Sprawozdanie Projekt Saper

Mikołaj Frąckowiak i Artur Arciszewski

Spis treści

Cel zadania

Celem zadania było napisanie w języku C programu symulującego grę w sapera. Miał on być rozbity na moduły i spełniać wymagania opisane w instrukcji.

Podejście do rozwiązania

Podeszliśmy do problemu napisania programu gry saper poprzez rozbicie ją na moduły. Główny plik zawiera implementację wszystkich funkcji wraz ze stworzeniem struktury wyglądu gry. Łączy on pozostałe pliki aby umożliwić spójną grę. Plik minefield zawiera implementacje kluczowych funkcji potrzebnych do działania samej gry. Plik scoreboard przechowuje funkcje do zapisywania i uzupełniania rankingu wyników, natomiast w pliku filegame znajdują się funkcje umożliwiające czytanie gry z pliku. Funkcje zawarte w pliku steps pozwalają na utrzymywanie poprzednich kroków co umożliwia ich zapis do pliku. Napisane w sobie funkcje zawierają testy sprawdzające za każdym razem poprawność danych co upewnia poprawność działania kodu i zapobiega wszelkim błędom.

Podręcznik użytkowania

Wprowadzenie

Gra w Saper to tekstowa implementacja klasycznej gry logicznej. Celem gry jest odkrycie wszystkich bezpiecznych pól na planszy przy jednoczesnym unikaniu min. Program pozwala na grę w różnych trybach trudności, wczytywanie i zapisywanie stanu gry oraz śledzenie wyników w tablicy najlepszych graczy.

Zasady gry

1. Plansza składa się z pól, z których niektóre zawierają miny.
2. Gracz ma do wyboru trzy akcje:
    - **Odkrycie pola** (r x y): Odkrywa pole o współrzędnych (x, y).
        - Jeśli pole zawiera minę, gra kończy się przegraną.
        - Jeśli pole nie zawiera min, ujawniana jest liczba min w sąsiedztwie.
    - **Postawienie/zdjęcie flagi** (f x y): Oznacza pole o współrzędnych (x, y) jako podejrzane o zawieranie miny lub usuwa flagę.
    - **Zapis stanu gry** (s filename): Zapisuje aktualny stan gry do pliku filename.
3. Gra kończy się wygraną, gdy wszystkie pola bez min zostaną odkryte.

Sposób uruchomienia programu

Wymagania systemowe

- Kompilator C kompatybilny z GCC (np. MinGW na Windows lub GCC na Linux/MacOS).
- Terminal lub konsola systemowa.

Kompilacja programu

Aby skompilować program, użyj następującego polecenia w terminalu:

gcc -o saper main.c minefild.c scorebord.c filegame.c steps.c -lm

alternatywnie można użyć plik makefile komendą: *make*

Uruchamianie programu

Program może być uruchamiany w dwóch trybach:

1. **Nowa gra** (domyślny tryb): ./saper
2. **Kontynuacja gry z pliku**: ./saper -f <nazwa_pliku>
    - Gdzie <nazwa_pliku> to ścieżka do pliku zapisanego wcześniej.
    - Jeśli użytkownik chciałby sam stworzyć plik do wczytania tak powinna wyglądać jego struntura:
        1. <x y> - wymiary tabeli oddzielone spacją.
        2. „m m mm m” – ciąg małych liter ‘m’ oraz spacji, gdzie litery oznaczają położenie min.
        3. „r 1 1 f 2 3” – ciąg kolejnych komend do wykonania
    - Jeśli użytkownik dostarczy plik z błędem zostanie on wyświetlony na terminalu, a dane zostaną najprawdopodobniej błędnie wczytane

Opcje programu

Tryby trudności

Po uruchomieniu nowej gry program zapyta o wybór poziomu trudności:

- **1 - Łatwy**: Plansza 9x9 z 10 minami.
- **2 - Średni**: Plansza 16x16 z 40 minami.
- **3 - Trudny**: Plansza 16x30 z 99 minami.
- **4 - Własny**: Użytkownik podaje wymiary planszy oraz liczbę min.

