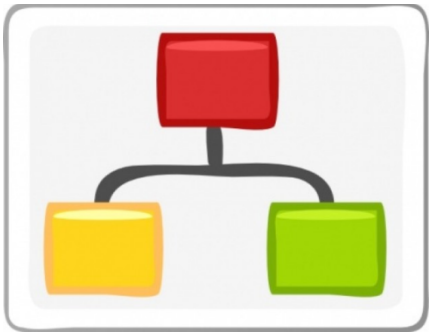

Podstawy Programowania 2020/2021

Instrukcja projektowa cz. 1

Zadanie 1 – Uogólniona gra karciana Trucizna

Autor: Tomasz Goluch

Wersja: 1.3

I. Opis gry karcianej: Trucizna

Cel: Zapoznanie z zasadami gry karcianej będącą podstawą pierwszego zadania części projektowej przedmiotu.

Zasady gry Trucizna, Reinera Knizia w polskiej wersji językowej są do ściągnięcia tutaj.

W oryginalnej gry mamy talię kart 50 kart. Każda z kart posiada kolor i wartość. W skład oryginalnej gry wchodzi 8 kart trucizny (zielone) o wartości 4 i Trzy kolory (niebieski czerwony i fioletowy) każdy składający się z 14 kart o wartościach: 1, 1, 1, 2, 2, 2, 4, 4, 5, 5, 5, 7, 7, 7. Oryginalnie gracze naprzemiennie, począwszy od rozpoczynającego umieszczają dowolną wybraną kartę z ręki na 3 kociołkach (stosach kart). Terminy kociołek i stos kart będziemy używać zamiennie.

Umieszczać w kociołkach można kartę identycznego koloru co umieszczone wcześniej. Wyjątkiem jest kolor zielony (trucizna) można ją odłożyć na dowolny stos, a jeżeli stos składa się tylko z kart trucizny to można do niego dołożyć kartę dowolnego koloru. Oczywiście na początku gry na pusty stos również można położyć dowolną kartę.

Ten z graczy który spowoduje przekroczenie sumy = 13 wartości kart w jednym z kociołków powoduje eksplozję i zabiera wszystkie karty do siebie (nie na rękę, tylko trzyma zakryte przed sobą).

Koniec rundy następuje kiedy wszyscy gracze pozbędą się kart z ręki. Następuje podliczenie kart pobranych przez każdego z graczy w wyniku eksplozji. Każda karta trucizny to 2 ujemne pkt., pozostałe warte są 1 pkt ujemny. Wyjątkiem są kolory których gracz uzbierał najwięcej (ma podczas podliczania największą liczbę kart danego koloru). Jest on uodporniony na dany kolor i takie karty warte są 0 pkt. Następnie kolejny gracz tasuje i rozdaje karty.

Całkowita rozgrywka składa się z 3 rund. Na koniec następuje podliczenie wszystkich pkt. ujemnych zdobytych podczas wszystkich rozegranych rund, ten kto ma ich najmniej wygrywa.

II. Uogólniona wersja gry karcianej: Trucizna

Cel: Zapoznanie z tematem pierwszego zadania części projektowej przedmiotu.

Tematem pierwszego zadania projektowego jest napisanie uogólnionej wersji gry Trucizna. Zakładamy, że w odróżnieniu od oryginału liczba graczy **n** może się zmieniać od 2 do **s**, gdzie **s** to suma wszystkich kart. Chcemy aby każdy z graczy otrzymał przynajmniej po jednej karcie do ręki. Zakładamy również, że nie obowiązują specjalne zasady dla 3 graczy opisane w instrukcji dotyczące grania niepełną talią kart. Taka rozgrywka dla dowolnej liczby przeciwników powinna być generowana za pomocą uniwersalnego algorytmu bez konieczności uwzględniania sytuacji specjalnych dla 3 i 2 graczy. Pozostałe elementy oryginalnej gry, które mogą ulec zmianie to:

- **k** – liczba kociołków/stosów kart, która zawsze będzie odpowiadać liczbie kolorów kart – 1. Spowodowane jest to tym, że do każdego stosu możemy dokładać dokładnie jeden kolor kart z wyłączeniem zielonego, który może trafić do każdego z nich

