Zabrze, 20 czerwca 2018

Laboratorium
Programowania Komputerów

Temat:
Szachy

Autor: Mateusz Chłopek
Informatyka, sem. 2, gr. 2, sekcja 2
Prowadzący: dr Adam Gudyś

1. Analiza i projektowanie

1.1Algorytmy, struktury danych

- Struktury danych

Odpowiednim wyborem dotyczącym struktur danych jest wykorzystanie tablicy wskaźników na strukturę. Ma ona określony z góry rozmiar (8 na 8 – tak jak szachownica). Zastosowanie tablicy wskaźników porządkuje dane i ułatwia tworzenie algorytmów dotyczących działań na szachownicy, konkretnych figurach szachowych itp.

- Algorytmy

Jako, że w szachach istnieją odrębne zasady (np. poruszania się) dla każdej z figur, odpowiednie funkcje będą polegać w większości na instrukcjach warunkowych (*if, if else*).

2. Specyfikacja zewnętrzna

2.1 Obsługa programu

Gra w szachy rozpoczyna się od momentu uruchomienia programu. W celu poruszenia się konkretną figurą należy kliknąć na nią lewym przyciskiem myszy (pole na którym się znajduje zostaje zaznaczone na żółto) i następnie kliknąć na pole docelowe. W przypadku, gdy niechcący została wybrana inna figura lub użytkownik zmienił decyzję co do jej wyboru, można kliknąć ją drugi raz albo wybrać inne pole, które nie jest możliwym celem przejścia wybranej wcześniej figury. Gra kończy się w momencie, gdy jeden z króli jest matowany. W takiej sytuacji wyświetlani jest komunikat o zakończonej rozgrywce i użytkownik jest pytany czy chce zagrać ponownie. Po wybraniu odpowiedzi, program przywraca początkowy stan szachownicy albo kończy działanie. W celu wcześniejszego wyjścia z programy należy kliknąć lewym przyciskiem myszy na napis „Wyjście” lub nacisnąć przycisk Esc.

2.2 Komunikaty

Przed rozpoczęciem działania programu, może zostać wyświetlony komunikat o braku pewnego pliku. Należy się upewnić, że w folderze z plikiem „Szachy.exe” znajdują się wszystkie potrzebne pliki, o których mowa w komunikatach.

3. Specyfikacja wewnętrzna

3.1 Zmienne

Szachownica[8][8]	Tablica wskaźników na strukturę *pionek*. Przechowuje wskaźniki na każde pole szachownicy.
piBi	Przykładowa zmienna (tutaj dotyczy **bi**ałego **pi**onka) przechowująca wskaźnik typu *ALLEGRO_BITMAP* – abstrakcyjny typ reprezentujący bitmapę.
wspProp	Zmienna typu *const float*. Wykorzystywana np. przy określaniu odpowiednich wartości współrzędnych pól szachownicy, rozmiaru

	czcionki – w celu prawidłowego wyświetlania się na różnych ekranach.
zdarzenie	Zmienna typu *ALLEGRO_EVENT*. Przy jej pomocy program otrzymuje informację kiedy i jak użytkownik dokonał interakcji – za pomocą myszy, klawiatury.
tabWym[8][8][2]	Tablica int'ów. Przechowuje współrzędne x oraz y (wyrażone w pikselach) dla każdego pola szachownicy – np. dla pola 4x3: x = tabWym[3][2][0], y = tabWym[3][2][1]

3.2 Struktury

`struct pionek {`
`  int  typ, kolor, x, y;`
`  int atak[2];`
`  ALLEGRO_BITMAP* obraz;`
`  };`
	Struktura *pionek* przechowuje informację na temat danego pola. Określa się go poprzez: typ figury (pionek, wieża, król), kolor (biały, czarny, pusty), współrzędne (wyrażone w pikselach), czy jest atakowany przez figury. Do każdego pola z figurą przypisana zostaje także odpowiednia bitmapa.