Sterowanie podczas gry

1. **Wykonywanie ruchów**:
    - Aby odkryć pole: r <x> <y> (np. r 3 4).
    - Aby postawić flagę: f <x> <y> (np. f 5 7).
2. **Zapis stanu gry**:

- Aby zapisać grę: s <nazwa_pliku> (np. s gra_zapis.txt).

3. **Podpowiedzi**:
    - Program wyświetla komunikaty o błędach, takich jak niepoprawne koordynaty, złe komendy czy format wejścia.

Tablica wyników

Po zakończeniu gry użytkownik może wpisać swoje imię, aby zapisać wynik. Najlepsze wyniki są przechowywane w tablicy wyników.

Funkcje tablicy wyników

- **Zapis wyniku**: Po zakończeniu gry wynik gracza zostanie automatycznie dodany do tablicy.
- **Wyświetlenie najlepszych wyników**: Program automatycznie wyświetla tablicę najlepszych wyników po zakończeniu gry.

Informacje dodatkowe

- Program obsługuje zapisywanie i wczytywanie stanu gry, co pozwala kontynuować grę w dowolnym momencie.
- Plansza jest wyświetlana w terminalu z kolorową grafiką ułatwiającą identyfikację flag, min oraz odkrytych pól.
- Gra sprawdza poprawność wprowadzanych danych i informuje o ewentualnych błędach.

Implementacja kodu

Plik *steps.c*

Struct CommandLog

Przechowuje wszystkie komendy wykonane przez gracza w trakcie gry. Pola:

- **char *data**: Wskaźnik na dynamicznie alokowany ciąg znaków, który zawiera zapis wszystkich komend.
- **int length**: Bieżąca długość zapisanych komend w buforze.
- **int capacity**: Maksymalna pojemność bufora przed kolejnym rozszerzeniem

void concatenate_to_string

Funkcja tworzy jeden ciąg znaków, łącząc jeden znak oraz dwie liczby całkowite. Używana jest do tworzenia zwięzłego zapisu danych w formacie tekstowym. Implementacja:

- Liczby są konwertowane na ciągi znaków za pomocą funkcji itoa.
- Wynikowy ciąg jest tworzony przez sprintf w formacie "%c %s %s".

CommandLog* init_command_log

Inicjalizuje strukturę CommandLog, która przechowuje log komend wykonanych w grze. Struktura CommandLog działa jak bufor tekstowy, który dynamicznie rozszerza swoją pojemność.

Implementacja:

- Alokowana jest pamięć dla struktury CommandLog oraz dla początkowego bufora o rozmiarze 256 znaków.
- Bufor jest inicjalizowany jako pusty ciąg (\0)

void add_command_to_log

Dodaje nową komendę do istniejącego logu. Jeśli brakuje miejsca w buforze, pojemność logu jest dynamicznie zwiększana.

Implementacja:

- Sprawdzane jest, czy aktualny bufor ma wystarczająco dużo miejsca.
- Jeśli nie, bufor jest podwajany przy użyciu realloc.
- Komenda jest dodawana do bufora za pomocą strcat, a na końcu dodawana jest spacja jako separator.
- Długość logu jest odpowiednio aktualizowana

void free_command_log

Zwolnienie pamięci zajmowanej przez strukturę CommandLog.

