GNU/LINUX Projekt zaliczeniowy - gra labirynt

Zofia Sikorska, Agata Leszczak

Czerwiec 2025

1 Wstęp

1.1 Cel Projektu:

Zaprezentowanie umiejętności zdobytych na zajęciach używając narzędzi takich jak:

- bash
- terminal
- GitHub
- LaTeX

1.2 Założenia projektu:

Stworzenie labiryntu używając basha oraz terminala. Przesłanie wszystkiego na repozytorium GitHuba oraz wykonanie opisu projektu w LaTeX.

2 Reguły gry

Gracz porusza się po labiryncie literą "P"używając do poruszania się klawiszy:

- w - góra ↑
- a - lewo ←
- s - dół ↓
- d - prawo →

Gracz po drodze zbiera punkty gdzie $ to 1 punkt a ♡ to 5 punktów. Aby ukończyć labirynt należy dojść do znaku "E". Labirynt składa się z 3 poziomów. Gra zawiera też tabelę punktacji z graczami i liczbą ich punktów oraz średnim czasem spędzonym nad poziomami.

3 Wyjaśnienie kodu

3.1 Zarządzanie danymi gracza

- player_name=" nazwa gracza
- current_level - aktualny poziom
- attempt - numer próby
- current_score - aktualna punktacja
- best_times - najlepszy czas dla poziomów
- level_times - czas przejść w bieżącej sesji
- current_start_time - czas rozpoczęcia poziomu

Algorytm load_level_scores

Wczytywanie najlepszychwyników gracza dla poszczególnych poziomów.

1. Sprawdzanie czy zmienna jest plikiem
2. Ustawianie separatora pól na przecinek i wczytywanie z każdej linii 3 zmiennych: name, level, hs_time.
3. Sprawdzanie czy wynik z pliku należy do gracza
4. Zapisywanie najlepszego wyniku
5. Przekazanie zawartości do pliku

Algorytm save_level_score

Zapisywanie najlepszych wyników gracza dla poszczególnych poziomów

1. Przyjmowanie 2 parametrów: poziomu i czasu, ktory osiągnął gracz
2. Tworzenie dwoch parametrów: tymczasowego pliku do przechowywania danych oraz flagi wsazującej czy wynik został zaktualizowany
3. Sprawdzanie czy nie ma zapisanego czasu dla danego poziomu lub czy nowy czas jest lepszy niż dotychczasowy → aktualizacja tablicy i zmiana flagi
4. Zapisywanie nowych lini do scorów
5. Jeśli flaga jest true, następuje dodanie nowego poziomu
6. Zastąpienie orginalnych plików tymczasowym

Algorytm save_overall_score

Zapisywanie i aktualizowanie ogólnego wyniku gracza

1. Inicjalizacja zmiennych total_time, total_score, temp_file oraz flagi
2. Sprawdzanie czy plik istnieje i jeśli tak to czytanie go dzieląc linie na trzy pola: name, score, time
3. Sprawdzanie czy wynik lub czas jest lepszy niż poprzedni, jeśli tak to jest on zapisywany dla temp file
4. Jeśli gracz nie występował wcześniej i zdobył wynikwiększy niż 0 to zostaje dodany do rankingu
5. Jeśli plik został zaktualizowany, sortujemy plik tymczasowy i zastępujemy stary plik nowym

Algorytm show_highscores

Wyświetlanie rankingów najlepszych graczy

1. Sprawdzanie czy plik ovaerall_fille istnieje
2. Wczytuje 5 najwyższych wyników
3. Obliczanie średniego czasu na poziom
4. Formatowane wyświetlenie znaków

Algorytm get_player_name

Pobranie nazwy gracza i przygtowanie podstawowych zmiennych

1. Czyszczenie ekranu i wyświetlanie nagłówka gry
2. Wczytywanie imienia, jeśli nie zostało podane imię jest przyznawane automatycznie jako Player1
3. Inicjalizacja zmiennych gry - resetowanie licznika punktów, pusta tabliza do przechowywania czasów przejścia poziomów, wywoływanie funkcji z najlepszymi wynikami gracza, ustawienie licznika prób

Algorytm show_image

1. Tworzenie zmiennej przechowywującej ścieżkę do pliku obrazu
2. Sprawdzanie czy plik istnieje
3. Otwieranie obrazka za pomocą programu xdg-open lub feh, sprawdzanie czy on istnieje → otwieranie pliku w domyślnej aplikacji → przekierowywanie wejścia stdout i stderr do "nicości"i uruchamianie procesu w tle
4. Zapamiętywanie procesu, który otworzył obrazu
5. Oczekiwanie 5 sekund i zamknięcie programu, który otworzył program

