Układanki z lat dziecinnych

Projekt zaliczeniowy nr **13** z przedmiotu **Zaawansowane C++** (wiosna 2018)

Wstępna analiza funkcjonalna

Treść projektu.

Napisać program symulujący "puzzle" (kwadrat o zadanej długości boków n, z polami ponumerowanymi od 1 do n*n-1, jedno pole puste. Użytkownik może przesunąć na puste pole dowolny z sąsiadujących elementów). Program powinien umożliwiać ułożenie pól wierszami od najmniejszego numeru do największego ręcznie lub automatycznie (w przypadku układania automatycznego mamy do czynienia z trudnym projektem).

Założenia projektu

Środowisko

Aplikacja zostanie napisana za pomocą darmowego środowiska Qt w wersji 5.10.1 i edytora Qt Creator 4.5.1. Wykorzystany zostanie kompilator Desktop Qt 5.10.1 MinGW 32bit lub Microsoft Visual C++ Compiler 15.0 lub 17.0 64bit AMD (jeśli uda mi się skonfigurować prawidłowo Qt do współpracy ze środowiskiem VS Community 2015 lub 2017)
Jeśli nie uda mi sięokiełznać środowiska Qt (nigdy z niego nie korzystałem), to aplikacja zostanie napisana za pomocą edytora Visual C++ Community 2017 z wykorzystaniem platformy Microsoft .NET Framework w wersji 4.5.2
Aplikacja będzie posiadała standardowy dla systemu MS Windows graficzny interfejs użytkownika w postaci okna dialogowego posiadającego pasek menu i pasek stanu (i ewentualnie pomocniczych okienek dialogowych do wyświetlania komunikatów)

Zakładane funkcjonalności

Aplikacja będzie umożliwiała:

- ustawienie parametrów, czyli np. rozmiaru układanki (długości boku kwadratu wyrażonego w liczbie pól). Ustawianie parametrów odbywa się w dodatkowym okienku (okno konfiguracji) wywoływanym za pomocą menu okna głównego.
- wyświetlanie aktualnego stanu układanki w postaci jej graficznej reprezentacji (każde pole w postaci kwadratu z odpowiadającym mu numerem; układanka rozwiązana ma pola ponumerowane jak w opisie projektu – poczynając od lewego górnego pola wierszami liczbami od 1 do n*n-1, ostatnie puste pole nie jest numerowane i jest wyraźnie wyróżnione)
- wygenerowanie losowego rozmieszczenia elementów układanki ręcznego rozmieszczenia elementów układanki (poprzez wykonywanie przesunięć zgodnie z regułami)
- ręczne rozwiązanie układanki – przez umożliwienie przesuwania elementów zgodnie z regułami – osiągnięcie stanu „ułożenia" aplikacja będzie odpowiednio sygnalizować, dodatkowo zliczana będzie liczba wykonanych ruchów i czas
- automatyczne rozwiązanie układanki za pomocą jednego z algorytmów poinformowanego przeszukiwania (na pewno postaram się zrealizować algorytm A*) – po znalezieniu rozwiązania aplikacja zaprezentuje je poprzez wykonanie kolejnych ruchów na planszy z odpowiednim opóźnieniem (z możliwością ustawienia tego opóźnienia w oknie konfiguracji). Dodatkowo prezentowane są statystyki (np. długość ścieżki (liczba ruchów), liczba sprawdzonych stanów (rozpatrywanych ruchów)

Dodatkowo, jeśli starczy mi czasu, chciałbym zrealizować następujące funkcjonalności:

- rozszerzenie planszy układanki do prostokąta o długościach boków n i m
- umieszczenie jako tła dla elementów układanki odpowiednio przeskalowanego obrazka – jako alternatywnego widoku dla liczbowego opisu elementów układanki (możliwe byłyby 3 tryby wyświetlania układanki – tylko numery elementów, tylko fragmenty obrazka przypisane do elementów, łączny – fragmenty obrazka z umieszczonym na nim numerem elementu)
- rozszerzenie planszy do dowolnego rozmiaru i kształtu (przy zachowaniu możliwości rozwiązania, czyli „spójności" planszy rozumianej jako możliwość przesunięcia pola pustego na dowolne pole planszy)

W dwóch powyższych funkcjonalnościach chodzi o umożliwienie zrealizowania układanki, jaką pamiętam z własnych lat dziecinnych – czyli w postaci kwadratowej planszy o n*n polach z dodatkowym pustym polem znajdującym się przy jednej z krawędzi (rys.1). Umożliwi to wyświetlanie pełnego obrazka na elementach układanki. Układanka zdefiniowana w treści projektu powodowałaby „zdekompletowanie" obrazka przez wycięcie jego prawego dolnego rogu.