- **g** – liczba zielonych kart w talii, zakładamy, że zielone karty będą zawsze i będą działały jako trucizna ale ich liczba i wartość w talii mogą się różnić

- **gv** – wartość zielonych kart, będzie ona identyczna dla wszystkich kart trucizny, w przeciwieństwie do pozostałych kolorów które mogą przyjmować różne wartości

- **o** – liczba kart innego koloru niż zielony

- **ov –** zbiór wartości każdego z kolorów kart nie będących trucizną o mocy **o**. Mówi on z jakich wartości będzie składał się każdy kolor inny od zielonego. Jak widać wynika z tego konkluzja, że wszystkie kolory (z wyjątkiem zielonego) będą zawierać te same karty co do wartości i ich liczby, tak jak ma to miejsce w oryginalnej grze.

- **e –** wartość po przekroczeniu której następuje eksplozja kociołka.

- **r –** liczba rund rozgrywanych podczas całej rozgrywki.

Po ustaleniu tych parametrów widzimy, że liczba kart dostępnych w talii równa jest: s = **g** + **k** * **o**. Oryginalna wersja była by opisana parametrami: $3 \leq \mathbf{n} \leq 6$, **k** = 3 , **g** = 8, **gv** = 4, **o** = 14, **ov** = {1, 1, 1, 2, 2, 2, 4, 4, 5, 5, 5, 7, 7, 7}, **e** = 13, **r** = 3, dodając wyjątki dotyczące wariantu dla 3 graczy z instrukcji.

III. Środowisko pracy

Cel: Zapoznanie z wykorzystywanym w projekcie oprogramowaniem.

Program będzie implementowany z użyciem języka C. Można używać dowolnych kompilatorów oraz edytorów tego języka. Używane narzędzia oraz IDE mogą współpracować z różnymi systemami operacyjnymi: Windows/Linux. Aby uzyskać większą zgodność z systemem STOS, który będzie platformą automatycznie weryfikującą początkowe funkcjonalności projektu (szczegóły w pkt. Kryteria Oceny) zaleca się wykorzystanie Visual Studio w wersji 2017 lub wyższej (na stosie wykorzystywana jest wersja kompilatora z Visual Studio 2017 wersja 15.9.5). Można używać standardowych bibliotek C (stdlib.h, string.h itd.). **W programie nie należy używać biblioteki C++ stl.** Projekt można realizować z wykorzystaniem paradygmatu programowania obiektowego a do kompilacji można wykorzystać kompilator języka C++.

IV. Kryteria oceny

Cel: Zapoznanie z wymaganymi, dodatkowymi oraz bonusowymi kryteriami oceny projektu.

Cały projekt oddają państwo w formie zgłoszeń na platformie STOS (https://stos.eti.pg.gda.pl/), trzeba mieć na niej założone konto o numerze indeksu (tylko 6 cyfr BEZ LITERKI S) i być dodanym do tego przedmiotu: https://stos.eti.pg.gda.pl/index.php?p=viewprzedmiot&cid=356.

Część projektu będzie można oddać w postaci krótkich zadań automatycznie rozwiązywanych przez platformę. Końcowy kod projektu również będzie umieszczony na platformie STOS ale nie będzie on podlegał automatycznej ocenie. Automatyczne ocenianie ma za zadanie ułatwić

osobom sprawdzającym ocenę projektów. Fakt nie zaliczenia zadania przez platformę STOS nie oznacza nie zaliczenia danej części projektu, jednak podstawą osobistej oceny przez prowadzącego jest kod umieszczony na platformie. Gwarantuje to, że został napisany przed terminem jego umieszczenia oraz, że przeszedł weryfikację antyplagiatową. Student może nadal osobiście udowodnić, że jego kod spełnia wymagane funkcjonalności. Zaliczenie testów na STOS'ie daje większy komfort psychiczny i zwiększa możliwość zaliczenia danej funkcjonalności projektu w prawie 100%. Odstępstwa mogą się pojawić w przypadku udowodnienia plagiatu (co może doprowadzić do uzyskania ujemnej liczby punktów z tej części projektu), słabej znajomości umieszczonego kodu oraz błędów logicznych, stylistycznych, niskiej przejrzystości kodu itp. Takie same elementy mają również wpływ na ocenę końcową kodów ocenianych ręcznie.