Implementacja:

- Zwalniana jest pamięć dla bufora data.
- Następnie zwalniana jest pamięć struktury CommandLog

Plik *minefield.c*

Struct minefild

Przechowuje dane dotyczące stanu gry na planszy Saper. Pola:

- **int *cover**: Tablica jednowymiarowa reprezentująca stan pól na planszy. Możliwe wartości:
    1. 1 – pole osłonięte,
    2. 2 – pole oznaczone flagą,
    3. 0 – pole odkryte.
- **int *mines**: Tablica jednowymiarowa przechowująca rozmieszczenie min i liczby sąsiednich min na planszy. Możliwe wartości:
    1. MINE (9) – pole z miną,
    2. 0-8 – liczba min w sąsiedztwie.
- **int x**: Liczba wierszy planszy.
- **int y**: Liczba kolumn planszy.
- **int multi**: Wartość punktowa za odkrycie jednego pola (może zależeć od poziomu trudności).
- **int score**: Aktualny wynik gracza, rośnie wraz z odkrywaniem bezpiecznych pól.

minefild* minefild_innit

Inicjalizuje planszę do gry w Saper. Tworzy planszę o podanych wymiarach i umieszcza określoną liczbę min.

Implementacja:

- Alokuje pamięć na strukturę minefild oraz tablice do przechowywania osłony (cover) i rozmieszczenia min (mines).
- Resetuje wartości planszy do stanu początkowego.
- Losowo rozmieszcza miny, unikając kolizji.

minefild* minefild_for_file

Tworzy planszę do gry na podstawie danych wczytanych z pliku. Oczekuje poprawnie zapisanego pliku zawierającego wymiary planszy i rozmieszczenie min.

Implementacja:

- Wczytuje wymiary planszy (x, y) i alokuje odpowiednią strukturę.
- Iteruje przez dane w pliku, przypisując miny do odpowiednich pozycji.

void minefild_free

Zwalnia pamięć zajmowaną przez strukturę minefild. Wywoływana po zakończeniu pracy z planszą, aby uniknąć wycieków pamięci.

Implementacja:

- Zwalnia pamięć dla tablic cover, mines oraz samej struktury.

int minefild_cord_to_ind

Przekształca współrzędne planszy (x, y) na indeks w tablicy jednowymiarowej.

Implementacja:

- Zwraca indeks zgodnie z formułą base->y * x + y.

void minefild_print

Wyświetla aktualny stan planszy w terminalu. Wykorzystuje znaki (#, F, M, .) do graficznej reprezentacji planszy.

Implementacja:

- Iteruje przez planszę, drukując odpowiednie znaki zależnie od stanu pola (miny, osłony, flagi).
- Zawiera obsługę kolorów ANSI dla lepszej czytelności.

int minefild_open

Odkrywa pole na planszy (x, y).

Implementacja:

- Aktualizuje stan pola w tablicy cover.
- Jeśli mina jest na polu, zwraca 1 (przegrana).
- Rekurencyjnie odkrywa pola sąsiadujące, jeśli na odkrytym polu nie ma min wokół.

void minefild_sopen

Wykonuje pierwszy ruch, przesuwając minę, jeśli wybrano pole z miną. Gwarantuje, że pierwszy ruch gracza nie kończy się przegraną.

Implementacja:

- Losowo przesuwa minę w inne miejsce, jeśli na wybranym polu znajduje się mina.

void minefild_flag

Ustawia lub usuwa flagę na polu (x, y). Implementacja:

- Przełącza stan pola między osłoniętym (1) a oznaczonym flagą (2).

void minefild_to_file

Zapisuje aktualny stan planszy oraz log komend do pliku. Plik może być później wykorzystany do odtworzenia gry.

Implementacja:

- Zapisuje wymiary planszy i rozmieszczenie min.
- Dodaje log komend jako ostatnią część zapisu.

int minefild_check_board

Sprawdza stan gry, określając, czy gra jest zakończona.

Założenia:

- 0 – gra trwa.
- 1 – przegrana (odkryto pole z miną).
- 2 – wygrana (wszystkie pola bez min są odkryte).

Implementacja:

- Iteruje przez planszę, sprawdzając stan pól.

Plik *filegame.c*

int f_move(□inefield* game, FILE* source)

Odczytuje i wykonuje pojedynczy ruch na planszy na podstawie danych z pliku. Komendy są zapisane w pliku w formacie <komenda> <x> <y>, gdzie:

- <komenda> to ‘r’ (odkryj pole) lub ‘f’ (postaw flagę),
- <x> i <y> to współrzędne pola.

