Krzysztof Urbańczyk
gr. IZ04P02
nr indeksu: 14 302

Warszawa, dn. 20.06.2014

Programowanie Obiektowe 2 – Projekt
"Szachy wykorzystujące polimorfizm oraz biblioteki STL"

Prowadzący: dr hab. Tomasz Werner

I. Algorytm

Program wykorzystuje polimorfizm oraz biblioteki STL do zaznaczania dostępnych elementów szachowych, przedstawiania dostępnych ruchów i poruszania się po szachownicy.

Polimorfizm:
Algorytm zakłada istnienie klasy szachownica, która w zasadzie jest tablicą [8][8] i jest odwzorowaniem szachownicy. Jest to tablica wskaźników na obiekty klasy "pionek". Klasa "pionek" jest klasą abstrakcyjną i zawiera parametry koloru, nazwy danego elementu szachowego oraz funkcję czysto wirtualną do późniejszego określenia ruchów każdego z elementów.

STL:
Ilość dostępnych ruchów dla danego elementu jest wartością zmienną, zależną od aktualnego stanu szachownicy. Do przekazywania współrzędnych dostępnych ruchów wykorzystałem wektory z biblioteki STL. Przy każdym wytypowaniu elementu funkcja pochodna od "pionek::ruchy" czyści z danych 2 wektory, a następnie bada położenia na szachownicy i dopisuje informacje o dostępnych ruchach. Współrzędne "x" są wpisane do tablicy V (vertical), a "y" do tablicy H (horizontal).

Algorytm:

1. p1.show() - metoda klasy wyświetlającą szachownicę z zainicjowanymi pochodnymi obiektami klasy "pionek".
2. Pobranie współrzędnych wybranego elementu.
3. Uruchomienie funkcji „ruchy”, która inicjuje tablice „ruchy”, położenia, oraz wektory V, H:
   3.1. „tab_polozen” wpisuje do tablicy położenia informacje na temat kolorów pionków w poszczególnych polach szachowych.
   3.2. „Pionek.ruchy” wykorzystuje tablice „polozenia”, współrzędne aktualnie wybranego elementu oraz oba wektory, które po tej operacji zawierają współrzędne dostępnych ruchów.
   3.3. Wektory są przepisane do tablicy „ruchy”, która jest potem wykorzystana do wyświetlenia dostępnych ruchów za pomocą funkcji „show_ruchy”.
   3.4. Funkcja „ruchy” zawiera również serię komentarzy do użytkownika związaną z niepoprawnymi wyborami oraz zwraca wartość false, jeżeli ruch nie został wykonany poprawnie.
   3.5. W przypadku przejścia całego algorytmu prawidłowo, funkcja „ruchy” prosi ponownie o akceptacje przesunięcia elementu. Za pomocą przeciążonego operatora obiekt szachownica podany do funkcji zostaje zmodyfikowany.
4. Zwrócona wartość bool przez funkcje „ruchy” jest przypisywana zmiennej „tura”, która tylko w przypadku wartości prawdziwej, pozwala na zakończenie danej tury. Ta wartość będzie prawdziwa tylko wtedy, kiedy wykonany ruch elementu był prawidłowy.

II. Klasy:

Klasa abstrakcyjna:

```
class pionek{
public:
	string name;
	string color;
pionek(string n, string c);
virtual void ruchy(vector<int> &V, vector<int> &H,const short &x, const short &y, const short
tab_polozen[8][8])=0;//przyjmuje globalne wektory ruchu, czyści i uzupełnia o nowe ruchy
};
```

Klasa zawiera funkcje czysto wirtualną „ruchy” oraz publiczne zmienne dotyczące nazwy pionka i koloru. Zmienne te są typu publicznego, aby klasa szachownica mogła wyświetlać kolor i nazwę elementu za pomocą metody show();

przykładowa klasa pochodna:

```
class queen : public pionek{
public:
queen(string c);
void ruchy(vector<int> &V, vector<int> &H,const short &x, const short &y, const short tab_polozen[8]
[8]);
};
```

Konstruktor wymaga tylko określenia koloru natomiast nazwa zostaje automatycznie przypisana zgodnie z typem klasy pochodnej:

```
pawn::pawn(string c):pionek("P",c){}
king::king(string c):pionek("K",c){}
knight::knight(string c):pionek("H",c){}
bishop::bishop(string c):pionek("G",c){}
rook::rook(string c):pionek("W",c){}
queen::queen(string c):pionek("H",c){}
```

Przykładowa funkcja ruchu:

```
void pawn::ruchy(vector<int> &V, vector<int> &H,const short &x, const short &y, const short
tab_polozen[8][8]){
	short a[2];
	V.clear();H.clear();

if(this->color=="b"){
//bicie lewo
	if(y>0 && tab_polozen[x+1][y-1]==2){
		a[0]=1;//0 dla pionu
		a[1]=-1; // 1 dla poziomu
	V.push_back(a[0]);H.push_back(a[1]);
	}
...

}
}
```

Metoda czyści wektory V i H, a następnie wprowadza dostępne współrzędne ruchy, jeżeli zostają spełnione warunki brzegowe oraz gdy dany element, który jest w pozycji bicia, jest przeciwnika.

Klasa szachownica:

```
class szachownica{
public:
       pionek *plan[8][8];
szachownica();
void show();
friend szachownica& operator<<(szachownica &P,const szachownica &M);
};
```

Klasa zawiera tablicę wskaźników do obiektów pionek. Konstruktor uzupełnia szachownice okładając elementy zgodnie z układem rozpoczynającej się rozgrywki. Posiada metodę drukująca szachownice wraz z elementami oraz zaprzyjaźnia funkcje przeciążającą operator "<<".

Inne funkcje:

```
szachownica& operator<<(szachownica &P,szachownica &M);
```

przeciążenie operatora "<<" jest wykorzystane do głębokiego kopiowania obiektów typu szachownica.

```
void show_ruchy(const szachownica &A, const short tab[8][8]);
```

procedura wyświetla aktualna szachownice wraz z dostępnymi ruchami wybranego elementu.

```
void tab_polozen(short tab[8][8], const szachownica &p1);
```

procedura wpisuje do tablicy polozen informacje o położeniach elementów w zależności od koloru.

```
bool ruchy(szachownica &p1,const short &x, const short &y);
```

Funkcja wykorzystuje powyższe procedury i metody do wykonania opisanego powyżej algorytmu.

III Układ kodu:

Blok prototypów klas.
Blok prototypów funkcji.
Main();
Konstruktory klasy bazowej pionek i pochodnych
Konstruktory klasy szachownica oraz definicje metod szachownica
Definicje pozostałych funkcji
Definicje metody ruchu dla klas pochodnych (z tytułu najobszerniejszego kodu).