Wymagania obowiązkowe (7 pkt.)

Wszystkie wymienione tutaj elementy należy zaimplementować. Brak któregokolwiek z poniższych elementów skutkuje otrzymaniem **0 pkt**. z tego projektu:

- **utworzenie talii (1pkt)** – wygenerowanie pliku zawierającego niepotasowaną talię kart do uogólnionej wersji gry. Parametry opisujące grę będą podane na wejściu. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=423&cid=356.

- **rozdanie kart (1pkt)** – wygenerowanie pliku zawierającego stan rozgrywki po rozdaniu niepotasowanej talii kart. Parametry opisujące grę będą podane na wejściu. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=194&cid=356.

- **wczytanie stanu gry cz.1 (0.5 pkt)** – wczytanie pliku reprezentującego pewien stan gry i wyświetlenie wybranych informacji które są z nim zgodne. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=833&cid=356.

- **wczytanie stanu gry cz.2 (0.5 pkt)** – kontynuacja powyższego podpunktu. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=910&cid=356

- **sprawdzenie liczby i wartości zielonych kart (0.5pkt)** – po wczytaniu stanu gry należy sprawdzić ile wczytano zielonych kart i jakie mają wartości. Jeśli nie wczytano żadnej takiej karty (nie było jej u żadnego z graczy oraz w kociołku) lub wartości przynajmniej dwóch z nich są różne należy wydrukować stosowny komunikat. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=912&cid=356

- **sprawdzenie liczby wszystkich kart (0.5pkt)** – po wczytaniu stanu gry należy sprawdzić czy liczba wszystkich wczytanych kolorów innych kart niż zielonych jest równa. Następnie należy wydrukować stosowny komunikat. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=913&cid=356

- **sprawdzenie wartości wszystkich kart (1pkt)** – po wczytaniu stanu gry należy sprawdzić czy wartości wszystkich wczytanych kolorów innych kart niż zielonych są identyczne. Następnie należy wydrukować stosowny komunikat. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=918&cid=356

- **sprawdzenie poprawności stanu gry (1pkt) –** po wczytaniu stanu gry należy sprawdzić czy liczba kart na ręce graczy jest poprawna oraz czy w żadnym ze stosów liczba kart nie przekracza maksymalnej dozwolonej i czy nie zawiera on niedozwolonej kombinacji kolorów. Następnie należy wydrukować stosowny komunikat. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=914&cid=356

- **wykonanie prostego posunięcia (0.5pkt) –** po wczytaniu stanu gry należy wykonać proste posunięcie aktywnym graczem i zapisać nowy stan gry w pliku. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=915&cid=356

- **obsługa eksplozji kociołka (0.5pkt) –** sprawdzenie po wykonaniu prostego posunięcia czy eksplodował kociołek, jeśli tak to przed zapisaniem nowego stanu gry należy przenieść z niego wszystkie karty z eksplodującego kociołka na stos kart aktywnego gracza. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=916&cid=356

Wymagania dodatkowe (8 pkt.)

Dodatkowe punkty można otrzymać za spełnienie poniższych podpunktów (nie muszą być one implementowane wszystkie i w podanej kolejności):

- **obsługa końca rundy (1pkt) –** sprawdzenie czy runda gry uległa zakończeniu, jeśli tak należy dodatkowo wyświetlić punktację jaką uzyskają gracze w tej rundzie. To wymaganie może zostać ocenione automatycznie przez STOS: https://stos.eti.pg.gda.pl/index.php?p=show&pid=917&cid=356

- **prosta rozgrywka (1pkt) –** uruchomienie 2 kopii programu w celu przeprowadzenia rozgrywki pomiędzy nimi. Programy nie muszą grać inteligentnie, wystarczy wybieranie dowolnej karty z dowolnego niepustego stosu ale każda kopia powinna grać zgodnie z zasadami, w szczególności, musi przeprowadzić weryfikację zgodności liczby i rodzaju kart, czy wartość kart w dowolnym z kociołków nie przekracza **e** – maksymalnej dozwolonej, czy liczba kart graczy na ręce się zgadza oraz czy nie nastąpił koniec rundy lub gry.

