Politechnika Śląska

Wydział Matematyk Stosowanej

Kierunek Informatyka

Gliwice, 30.01.2024

Programowanie I

**projekt zaliczeniowy**

**"*Sztuka zbierania*"**

**Maciej Sztucki gr. 2 lab. 4**

**1. Opis projektu.**

*Projekt jest grą polegającą na zbieraniu punktów rozsianych losowo na jednej z trzech przygotowanych map. Gracz staje naprzeciw komputerowemu przeciwnikowi, który również stara się uzbierać wymaganą ilość punktów.*

**2. Funkcjonalności.**

* void print(vector<vector<char» mapa, int punkty\_gracza, int punkty\_bota) Ta funkcja wyświetla aktualną planszę oraz punkty graczy.
* dodanie\_punktu(vector<vector<char» mapa) Ta funkcja losowo wybiera dozwolony punkt na planszy i zwraca jego współrzędne aby móc dodać tam punkt do zebrania.
* stack<punkt> bfs(vector<vector<char» mapa, pair<int,int> poz\_bota) Ta funkcja zwraca stos punktów od pozycji początkowej bota do punktu do zebrania
* int gra() Ta funkcja zarządza tokiem gry
* int generowanie() Funkcja służy losowemu wybraniu numeru mapy na której rozgrywana będzie rozgrywka
* vector<vector<char» mapa() Funkcja służy rozlokowaniu 5 losowych punktów na wybranej mapie i zwraca mapę już z punktami.

**3. Przebieg realizacji.**

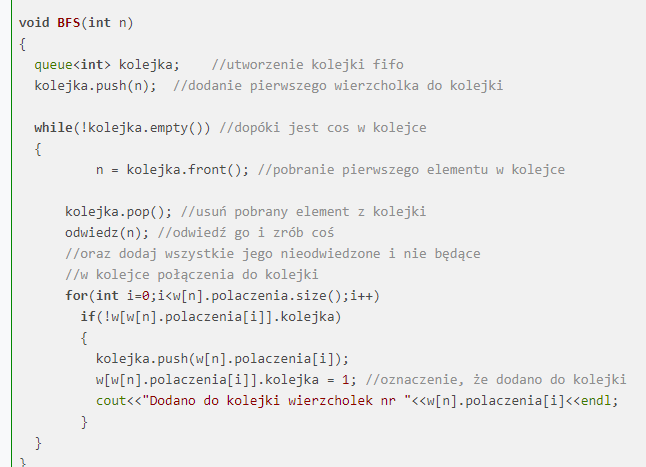
*Pliki składowe:*

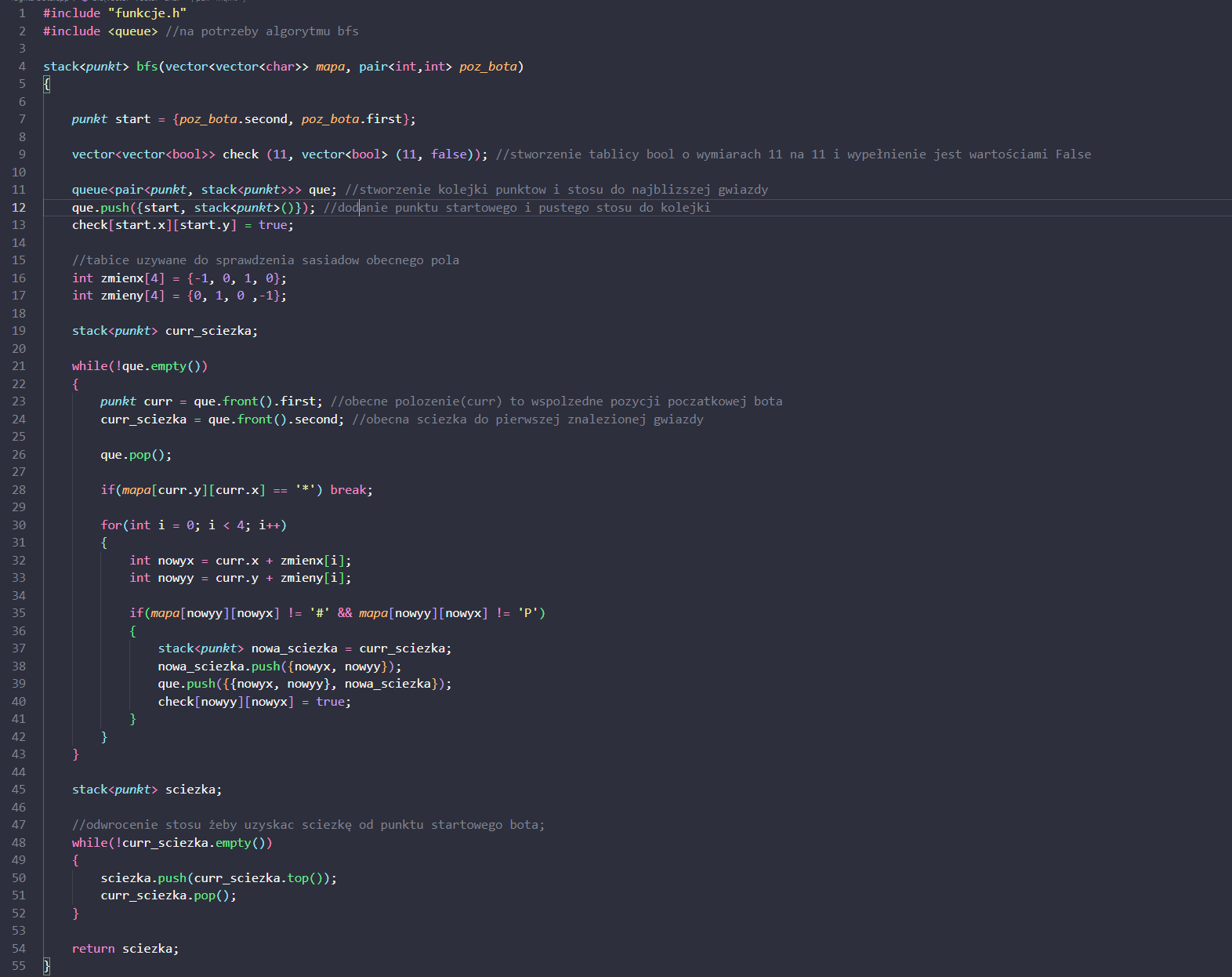
1. „faktyczna gra.cpp” Plik w którym znajduje się gra
2. „funkcje.h” Plik łączący wszystkie inne
3. „logika bota.cpp” Plik w którym znajduje się funkcja zwracająca ścieżkę bota do najbliższego punktu
4. „main.cpp” Menu gry
5. „przygotowanie mapy.cpp” Plik przygotowujący mapę do gry
6. „a.exe” plik z grą