3.3 Funkcje

```
void rysujTekst(ALLEGRO_FONT* czcionkaMala, ALLEGRO_FONT* czcionkaDuza,
ALLEGRO_COLOR zolty, char* czas1, char* czas2, int szer, int wys);
```

Funkcja wyświetla na ekranie napisy (o czcionkach **czcionkaMala** i **czcionkaDuza** oraz o kolorze **zolty**) – „Szachy", „Wyjście" oraz czas rozgrywania rund graczy (**czas1** i **czas2**).

```
void rysujPola(struct pionek* Szachownica[8][8], ALLEGRO_BITMAP* szachow,
ALLEGRO_COLOR zolty, float Wsp, int szer, int wys, int rysObr, float x, float y, int
wymPole);
```

Funkcja wyświetla na ekranie szachownicę, figury oraz w odpowiednich sytuacjach zaznacza wybrane pole (**rysObr**). Bitmapy są skalowane (**Wsp** = **wspProp**) i wyświetlane z odpowiednimi dla danego ekranu współrzędnymi. **Szer** oraz **wys** – wyiary ekranu, **wymPole** – wymiary pola szachownicy, **x**, **y** – wsp. zaznaczonego pola.

```
void rysujSzach(int czySzach1, int czySzach2, int szer, int wys, float Wsp,
ALLEGRO_BITMAP* szachB, ALLEGRO_BITMAP* szachC, ALLEGRO_BITMAP* szachBObr,
ALLEGRO_BITMAP* szachCObr);
```

Funkcja wyświetla napis „SZACH" w sytuacji gdy jeden z króli jest szachowany. **CzySzach1**, **czySzach2** – sprawdzany warunek.

```
void rysujMat(int czyMat1, int czyMat2, int szer, int wys, float Wsp, ALLEGRO_BITMAP*
mat, ALLEGRO_BITMAP* matObr, ALLEGRO_BITMAP* matC, ALLEGRO_BITMAP* matCObr);
```

Działanie analogiczne do funkcji *rysujSzach*, tyle, że zajmuje się przypadkiem matowania.

```
int ktorePole(int x, int y, int szer, int wys, int wymPole);
```

Funkcja określa na które pole kliknął użytkownik porównując współrzędne kliknięcia z współrzędnymi pól szachownicy.
Wynik zwracany jest w postaci liczby dwucyfrowej, gdzie: cyfra dziesiątek – wsp. x, cyfra jedności – wsp. y.

```
void zaznaczAtakowane(struct pionek* Szachownica[8][8], int pionek, int krol);
```

Funkcja zaznacza, które pola są atakowane przez figury białe i czarne. Służy do tego tablica w strukturze *pionek* (*atak[]*). Zmienna **krol** określa czy funkcja ma sprawdzać pola atakowane przez króla a zmienna **pionek** określa czy funkcja ma zaznaczać pola wg. przemieszczania się pionka czy wg. atakowania.
Funkcja zaznacza pola atakowane wzdłuż potencjalnej drogi przemieszczania się figur do momentu napotkania innej figury różnej od króla – ma to zapewnić, że król nie będzie zasłaniał pola znajdującego się za nim. W przeciwnym wypadku, program nie wykryłby sytuacji matowania, gdyż wg. niego król mógłby mieć jeszcze jedno bezpieczne pole do wykorzystania.

```
int czyPoprawny(int x1, int x2, int y1, int y2, struct pionek* Szachownica[8][8], int kolor, int czyRuch[3][2]);
```

Funkcja sprawdza czy ruch jaki chce wykonać użytkownik jest poprawny, czyli zgodny z zasadami poruszania się dla każdej z figur. Ważne jest, aby figury (poza konikiem) nie mogły przeskoczyć innych figur, zbić króla, ani innej figury tego samego koloru. Dodatkowo sprawdzana jest również sytuacja roszady.
Wynik: 1 - poprawny, 0 – niepoprawny.

```
void zamienZawartoscPola(int x1, int x2, int y1, int y2, struct pionek* Szachownica[8][8], int zbite[2]);
```

Funkcja zamienia zawartości elementów tablicy wskaźników po przemieszczeniu figury. Pole na które przemieszczona zostaje figura otrzymuje nowy typ, kolor oraz obraz (bitmapę). Obsługiwana jest także sytuacja roszady, gdzie to przemieszczane są naraz dwie figury.

```
void ktoAtakuje(struct pionek* Szachownica[8][8], int kolor, int WspAtak[2][2])
```

Funkcja sprawdza która figura przeciwnika szachuje króla (o kolorze **kolor)**. Współrzędne tej figury oraz króla zapisywane są w tablicy **WspAtak**. Poszukiwanie zaczynane jest od króla i następnie funkcja przechodzi po kolejnych polach (na ukos, w pionie, poziomie oraz na pola potencjalnego atakującego konika) do momentu natrafienia na figurę przeciwnika. Sytuacją szczególną jest natrafienie na pionka, który to może atakować tylko „do przodu" i tylko o jedno pole, czyli zbyt oddalony pionek nie może być figurą atakującą.

```
int sprSzach(struct pionek* Szachownica[8][8], int kolor);
```