  Każda kopia powinna otrzymywać informacje o numerze gracza którego reprezentuje np. jako parametr podczas wywołania programu przekazywany do funkcji main albo wczytywany z pliku konfiguracyjnego. Programy mogą synchronizować i komunikować się w dowolny sposób, np.:

  - poprzez plik reprezentujący. aktualny stan gry. Może istnieć zewnętrzny arbiter (dodatkowy program), który przechowuje stan gry i jest odpowiedzialny za przekazanie go podczas wywołania kolejnych graczy. Bez arbitra – informacja

może być zapisana wewnątrz pliku reprezentującego aktualny stan gry a programy z pewnym interwałem odczytują plik sprawdzając czy nasępiła ich kolej.

- poprzez implementację jednego programu uruchamiającego dwie instancje gry wymieniających strukturę zawierającą stan gry. Jest to najłatwiejsza forma ponieważ nie wymaga komunikacji pomiędzy 2 procesami.

- z wykorzystaniem potoków, kolejek, usług bądź innych kanałów komunikacji zamiast plików

Ponadto podczas przygotowania rozgrywki należy pamiętać aby przed rozdaniem potasować talię kart. Ponadto karty wszystkich graczy na ręce są jawne, zatem zawsze można przeprowadzić weryfikację czy ktoś nie ma ich za dużo bądź za mało.

- **wybór najmniejszej karty** (**1pkt**) – implementacja gracza wybierającego jako posunięcie kartę o najmniejszej wartości i odkładającego ją na stos o najmniejszej sumie wartości odłożonych kart.

- **pozbywanie się największych kart** (**1pkt**) **–** w przypadku kiedy nasze posunięcie z poprzedniego pkt. prowadzi do eksplozji należy pozbyć się największej karty zapełniając najbardziej pusty kociołek. Pozwoli to na pozbycie się niebezpiecznej karty i zwiększa prawdopodobieństwo pobrania małej liczby kart.

- **wybór optymalnej karty** (**1pkt**) **–** implementacja gracza dorzucającego jak największą kartę ale taką która nie spowoduje eksplozji, jeśli to niemożliwe dorzuca największą kartę jak w wymaganiu **pozbywanie się największych kart**.

- **maksymalizacja koloru** (**1pkt**) – w przypadku kiedy gracz musi dokonać eksplozji dokładając kartę dokonuje maksymalizacji jednego z kolorów. Robi to w ten sposób, że sprawdza jaki będzie stan kart leżących przed nim w przypadku wszystkich możliwych eksplozji. Porównuje te stany i wybiera ten który spowoduje, maksymalizację kart dowolnego konkretnego koloru poza zielonym (liczba kart jednego z możliwych kolorów poza zielonym leżąca przed nim będzie największa).

- **złożona rozgrywka (1pkt)** – uruchomienie rozgrywki dla dużej liczby osób (od 6 do 20 graczy).

- **testy skuteczności graczy (1 pkt)** – przeprowadzenie zautomatyzowanych testów porównujących przynajmniej dwa programy o znacznie różniącej się strategii (kilkaset rozgrywek). Przykładowo: porównanie liczby wygranych programu grającego losowo z programem próbującym unikać eksplozji, umieszczającym największą kartę która jeszcze nie spowoduje przekroczenia maksymalnej liczby punktów na stosie/kociołku.

Wymaganie bonusowe (**3 pkt.**)

- **Uczestnictwo w turnieju** (**1pkt**) **–** uczestnictwo w turnieju, co oznacza implementację programu który poprawnie rozegra wszystkie wymagane partie z wszystkimi zgłoszonymi przeciwnikami. Nie musi wygrać z żadnym z nich ale nie może się zawiesić albo grać błędnie.

