Dokumentacja projektu PROI

Autorzy

Franciszek Sakowski, nr albumu 309439

Łukasz Zawistowski, nr albumu 316916

Wprowadzenie

Omawianym w tym dokumencie projektem jest gra pod tytułem “2048”. Gra odbywa się na planszy o rozmiarach 4x4, mogącej pomieścić do 16 bloków. Każde pole planszy może zawierać lub nie zawierać bloku, a każdy blok posiada wartość liczbową będącą potęgą dwójki. W każdej turze gracz wykonuje ruch za pomocą strzałek przesuwając tym samym wszystkie bloki maksymalnie do lewej, prawej, góry lub dołu. Dodatkowo, jeśli w danej turze dwa bloki o takiej samej wartości znajdują się obok siebie, a gracz wybierze taki kierunek, że jeden z bloków zostanie przesunięty w stronę drugiego to bloki łączą się w jeden, o wartości będącej sumą obydwu bloków. Po każdym łączeniu bloków, dodawana jest odpowiednia wartość do wyniku gracza. Po wykonaniu ruchu i połączeń, na losowym, niezajętym dotychczas miejscu planszy pojawia się nowy blok o wartości 2 lub 4. Gra kończy się, gdy nie można dodać nowego bloku na planszę albo gdy powstał blok o wartości 2048 który oznacza wygraną.

Omówienie projektu

Do rozwiązania została użyta biblioteka SDL będąca wsparciem dla wizualizacji obiektów utworzonych, jak i obsługi sterowania. Drugą zewnętrzną biblioteką jest SDL_tff będąca niejako rozszerzeniem podstawowego SDL o czcionki. Jest ona istotna przede wszystkim w momencie nadrukowywania na bloki wartości liczbowych, umożliwiają też komunikację z użytkownikiem.

Architektura aplikacji

Projekt został utworzony w oparciu o wydzielenie plików nagłówkowych .h oraz .cpp w ramach łatwiejszej czytelności kodu oraz wzorzec projektowy MVC. Głównym jego zamysłem jest wydzielenie odpowiednich obszarów w celu utworzenia uporządkowanego rozwiązania, czytelnego, mogącego być łatwo skalowalne, innymi słowy takiego który jest nie wymusza lawinowych zmian w momencie rozwoju.

Zgodnie z akronimem wyróżnione zostały następujące klasy w programie. Są to kolejno:

- Model, czyli struktury danych oraz elementy odpowiadające za rozpatrywanie abstrakcyjnych problemów, przykładowo takich jak utworzenie obiektu, wykonanie operacji na danych. W ramach tego obszaru są klasy takie jak Board, Block, Scene i program main.

- View, oznacza pokrótce część odpowiedzialną za zwizualizowanie logiki dziejącej się w niższych warstwach abstrakcji. Jest to warstwa z którą użytkownik programu na jedynie do czynienia. Znajdują się w niej głównie obiekty graficzne. Przypada na nią klasa Visualization.

- Controller, ostatni element, będący rozwinięciem akronimu, ma na celu zaserwowanie z modelu wyniku w ramach, akcji wykonanych na widoku przez użytkownika. Są to przykładowo zmiany na inne widoki, zmienić model. W rozwiązaniu jest to klasa Steering.

Warto dodać, iż standardowym dla projektów zaimplementowanych w języku C++ jest funkcja uruchomienia aplikacji, main. Znajduje w niej się wywołanie widoku oraz uruchomienie pętli głównej programu.

Opisy klas

Board - jest to z perspektywy samej gry jeden z ważniejszych elementów. Znajduje się w nim reprezentacja planszy będącą macierzą o wielkości 4 na 4. Każda wchodząca w do niej wartość jest potem reprezentowana. Inną istotną funkcją klasy jest wykonywanie transformacji macierzy.

Pola klasy:

brd – omówiona wyżej macierz reprezentująca planszę.