Rys.1 układanka o 4x4=16 elementach i z dodatkowym polem pustym przy dolnej krawędzi po lewej

Realizacja projektu

Aplikacja zostanie zrealizowana wg założeń wzorca architektonicznego MVC. Poniżej przedstawiam wstępny szkic głównych części aplikacji w podziale wg tego wzorca. Szkic zawiera wyszczególnienie głównych klas każdej części i ich najważniejsze zadania.

View

Klasy odpowiedzialne za graficzny interfejs użytkownika (GUI). Zadaniem GUI jest wyświetlanie stanu aplikacji (układanki, algorytmu przeszukującego, parametrów) oraz rejestrowanie interakcji użytkownika. GUI jest pozbawione jakichkolwiek elementów sterowania – za sterowanie odpowiedzialny jest Kontroler.
GUI składa się przede wszystkim z klasy realizującej głowne okno aplikacji – CmainWindow. Klasa ta zostanie opracowana za pomocą edytora interfejsu graficznego w Qt. Klasa – oprócz kontrolek - będzie zawierała wskaźnik na obiekt kontrolera – do przekazywania mu akcji wykonanych przez użytkownika. Klasa będzie zawierała również wskaźniki na wszystkie kontrolki, które wymagają sterowania ich stanem na podstawie przekazanych z kontrolera dyspozycji (np. wyświetlanie aktualnego stanu układanki, informacji tekstowych itp.). Wizualizacja układanki będzie zrealizowana za pomocą przycisków – każdy element układanki będzie osobnym przyciskiem. Jeśli to rozwiązanie nie będzie efektywne dla dużych rozmiarów planszy, to zostanie zastąpione przez obiekt klasy QGraphgicsView – za pomocą którego (na którym) aktualny stan układanki będzie rysowany w postaci bitmapy.
Okno będzie zawierało proste menu, składające się z pozycji:
W grupie „Menu”:

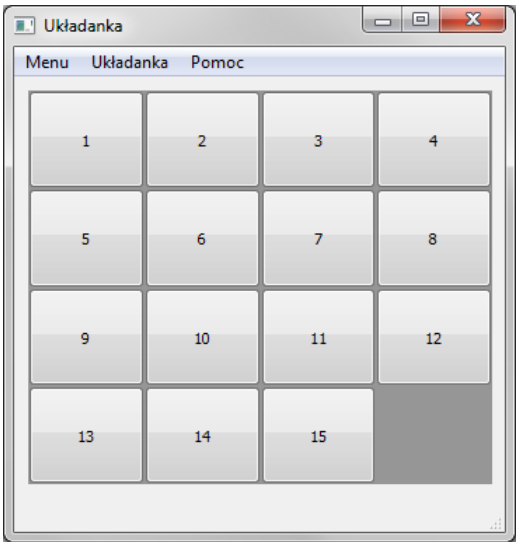
- konfiguracja (wyświetlanie prostego okna dialogowego do ustawienia parametrów: rozmiaru planszy itp.)
- wyjście –kończy działanie aplikacji

W grupie „Układanka”:

- Tasuj! – powoduje wykonanie szeregu losowych przesunięć elementów układanki zgodnie z regułami. Liczba przesunięć będzie losowa (np. z przedziału zadanego w oknie konfiguracyjnym)
- Rozwiąż! – uruchamia algorytm przeszukiwania znajdujący rozwiązanie (stan ułożenia)
- Resetuj – przywraca stan ułożenia bez szukania rozwiązania zgodnie z regułami
- Ręczne ustawienie – przełącznik umożliwiający ręczne przetasowanie układanki. Gdy jest ustawiony aplikacja nie mierzy liczby kroków ani czasu, gdy jest wyłączony – aplikacja mierzy liczbę wykonanych ruchów i czas. Czas mierzony jest od pierwszego przesunięcia elementu.

W skład części View będzie wchodził również interfejs ograniczający dostęp kontrolera do GUI tylko do potrzebnych metod. Będą to m.in. metody:

- wyświetlania układu planszy na podstawie przekazanych argumentów (ciąg kolejnych elementów)
- resetowania układu planszy na podstawie jej rozmiaru – zostanie wyświetlona „ułożona” układanka o danym rozmiarze
- wyświetlenia okna konfiguracji (argumenty: aktualne ustawienia)
- wyświetlenia informacji w pasku Status Bar
- wyświetlenia okienek z pomocą/informacjami O programie
- wyświetlania informacji o aktualnym stanie algorytmu przeszukiwania



Rys.2 projekt głównego okna aplikacji

Czynności wykonane przez użytkownika będą przekazywane do Kontrolera za pośrednictwem jedynej metody interfejsu Kontrolera udostępnionej GUI – wyślijWiadomość(obiekt klasy Wiadomość). Klasa wiadomość będzie klasą abstrakcyjną, z której dziedziczyć będzie hierarchia klas pochodnych – każda z nich będzie opisywała konkretną czynność użytkownika – np. kliknięcie elementu układanki o danym numerze, wybranie opcji z menu, wciśnięcie przycisku zamykania okna głównego itp.