Algorytm show_stats

1. Przygotowanie dwóch linii tekstu z imieniem gracza oraz poziomem, liczną prób i aktualnym wynikiem
2. Sprawdzanie czy istnieje najlepszy czas i wyświetlenie go
3. Wyświetlenie aktualnego czasu

Algorytm find_start_position

1. Iteracja po wierszach labiryntu i przypisanie aktualnego wierszu do zmiennej line
2. Iteracja po znakach wierszu
3. Wyszukiwanie pozycji gracza perz wycinanie pojedyńczego znaku z wiersza i pozycji j i sprawdzanie czy to znak P
4. Zapisywanie pozycji gracza jako aktualnej pozycji i pozycji startowej (do resetowania poziomu)
5. Ustawianie domyślnej pozycji startowej jeśli P nie zostanie znalezione

Algorytm set_current_maze

Przygotowywanie bieżącego labiryntu dla aktualnego poziomu gry

1. Usuwanie istniejącej tablicy nagród
2. Tworzenie nowej globalnej tablicy do śledzenia zebranych nagród
3. Ustawianie odpowiedniego labiryntu, dla odpowiedniego poziomu
4. wywołanie find_start_position, które znajduje P w labiryncie

Algorytm move_player

1. Zainicjalizowanie parametrów do przesówania gracza
2. Rozpoczęcie pomiaru czasu gry
3. Sprawdzanie czy nowa pozycja gracza mieści się w labiryncie
4. Pobieranie znaku w docelowej pozycji
5. Obsługa wyjścia "E"
    - Aktualizacja pozycji gracza
    - Obliczenie czasu przejścia poziomu
    - Zapis wyniku poziomu
    - Wyświetlenie komunikatu o ukończeniu poziomu
    - Przejście do następnego poziomu (jeśli nie jest to poziom 3)

- Jeśli jest to ostatni poziom to zakończenie gry → obliczanie całkowitego czasu, zapisanie końcowego wyniki, pokazanie obrazku zwycięstwa
- Zapytanie o ponowną grę; jeśli tak to następuję resetacja parametrów

6. Zbieranie nagród
    - Sprawdzanie czy nagroda nie została zebrana wcześniej
    - Dodanie odpowiednich punktów za nagrody
    - Zapisanie pozycji zebranej nagrody
    - Aktualizacja pozycji gracza

Algorytm display_maze

Wyświetlanie aktualnego stanu labiryntu, gracza, zebranych nagród oraz instrukcji sterowania

1. Czyszczenie ekranu i pokazanie statystyk
2. Rysowanie ramki labiryntu
3. Wyświetlanie komórek labiryntu
    - Sprawdzanie pozycji gracza
    - Sprawdzanie pozycji startowej
    - Obsługa zebranych nagród → jeśli nagroda została zebrana zostaje zamieniona na puste pole
    - Wyświetlanie "#"jako grubszej ściany
4. Wyświetlanie instrukcji

Algorytm game_over

1. Czyszczenie ekranu i wyświetlanie komunikatu
2. Wyświetlanie obrazka porażki
3. Zapytanie o ponowną grę

Algorytm main_game_loop

1. Rozpoczęcie nieskończonej pętli
2. Rozpoczęcie funkcji display_maze
3. Wczytywanie pojedyńczych klawiszy
4. Obsługa sterowania klawiszami w, a, s, d, q
5. Jeśli status wynosi 1 (poziom ukończony) przechodzi do następnego poziomu

Na końcu znajduje się zwenętrzna pętla gry, w której znajdują sie funkjce do pobierania nazwy gracza, inicjalizacji labiryntu i uruchomienie głównej pętli do rozgrywki

Dodatkowy plik do tworzenia wykresu wyników graczy

1. Przygotowanie pliku CSV z danymi graczy (nazwa, punkty, czas) na podstawie pliku wyników
2. Konfiguracja parametrów wykresu w gnuplot:
   - Format wyjściowy: PNG o rozmiarze 801x600 px
   - Tytuł wykresu i etykiety osi
   - Styl histogramu z pełnymi słupkami
3. Generowanie histogramu:
   - Dane: kolumna punktów z pliku CSV
   - Etykiety osi X: nazwy graczy
   - Kolor słupków: niebieski (#3498db)
4. Zapis wykresu do pliku maze_stats.png
5. Wyświetlenie komunikatu potwierdzającego