

Pasjans

Zadanie projektowe

Cel: Stworzenie klasycznej gry w Pasjansa. Gra wykonana musi zostać w jednym z języków C++/C#/Python i powinna w pełni działać w konsoli.

1. Opis gry

Pasjans to jednoosobowa gra karciana, w której celem jest uporządkowanie wszystkich kart według koloru i wartości w czterech stosach końcowych (od asa do króla).

Przygotowanie gry:

Talia kart

- Do gry używa się standardowej talii 52 kart (bez jokerów).
- Karty mają cztery kolory: **kier** (♥), **karo** (♦), **pik** (♠) i **trefl** (♣). Karty kier i karo powinny być wyświetlane innym kolorem czcionki niż karty pik i trefl.
- Każdy kolor zawiera karty o wartościach od **asa (A)** do **króla (K)**.

Układ początkowy

- **Kolumny gry**
 - Tworzy się **7 kolumn**, w których układ kart wygląda następująco:
 - **Pierwsza kolumna:** 1 karta (odkryta)
 - **Druga kolumna:** 2 karty (1 zakryta, 1 odkryta)
 - **Trzecia kolumna:** 3 karty (2 zakryte, 1 odkryta)
 - ...
 - **Siódma kolumna:** 7 kart (6 zakrytych, 1 odkryta)
 - Tylko **ostatnia karta w każdej kolumnie jest odkryta**.
- **Stos rezerwowy**
 - Reszta kart trafia na stos rezerwowy – można z niego dobierać karty (po przejrzeniu wszystkich kart stos powinien zostać przetasowany).
 - Należy zaimplementować 2 poziomy trudności - łatwy i trudny. Na poziomie łatwym można dobierać po 1 karcie, na poziomie trudnym

dobiera się 3 karty, ale użyć można tylko wierzchniej.

- **Stosy końcowe**

- Cztery puste miejsca, na które należy przenieść wszystkie karty według kolorów – **od asa do króla**.

Rozgrywka

Przenoszenie kart w kolumnach

- Można przesuwać karty, które są ułożone w kolejności malejącej ($K \rightarrow Q \rightarrow J \dots 2 \rightarrow A$).
- Karty muszą być układane naprzemiennie kolorami (czarna-czerwona-czarna-czerwona).
- Można przenosić pojedynczą kartę lub całą sekwencję poprawnie ułożonych kart.

Przenoszenie kart na stosy końcowe:

- Karty można przenieść na stosy końcowe tylko w kolejności od asa do króla, np.:
 - $A\spadesuit \rightarrow 2\spadesuit \rightarrow 3\spadesuit \rightarrow \dots \rightarrow K\spadesuit$
 - $A\heartsuit \rightarrow 2\heartsuit \rightarrow 3\heartsuit \rightarrow \dots \rightarrow K\heartsuit$
- Każdy stos końcowy może zawierać tylko jeden kolor.

Odkrywanie zakrytych kart:

- Gdy przeniesiemy ostatnią odkrytą kartę z kolumny, zakryta karta pod nią zostaje odkryta.
- Jeśli kolumna stanie się pusta, można w jej miejsce przenieść tylko króla (K) lub całą sekwencję kart zaczynającą się od króla.

Dobieranie kart ze stosu rezerwowego:

- Można losowo dobierać karty ze stosu:
 - Dobieranie według zasad opisanych w punkcie **Stos rezerwowy**.
 - Jeśli stos dobierania się wyczerpie, należy go przetasować i ponownie użyć.

Cofanie ruchów:

- Gra powinna umożliwiać cofnięcie 3 ostatnich ruchów gracza.

Zakończenie gry

- Wygrana:

Po ułożeniu wszystkich kart na stosach końcowych powinien pojawić się ekran wygranej. Gra powinna zliczać liczbę ruchów, w których gracz ułożył pasjansa i zapisywać ją w rankingu najlepszych wyników.

- Przegrana:
 - W trakcie gry powinna być dostępna opcja zakończenia gry i rozpoczęcia od nowa.

2. Założenia projektowe

- Należy zaimplementować grę w pasjansa według opisanych zasad.
- Projekt musi zostać wykonany w jednym z języków: C++/C#/Python.
- Projekt musi być aplikacją konsolową - musi działać w terminalu.
- Można korzystać z dowolnych bibliotek dla wybranego języka. W przypadku projektów w Pythonie wszystkie zależności muszą być instalowane za pomocą menadżera pakietów, np.pip - w dokumentacji należy umieścić [odpowiednio sformatowany plik requirements.txt](#).
- Zalecane środowiska: Programy napisane w językach C++/C# sprawdzane będą w środowiskach Visual Studio i Visual Studio Code.
- Wszystkie pliki projektu należy zapakować w plik .zip/.rar i nazwać go według szablonu: *16-18_język_imie_nazwisko*.

3. Dokumentacja

Do projektu musi zostać załączony plik README, w którym zostaną opisane:

- Sposób uruchomienia projektu, czyli lista kroków, jakie należy wykonać, aby uruchomić projekt.
- Instrukcje rozgrywki dla użytkownika (sterowanie itp.)
- Opis poszczególnych klas, modułów i metod/funkcji. Opis ten może być w formie komentarzy w kodzie lub w osobnym pliku.

4. Punktacja

- 30 pkt - spełnienie założeń projektowych (implementacja mechanik opisanych w części 1. "Opis gry")
- 30 pkt - architektura rozwiązania (organizacja kodu, dobre praktyki programistyczne)
- 20 pkt - poziom techniczny projektu (poziom użytych technologii)
- 15 pkt - kreatywność, innowacyjność i aspekt wizualny
- 5 pkt - dokumentacja, przygotowana zgodnie z opisem w części 3. Dokumentacja

5. Wskazówki

- Program działający w pełni w terminalu oznacza, że program działa od początku do końca w oknie konsoli, w którym został uruchomiony, bez otwierania żadnych dodatkowych okien.
- Twoim priorytetem powinien być poprawnie działający program. Lepiej jest przygotować mniejszy projekt, który działa bez błędów i jest dokończony, niż bardziej rozbudowany z błędami/bugami lub taki, którego nie zdążysz skończyć
- Zwróć uwagę na sposób kodowania znaków, w zależności od systemu znaki mogą wyświetlać się inaczej.
- Pamiętaj, że kod jest Twoją wizytówką. Zastanów się, kiedy warto skorzystać z bibliotek, które mają zaimplementowane pewne operacje, a kiedy możesz napisać coś samodzielnie i pochwalić się swoimi umiejętnościami programistycznymi. Dobry kod to nie tylko taki kod który działa, ale również taki który jest schludny i estetyczny. Staraj się korzystać z podejść SOLID, KISS, DRY, aby poprawić jakość kodu.