Metody:

Swipe() – ma na celu wykonanie określonego przemieszczenia się liczb na macierzy. Domyślnie jest to ruch liczb z dołu na górę, z zachowaniem określonych warunków. Liczby zero dodawane są na sam dół, bądź przenoszone, jako ostatnie. Przykładowo wektor [2 2 0 0] da wynik [4 0 0 0]. Nastąpiło w nim sumowanie dwóch liczb takich samych, a na końcu dostawiono liczbę zero.

TransformLeft(), TransformDown(), TransformRight() – są to metody wykonujące transformację na na zmiennej brd w celu przygotowania jej do wykonania ruchu przez metodę Swipe(). Następne są one używane do odwrócenia modyfikacji.

ReverseRight() – odwraca transformację wykonaną przez metodę TransformRight(), metoda ta jako jedyna wymagała predefiniowania odwrócenia.

DrawBoard() – jest to funkcja pomagająca w debugowaniu macierzy.

AddRandom() – ma na celu dodanie liczby od 2 lub 4 do macierzy w losowym miejscu pojawienia się zera.

GetBoard() – zwraca obecny stan macierzy

GetScore() – zlicza wszystkie punkty na planszy

Clear() – usuwa liczby z macierzy i wprowadza zera.

Steering – klasa sterująca grą, obsługująca wejścia użytkownika i zwracająca informacje o aktualnym stanie gry.

Pola:

New_event – obiekt przechowujący najnowszy event, tj. wciśnięcie przycisku na klawiaturze, ruch myszką itp.

KeyPressedNow – używając pętli while do obsługi gry, charakterystyka SDL wymusza osługiwanie eventu przy każdym jej obrocie. Powyższe pola pozwala obsłużyć pojedynczy event tylko raz.

Brd – instancja klasy Board, przechowująca planszę

IsGameLost, isGameWon, gameRestart – pola typu bool pozwalające określić aktualny stan gry – wygrana/przegrana/w trakcie/wymagająca restartu

Metody:

CurrentBoard() - zwraca aktualny stan planszy

IsNewEvent() - pomaga ustalić, czy pojawił się nowy event, np. użytkownik wcisnął nową strzałkę

IsQuit() - zwróci prawdę, jeśli gra z jakiegoś powodu się zakończy

HandleKeyboard() - główna metoda klasy, w przypadku istnienia nowego eventu odpowiada za obsługę wejść użytkownika w postaci wciśnięcia jednej ze strzałek, wywołując metody klasy Board oraz ustawiając odpowiednio wszystkie flagi

Scene – klasa bezpośrednio odpowiedzialna za rysowanie statycznych elementów gry – tła, planszy, nazwy gry oraz wyniku. Jej podklasa Visualization odpowiedzialna jest za rysowanie zmieniających się, dynamicznych części gry.

Pola klasy:

window, renderer, Arial – obiekty klas udostępnianych przez SDL i SDL_ttf służące do realizacji okna i wizualizowania planszy.

steer – obiekt klasy Steering, dzięki niemu klasa Visualization odbiera wszystkie zmiany na planszy spowodowane ruchami użytkownika.

Metody:

DrawScene() - celem tej metody jest ustalenie koloru tła oraz wyrysowanie kwadratowego obszaru, na którym prezentowany będzie aktualny stan planszy

ScoreText() - metoda renderuje aktualny wynik użytkownika, poprzedzony napisem “Wynik”.

GameNameText() - jej zadaniem jest wygenerowanie w lewym górnym rogu okna napisu z nazwą gry - “2048”.

Visualization – podklasa klasy Scene odpowiedzialna za graficzną wizualizację elementów samej gry w oknie.

Pola klasy:

window, renderer, Arial – obiekty klas udostępnianych przez SDL i SDL_ttf służące do realizacji okna i wizualizowania planszy.

steer – obiekt klasy Steering, dzięki niemu klasa Visualization odbiera wszystkie zmiany na planszy spowodowane ruchami użytkownika.

window_width, window_height - szerokość i wysokość wyświetlanego okna, odgórnie ustalone na 800x600.

