**Podręcznik użytkownika programu**

**Opis zasad gry**

Gra jest komputerową wersją klasycznego Saper. Celem gracza jest odkrycie wszystkich pól na planszy, na których nie znajdują się miny, oraz oznaczenie wszystkich min za pomocą flag. Jeśli gracz odkryje pole z miną, gra zostaje zakończona przegraną.

Na planszy gracz może wykonać następujące ruchy:

* **Odkrycie pola**: wpisanie polecenia r x y (r – otwarcie, x i y - współrzędne wybranego pola (np. r 3 5)).
* **Postawienie flagi**: wpisanie polecenia f x y (f- postawienie flagi, x i y - współrzędne wybranego pola (np. f 2 4)).

Podczas gry, plansza wyświetlana jest w widoku siatki, gdzie:

* Pola niewidoczne oznaczane są znakiem #.
* Pola z flagą oznaczane są znakiem F.
* Pola odkryte mogą zawierać liczbę min w sąsiedztwie lub być puste.

**Sposób uruchomienia programu**

**Kompilacja**

Gra jest kompilowana poprzez podanie polecenia „make” w terminale Linux’owym. Jest to dodane dla ułatwienia kompilacji programu przy zmianach.

Zawartość ścieżki kompilującej: gcc –Wall –g proj.c sotw.c –o project\_output

1. **Uruchomienie w trybie gry manualnej:** Program uruchamiamy bez dodatkowych parametrów:

./project\_output

Po uruchomieniu gracz zostanie poproszony o wybranie poziomu trudności:

* + **1:** Plansza 9x9 z 10 minami
  + **2:** Plansza 16x16 z 40 minami
  + **3:** Plansza 16x30 z 99 minami
  + **4:** Plansza i ilość min jest podana przez gracza. Przy podaniu ilości min większej od samej planszy program poprosi o ponowne wpisywanie tej wielkości

1. **Uruchomienie w trybie automatycznym:** Program przyjmuje jako argument ścieżkę do pliku z wcześniej zdefiniowaną planszą oraz ruchami:

./project\_output -a plansza.txt

Plik powinien zawierać:

* + Wymiary planszy (n, m)
  + Zawartość planszy (gdzie -1 oznacza minę, a 0 pole puste)
  + Kolejne ruchy w formacie: x y

Po zakończeniu gry w trybie automatycznym jest wypisana linijka, zawierająca trzy liczby: liczbę poprawnych kroków, liczbę punktów oraz czy gra zakończyła się powodzeniem (0 - niepowodzenie, 1 – wygrana)

**Opis poszczególnych opcji**

* **Wybór poziomu trudności:** Pozwala dostosować planszę i liczbę min do doświadczenia gracza.
* **Wyniki:** Po zakończeniu gry, gracz może wpisać swoje imię, a jego wynik zostanie zapisany w pliku Score2.txt wraz z datą i czasem zakończenia gry. Wynik jest mnożony przez poziom trudności oprócz własnego poziomu „4”. Jeśli gracz wchodzi w top 5 graczów, jego imie będzie wypisane wraz z czwora innymi graczami i zapisane do innego pliku Score.txt.
* **Przerwanie gry:** W każdej chwili gracz może wpisać komendę CTRL+Z, aby zakończyć rozgrywkę bez zapisywania wyniku.

**Szczegóły implementacji programu**

**Podział programu na moduły**

Program został podzielony na następujące moduły:

1. **Główny plik programu (proj.c):** Zawiera logikę gry, obsługę wejścia użytkownika i wyświetlanie planszy.
2. **Moduł funkcji pomocniczych (sotw.h i sotw.c):**
   * spawnmines: Losowe rozmieszczanie min na planszy.
   * write\_b: Wyświetlanie aktualnego stanu planszy z kolorowym oznaczeniem.
   * wheremines: Obliczanie liczby min w sąsiedztwie danego pola.
   * rasp: Rekurencyjne odkrywanie pustych pól.
   * check: Sprawdzanie, czy gra została wygrana.