Opis algorytmu BFS

Z angielskiego breadth-first search służy to przeszukiwania grafu (w przypadku tego projektu planszy). Projekt wykorzystuje ten algorytm aby znaleźć najkrótszą ścieżkę do punktu do zebrania.

Przykładowa implementacja BFS ze strony https://www.algorytm.edu.pl/grafy/bfs.html



Implementacja BFS w projekcie

Działanie algorytmu

Tworzymy wektor wektorów bool-ów i wypełniamy je fałszem

Algorytm dodaje początkową pozycję do kolejki (queue) i oznacza ją jako odwiedzoną. Następnie wchodzimy do pętli która działa do momentu sprawdzenia wszystkich wierzchołków grafu lub dotarcia do najbliższej gwiazdy. Curr to obiekt struktury punkt zawierający obecne współrzędne dla których rozpatrujemy sąsiadów. curr\_sciezka to sciezka od startu (poz\_bota) od obecnego punktu Curr.

Sprawdzamy czy nasza obecna pozycja to punkt do zebrania ‘\*’ jeżeli tak to przerywamy pętlę odwracamy stos żeby uzyskać kolejność od startu do naszego punktu a nie na odwrót i zwracamy odwrócony stos.

W innym przypadku sprawdzamy czy sąsiedzi obecnego punktu spełniają warunek czy można na nie wejść(Nie są graczem lub ścianą ‘#’) jeżeli spełniają warunek to do nowego stosu kopiujemy zawartość curr\_sciezka i punkt spełniający warunki, następnie dodajemy ten sam punkt do kolejki punktów do sprawdzenia a odwiedzony punkt oznaczamy jako odwiedzony.

Opis bibliotek

conio.h

* \_kbhit() sprawdza czy został naciśnięty klawisz na klawiaturze
* \_getch() zbiera input z klawiatury bez potrzeby potwierdzenia go enterem (zwraca int) dlatego potrzebna była konwersja na char

chrono

* chrono::steady\_clock::now() zwraca obecny czas
* chrono::duration\_cast<chrono::miliseconds> rzutuje na milisekundy

thread

* this\_thread::sleep\_for() zatrzymuje wątek na zadany czas w milisekundach

vector

* pozwala na użycie vectorów

stack

* pozwala na użycie stosów
* stos.push() dodanie elementu do stosu;
* stos.pop() usunięcie z góry elementu stosu;
* stos.top() zwraca górny element stosu
* sots.empty() sprawdza czy stos jest pusty

queue

* pozwala na użycie queue (kolejki)
* q.push() dodanie na koniec kolejki
* q.empty() sprawdza czy kolejka jest pusta
* que.front() zwraca pierwszy element kolejki
* que.pop() usuwa pierwszy element kolejki

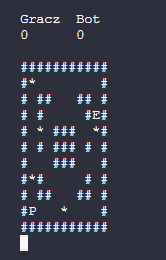
ctime

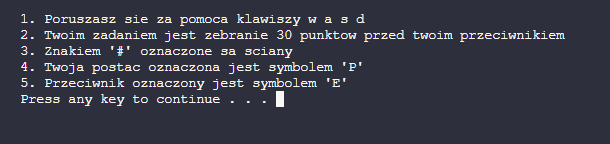
* srand(time(NULL)) służy randomizacji seeda na podstawie którego losowane są liczby

random

* rand() zwraca losową liczbę

**4. Instrukcja użytkownika.**





**5. Podsumowanie i wnioski.**

*Udało się zrealizować projekt jako aplikację konsolową, jednakże można to rozwinąć przy użyciu bibliotek graficzny, dzięki temu unikniemy mrugania planszy przy jej ciągłym rysowaniu na nowo. Trudnością była odpowiednia implementacja algorytmu bfs ze względu na potrzeby projektu. Inną trudnością z którą spotkałem się na samym początku było zwracanie dwuwymiarowych tablic których początkowo chciałem używać. Jednak po zgłębieniu tematu stwierdziłem że lepszą i bardziej komfortową opcją będzie operowanie na wektorach, które mają w siebie wbudowane funkcje takie jak dynamiczna alokacja pamięci. Kolejnym napotkanym problemem są ruchy wykonywane przez bota, które z niewiadomego mi są blokowane przez ruchy gracza dlatego przy dalszym rozwoju projektu można się wgłębić w ten temat. Ostatnim co mi przychodzi do głowy jest stworzenie generatora losowych plansz gdzie przy wykorzystaniu algorytmu bfs można by sprawdzać poprawność generacji tzn. Czy nie powstały jakieś obszary do których nie da się dostać.*