Metody:

Klasa posiada konstruktor tworzący okno, ładujący używaną czcionkę oraz inicjalizujący obiekt klasy Steering, a także destruktor kasujący okno i kończący procesy wizualizacyjne.

Za pomocą metod IsWindowCreated(), IsRendererCreated() i IsFontLoaded() jest w stanie określić, czy powyższe elementy został utworzone prawidłowo, a jeśli nie wypisze informacje o błędzie.

Następnie klasa udostępnia kilka metod służących tworzeniu i wizualizacji poszczególnych elementów. Działają one dzięki metodzie CreateRectangle() tworzącej obiekt prostokąta typu SDL_Rect, który następnie może zostać wyrenderowany.

DrawGrid() - wyrysowuje wszystkie 16 bloków znajdujących się na planszy, rozpoczynając od uzyskania aktualnego stanu planszy przez metody klas Steering i Board. Korzystając z metody GetValue() klasy Block przyporządkowuje każdemu blokowi odpowiedni kolor. Następnie, jeśli wartość bloku (pole value udostępniane metodą GetValue()) jest różna od zera, wizualizuje się ją w odpowiednim miejscu przy pomocy poniższe metody DrawNumber()

DrawNumber() - za argumenty przyjmuje współrzędne (lewy-górny róg) wypisywanego tekstu oraz samą wartość do wypisania. Następnie zamienia typ wartości z int na char[] i dzięki metodom biblioteki SDL renderu napis na ekranie.

DrawYes() - wizualizuje w oknie prostokątny obszar koloru zielonego, a następnie wypisuje na środku tego obszaru napis “Yes”.

GameWonScreen() - w przypadku spełnienia warunku na wygranie gry zostanie uruchomiona powyższa metoda, zmieniająca obraz wizualizowany w oknie na specjalną scenę. Podobnie jak poprzednie metody rozpoczyna od wypisania napisu – tym razem o treści “You won. Play again?”, a na koniec tworzy przycisk umożliwiający ponowną rozgrywkę za pomocą DrawYes(). Jego wciśnięcie zresetuje grę i ustawi planszę w stan początkowy.

GameLostScreen() - analogiczna do GameWonScreen() metoda, udostępniają specjalną scenę po przegranej. Treścią napisu “You lost. Play again?”.

