Politechnika Śląska w Gliwicach
Wydział Automatyki, Elektroniki i Informatyki


# Programowanie Komputerów 2

## Gra Klocki

| | |
|---|---|
| autor | **Dorian Barański** |
| prowadzący | **mgr inż. Maciej Długosz** |
| rok akademicki | **2018/2019** |
| kierunek | **informatyka** |
| rodzaj studiów | **SSI** |
| semestr | **4** |
| termin laboratorium / ćwiczeń | **wtorek, 12:00 – 13:30** |
| grupa | **6** |
| data oddania sprawozdania | **18.06.2019** |

# 1 Temat

Napisać program wzorowany na grze „Tetris”. Przesuwające się w dół elementy o różnych kształtach powinny być układane przez użytkownika w taki sposób, aby tworzyły ciągłe wiersze, które automatycznie znikają zwiększając pulę punktową użytkownika. Gra powinna mieć funkcję tworzenia historii wyników, która będzie przechowywana w pliku binarnym.

## 2.1 Algorytmy

Gra jest wyświetlana dzięki cyklicznemu czyszczeniu konsoli i ponownemu wypisywaniu tablicy (która zawiera planszę) w momencie gdy nastąpiły na niej jakieś zmiany. Obrót klocków zrealizowany został za pomocą wzorów na macierzach. Punktem obrotu jest wybrany segment klocka. Ruch klocków przebiega następująco:

- najpierw występuje inkrementacja lub dekrementacja współrzędnych klocka
- następnie sprawdzane są wystąpienia ewentualnych kolizji, jeżeli zostanie wykryta, wówczas klocek wraca na swoje miejsce, jeżeli nie, klocek znajduje się na nowej pozycji.

Opóźnienia generowane są poprzez porównywanie liczb zwracanych przez funkcje biblioteczne.

## 2.2 Struktury danych

Struktura danych wykorzystana w programie to lista jednokierunkowa. Za jej pomocą przechowywane są wyniki wczytane wcześniej z pliku binarnego. Elementy listy zawierają nazwę gracza oraz ilość zdobytych przez niego punktów.

## 2.3 Analiza problemu

Zadanie przedstawia problem implementacji prostej gry wzorowanej na grze „Tetris”.

# 3 Specyfikacja zewnętrzna

Program uruchamiany jest z wiersza poleceń poprzez przekazanie nazwy programu oraz nazwy pliku przechowującego wyniki po przełączniku –i. Przykładowe wywołanie programu:

```
TETRIS.exe –i wyniki.bin
```

**Obsługa programu:**
Po uruchomieniu programu użytkownik rozpoczyna od wpisania swojej nazwy. Następnie kliknięcie przycisku S powoduje rozpoczęcie rozgrywki. Przycisk P odpowiada za wstrzymanie gry, a ponowne użycie tego klawisza wznawia ją. Po zakończeniu rozgrywki klawisz R powoduje zrestartowanie gry, natomiast przycisk E wyłącza program.

**Poruszanie:**
Odbywa się głównie za pomocą strzałek.
- ruch w boki – strzałki boczne
- ruch w dół ( przyspieszenie opadania klocka ) – strzałka w dół
- obrót klocka – strzałka w górę

# 4 Specyfikacja wewnętrzna

Program został zrealizowany zgodnie z paradygmatem strukturalnym. W programie rozdzielono interfejs od logiki aplikacji.

## 4.1 Typy zdefiniowane w programie

W programie zdefiniowano następujący typy:

```
struct zapis {
int wynik;
char imie[10];
};
```

Typ ten przechowuje pojedynczy rekord(wynik wraz z nazwą).

```
struct element {
 struct element* nastepny;
 struct zapis zapis;
};
```

Typ ten służy do zbudowania listy wyników.

```
struct klocek {
 int x1, y1;
 int x2, y2;
 int x3, y3;
 int x4, y4;
};
```

Typ ten przechowuje informacje o współrzędnych klocka.

## 4.2 Ogólna struktura programu

Na początku wywołana zostaje funkcja:

```
int czytajArgumentyWierszaPolecen(int argc, char** argv, char** nazwaPlikuWejsciowego);
```

Funkcja wczytuje nazwę pliku podaną po przełączniku -i. Gdy uda się wczytać nazwę pliku wywołana zostaje funkcja:

```
void wczytaj(struct zapis najlepsze[10],char* nazwaPlikuWejsciowego);
```

Odpowiada ona za wczytanie wyników z pliku oraz zapisanie do pamięci 10 najlepszych. To tutaj program alokuje a następnie dealokuje dynamiczną listę. Następnie wywołana zostaje funkcja:

```
void poczatekGry(char plansza[wysokoscPlanszy][szerokoscPlanszy], struct zapis najlepsze[10], HANDLE* hOut, struct zapis* test);
```