Implementacja:

- Wczytuje komendę i współrzędne z pliku przy użyciu fscanf.
- Waliduje wczytane dane, sprawdzając poprawność komendy oraz zakres współrzędnych.
- Wykonuje komendę, np. odkrywa pole (minefild_open) lub oznacza je flagą (minefild_flag).

Zwraca:

- 1, jeśli w pliku nie ma więcej komend (EOF),
- 0, jeśli wykonano poprawną operację,
- 1, jeśli ruch odkrył pole z miną (przegrana).

Minefild* file_game

Rozgrywa całą grę na podstawie danych z pliku wejściowego, wczytując planszę i kolejne ruchy. Plik wejściowy zawiera dane o planszy (minefild_for_file) i listę komend. Gra kończy się, gdy:

- Gracz odkryje pole z miną (przegrana),
- Wszystkie pola bez min zostaną odkryte (wygrana),
- W pliku zabraknie komend.

Implementacja:

- Tworzy planszę, wczytując jej dane z pliku przy użyciu minefild_for_file.
- W pętli wykonuje kolejne ruchy (f_move) i sprawdza stan gry (minefild_check_board).
- Zlicza liczbę poprawnych kroków (poprawne_kroki) oraz aktualizuje wynik (game->score).
- Po zakończeniu gry wyświetla statystyki i wynik rozgrywki (wygrana lub przegrana).
- Zwraca wskaźnik do ukończonej planszy (minefild).

Plik *scorebord*

scorebord* scorebord_load

Wczytuje tablicę wyników (scoreboard) z pliku i tworzy jej reprezentację w pamięci. Jeśli plik zapisu (SAVE_FILE) nie istnieje, tworzy pustą tablicę wyników. Plik zawiera liczbę wpisów, wyniki oraz przypisane do nich imiona graczy.

Implementacja:

- Otwiera plik w trybie odczytu.
- Jeśli plik nie istnieje, alokuje pustą tablicę wyników.
- Wczytuje liczbę wpisów i iteruje przez kolejne rekordy, odczytując wyniki i imiona graczy.
- Zwalnia zasoby pliku po zakończeniu.

Void scorebord_add

Dodaje nowy wynik do tablicy wyników i sortuje ją w kolejności malejącej. Wynik gracza jest dopisywany na końcu, a następnie przesuwany na właściwą pozycję w zależności od wartości wyniku.

Implementacja:

- Rozszerza tablice wynik i gracz o jedno miejsce, używając realloc.
- Przypisuje nowy wynik i imię gracza na końcu tablicy.
- Przesuwa elementy, aby zachować posortowaną kolejność wyników (malejącą).

Void scorebord_save

Zapisuje aktualny stan tablicy wyników do pliku. Plik zapisu (SAVE_FILE) przechowuje liczbę wpisów, a następnie kolejne wyniki i przypisane imiona graczy.

Implementacja:

- Otwiera plik w trybie zapisu.
- Zapisuje liczbę wyników (n), a następnie iteruje przez tablicę wyników, zapisując każdy rekord w formacie <wynik> : <imię>.
- Po zapisaniu zwalnia pamięć tablicy wyników.

Void scorebord_print

Wyświetla tablicę wyników w terminalu. Wyświetla tylko najlepsze 5 wpisów. Format wyświetlania to „<pozycja>. <imię> : <wynik>”.

Implementacja:

- Iteruje przez tablicę wyników i drukuje kolejne rekordy w kolejności od najlepszych.

Int load_name

Wczytuje pojedyncze imię gracza.

Implementacja:

- Iteruje przez znaki w pliku i zapisuje je w buforze (retVal), aż napotka \n, EOF lub osiągnie maksymalną długość.

Zwraca:

- -1, jeśli osiągnięto koniec pliku,
- 0, jeśli wczytano pełne imię,
- 1, jeśli wciąż są dane do odczytania.