Model

Model składa się z dwóch osobnych części – jedna z nich odpowiada za reprezentację układanki, druga – algorytmu przeszukiwania A*.

Układanka

W głównej klasie (CUkladanka) pamiętana będzie przede wszystkim reprezentacja planszy układanki. Plansza będzie reprezentowana przez tablicę jednowymiarową int o rozmiarze n*n oraz osobną zmienną pamiętającą rozmiar planszy (czyli wartość n). Aktualny stan układanki będzie opisany przez ciąg n*n liczb: od 0 do n*n-1 – odczytanych kolejno wierszami od lewego górnego rogu układanki, gdzie 0 będzie oznaczało pole puste. Chciałbym do tej reprezentacji wykorzystać szablon z jakimś nietrywialnym iteratorem umożliwiającym poruszanie się po tej tablicy zgodnie regułami układanki (a więc iterator

wskazywałby puste pole i umożliwiał sprawdzanie jego sąsiadów). W szablonie można by przechowywać liczby całkowite – jak to jest przedstawione wcześniej, albo obiekty specjalnej klasy opisującej element układanki (CElement) – za pomocą której można by przechowywać np. układankę graficzną (każdy element wskazywałby obiekt reprezentujący fragment obrazka oraz indeks odpowiadający jego pozycji w stanie ułożenia).
Klasa CUkladanka zawiera szereg metod udostępnionych przez odpowiedni interfejs Kontrolerowi:

- resetuj(rozmiar) – resetuje reprezentację układanki do stanu początkowego (ułożenia) na planszy o zadanym rozmiarze
- sprawdzRuch(pole) – zwraca informację (bool), czy możliwe jest wykonanie ruchu polegającego na przesunięciu elementu ze wskazanego pola planszy (a więc jeśli w sąsiedztwie tego pola jest pole puste – true, w p.p. – false)
- sprawdzRuch(pole,stan) – metoda przeciążona - zwraca informację (bool), czy możliwe jest wykonanie ruchu polegającego na przesunięciu elementu ze wskazanego pola planszy dla ustawienia opisanego przez argument stan – metoda ta będzie wykorzystywana przez algorytm przeszukujący
- wykonajRuch(pole) – próbuje wykonać zadany ruch i zwraca true, jeśli ruch został wykonany (był poprawny)
- czyUlozona() – zwraca true jeśli stan układanki to stan ułożenia (wszystkie elementy są na swoich polach)
- czyUlozona(stan) – zwraca true jeśli zadany jako argument stan układanki to stan ułożenia (wszystkie elementy są na swoich polach) – na potrzeby algorytmu przeszukującego
- podajOcene(stan) – zwraca wartość liczbową opisującą ocenę zadanego stanu układanki – na potrzeby algorytmu przeszukującego (heurystyki). Liczba ta powinna być uzależniona od liczby elementów znajdujących się poza swoim docelowym miejscem (i ewentualnie odległości od tego miejsca).
- zwrocMozliweRuchy() -

Algorytm

W głównej klasie (CAlgorytm) zaimplementowany jest algorytm A* bez odniesienia do aktualnego problemu (rozwiązywanie układanki). Takie rozwiązanie pozwoli wykorzystać tę klasę do innych zadań. W Algorytmie A* tworzone jest drzewo przeszukiwań, w którym w każdym węźle reprezentowany jest konkretny stan problemu. Algorytm A* to przeszukiwanie poinformowane – informację stanowi ocena łączna danego stanu problemu złożona z dotychczas przebytej drogi (od stanu początkowego) oraz szacowanej odległości do stanu końcowego (heurystyka). Zatem w każdym węźle tworzonego drzewa przeszukiwań musimy umieć określić – przebytą dotychczas drogę, szacowaną drogę do przebycia oraz możliwe do wykonania ruchy (do osiągnięcia kolejnego stanu) wraz z ich kosztem. Jeśli więc stworzymy klasę CWezel realizującą te wszystkie metody, to wtedy klasa CAlgorytm może być zaimplementowana jako szablon klasy z parametrem klasy CWezel. W takim rozwiązaniu wszystkie metody z klasy CAlgorytm (realizujące algorytm przeszukiwania) będą rozdzielone od konkretnego problemu, natomiast problem będzie zdefiniowany w klasie CWezel.
Klasa CAlgorytm będzie realizowała standardowy algorytm A*:

1. Pobierz węzeł startowy S i wstaw go do zbioru OPEN, ustaw jego funkcję oceny f=0 i koszt dojścia g=0
2. Pobierz ze zbioru OPEN węzeł X o najmniejszej wartości funkcji oceny f i wstaw go do zbioru CLOSED
3. Jeśli X jest węzłem końcowym – zakończ działanie, zrekonstruuj ścieżkę dotarcia od S do X i zwróć ją jako rozwiązanie
4. Znajdź węzły – następniki węzła X – zbiór Y1,Y2,...,Yn
5. Dla każdego następnika Yi:
    a. oblicz koszt dojścia g'(Yi)=g(X)+c(X,Yi), gdzie g(X) to koszt dotarcia do aktualnego węzła X, a c(X,Yi) to koszt przejścia z węzła X do węzła Yi
    b. Jeśli Yi nie jest elementem zbiorów OPEN i CLOSED – dodaj go do zbioru OPEN i ustaw jego koszt dojścia na g=g'(Yi), a funkcję oceny na f=g'(Yi)+h(Yi), gdzie h(Yi) jest wartością heurystyki (szacowania odległości tego węzła od węzła końcowego)
    c. Jeśli Yi należy do zbioru OPEN lub (należy do zbioru CLOSED i znaleziony teraz koszt dotarcia jest mniejszy niż zapamiętany, czyli g'(Yi)<g(Yi)) to:
        i. Zapamiętaj nowe wartości funkcji oceny i kosztu dotarcia dla tego węzła: g=g'(Yi), f=g'(Yi)+h(Yi)
        ii. Jeśli węzeł Yi był w zbiorze CLOSED – usuń go z tego zbioru i wstaw do zbioru OPEN
        iii. Usuń z drzewa przeszukiwań starą ścieżkę od węzła S do węzła Yi (znaleziona obecnie jest lepsza – ma lepszą funkcję kosztu dojścia)
6. Wróć do kroku 2.

Powyższy algorytm – a dokładnie jeden jego krok (czyli od p.2 do p.6) będzie realizowany za pomocą metody KrokAlgorytmu(). Metoda ta będzie udostępniona przez interfejs kontrolerowi. Umożliwienie kontrolerowi uruchamiania tylko pojedynczych kroków algorytmu umożliwi proste zarządzanie komunikacją pomiędzy składowymi aplikacji.
Implementacja zbiorów OPEN I CLOSED zostanie przeprowadzona za pomocą posortowanych list – chodzi o zoptymalizowanie wyszukiwania węzłów w tych zbiorach, tak aby metody sprawdzające czy dany węzeł jest już elementem tych zbiorów działały szybko (co może być istotne przy dużych rozmiarach planszy)
Definiowanie problemu do rozwiązania dla algorytmu będzie polegało na utworzeniu przez kontroler odpowiednio sparametryzowanego obiektu klasy CWezel i przekazaniu go obiektowi klasy CAlgorytm jako węzeł startowy.

Klasa CWezel – obiekt tej klasy pamiętałby konkretny stan układanki – czyli obiekt klasy CUkladanka

Kontroler

Klasa realizująca sterowanie aplikacją. Przechowuje wskaźniki do obiektu GUI (głównego okna) oraz do obiektów modelu (CUkladanka i CAlgorytm). Za pomocą metody wyślijWiadomość udostępnionej GUI przez interfejs otrzymuje informacje o czynnościach wykonanych przez użytkownika (w postaci obiektów z hierarchii klas wiadomości). Na ich podstawie oraz na podstawie zmiennej opisującej własny stan (typ wyliczeniowy: stan_ustawianie_ręczne, stan_rozwiązywanie_ręczne, stan_rozwiązywanie_automatyczne, stan_bezczynny) wykonuje odpowiednie scenariusze, np. w przypadku wybrania przez użytkownika opcji Rozwiąż! z menu:

1. Ustaw stan na stan_rozwiązywanie_automatyczne
2. Uruchom metodę dostarczoną przez CAlgorytm KrokAlgorytmu
3. Jeśli metoda zwróciła false (algorytm w tym kroku nie znalazł rozwiązania)
    a. Sprawdź czy użytkownik nie przerwał działania algorytmu – jeśli tak: przejdź do obsługi tego zdarzenia
    b. Wróc do p.2
4. Jeśli metoda zwróciła true (algorytm w tym kroku znalazł rozwiązanie)
    a. Wywołaj metodę dostarczoną przez interfejs CAlgorytm zwracającą ścieżkę (sekwencję kolejnych stanów)
    b. W pętli z ustalonym opóźnieniem za pomocą dostarczonej przez odpowiedni interfejs z GUI nakazuj wyświetlać kolejne układy elementów
    c. Zmień stan na stan_bezczynny

Obiekt Kontroler (wraz z obiektami modelu) i obiekt GUI powinny działać w osobnych wątkach.