**Interfejs kluczowych funkcji**

1. **Funkcja spawnmines**
   * **Opis:** Umieszcza miny na planszy w losowych miejscach.
   * **Prototyp:**

void spawnmines(int\*\* a, int n, int m, int level);

* + **Parametry:**
    - a: wskaźnik na planszę.
    - n, m: wymiary planszy.
    - level: poziom trudności.

1. **Funkcja rasp**
   * **Opis:** Rekurencyjnie odkrywa puste pola oraz oznacza pola sąsiadujące z liczbami.
   * **Prototyp:**

void rasp(int\*\* a, char\*\* b, int n, int m, int o, int p);

* + **Parametry:**
    - a: wskaźnik na planszę gry.
    - b: widok gry dla gracza.
    - n, m: wymiary planszy.
    - o, p: współrzędne aktualnego pola.

1. **Funkcja check**
   * **Opis:** Sprawdza, czy wszystkie pola niebędące minami zostały odkryte.
   * **Prototyp:**

int check(int\*\* a, int n, int m, int \*points);

* + **Parametry:**
    - a: wskaźnik na planszę gry.
    - n, m: wymiary planszy.
    - points: licznik punktów gracza.

**Opis ważniejszych struktur**

* **Struktura player** Przechowuje dane o graczu:
* typedef struct {
* char name[255];
* int score;

} player;

* **Tablice dynamiczne:**
  + Plansza gry (a) jest reprezentowana jako dynamicznie alokowana tablica dwuwymiarowa typu int.
  + Widok gry dla gracza (b) jako tablica typu char.

**Mechanizmy zarządzania pamięcią**

* Wszystkie tablice są dynamicznie alokowane przy użyciu malloc.
* Pamięć jest zwalniana po zakończeniu gry, aby uniknąć wycieków pamięci.
* Obsługa błędów alokacji została zaimplementowana w celu zapewnienia stabilności programu.

**Opis i wyniki testów**

Program został przetestowany w następujących przypadkach:

1. **Tryb manualny:**
   * Działanie na wszystkich poziomach trudności.
   * Obsługa błędów wejścia (np. niepoprawne współrzędne, nielegalne ruchy).
2. **Tryb automatyczny:**
   * Wczytywanie planszy z pliku i analiza wyników gry.
   * Wyniki (liczba ruchów, wynik, status zwycięstwa) poprawne dla przykładowych plików testowych.
3. **Stres-testy:**
   * Długotrwałe rozgrywki na dużych planszach.
   * Testowanie różnych kombinacji ruchów gracza.

**Podział pracy w zespole**

**Artsiom Markevich: implementacja zasad gry i planszy**

**Serafim Sidubayeu: obsługa wczytywania z pliku tekstowego, obsługa flag**

**Podsumowanie**

**Czy wszystkie funkcjonalności udało się zaimplementować?**

Tak, wszystkie planowane funkcjonalności zostały zaimplementowane, w tym:

* Obsługa dynamicznych plansz.
* Kolorowe oznaczenia na planszy.
* Zapisywanie wyników graczy.

**Problemy i sposoby ich rozwiązania**

1. **Problem:** Błędy przy alokacji dynamicznej tablic.
   * **Rozwiązanie:** Dokładne sprawdzanie wyników funkcji malloc i poprawne zwalnianie pamięci.
2. **Problem:** Niepoprawne wypisywanie ilości min przy kątach na już otwartych miejscach
   * **Rozwiązanie:** Korekta indeksów w funkcji wheremines.

**Wnioski**

* Dynamiczna alokacja pamięci umożliwiła elastyczną implementację gry.
* Dalszym krokiem może być dodanie interfejsu graficznego i lepsza obsługa błędów.
* Projekt pozwolił nam pogłębić znajomość zarządzania pamięcią w C oraz pracy zespołowej.

Bibliografia:

Artsiom Markevich, Serafim Sidubayeu