| SPRAWOZDANIE | |
|---|---|
| **Tytuł projektu:** | **Wykonali:** |
| *Minesweeper.*<br>*Gra w Sapera.* | 1. *Anastasiia Dmytrenko*<br>2. *Jan Szulc* |

# Spis treści

# Podręcznik użytkownika programu

***Opis zasad gry:***

Minesweeper to klasyczna gra logiczna, w której celem jest oczyszczenie planszy z ukrytych min, nie detonując żadnej z nich. Gra rozpoczyna się z zakrytą planszą zawierającą miny rozłożone losowo. Podstawowe zasady:

- Każda komórka na planszy może być pusta, zawierać minę lub numer wskazujący liczbę min w sąsiednich komórkach.
- Kliknięcie na komórkę odsłania jej zawartość. Jeśli komórka zawiera minę, gra kończy się przegraną.
- Jeśli odkryta komórka nie ma sąsiednich min, odsłaniane są sąsiednie komórki aż do komórek, które mają miny w sąsiedztwie.
- Aby oznaczyć komórkę, którą podejrzewasz o minę, możesz umieścić na niej flagę.
- Gra kończy się wygraną, gdy wszystkie komórki bez min są odsłonięte.

**Budowanie programu:**

Program Minesweeper korzysta z pliku Makefile, co umożliwia szybkie i łatwe budowanie projektu. Aby skompilować program, wystarczy wykonać w konsoli poniższe polecenie:

*make*

W wyniku tego procesu powstanie plik wykonywalny o nazwie minesweeper.

**Uruchamianie programu:**

Po skompilowaniu aplikacji można ją uruchomić za pomocą polecenia:

*make run*

To polecenie automatycznie uruchomi grę Minesweeper w trybie domyślnym.

**Testowanie programu:**

Dla celów testowych przygotowano oddzielny zestaw testów jednostkowych. Aby je uruchomić, należy wykonać:

*make test*

W wyniku tego polecenia zostaną uruchomione wszystkie zdefiniowane testy, a ich wyniki zostaną wyświetlone w konsoli.

**Czyszczenie plików pomocniczych:**

Aby usunąć wszystkie pliki obiektowe (*.o) oraz pliki wykonywalne, można skorzystać z polecenia:

*make clean*

To polecenie pozwala utrzymać porządek w katalogu projektu, usuwając tymczasowe pliki wygenerowane podczas procesu kompilacji.

**Sposób uruchamiania programu:**

Program Minesweeper można uruchomić za pomocą konsoli, wpisując następujące polecenie:

*./minesweeper [opcje]*

Dostępne są następujące opcje uruchamiania:

1. **-i** : uruchomienie gry w trybie interaktywnym. Program poprosi o podanie wymiarów planszy, liczby min i innych parametrów bezpośrednio w konsoli.
2. **-f <nazwa_pliku>** : wczytanie planszy z pliku. Plik powinien zawierać predefiniowaną planszę w określonym formacie.
3. Opcje można łączyć, na przykład:

*./minesweeper -i -f plansza.txt*

W powyższym przypadku gra uruchomi się w trybie interaktywnym, ale plansza zostanie wczytana z pliku "plansza.txt".

**Format pliku planszy:**

Plik planszy (np. plansza.txt) powinien mieć następujący format:

*wiersze: <liczba wierszy>*

*kolumny: <liczba kolumn>*

*‘Enter’*

*pozycje_min:*

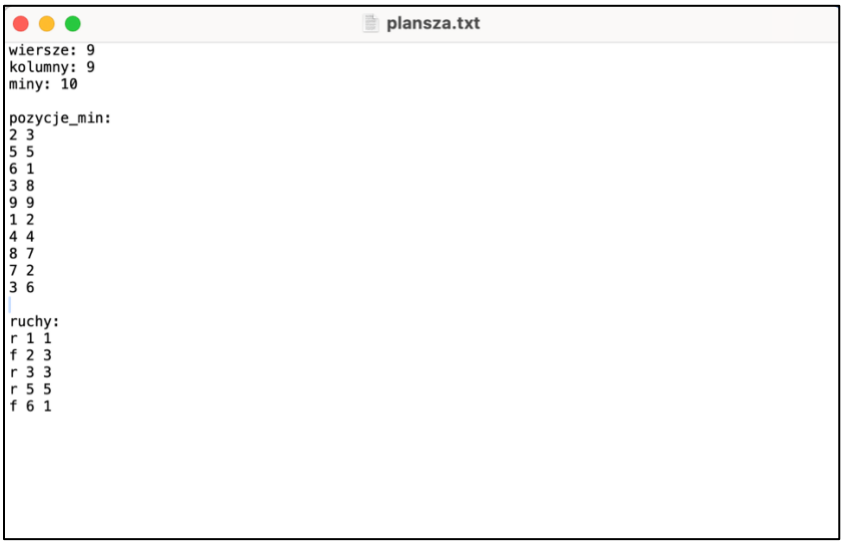
*‘pozycje min w formacie x y (kolumna wiersz), np.:’ 2 3*