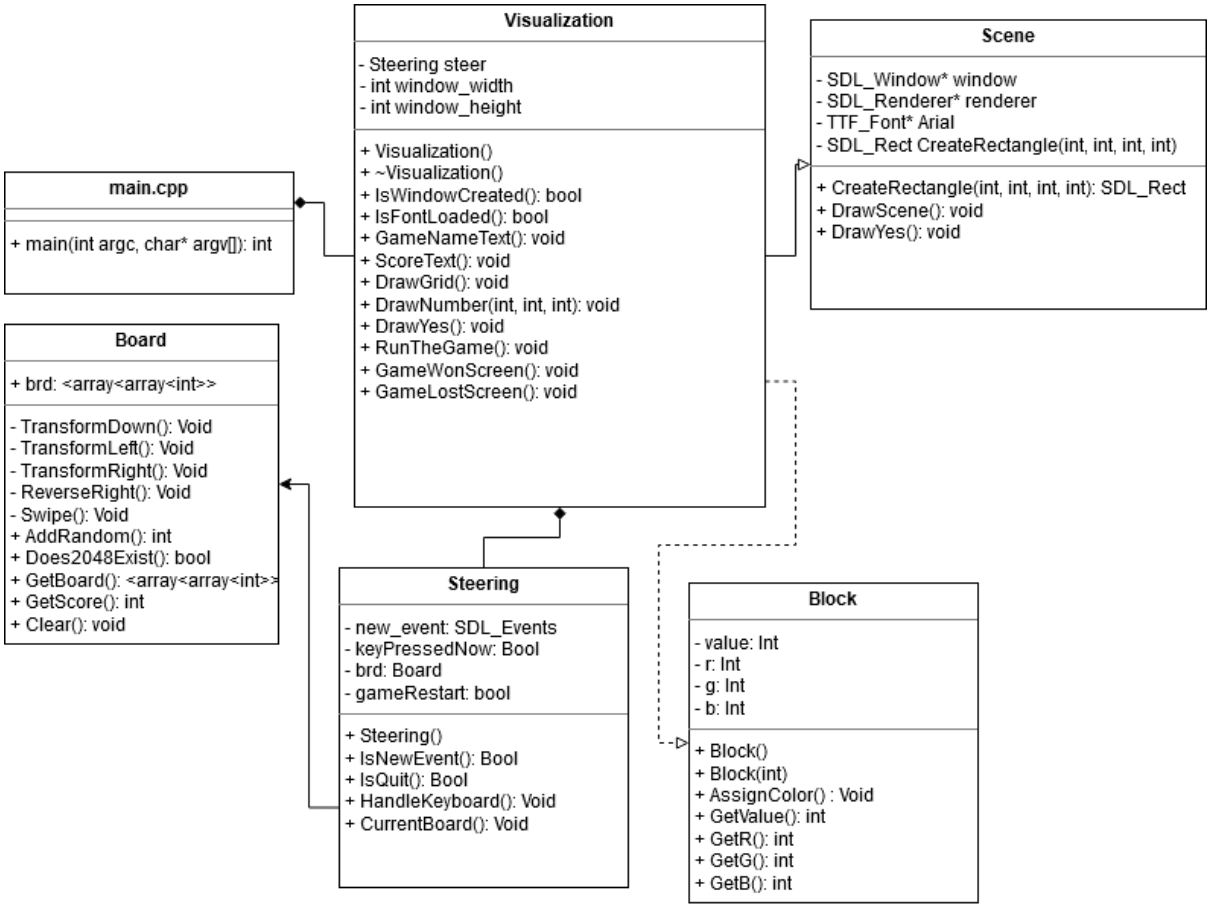
Główną metodą klasy jest RunTheGame(), obsługująca wizualizację gry i korzystająca jedynie ze wcześniej zaimplementowanych metod. Jest głównym zadaniem jest ustalenie, co w danym obrocie pętli wyrenderować w oknie gry. Do ustalenia tego posługuje się flagami obiektu steer, które są odpowiednio ustawiane w zależności od wejścia użytkownika (przyciskanie jednej z czterech strzałek) oraz stanu gry (gra wygrana/przegrana/w trakcie/wymagająca resetu). RunTheGame() wywołuje metodę HandleKeyboard() klasy Steering, a następnie metody klasy Visualization w zależności od aktualnej sytuacji.

Block – klasa reprezentująca pojedynczy blok znajdujący się na planszy.

Polami obiektów tej klasy są value, r, b, g. Pierwszy z nich przechowuje aktualną wartość bloku, będącą potęgą dwójki. Pozostałe trzy reprezentują kolor bloku w formacie RGB, wyznaczany za pomocą metody AssignColor(). Metoda ta wyznacza liczby r, g, b (0<= r, g, b <=255) na podstawie aktualnej wartości bloku (value), przyporządkowując odpowiedni kolor – od białego, przez pomarańczowy i czerwony, po żółty.

Diagram UML

Klasy i metody wymienione w poprzednim rozdziale zostały ujęte w diagram UML widoczny poniżej.



Rozwój projektu

W ramach dalszego rozwoju projektu planowane jest dodanie wyboru poziomu trudności, np. łatwy, średni i trudny polegającego na kolejno zwiększaniu liczby pól oraz wymogu uzyskania docelowo większej liczby punktów na pojedynczym bloku potrzebnych do wygrania gry, dalej będących wielokrotnością liczby dwa.