- **Zwycięstwo w turnieju (0.5-2pkt) –** uzyskanie odpowiednio wysokiego miejsca. Szczegółowe informacje odnośnie progów gwarantujących określoną liczbę pkt. będą dostępne dopiero po przeprowadzeniu turnieju, ponieważ zależą od liczby zgłoszonych kandydatów, liczby zdyskwalifikowanych programów oraz ogólnego poziomu zaimplementowanych w nich algorytmów.

Początkowe 11 zadań odpowiadających kolejnym wymaganiom funkcjonalnym projektu może być pisane w formie krótkich osobnych programów lub w formie większego programu który można „dostosowywać do wymagań STOS'u" nie ma to znaczenia na ich ocenę. Ważne aby w podpunkcie **prosta rozgrywka** złożyć te wszystkie funkcjonalności w jeden grający program. Taki program powinien:

1. wiedzieć którym jest graczem,
2. potrafić wczytać aktualny stan gry,
3. w przypadku kiedy posunięcie należy do niego wykonać ruch zgodnie z zasadami,
4. w przypadku wykrycia niezgodności stanu gry wyświetlić błąd,
5. poinformować o wyniku jeśli wczytany stan lub stan po wykonaniu posunięcia jest końcowy.

Ponadto w podpunkcie **prosta rozgrywka** należy uruchomić przynajmniej 2 kopie programu i przeprowadzić rozgrywkę od początku do końca, czyli:

1. Należy w zależności od parametrów **n, k, g, gv, o, ov, e** należy wygenerować stan początkowy gry, należy potasować i rozdać karty.
2. Przeprowadzić sekwencję poprawnych posunięć prowadzących do osiągnięcia końcowego stanu rundy.
3. Wyświetlić aktualną punktację graczy/programów
4. Powtórzyć **r** razy powyższe punkty.

V. Turniej

Cel: Zapoznanie z realizacją odnośnie turnieju.

Oddawanie zadań na STOS'ie ma jeszcze jedno zadanie, pozwala ono na ujednolicenie formatu pliku pozwalającego na zapisanie stanu rozgrywki pomiędzy posunięciami kolejnych graczy. Pozwoli to na przeprowadzenie rozgrywek pomiędzy różnymi wersjami własnych programów ale przede wszystkim pomiędzy programami różnych autorów. W finale powinno to pozwolić na rozstrzygnięcie konkursu na najlepszego gracza. Turniej o którym mowa będzie polegał na przeprowadzeniu serii rozgrywek pomiędzy dwoma bądź większą liczbą programów grających jednocześnie. Jego dokładna forma nie jest jeszcze określona i jest dosyć trudna do opisania w zupełności w tej instrukcji. To czy będzie to pojedynek typu

każdy z każdym albo czy turniej odbędzie się etapami gdzie zwycięzcy będą przechodzić do kolejnych etapów, jak i to gdzie zostanie przeprowadzony i przez kogo (czy prowadzący będą uruchamiać programy, czy zrobimy wspólne spotkanie w sali laboratoryjnej gdzie turniej zostanie przeprowadzony, czy może studenci będą wymieniali się binarnymi wersjami programu i uzgadniać kto z nich wygrał) to wszystko po uściśleniu zostanie opisane na STOS'ie w treści przedmiotu: https://stos.eti.pg.gda.pl/index.php?p=viewprzedmiot&cid=356 w zakładce TURNIEJ. Dodatkową niewiadomą jest rodzaj rozgrywki (czy karty przeciwników są tajne, jawne albo obie te wersje) oraz parametry opisujące konkretne rozgrywki jakie będą przeprowadzone, zatem programy należy tak pisać aby były przygotowane na różną liczbę graczy, kociołków/kolorów kart, liczby i wartości kart, stałej eksplozji itp. Raczej pewne jest to, że komunikacja pomiędzy programami będzie się odbywać z wykorzystaniem pliku o ściśle określonym formacie a środowiskiem uruchomieniowym będzie Windows 10. Oznacza to, że preferowane będą programy skompilowane przez kompilator Microsoft'u *cl.exe* z wykorzystaniem i *msbuild.exe* z Visual Studio.