Funkcja sprawdza czy król (o kolorze **kolor**) jest szachowany.
Wynik: 1- tak, 0 – nie.

```
int sprOtocz(struct pionek* Szachownica[8][8], int kolor);
```

Funkcja sprawdza czy pola wokół króla są atakowane przez figury przeciwnika – sprawdzane przy matowaniu.
Wynik: 1 – tak, 0 – nie.

```
int czyRatunek(struct pionek* Szachownica[8][8], int WspAtak[2][2], int kolor);
```

Funkcja określa czy szachowany król (o kolorze **kolor**), potencjalnie matowany, może zostać „uratowany" przez inną figurę – poprzez „zasłonięcie" go lub zbicie figury szachującej. W tablicy **WspAtak** przechowywane są współrzędne szachowanego króla i figury szachującej. Określane jest to na podstawie tablicy *atak[]* przechowywanej w strukturze *pionek*. Jeśli pomiędzy królem a figurą szachującą (i wraz z nią) jest przynajmniej jedno pole atakowane przez figurę o kolorze **kolorb**, a więc pewna figura może przemieścić się na to miejsce, to „ratunek" jest możliwy.
Wynik: 1 – możliwy „ratunek", 0 – niemożliwy.

```
void poczRuch(struct pionek* Szachownica[8][8], int czyRuch[3][2]);
```

Funkcja sprawdza czy użytkownik wykonał chociaż jeden ruch lewą wieżą, królem lub prawą wieżą.
Wynik zapisywany jest w tablicy czyRuch: 1 – wykonał, 0 – nie wykonał. Sprawdzane jest to w przypadku roszady.

```
void przywrFig(struct pionek* Szachownica[8][8], int zbite[2], int x, int y, ALLEGRO_BITMAP* tab[12]);
```

Funkcja przywraca zbitą figurę. Wymagane przy niepoprawnym ruchu wykonanym przez gracza (poprawny ruch dla danej figury, lecz w skutek tego własny król był szachowany).

`int wylaczKryjace(struct pionek* Szachownica[8][8], int WspAtak[2][2], int kolor, int ukryte[2][8]);`
Funckja „wyłącza” figury zasłaniające króla (o kolorze **kolor**). Rozpoczyna się to od pozycji króla i następnie funkcja przechodzi po kolejnych polach (na ukos, w pionie i poziomie). Gdy natrafi na figurę o tym samym kolorze, wywołuje funkcję *dodajUkryte()*. W funkcji, sprawdzana jest od razu sytuacja „ratowania” króla za pomocą funkcji *czyRatunek()*. Funkcja zwraca wartość 1, gdy po „ukryciu” figur, możliwy jest „ratunek” króla. Zwraca 0, gdy nie jest to możliwe.

`void dodajUkryte(struct pionek* Szachownica[8][8], int i, int j, int ukryte[2][8], int licznik);`
Funkcja zapisuje w tablicy **ukryte[][]** wsp. figury zasłaniającej króla. Następnie zamienia jej kolor na wartość -1. W ten sposób, inne funkcje będą widziały te pole jako puste.

`void pokazUkryte(struct pionek* Szachownica[8][8], int ukryte[2][8], int licznik, int kolor);`
Funkcja ta „odkrywa” figury ukryte przez funckję *dodajUkryte()*.

`void initTabWym(int tabWym[8][8][2], int szer, int wys, int wymPole);`
Funkcja inicjalizuje tablicę tabWym odpowiednimi współrzędnymi (wyrażonymi w pikselach) dla danych pól.

`void initSzachWym(struct pionek* Szachownica[8][8], int szer, int wys, int wymPole);`
Funkcja inicjalizuje tablicę wskaźników Szachownica wartościami współrzędnych pól (wyrażone w pikselach). Funkcja przed tym również przydziela pamięć na każdy element tablicy.

`void initSzachownica(struct pionek* Szachownica[8][8], ALLEGRO_BITMAP* tab[12]);`
Funkcja inicjalizuje tablicę wskaźników odpowiednimi wartościami– typ, kolor, bitmapa, czy jest atakowane przez figurę białą lub czarną (początkowo 0).

