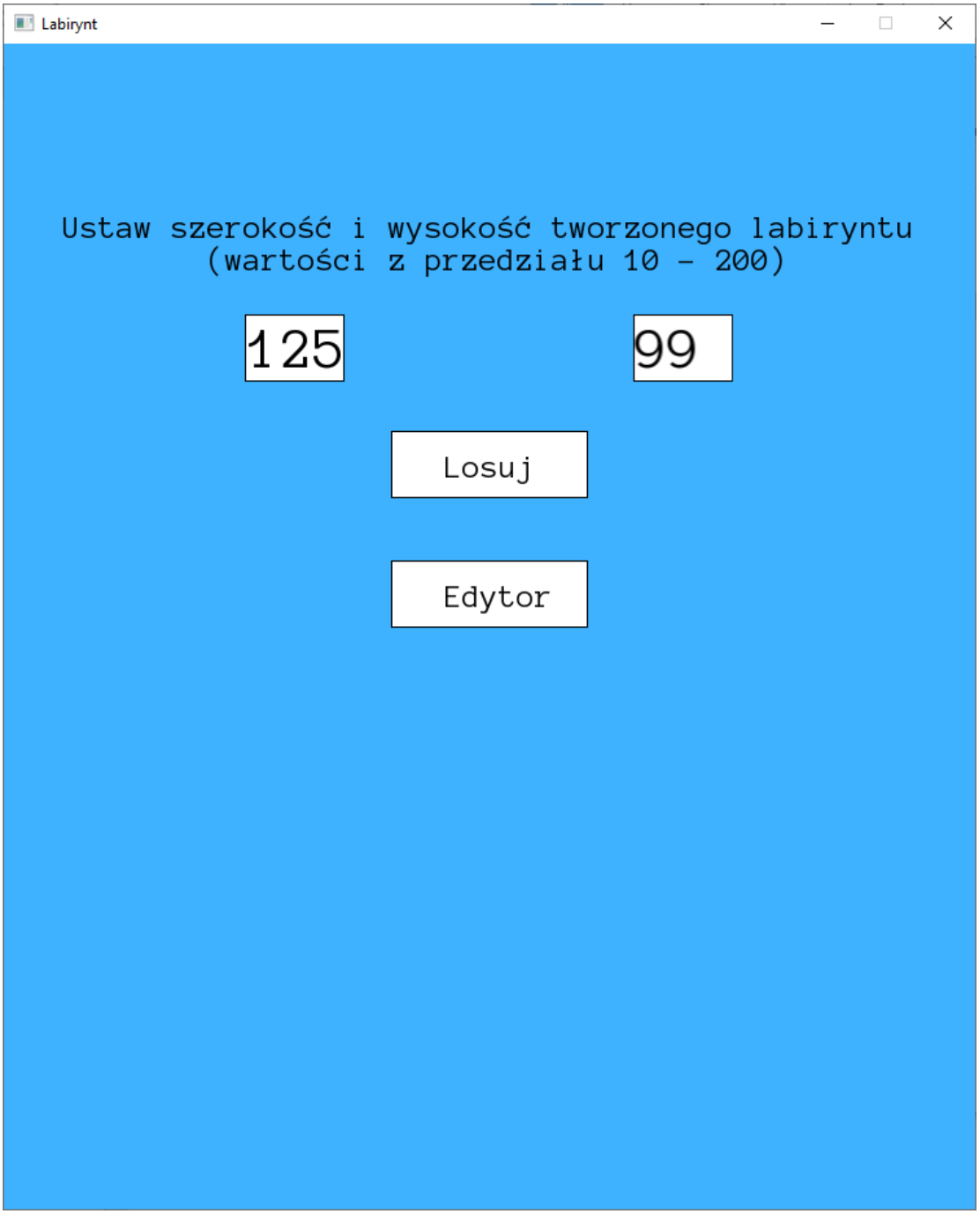
Labirynt

Dokumentacja projektu z przedmiotu
Programowanie w języku C++

Mikołaj Pałka

167828

SPIS TREŚCI

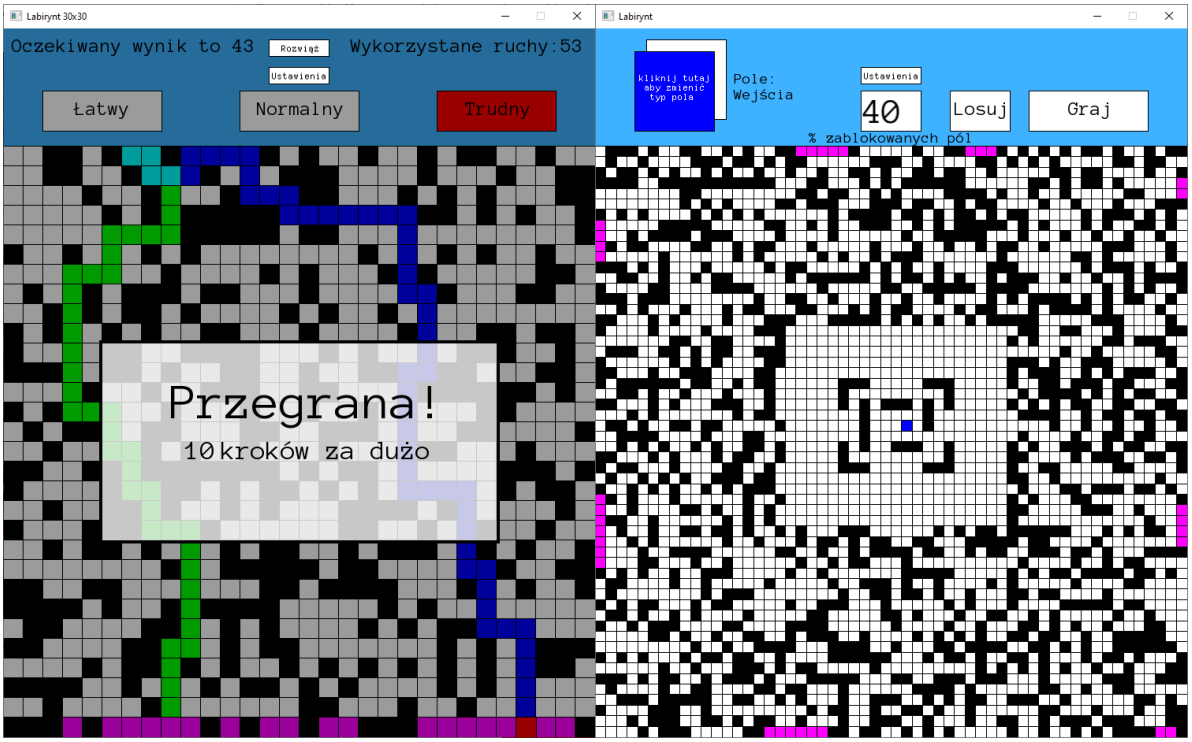


OPIS PROJEKTU.

Labirynt to gra logiczna 2D stworzona w języku C++ z użyciem biblioteki programistycznej SFML 2.5.1.

Zadaniem gracza jest przejście labiryntu złożonego z pól białych, po których można przejść, pól czarnych, na które nie można wejść, oraz pól o kolorze magenty, które stanowią zakończenie labiryntu. Pola odwiedzone przez gracza malowane są na kolor niebieski. Gracz wygrywa jeśli uda mu się przejść labirynt przy użyciu najmniejszej ilości kroków. W razie porażki, gra pokazuje optymalną drogę kolorem zielonym (kolorem turkusowym malowane są pokrywające się pola ścieżki wybranej przez gracza i ścieżki optymalnej).

Po włączeniu gry gracz jest poproszony o wpisanie rozmiarów labiryntu. Boki planszy labiryntu mogą składać się od 10 do 200 pól. Następnie gracz może wybrać między wylosowaniem labiryntu lub stworzeniem własnego poziomu o podanych rozmiarach za pomocą edytora planszy.



Gracz może wylosować nowy labirynt o takich samych wymiarach klikając na jeden z przycisków trudności decydujących o ilości pól czarnych na wylosowanej planszy. Możliwy jest także powrót do okna wpisywania rozmiaru i wyboru między planszą losową a edytorem za pomocą przycisku „Ustawienia”. Przycisk „Rozwiąż” pokazuje optymalną drogę bez przechodzenia labiryntu.

Sterowanie:

- strzałki: przejście na pole obok
- Z: cofnięcie się o jeden krok
- Q: wyjście z programu

KLASA OKNO.