*‘Enter’*

*ruchy:*

*‘ruchy w formacie r/f x y (kolumna wiersz), np.:’ r 12 3*

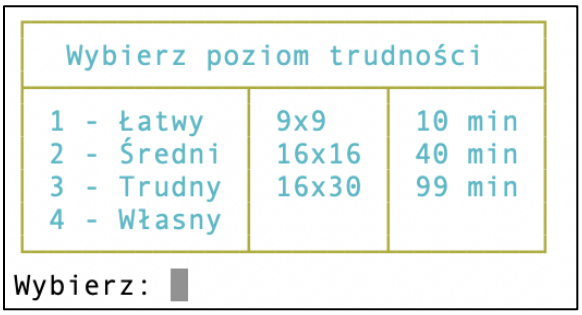
Przykład pliku planszy:




**Wybór poziomu trudności:**

Poziom trudności w grze może być wybrany na początku każdej sesji gry. Użytkownik może wybrać między 4 poziomami trudności:

- Łatwy
- Średni
- Trudny
- Własny




Każdy poziom różni się wielkością planszy oraz liczbą min na niej rozmieszczonych.

Jeśli wybierana jest opcja „Własna”, możemy ręcznie podać wymiary planszy oraz liczbę min. Każdy wymiar ma być większy od 5. W razie wprowadzenia błędnych danych program o tym poinformuje i powróci do wyboru poziomu trudności.

**Rozgrywka:**

Podczas gry użytkownik może wykonywać następujące komendy:

1. ***r <x> <y>:*** Odkrycie pola na pozycji (x, y).
2. ***f <x> <y>:*** Oznaczenie pola na pozycji (x, y) flagą (jako podejrzane o minę).

Przy każdym ruchu program aktualizuje planszę i wyświetla jej obecny stan. Gra kończy się, gdy wszystkie pola bez min zostaną odkryte lub użytkownik odkryje minę.

**Mechanizm gry i zapisywanie wyników:**

- **Aktualizacje podczas gry:** Podczas gry, stan planszy oraz liczba pozostałych min jest aktualizowana na bieżąco w interfejsie użytkownika.
- **Wyświetlanie końcowego wyniku:** Po zakończeniu gry, czy to przez wykrycie wszystkich min, czy przez detonację miny, program wyświetla końcowy wynik, który liczy się ze wzoru *liczba_odsłoniętych_pól x mnożnik_trudności_planszy.*
- **Zapisywanie wyników:** Każdy końcowy wynik jest zapisywany w wyznaczonym pliku *Wyniki.cfg* po podaniu przez użytkownika imienia i nazwiska.
- **Lista najlepszych wyników:** Po zakończeniu sesji gry, program wyświetla listę pięciu najlepszych wyników zarejestrowanych do tej pory, co pozwala na rywalizację i śledzenie postępów między sesjami gry.

## Szczegóły implementacyjne

**Podział programu na moduły:**

Program Minesweeper został podzielony na szereg modułów, z których każdy odpowiada za określony aspekt funkcjonalności gry. Podział ten zapewnia lepszą czytelność, ułatwia debugowanie oraz umożliwia dalsze rozszerzanie programu. Kluczowe moduły to:

1. **Moduł główny (main.c)**:
    - Odpowiada za logikę rozgrywki.
    - Zarządza komunikacją z użytkownikiem oraz obsługuje wprowadzanie komend.
    - Wywołuje funkcje z innych modułów w celu obsługi planszy, wyników i wyświetlania.
2. **Moduł ładowania planszy (loader.c, loader.h)**:
    - Obsługuje wczytywanie planszy z plików w formacie tekstowym.
    - Zapewnia walidację poprawności danych wejściowych (rozmiar planszy, liczba min).
3. **Moduł zarządzania planszą (array.c, array.h)**:
    - Tworzy dynamiczne tablice reprezentujące planszę gry oraz pola z minami.
    - Obsługuje manipulację danymi planszy, w tym oznaczanie odkrytych pól i flag.
4. **Moduł wyświetlania (display.c, display.h)**:
    - Odpowiada za wyświetlanie aktualnego stanu gry w konsoli.
    - Zapewnia czytelny interfejs użytkownika.
5. **Moduł wyników (scores.c, scores.h)**:
    - Zarządza zapisami wyników w pliku konfiguracyjnym (Wyniki.cfg).
    - Obsługuje wyświetlanie najlepszych wyników graczy.
6. **Moduł instrukcji (instruction.c, instruction.h)**:
    - Wyświetla pomoc oraz informacje o obsłudze gry na żądanie użytkownika.

**Interfejs kluczowych funkcji:**

Każdy moduł udostępnia zestaw funkcji, z których najważniejsze to:

1. ***void runGame(int x, int y, int minesCount, int countingPoints)*** *(main.c)*:
    - Implementuje główną pętlę gry.
    - Obsługuje komendy użytkownika (odkrywanie pól, oznaczanie flag, zakończenie gry).
2. ***char*** generateArray(int x, int y, const char* initialValue)*** *(array.c)*:
    - Tworzy dynamiczną tablicę 2D reprezentującą planszę.
    - Wypełnia tablicę domyślnym symbolem.
3. ***int loadBoardFromFile(const char* filePath)****(loader.c)*:
    - Wczytuje planszę z pliku i zapisuje dane do podanych wskaźników.
    - Sprawdza poprawność formatu pliku.
4. ***void printDisplay(char*** display, int x, int y, int minesCount, int flagsCount)*** *(display.c)*:
    - Wyświetla aktualny stan planszy w konsoli.
5. ***void saveResult(int points)*** *(scores.c)*:
    - Zapisuje wynik gracza do pliku wyników.
6. ***void displayTopScores()*** *(scores.c):*
    - Aktualizuje istniejące wyniki graczy, sortuje je oraz wypisuje 5 najlepszych graczy w danej sesji.
7. ***void instruction()*** *(instruction.c)*:
    - Wyświetla szczegółowe instrukcje dotyczące gry.

**Wazniejsze struktury danych:**

Program wykorzystuje dynamiczne tablice 2D do reprezentacji planszy oraz układu min. Kluczowe elementy to:

- ***Tablica char*** display***: Reprezentuje aktualny widok planszy (odkryte pola, nieodkryte pola, flagi).
- ***Tablica char*** mines***: Przechowuje informacje o rozmieszczeniu min na planszy.
- ***struct Score***: Przechowuje w strukturze dane gracza – jego imię, nazwisko oraz wynik końcowy gry.

**Opis testów**:

Plik test_functions.c zawiera implementację testów jednostkowych, które weryfikują działanie kluczowych funkcji programu Minesweeper. Testy są zaprojektowane, aby zapewnić poprawność działania podstawowych elementów takich jak generowanie planszy, rozmieszczanie min oraz sprawdzanie warunków zwycięstwa.

1. **Test funkcji *test_generateArray***
    - **Cel**: Sprawdzenie, czy funkcja generateArray poprawnie inicjalizuje dynamiczną tablicę o zadanych wymiarach i wypełnia ją odpowiednim wartością.
    - **Scenariusz**:

        - Tworzenie tablicy 3x3 wypełnionej wartością "[redacted]" (symbol nieodkrytego pola).
        - Iteracja przez wszystkie pola tablicy i porównanie ich z oczekiwaną wartością.
        - Zwolnienie pamięci po zakończeniu testu za pomocą freeArray.
    - **Wynik**:
        - Test przechodzi, jeśli wszystkie pola tablicy mają poprawne wartości.
        - Wyjście w przypadku sukcesu: [PASS] generateArray initializes the array correctly.
        - Wyjście w przypadku błędu: [FAIL] generateArray did not initialize correctly.
2. **Test funkcji *test_generateMines***
    - **Cel**: Weryfikacja poprawności rozmieszczania min na planszy przez funkcję generateMines.
    - **Scenariusz**:
        - Tworzenie planszy 5x5 i rozmieszczenie 3 min.
        - Iteracja przez wszystkie pola planszy i zliczenie liczby min (oznaczonych symbolem " o ").
        - Porównanie liczby faktycznych min z wartością oczekiwaną (3).
        - Zwolnienie pamięci (z komentarzem sugerującym pominięcie tego kroku w kodzie prezentacyjnym).
    - **Wynik**:
        - Test przechodzi, jeśli liczba min na planszy jest zgodna z oczekiwaną wartością.
        - Wyjście w przypadku sukcesu: [PASS] generateMines places the correct number of mines.
        - Wyjście w przypadku błędu: [FAIL] generateMines placed X mines instead of Y.
3. **Test funkcji *test_isWin***
    - **Cel**: Sprawdzenie, czy funkcja isWin poprawnie ocenia, czy gra została wygrana.
    - **Scenariusz**:
        - Inicjalizacja planszy 3x3 oraz tablicy min.
        - Dodanie miny na pole (0, 0) i oznaczenie tego pola jako nieodkryte.
        - Wywołanie funkcji isWin i analiza wyniku:
            - Gra nie powinna być uznana za wygraną, jeśli istnieje nieodkryte pole, które nie zawiera miny.
    - **Wynik**:
        - Test przechodzi, jeśli funkcja poprawnie wykrywa, że gra nie została wygrana.
        - Wyjście w przypadku sukcesu: [PASS] isWin correctly detects game is not won.
        - Wyjście w przypadku błędu: [FAIL] isWin incorrectly reports game as won.

**Struktura testów:**

Każdy test jest niezależny i rozpoczyna się od inicjalizacji wymaganych danych wejściowych (np. dynamiczne tablice). Po wykonaniu testu zasoby są zwalniane, aby uniknąć wycieków

pamięci. Wyniki testów są wyświetlane w konsoli, co pozwala na szybką ocenę poprawności działania programu.

**Podsumowanie wyników:**

Wszystkie zaimplementowane testy skutecznie weryfikują kluczowe funkcjonalności programu:

- Poprawna inicjalizacja tablic (planszy gry).
- Prawidłowe rozmieszczanie min na planszy.
- Poprawna detekcja zwycięstwa w grze.

Testy stanowią solidną podstawę do dalszej rozbudowy programu i gwarantują, że podstawowe funkcje działają zgodnie z oczekiwaniami.

## Podział pracy w zespole

W projekcie Minesweeper uczestniczyły dwie osoby: **Anastasiia** i **Jan**. Tak wyglądał szczegółowy podział obowiązków oraz zakres współpracy nad poszczególnymi etapami projektu:

**1. Wspólna praca nad algorytmem**:

- Wspólnie opracowaliśmy koncepcję algorytmu generowania planszy oraz mechanizmu działania gry.
- Dyskusje dotyczyły przede wszystkim logiki rozmieszczania min oraz funkcjonalności odkrywania i oznaczania pól.

**2. Implementacja logiki generowania planszy**:

- Jako bardziej doświadczony programista, Janek zrealizował implementację logiki generowania planszy.
- Opracował funkcje *generateArray* (tworzenie planszy) oraz *generateMines* (rozmieszczanie min w losowych lokalizacjach) oraz wizualnie estetyczne wyświetlanie planszy gry.

**3. Implementacja funkcji obliczania wyników i zarządzania graczami**:

- Anastasiia zajęła się implementacją funkcji odpowiedzialnych za:
    - Obliczanie punktów gracza w czasie rzeczywistym.
    - Zarządzanie listą graczy oraz tworzeniem rankingu.
    - Zapis i odczyt wyników do/z pliku Wyniki.cfg.

**4. Tworzenie testów aplikacji**:

- Janek opracował zestaw testów jednostkowych w pliku *test_functions.c.*
- Testy obejmowały sprawdzanie poprawności generowania planszy, rozmieszczania min, obsługi wyników oraz warunków zwycięstwa.

**5. Tworzenie Makefile oraz obróbka argumentów wywołania**:

- Anastasiia utworzyła plik Makefile, upraszczając proces budowania aplikacji.
- Zaimplementowała obsługę argumentów wywołania programu, takich jak:
  - Wyświetlanie instrukcji gry (-i).
  - Wczytywanie planszy z pliku (-f <nazwa_pliku>).

