Dokumentacja do projektu

Przedmiot: **PROI**
Autor: **Rafał Kulus**

Temat projektu:
Ustawianie figur na szachownicy.

Informacje szczegółowe:
Projekt spełnia następujące wymagania:

- Klasy autonomiczne ze zmienną strukturą obiektów
  - class Setup;
  - class Interface;
- Reprezentacja grupowa
  - class Piece;
- Szablony klas i funkcji
  - class PieceInterface;
  - class Interface::InterfaceImpl;
- Dziedziczenie
  - class Piece;
- Funkcje wirtualne
  - class Piece;
- Obsługa sytuacji wyjątkowych
  - class IOHandler;
- Współpraca ze strumieniami
  - class IOHandler;
- Klasy kontenerowe i iteratory
  - class Quiz;
  - class Utilities;
- Dokumentacja

Informacje dodatkowe:
Ze względu na zastosowanie w projekcie reprezentacji grupowej i szablonów, nawet algorytm brute-force nie radzi sobie z każdym problemem. Nie zmienia to jednak faktu, że jego skuteczność jest większa od standardowego algorytmu.

Link do projektu na GitHubie:
https://github.com/BlueAlien99/PROI_Projekt3

“Interface.h”

Najważniejsza klasa. Stanowi serce programu.
Zawiera algorytm i obsługę szachownicy.

`Interface(uint xx, uint yy, uint pa = 0, uint ro = 0, uint bi = 0, uint qu = 0, uint kn = 0, uint ki = 0){`

Konstruktor. Parametry: xx (szerokość szachownicy), yy (wysokość szachownicy), pa (liczba pionków), ro (liczba wież), bi (liczba gońców), qu (liczba hetmanów), kn (liczba skoczków), ki (liczba królów).

`}`

`~Interface(){`

Destruktor.

`}`

`int isSqFree(uint xx, uint yy){`

Funkcja zwraca:

- 1 jeśli pole (xx, yy) nie jest bite.
- 0 jeśli pole (xx, yy) jest bite lub zajęte.
- -1 jeśli pole (xx, yy) znajduje się poza szachownicą.

`}`

`bool algorithm(bool interact = 0, bool batch = 0){`

Funkcja uruchamia algorytm dla posiadanej konfiguracji. Wywołanie z parametrami:

- interact == 0, batch == 0;
- interact == 0, batch == 1; wyeksportowanie wyniku
- interact == 1, batch == 0; szukanie rozwiązania w pliku, pytanie o użycie brute-force, pytanie o wyeksportowanie wyniku
- interact == 1, batch == 1; użycie brute-force, wyeksportowanie wyniku

`}`

`std::string getBoardStr(bool debug = 0){`

Funkcja zwraca stringa zawierającego obecny stan szachownicy. Szachownica jest przedstawiona w sposób opisowy, zrozumiały dla użytkownika.

- Jeśli nie został uruchomiony algorytm, string będzie zawierał pustą szachownicę.
- Jeśli został uruchomiony algorytm, debug == 0, a rozwiązanie znaleziono, string będzie zawierał szachownicę tylko z podstawowymi informacjami, tzn. będą na niej oznaczone tylko figury.

- Jeśli został uruchomiony algorytm, debug == 1, a rozwiązanie znaleziono, string będzie zawierał szachownicę w trybe debugu, tj. każde wolne pole będzie oznaczone liczbą, przez ile figur jest bite.
- Jeśli został uruchomiony algorytm, a rozwiązania nie znaleziono, jak wyżej, lecz string będzie zawierał niekompletną szachownicę.

}

`std::string getConfigStr(){`

Funkcja zwraca stringa zawierającego obecną konfigurację szachownicy. Konfiguracja jest przedstawiona w sposób opisowy, zrozumiały dla użytkownika.

}

“IOHandler.h”

Klasa odpowiedzialna za każdą operację zapisu do / odczytu z pliku w programie.

`static void cleanup(){`

Funkcja usuwa wszystkie pliki tekstowe utworzone przez program.