Klasa odpowiadająca za wyświetlanie grafiki w oknie, obsługi zdarzeń i aktualizowania stanu gry w pętli. Do przechowywania planszy i jej stanów użyto wektorów z biblioteki standardowych szablonów STL. Plansza złożona jest z pól stworzonych przy użyciu obiektu `sf::RectangleShape`. Gracz porusza się po dwuwymiarowym wektorze przechowującym rodzaj danego pola na planszy na podstawie czego gra decyduje czy gracz może taki ruch wykonać lub czy znalazł pole wyjścia. Pozycja gracza w wektorze przekłada się na pozycję gracza na ekranie.

Metody publiczne:

- `okno();` – konstruktor inicjujący zmienne
- `~okno();` – destruktor
- `bool oknoOtwarte();` – metoda sprawdzająca czy okno gry jest otwarte
- `void rysuj();` – metoda rysująca i wyświetlająca elementy graficzne w oknie
- `void aktualizuj();` – metoda do obsługi zdarzeń (np. naciśnięcie przycisku, ruch gracza za pomocą klawiszy strzałek, dojście do końca labiryntu)

Metody prywatne:

- `void resetZmienne();` – metoda zerująca zmienne
- `void tworzenieOkna();` – metoda tworząca okno
- `void tworzeniePlanszy();` – metoda tworząca planszę
- `void kolorowaniePlanszy();` – metoda ustawiająca kolory pól planszy
- `void restart();` – metoda losująca nowy labirynt
- `void optymalnaDroga();` – metoda malująca optymalną drogę na planszy
- `void PustyLabirynt();` – metoda tworząca pustą planszę do edytowania
- `bool sczytajKolory();` – metoda sczytująca kolory z edytowanej planszy w celu określenia ich rodzaju oraz sprawdzająca czy została podana odpowiednia ilość wejść i wyjść
- `void resetWektory();` – metoda zerująca wektory oraz ustawiająca ich rozmiary
- `void poziomEdytora();` – metoda wczytująca planszę stworzoną w edytorze

Pola prywatne:

- pola biblioteki SFML odpowiadające za zawartość wyświetlanych przycisków i tekstu:

```
sf::Font czcionka;
sf::Text gornyPasek;
sf::Text wynikTxt;
sf::Text koniecTxt;
sf::Text krokiTxt;
sf::Text kroki2Txt;
sf::Text instrukcja;
sf::Text latwyTxt;
sf::Text normalnyTxt;
sf::Text trudnyTxt;
sf::RectangleShape latwy;
sf::RectangleShape normalny;
sf::RectangleShape trudny;
sf::RectangleShape koniecTlo;
sf::RectangleShape koniecTlo2;
sf::Text szerokoscWczytaj;
sf::Text wysokoscWczytaj;
sf::Text szansaWczytaj;
sf::Text generujTxt;
sf::Text ustawieniaTxt;
sf::Text menuTxt;
sf::Text rozwiazanieTxt;
sf::Text edytorTxt;
sf::Text poziomEdytoraTxt;
sf::Text EdytorGenerujTxt;
sf::Text ostrzezenieEdytor;
sf::Text EdytorInstrukcja1;
sf::Text EdytorInstrukcja2;
sf::RectangleShape szerokoscPoleTxt;
sf::RectangleShape wysokoscPoleTxt;
sf::RectangleShape szansaPoleTxt;
sf::RectangleShape guzikMenu;
sf::RectangleShape guzikGeneruj;
sf::RectangleShape poleKolor;
sf::RectangleShape poleKolorBialy;
sf::RectangleShape guzikRozwiazanie;
sf::RectangleShape guzikEdytor;
sf::RectangleShape guzikPoziomEdytora;
sf::RectangleShape guzikEdytoraGeneruj;
```