`int zamienPionek(struct pionek* Szachownica[8][8], ALLEGRO_BITMAP* kaBi, ALLEGRO_BITMAP* kaCz);`
Funkcja obsługuje sytuację promocji – zamiany pionka na królową przy dojściu jego do końca szachownicy. Funkcja zwraca współrzędną x takiego pionka – potrzebne w sytuacji, gdy pionek po przemieszczeniu się i promocji odsłonił króla.

`void zamienNaPionek(struct pionek* Szachownica[8][8], int xPi, int kolor, ALLEGRO_BITMAP* piBi, ALLEGRO_BITMAP* piCz);`
Funkcja obsługuje sytuację, gdy pionek po promocji odsłonił króla. Wtedy wraca na poprzednią pozycję i przywracany mu jest poprzedni typ.

`void stoperF(int czasS[2], char** czas1, char** czas2, int nowyCzas, int* sekunda, int Gracz);`
Funkcja zajmuje się mierzeniem czasu rozgrywania rund dla obu graczy.

Program zaczyna się funkcjami inicjalizującymi. Następnie wkracza do dwóch pętli while. W pierwszej ustawiane są wartości początkowe zmiennych (wymagane przy ponownym rozegraniu gry). W głównej pętli program wykonuje funkcję *zaznaczAtakowane()* oraz czeka na zdarzenia przychodzące od użytkownika (kliknięcie myszką lub naciśnięcie przycisku na klawiaturze). Następnie w zależności co zrobił użytkownik, podejmowane są różne akcje. Jeśli użytkownik kliknął na napis „Wyjście”, wyskakuje zapytanie o potwierdzenie zamiaru wyjścia z programu. Jeśli natomiast użytkownik kliknął na planszę, sprawdzana jest pole (*ktorePole()*)

które wybrał. Dla gracza 1 program podświetli pole tylko, gdy jest na nim figura biała (dla gracza 2 – figura czarna). Przy następnym kliknięciu sprawdzana jest pole docelowe i program korzysta z funkcji *czyPoprawny()*. Po przemieszczeniu (*zamienZawartoscPola()*), sprawdzane jest od razu, czy przypadkiem ten ruch nie powoduje odsłonięcie własnego króla na szachowanie (co nie powinno się wydarzyć). W takim wypadku „cofany" jest ruch figury. Jeśli natomiast ruch był całkowicie poprawny, sprawdzana jest sytuacja szachowania króla przeciwnika (*sprSzach()*). Jeśli tak się wydarzyło, należy sprawdzić sytuację matowania. Jeśli otoczenie króla jest również atakowane (*sprOtocz()*) – nie może się poruszyć, należy upewnić się czy przypadkiem nie może on zostać „uratowany" przez inną figurę. „Ukrywane" są figury już zasłaniające króla (*wylaczKryjace()*) i następnie sprawdzany jest ewentualny ratunek (*czyRatunek()*). Gdy to się nie powiedzie, zmienna czyMatB/czyMatC zmienia wartość na 1. Program przechodzi do rysowania (czasu, napisów, planszy, figur) i następnie sprawdza wartość zmiennych czyMatB i czyMatC. Jeśli któraś z nich ma wartość 1, wyświetlany jest komunikat o zakończonej rozgrywce i zapytanie o jej ponowne rozpoczęcie. Jeśli użytkownik wybierze taką opcję, program kończy wykonywanie głównej pętli. W przeciwnym wypadku, program wyjdzie także z kolejnej pętli i w ten sposób będzie mógł zakończyć swoje działanie. W przypadku, gdy nie doszło do matowania króla, program wróci na początek pętli.

4. Testowanie

Program testowany był przede wszystkim pod względem wykrywania momentów szachowania i matowania królów. W szachach istnieje mnóstwo różnych sytuacji, gdy taki stan zachodzi i ważne jest, aby przetestować jak największą liczbę z nich.

5. Wnioski

W szachach istnieją osobne zasady przemieszczania się dla każdej z figur. Z tego powodu, część funkcji w głównej mierze opiera się na instrukcjach warunkowych (*if*), gdyż nie da się stworzyć jednej funkcji „uniwersalnej" dla każdej figury. Należało pamiętać również o tym, że figury mogą znaleźć się na brzegach szachownicy i sprawdzanie pola obok nich (wykroczenie poza szachownicę) mogłoby przynieść nieoczekiwany skutek. Największą uwagę należało poświęcić królowi i sytuacji szachowania oraz matowania. Istnieje ogromna liczba różnych wariantów wygrania rozgrywki i należało stworzyć funkcje, które by wykrywały to poprawnie.