}

`static void genProblemsTxt(){`

Funkcja generuje plik “problems.txt” z komentarzem pomocy. Użytkownik może następnie umieścić w pliku konfiguracje, które zostaną rozwiązane jedna po drugiej.

}

`static void solveProblems(bool brute){`

Funkcja rozwiązuje po kolei wszystkie konfiguracje umieszczone w pliku “problems.txt”. Jeśli brute == 1, program spróbuje dany problem rozwiązać także metodą brute-force, jeśli standardowy algorytm zawiedzie.

}

`static void findSolution(std::string config, std::vector<int> *board){`

Funkcja przeszukuje plik “solutions.txt” w poszukiwaniu konfiguracji config, a następnie importuje rozwiązanie (szachownicę) do board.

}

`static void saveSolution(bool result, std::string config, std::string board){`

Funkcja zapisuje rozwiązanie board konfiguracji config w pliku “solutions.txt”. Jeśli konfiguracja nie ma rozwiązania, wówczas result == 0 i board nie ma znaczenia.

}

`static void exportVisualised(bool result, std::string config, std::string board){`

Funkcja zapisuje rozwiązanie board konfiguracji config w pliku “solutions_visualised.txt” w formie zrozumiałej i czytelnej dla użytkownika. Jeśli konfiguracja nie ma rozwiązania, wówczas result == 0 i board nie ma znaczenia.

}

`static void saveQuizScore(std::string name, uint correct, uint total, int time, int score){`

Funkcja zapisuje do pliku “scoreboard.txt” statystyki gracza po ukończeniu quizu. Parametry: name (nazwa gracza), correct (prawidłowe odpowiedzi), total (liczba pytań), time (czas trwania quizu), score (uzyskane punkty).

`}`

`static void printScoreboard(){`

Funkcja odczytuje z pliku “scoreboard.txt” statystyki graczy i prezentuje je w postaci czytelnej tabeli.

`}`

“Piece.h”

Klasa reprezentująca pojedynczą figurę.

`Piece(){`
Konstruktor.
`}`

`virtual ~Piece(){`
Destruktor.
`}`

`Piece(const Piece &piece){`
Konstruktor kopiujący.
`}`

`int getID(){`
Funkcja zwraca ID figury.
`}`

`void setLocation(int x, int y){`
Funkcja ustawia pole figury na (x,y).
`}`

`std::pair<int,int> getLocation(){`
Funkcja zwraca koordynaty pola, na którym stoi figura.
`}`

`int moveChar(){`
Funkcja zwraca liczbę opisującą charakterystykę ruchu figury. Liczba ta jest sumą następujących składników:

- 1 - figura może poruszać się pionowo
- 2 - figura może poruszać się poziomo
- 4 - figura może poruszać się na skos

`}`

`std::vector<std::pair<int,int> > moveSquares(){`
Funkcja zwraca vector relatywnych koordynatów pól, na które może się przemieścić figura podczas wykonywania ruchu.
`}`

virtual int getRelativeValue(){

Funkcja zwraca relatywną wartość figury.

}

Klasy pochodne:

```
class Pawn: public Piece;

class Rook: public Piece;

class Bishop: public Piece;

class Queen: public Piece;

class Knight: public Piece;

class King: public Piece;
```

Każda klasa pochodna posiada swój konstruktor oraz (z wyjątkiem klasy King) funkcję int getRelativeValue().

“PieceInterface.h”
template <class T>

Klasa reprezentuje zbiór figur jednego rodzaju.