Plik main.c

void clear_stdin

Opróżnia bufor wejściowy (stdin) po niepoprawnym wprowadzeniu danych przez użytkownika. Usuwa wszystkie znaki do końca linii w celu uniknięcia problemów z kolejnymi operacjami wejścia.

Minefild* setup

Inicjalizuje planszę do gry na podstawie wybranego poziomu trudności. Użytkownik wybiera jeden z czterech poziomów trudności lub podaje własne parametry planszy.

Implementacja:

- Dla poziomów łatwy/sredni/trudny używa predefiniowanych ustawień (rozmiar, liczba min).
- Dla niestandardowego poziomu prosi użytkownika o wymiary planszy i liczbę min, sprawdzając poprawność danych.

Int move

Obsługuje pojedynczy ruch użytkownika (odkrycie pola, postawienie flagi, zapis stanu gry do pliku).

Implementacja:

- Wczytuje komendę i odpowiednie parametry (x, y lub nazwę pliku).
- Waliduje poprawność danych.
- Dla komendy 's' zapisuje stan gry i log do pliku.
- Dodaje ruch do logu (CommandLog) i wykonuje odpowiednią akcję (minefild_open, minefild_flag).

void start_move

Wykonuje pierwszy ruch w grze, z gwarancją, że wybrane pole nie zawiera miny.

Implementacja:

- Wczytuje współrzędne pierwszego ruchu.
- Dodaje ruch do logu.
- Wywołuje funkcję minefild_sopen w celu odkrycia pola.

void kont_game

Kontynuuje grę na istniejącej planszy (np. po wczytaniu z pliku). Obsługuje kolejne ruchy gracza, aż do wygranej, przegranej lub zapisania gry.

Implementacja:

- Tworzy nowy log komend.
- Iteruje, wykonując kolejne ruchy (move) i drukując stan planszy (minefild_print).
- Po zakończeniu gry wczytuje imię gracza i zapisuje wynik do tablicy wyników.

void hand_game

Obsługuje całą rozgrywkę dla nowej gry. Gracz rozpoczyna nową grę, wykonuje ruchy i zapisuje wynik po zakończeniu.

Implementacja:

- Inicjalizuje planszę (setup) oraz log komend.
- Obsługuje ruchy gracza, aktualizując stan planszy i sprawdzając wynik (minefild_check_board).

- Po zakończeniu rozgrywki wczytuje imię gracza i aktualizuje tablicę wyników.

int main

Uruchamia program i obsługuje argumenty wiersza poleceń. Jeśli podano argument -f <plik>, wczytuje zapisany stan gry z pliku. Natomiast jeśli jest brak argumentów, rozpoczyna nową grę.

Implementacja:

- Wczytuje i analizuje argumenty (getopt).
- Jeśli plik jest podany, wczytuje planszę i kontynuuje grę jeśli to możliwe (kont_game).
- W przeciwnym razie rozpoczyna nową grę (hand_game).

Podział pracy w zespole

Artur zajmował się napisaniem plików minefild oraz scorbord natomiast Mikołaj napisał pliki filegame oraz steps. Obydwoje zajmowaliśmy się pisaniem pliku main.c gdzie Artur zajął się częścią związaną z samą grą w terminalu oraz tabelą wyników, a Mikołaj implementacją czytania gry z pliku, możliwością jej kontynuowania oraz opcją jej zapisu. Mikołaj był również odpowiedzialny za napisanie sprawozdania do projektu.

Podsumowanie

Napisanie programu gry saper było ciekawym i wymagającym podsumowaniem semestru pracy w języku C. Wymagało użycia informacji nabytych na zajęciach i nie tylko. Program nie sporządził trudności jakich nie udało nam się rozwiązać, jednak implementowanie kolejnych funkcji wymagało czasami powrotu do już napisanego kodu aby dokonać zmian bądź napisania go od nowa aby umożliwić działanie obecnie tworzonej funkcji. Program był rozbudowany i wymagający, ale znajdował się w naszym zasięgu możliwości.