standardami programowania w języku C++.

Rekomendacja: Projekt w pełni kwalifikuje się jako wzorcowa implementacja zaliczeniowa z zakresu Programowania Obiektowego (OOP).