PieceInterface(int n = 0){
Konstruktor. Parametr n oznacza liczbę figur typu T, które będą przechowywane.
}

~PieceInterface(){
Destruktor.
}

int getID(){
Funkcja zwraca ID przechowywanego rodzaju figur.
}

void setLocation(uint n, int x, int y){
Funkcja ustawia pole n-tej figury na (x,y).
}

std::pair<int,int> getLocation(uint n){
Funkcja zwraca koordynaty pola, na którym stoi n-ta figura.
}

int moveChar(){
Funkcja zwraca liczbę opisującą charakterystykę ruchu przechowywanego rodzaju figur. Liczba ta jest sumą następujących składników:

- 1 - figura może poruszać się pionowo
- 2 - figura może poruszać się poziomo
- 4 - figura może poruszać się na skos

}

std::vector<std::pair<int,int> > moveSquares(){
Funkcja zwraca vector relatywnych koordynatów pól, na które może się przemieścić przechowywany rodzaj figur podczas wykonywania ruchu.
}

int getCount(){
Funkcja zwraca liczbę figur, które przechowuje.
}

“Quiz.h”

Klasa zawiera najważniejsze mechanizmy potrzebne do działania quizu.

```
struct Player{
```

`std::string name;` Nazwa gracza

`uint totalQues;` Liczba pytań

`uint correctQues;` Prawidłowe odpowiedzi

`int time;` Czas trwania quizu

`int score;` Liczba uzyskanych punktów

```
Player(uint totalQues = 0)
: name(""), totalQues(totalQues), correctQues(0), time(0), score(0) {
```

Konstruktor.

```
}
};
```

```
struct PtrCmp{
    bool operator()(const Player *p1, const Player *p2){
```

Funkcja porównująca dwóch graczy. Używana do sortowania po wyniku (większy), a następnie po czasie (mniejszy).

```
    }
};
```

```
static void run(){
```

Funkcja uruchamia quiz.

```
}
```

“Setup.h”

Klasa stanowi główny interfejs interakcji z użytkownikiem.

Setup(){
Konstruktor.
}

~Setup(){
Destruktor.
}

void help(){
Funkcja czyści ekran i wypisuje pomoc (menu).
}

void run(){
Funkcja stanowi główne menu programu.
}

void newBoard(){
Funkcja przeprowadza z użytkownikiem dialog nt. konfiguracji nowej szachownicy.
}

void algorithm(){
Jeśli została wprowadzona konfiguracja szachownicy (newBoard()), funkcja wywoła algorytm, który spróbuje znaleźć rozwiązanie dla wprowadzonej konfiguracji.
}

void printDebBoard(){
Jeśli została wprowadzona konfiguracja szachownicy (newBoard()):

- i nie został uruchomiony algorytm, funkcja wypisze pustą szachownicę.
- i został uruchomiony algorytm, a rozwiązanie znaleziono, funkcja wypisze szachownicę w trybe debugu, tj. każde pole będzie oznaczone liczbą, przez ile figur jest bite.
- i został uruchomiony algorytm, a rozwiązania nie znaleziono, jak wyżej, lecz szachownica nie będzie kompletna.

}

“Tests.h”

Klasa zawiera testy algorytmu rozstawiania figur.

void run(){

Funkcja pyta użytkownika, czy chce uruchomić test algorytmu, a następnie wykonuje kilka predefiniowanych testów. W przypadku wyników innych od oczekiwanych, program zostanie zamknięty z powodu “niespełnionych” assertów.

}

“Tutorial.h”

Klasa zawiera wszystkie funkcje związane z tutorialem gry w szachy.

static void run(){

Funkcja uruchamia tutorial, który opisuje podstawy gry w szachy.

}

“Utilities.h”

Klasa zawiera funkcje, które nie są ekskluzywne dla projektu.

static uint get_uint(){

Funkcja wczytuje od użytkownika stringa, a następnie szuka w nim pierwszego wystąpienia nieujemnej liczby całkowitej, którą następnie zwraca. Jeśli string nie zawiera cyfry, zwracane jest 0. Jeśli liczba składa się z ponad 9 cyfr, zwracana jest liczba złożona z pierwszych dziewięciu cyfr.

}

static void getPermutations(std::string str, std::set<std::string> *perms, uint lock = 0){

Funkcja generuje wszystkie permutacje stringa str, które umieszcza w kontenerze perms. Dzięki użyciu std::set, w kontenerze znajdą się tylko unikalne permutacje (bez powtórzeń).

}