Przygotowuje ona planszę oraz zmienne do gry. Następnie cyklicznie wywoływany jest szereg funkcji odpowiedzialnych za grę, komunikacje z użytkownikiem oraz wyświetlanie planszy.

```
void usunWiersz(char plansza[wysokoscPlanszy][szerokoscPlanszy], int* wynik, struct zapis najlepsze[10], int nastepny);
```

Zajmuje się poszukiwaniem pełnych wierszy na planszy. Usuwa je oraz sprawia że pozostałe klocki „spadają”. Dodatkowo to ona zajmuje się dodawaniem punktów do wyniku gracza.

```
int nowyKlocek(char plansza[wysokoscPlanszy][szerokoscPlanszy], int tablica[7][8], struct klocek* obecny, int losowa, struct zapis najlepsze[10], HANDLE* hOut, struct zapis* test, int wynik, char* nazwaPlikuWejsciowego);
```

Funkcja jest odpowiedzialna za dodanie nowego klocka na planszę. Wystąpienie kolizji jest sygnałem do zakończenia gry więc w takim wypadku wywoływana jest funkcja:

```
int    KoniecGry(char    plansza[wysokoscPlanszy][szerokoscPlanszy],
struct zapis najlepsze[10], HANDLE* hOut, struct zapis* test, struct
klocek* obecny, int wynik, char* nazwaPlikuWejsciowego);
```

Wyświetla ekran końcowy, czeka na decyzję użytkownika. Wywołuję funkcję odpowiedzialną za zapisanie wyniku do pliku.

```
void zapisz(struct zapis* rekord, char* nazwaPlikuWejsciowego);
```

Funkcja zapisuje wynik gracza do pliku. Czeka na decyzję gracza czy gra ma rozpocząć od nowa czy też ma się wyłączyć.

```
void wypisaniePlanszy(char plansza[wysokoscPlanszy][szerokoscPlanszy], int
wynik, int nastepny, struct zapis najlepsze[10]);
```

Funkcja odpowiedzialna za wypisanie planszy w oknie konsoli. Wyświetla ona również następny klocek, 10 najlepszych wyników wczytanych z pliku oraz obecny wynik gracza. Przez większość czasu za komunikacje gracza z grą odpowiada funkcja:

```
void reakcjaNaKlawisze(char plansza[wysokoscPlanszy][szerokoscPlanszy],
int *zmiana, struct klocek* obecny, int* nowy, int obrot);
```

W zależności od wciśniętego przez gracza przycisku może ona wywołać różne funkcje:

```
void pauza();
```

```
void obroc(char plansza[wysokoscPlanszy][szerokoscPlanszy], int *zmiana,
struct klocek* obecny);
```

```
void przesunWBok(char plansza[wysokoscPlanszy][szerokoscPlanszy], int
*zmiana, struct klocek* obecny, int n);
```

```
void przesunWDol(char plansza[wysokoscPlanszy][szerokoscPlanszy], int
*zmiana, struct klocek* obecny, int*nowy);
```

Warto zwrócić uwagę że ostatnia z tych funkcji jest też w programie wywoływana cyklicznie wraz z upływem czasu.

## 4.3 Szczegółowy opis implementacji funkcji

```
void zapisz(struct zapis* rekord, char* nazwaPlikuWejsciowego);
```

Funkcja dopisuje wynik i nazwę gracza na koniec pliku binarnego przechowującego wszystkie wyniki.

**Parametry funkcji:**

| | |
|---|---|
| *rekord* | wskaźnik na strukturę przechowującą imię oraz wynik gracza we właśnie zakończonej rozgrywce |
| *nazwaPlikuWejsciowego* | nazwa pliku przechowującego wyniki |

```
void wczytaj(struct zapis najlepsze[10],char* nazwaPlikuWejsciowego);
```

Funkcja wczytuje wszystkie wyniki z pliku do pamięci, następnie zachowuje 10 najlepszych wyników w statycznej tablicy a zaalokowaną wcześniej pamięć zwalnia.

**Parametry funkcji:**

| | |
|---|---|
| *najlepsze* | statyczna 10 elementowa tablica przechowująca 10 najlepszych wyników wczytanych z pliku |
| *nazwaPlikuWejsciowego* | nazwa pliku przechowującego wyniki |

```
void przesunKursor(int x, int y);
```

Funkcja przesuwa kursor w konsoli na podane współrzędne.