**6. Podział na moduły**:

- Wspólnie zaplanowaliśmy podział programu na moduły
  (np. display, array, scores, loader) w celu zachowania czytelności i organizacji kodu.
- Przeprowadziliśmy testy integracyjne dla wszystkich modułów.

**7. Dodanie elementów kolorystycznych**:

- Janek dodał elementy kolorystyczne w wyświetlaniu planszy oraz wyników w konsoli, poprawiając estetykę i czytelność interfejsu.

**8. Sprawozdanie**:

- Anastasiia napisała szczegółowe sprawozdanie do projektu, opisując wszystkie aspekty techniczne i organizacyjne pracy. Janek pomógł z ogarnięciem działów takich jak opis użytych struktur danych, ważniejszych funkcji oraz opis testów.

## Podsumowanie

Pracując nad Minesweeperem, mieliśmy jasny cel: stworzyć grę, która nie tylko działa poprawnie, ale jest też intuicyjna i przyjemna w odbiorze. Teraz, kiedy patrzymy na efekt końcowy, możemy śmiało powiedzieć, że udało nam się osiągnąć to, co zaplanowaliśmy, choć po drodze napotkaliśmy kilka wyzwań.

**Czy wszystkie funkcjonalności zostały zaimplementowane?**

Tak, wszystkie kluczowe funkcjonalności zostały zaimplementowane. Gra oferuje generowanie planszy o różnych rozmiarach, losowe rozmieszczanie min, oznaczanie pól flagami oraz obliczanie wyników. Użytkownik ma możliwość wczytania planszy z pliku, co było jednym z bardziej zaawansowanych elementów projektu. Dodatkowo zadbaliśmy o estetykę wyświetlania, wprowadzając kolorowe elementy w konsoli, co znacząco podniosło jakość interfejsu mimo tego, że program jest realizowany w powłoce tekstowej.

**Problemy i wyzwania:**

Największym wyzwaniem było opracowanie algorytmu, który gwarantowałby, że pierwsze kliknięcie gracza nigdy nie trafi na minę. Spędziliśmy sporo czasu na próbach, zanim udało się opracować rozwiązanie, które działało. Kolejnym wyzwaniem była obsługa plików – nie tylko musieliśmy stworzyć algorytm wczytywania zapisanych plansz, ale też zadbać o ich poprawną interpretację.

Także wymagało dużej uwagi zarządzanie pamięcią. Dynamiczne tablice były podstawą naszego programu, więc musieliśmy dokładnie pilnować, aby każda alokacja pamięci była później zwalniana.

**Jak rozwiązywaliśmy problemy?**

Nasza współpraca opierała się na otwartej komunikacji i wspólnym podejmowaniu decyzji. Gdy pojawiał się problem, siadaliśmy razem, analizowaliśmy sytuację i wymyślaliśmy możliwe rozwiązania.
Często korzystaliśmy z podejścia „iteracyjnego" – wdrażaliśmy niewielkie fragmenty kodu, testowaliśmy je, a następnie wprowadzaliśmy poprawki.

**Wnioski**:

Praca nad Minesweeper nauczyła nas wielu rzeczy. Po pierwsze, uświadomiliśmy sobie, jak ważna jest odpowiednia organizacja kodu. Podział na moduły nie tylko ułatwił pracę, ale także sprawił, że cały projekt stał się bardziej czytelny. Po drugie, nauczyliśmy się jeszcze bardziej wykorzystywać różne komendy oraz funkcje w języku C na praktyce – dzięki temu zauważyliśmy, że z każdym kolejnym etapem realizacji wymogów nasza praca stawała się szybsza, a pojawiające się błędy byliśmy w stanie szybciej wychwycić i naprawić. Wreszcie, praca zespołowa okazała się kluczowa dla sukcesu projektu. Wspólne omawianie problemów, dzielenie się obowiązkami i wzajemne wsparcie sprawiły, że projekt był nie tylko wyzwaniem technicznym, ale także przyjemnością oraz ważnym zarządzania czasem – dzięki temu ze pracowaliśmy we dwójkę byliśmy w stanie realizować wszystkie funkcjonalności o wiele szybciej.
To była świetna lekcja zarówno programowania, jak i pracy zespołowej!