- `sf::RenderWindow* oknoGry;` – zmienna okna, w którym wyświetlane są elementy graficzne
- `sf::Event zdarzenie;` – zmienna przechowująca ostatnie zdarzenie
- `sf::RectangleShape kwadratGracz;` – obiekt gracza
- `std::vector<std::vector<sf::RectangleShape>> kwadraty;` – wektor wektorów obiektów kwadratów planszy
- `sf::Vector2i pozycjaGracz;` – wektor przechowujący położenie gracza
- `sf::Vector2f pozycjaKursor;` – wektor przechowujący położenie kursora
- `std::vector<std::vector<int>> czarne;` – wektor wektorów przechowujący rodzaj poszczególnych pól
- `std::vector<std::vector<int>> odwiedzone;` – wektor wektorów przechowujący stan pól
- `std::vector<int> ruchy;` – wektor przechowujący ruchy gracza
- `int n;` – wysokość planszy
- `int m;` – szerokość planszy
- `float szansa;` – szansa na pokolorowanie pola na czarno ustawiana przez gracza
- `float skala;` – skala rozmiaru pola
- `int kolor;` – zmienna ustalająca wybrany kolor pola w edytorze planszy
- `int wynik;` – zmienna przechowująca bieżący wynik (liczbę kroków) gracza
- `int oczekiwanyWynik;` – zmienna przechowująca optymalną liczbę kroków
- `float szansaCzarny;` – zmienna przechowująca szanse pokolorowania pola na czarno ustawiana przyciskami trudności
- `std::string daneN;` – zmienna przechowująca zawartość pola tekstowego do ustawiania wysokości labiryntu
- `std::string daneM;` – zmienna przechowująca zawartość pola tekstowego do ustawiania szerokości labiryntu
- `std::string daneSzansa;` – zmienna przechowująca zawartość pola tekstowego do ustawiania szansy pokolorowania pola na czarno
- pola typu bool do zarządzania stanem gry:

`bool koniecGry;` – zmienna określająca czy gracz ukończył labirynt
`bool menu;` – zmienna określająca czy gracz jest w oknie Ustawienia
`bool szerokoscWpisz;` – zmienna określająca czy pole tekstowe szerokości jest aktywne
`bool wysokoscWpisz;` – zmienna określająca czy pole tekstowe wysokości jest aktywne
`bool szansaWpisz;` – zmienna określająca czy pole tekstowe szansy pokolorowania pola na czarno jest aktywne
`bool sprawdzanie;` – zmienna określająca czy należy wyświetlić optymalną ścieżkę po zakończeniu labiryntu
`bool stopRuch;` – zmienna określająca czy gracz może się poruszać
`bool malowanie;` – zmienna określająca czy gracz maluje pola
`bool mazanie;` – zmienna określająca czy gracz maże pola
`bool edytor;` – zmienna określająca czy gracz jest w oknie edytora
`bool rozw;` – zmienna określająca czy gracz kliknął w guzik „Rozwiąż”
`bool pozEdytora;` – zmienna określająca czy przechodzony jest plansza edytora
`bool wstawianieStart;` – zmienna określająca czy można wstawić pole wejścia w edytorze

- `algorytm bfs;` – obiekt klasy algorytm

KLASA POLE.

Klasa pomocnicza definiująca pole labiryntu do użytku w algorytmie wyszukującym optymalną drogę przez labirynt. Przechowuje koordynaty i odległość obiektu od początku labiryntu.

Metody publiczne:

- `pole(int w, int k, int o):wiersz(w), kolumna(k), odleglosc(o){};` – konstruktor parametrowy obiektu

Pola publiczne:

- `int wiersz;` – zmienna przechowująca położenie pola na wysokości
- `int kolumna;` – zmienna przechowująca położenie pola na szerokości
- `int odleglosc;` – zmienna przechowująca odległość pola od pola wejściowego

KLASA ALGORYTM.

Klasa odpowiadająca za znajdywanie optymalnej ścieżki przez labirynt. Do wykonania tego zadania użyty został algorytm BFS – przeszukiwania wszerz – zmodyfikowany tak, aby działał na kracie – wektorze wektorów w programie.

Metody publiczne:

- `int odlegloscMinimalna(std::vector<std::vector<int>> plansza);` – funkcja przeszukująca wszerz podaną planszę w celu znalezienia pola wyjścia. Do zrealizowania funkcji zastosowana jest obiekt kolejki z STL. Obiekty klasy Pole.h są w pętli kolejkowane i oznaczane jako odwiedzone. Jeśli nie zostanie znalezione pole wyjściowe funkcja zwraca wartość ujemną, w przeciwnym razie zwraca optymalną liczbę kroków potrzebnych do przejścia labiryntu.
- `void rec(std::vector<std::vector<int>>& plansza, int optymalneKroki);` – funkcja przyjmuje planszę i ustawia stan pól zawierających się w optymalnej ścieżce do późniejszego pokolorowania na zielono przez funkcję `void okno::optymalnaDroga().`

Pola publiczne:

- `int n;` – wysokość planszy
- `int m;` – szerokość planszy