**Parametry funkcji:**

| | |
|---|---|
| x | współrzędna pionowa (licząc od góry) |
| y | współrzędna pozioma (licząc od lewej) |

```
void wypisaniePlanszy(char plansza[wysokoscPlanszy][szerokoscPlanszy], int wynik, int nastepny, struct zapis najlepsze[10]);
```

Funkcja czyści konsolę a następnie wypisuje cała planszę wraz z informacjami o wyniku, następnym klocku oraz tabele 10 najlepszych wyników.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *wynik* | zmienna przechowuje informacje o dotychczasowo uzyskanym przez gracza wyniku |
| *nastepny* | zmienna przechowuje informacje jaki rodzaj klocka pojawi się następnie na planszy |
| *najlepsze* | statyczna 10 elementowa tablica przechowująca 10 najlepszych wyników wczytanych z pliku |

```
void przesunWBok(char plansza[wysokoscPlanszy][szerokoscPlanszy], int *zmiana, struct klocek* obecny, int n);
```

Funkcja przesuwa klocek o jedną kratkę w bok. W wypadku wykrycia kolizji klocek wraca na początkową pozycję.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *zmiana* | zmienna przechowująca informacje o tym czy na planszy nastąpiła jakaś zmiana |
| *obecny* | wskaźnik na strukturę przechowującą obecne współrzędne klocka |
| *n* | zmienna definiuje czy przesunięcie występuje w prawo(1) czy w lewo(-1) |

```
void przesunWDol(char plansza[wysokoscPlanszy][szerokoscPlanszy], int *zmiana, struct klocek* obecny, int*nowy);
```

Funkcja przesuwa klocek o jedną kratkę w dół. W wypadku wykrycia kolizji klocek wraca na początkową pozycję, zostaje unieruchomiony a funkcja zapisuje informacje o tym że powinien pojawić się nowy klocek.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *zmiana* | zmienna przechowująca informacje o tym czy na planszy nastąpiła jakaś zmiana |
| *nowy* | zmienna przechowująca informacje o tym czy na planszy powinien pojawić się nowy klocek |
| *obecny* | wskaźnik na strukturę przechowującą obecne współrzędne klocka |

```
void obroc(char plansza[wysokoscPlanszy][szerokoscPlanszy], int *zmiana,
struct klocek* obecny);
```

Funkcja obraca klocek o 90 stopni zgodnie z ruchem wskazówek zegara. W wypadku wykrycia kolizji klocek wraca na początkową pozycję.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *zmiana* | zmienna przechowująca informacje o tym czy na planszy nastąpiła jakaś zmiana |
| *obecny* | wskaźnik na strukturę przechowującą obecne współrzędne klocka |

```
void reakcjaNaKlawisze(char plansza[wysokoscPlanszy][szerokoscPlanszy],
int *zmiana, struct klocek* obecny, int* nowy, int obrot);
```

Funkcja, w zależności od wciśniętego przez użytkownika klawisza, wywołuje żądane funkcje.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *zmiana* | zmienna przechowująca informacje o tym czy na planszy nastąpiła jakaś zmiana |
| *obecny* | wskaźnik na strukturę przechowującą obecne współrzędne klocka |
| *nowy* | zmienna przechowująca informacje o tym czy na planszy powinien pojawić się nowy klocek |
| *obrot* | zmienna przechowująca informacje czy dany klocek się obraca |

```
void usunWiersz(char plansza[wysokoscPlanszy][szerokoscPlanszy], int*
wynik, struct zapis najlepsze[10], int nastepny);
```

Funkcja sprawdza czy na planszy znajdują się całkowicie wypełnione wiersze. Gdy je znajdzie usuwa je a następnie obniża wszystkie klocki znajdujące się powyżej o jeden w dół. W zależności od ilości wyczyszczanych wierszy dodaje punkty do wyniku.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *wynik* | zmienna przechowuje informacje o dotychczasowo uzyskanym przez gracza wyniku |
| *najlepsze* | statyczna 10 elementową tablica przechowująca 10 najlepszych wyników wczytanych z pliku |
| *nastepny* | zmienna przechowuje informacje jaki rodzaj klocka pojawi się następnie na planszy |

```
void poczatekGry(char plansza[wysokoscPlanszy][szerokoscPlanszy], struct
zapis najlepsze[10], HANDLE* hOut, struct zapis* test);
```

Funkcja ustawia planszę oraz zmienne do startowych wartości.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *najlepsze* | statyczna 10 elementowa tablica przechowująca 10 najlepszych wyników wczytanych z pliku |
| *hOut* | uchwyt na wyjście konsoli |
| *test* | wskaźnik na strukturę przechowującą imię oraz wynik gracza |

```
int nowyKlocek(char plansza[wysokoscPlanszy][szerokoscPlanszy], int
tablica[7][8], struct klocek* obecny, int losowa, struct zapis
najlepsze[10], HANDLE* hOut, struct zapis* test, int wynik, char*
nazwaPlikuWejsciowego);
```

Funkcja jest odpowiedzialna za pojawienie sie na planszy nowego klocka.
W wypadku wystąpienia kolizji wywoływana jest funkcja KoniecGry.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *tablica* | statyczna dwuwymiarowa tablica przechowująca początkowe współrzędne klocków |
| *obecny* | wskaźnik na strukturę przechowującą obecne współrzędne klocka |
| *losowa* | zmienna przechowuje informacje o typie klocka który powinien pojawić się na planszy |
| *najlepsze* | statyczna 10 elementowa tablica przechowująca 10 najlepszych wyników wczytanych z pliku |
| *hOut* | uchwyt na wyjście konsoli |
| *test* | wskaźnik na strukturę przechowującą imię oraz wynik gracza |
| *wynik* | zmienna przechowuje informacje o dotychczasowo uzyskanym przez gracza wyniku |
| nazwaPlikuWejsciowego | nazwa pliku przechowującego wyniki |

**Wartość zwracana:**

0 użytkownik zdecydował się na wyłączenie gry
1 użytkownik zdecydował się na rozpoczęcie gry od nowa
2 gra jeszcze się nie zakończyła

```
int czytajArgumentyWierszaPolecen(int argc, char** argv, char**
nazwaPlikuWejsciowego);
```

Funkcja odczytuje argumenty podane przez użytkownika w wierszu poleceń. Wczytuje nazwę pliku podaną po przełączniku -i.

**Parametry funkcji:**

| | |
|---|---|
| *argc* | ilość argumentów wczytanych z wiersza poleceń |
| *argv* | tablica przechowująca argumenty |
| *nazwaPlikuWejsciowego* | nazwa pliku przechowującego wyniki |

```
int KoniecGry(char plansza[wysokoscPlanszy][szerokoscPlanszy], struct
zapis najlepsze[10], HANDLE* hOut, struct zapis* test, struct klocek*
obecny, int wynik, char* nazwaPlikuWejsciowego);
```

Funkcja wyświetla ekran końcowy, czeka na decyzje użytkownika. Wywołuje również funkcje odpowiedzialna za zapisanie wyniku do pliku.

**Parametry funkcji:**

| | |
|---|---|
| *plansza* | statyczna dwuwymiarowa tablica przechowująca planszę gry |
| *tablica* | statyczna dwuwymiarowa tablica przechowująca początkowe współrzędne klocków |
| *obecny* | wskaźnik na strukturę przechowującą obecne współrzędne klocka |
| *losowa* | zmienna przechowuje informacje o typie klocka który powinien pojawić się na planszy |
| *najlepsze* | statyczna 10 elementowa tablica przechowująca 10 najlepszych wyników wczytanych z pliku |
| *hOut* | uchwyt na wyjście konsoli |
| *test* | wskaźnik na strukture przechowującą imię oraz wynik gracza |
| *wynik* | zmienna przechowuje informacje o dotychczasowo uzyskanym przez gracza wyniku |
| *nazwaPlikuWejsciowego* | nazwa pliku przechowującego wyniki |

**Wartość zwracana:**

| | |
|---|---|
| 0 | użytkownik zdecydował się na wyłączenie gry |
| 1 | użytkownik zdecydował się na rozpoczęcie gry od nowa |

```
void pauza();
```

Funkcja zachowuje sterowanie tak długo aż użytkownik nie wciśnie klawisza odpowiedzialnego za wyłączenie pauzy.

**Wartość zwracana:**

| | |
|---|---|
| 0 | udało się wczytać nazwę pliku wejściowego |
| 1 | nazwy pliku nie udało się wczytać |

# 5 Testowanie

Program został przetestowany na systemie Windows. Sprawdzony pod kątem wycieków pamięci. Miganie konsoli spowodowane jej nienadążającym wyjściem udało się zminimalizować do minimum. Początkowo wysokie zużycie procesora zmalało niemal 10-krotne dzięki dodaniu krótkiej instrukcji zmniejszającej częstotliwość wykonywania głównej pętli (zachowując niemal taką samą płynność rozgrywki).

# 6 Wnioski

Gra Tetris opiera się na kilku podstawowych algorytmach odpowiadających między innymi za ruch klocka i jego obrót, co zostało zrealizowane za pomocą prostych obliczeń na macierzach. Wszystko złączone w jedną całość prezentuje się bardzo dobrze i nie sprawiało większych problemów. Jedyny, na który się napotkałem to wolna praca wyjścia konsoli podczas odświeżania obrazu, co skutkuje jej miganiem i nie jest to przyjemne dla oka. Jednym ze sposobów eliminacji mogłoby być ograniczenie częstotliwości odświeżania do minimum, jednak nie udało mi się tego dokonać w stu procentach.