Przewodnik po projekcie so_long

Spis treści

Wprowadzenie

Projekt so_long to prosta gra 2D, w której gracz porusza się po mapie, zbiera przedmioty i próbuje dotrzeć do wyjścia. Projekt wykorzystuje bibliotekę MLX do renderowania grafiki.

Struktury danych

Projekt wykorzystuje kilka kluczowych struktur:

```c
typedef struct s_point {
    size_t px_x;
    size_t px_y;
} t_point;

typedef struct s_sprite {
    int px_w;
    int px_h;
    void *img;
} t_sprite;

typedef struct s_player {
    t_point pos;
    t_point start_pos;
    t_sprite sprite;
} t_player;

typedef struct s_tile {
    char t;
    int v;
} t_tile;
```

```
typedef struct s_map {
    char *path;
    size_t g_h;
    size_t g_w;
    char **grid;
    t_tile **tiles;
    int exit_accessible;
    int fd;
    int accessible_collectibles;
} t_map;

typedef struct s_vars {
    void *mlx;
    void *win;
    void *img;
    char *addr;
    t_player player;
    t_map map;
    t_sprite w_sp;
    t_sprite c_sp;
    t_sprite s_sp;
    t_sprite e_sp;
    t_sprite f_sp;
    t_sprite p_sp;
    t_sprite yw_sp;
    t_sprite *digits_sp;
    BOOL won;
    int moves;
    int collected;
    int collectibles;
    BOOL exit_unlocked;
    int exit_found;
    int start_found;
    int bits_per_pixel;
    int line_length;
    int endian;
} t_vars;
```

Każda struktura ma swoje zastosowanie:
- `t_point`: Reprezentuje pozycję na mapie.
- `t_sprite`: Przechowuje informacje o sprite'ach (obrazkach).
- `t_player`: Zawiera informacje o graczu.
- `t_tile`: Reprezentuje pojedynczy kafelek na mapie.
- `t_map`: Przechowuje informacje o całej mapie.
- `t_vars`: Główna struktura przechowująca wszystkie zmienne stanu gry.

Główne funkcje

Inicjalizacja gry

```c
void init_game(t_vars *vars, char *map_path) {
    // Przypisanie ścieżki do pliku mapy
    vars->map.path = map_path;

    // Inicjalizacja licznika zebranych przedmiotów
    vars->collected = 0;

    // Ustawienie flagi odblokowania wyjścia na FALSE (wyjście zablokowane)
    vars->exit_unlocked = FALSE;

    // Inicjalizacja flagi dostępności wyjścia na FALSE (ścieżka do wyjścia nie znaleziona)
    vars->map.exit_accessible = FALSE;

    // Zerowanie licznika dostępnych przedmiotów do zebrania
    vars->map.accessible_collectibles = 0;

    // Zerowanie całkowitej liczby przedmiotów do zebrania na mapie
    vars->collectibles = 0;

    // Ustawienie flagi wygranej na FALSE (gra nie jest jeszcze zakończona)
    vars->won = FALSE;

    // Zerowanie licznika ruchów gracza
    vars->moves = 0;

    // Zerowanie licznika znalezionych wyjść
    vars->exit_found = 0;

    // Zerowanie licznika znalezionych punktów startowych
    vars->start_found = 0;
}

```

Główna pętla gry

```c
int main(int ac, char *av[]) {
    t_vars vars;

    // Sprawdzenie argumentów
    if (ac != 2) {
        map_error("Nieprawidłowa liczba argumentów.");
    }

    // Inicjalizacja gry
    init_game(&vars, av[1]);

    // Parsowanie mapy
    parse_map(&vars.map);
    fill_grid(&vars);

    // Inicjalizacja MLX
    vars.mlx = mlx_init();
    vars.win = mlx_new_window(vars.mlx, vars.map.g_w * SIZE, vars.map.g_h * SIZE,
WIN_NAME);

    // Ładowanie sprite'ów
    load_sprites(&vars);

    // Ustawienie hooków
    mlx_hook(vars.win, 2, 1L << 0, key_handler, &vars);
    mlx_hook(vars.win, 17, 1L << 0, close_window, &vars);
    mlx_loop_hook(vars.mlx, render, &vars);

    // Rozpoczęcie pętli gry
    mlx_loop(vars.mlx);
}
```

Parsowanie mapy

Parsowanie mapy odbywa się w kilku krokach:

1. Otwarcie pliku mapy
2. Zliczenie wymiarów mapy
3. Alokacja pamięci dla siatki mapy
4. Wczytanie zawartości mapy
5. Walidacja mapy

```c
int parse_map(t_map *map) {
    int fd;         // Deskryptor pliku
    char *line;     // Zmienna do przechowywania wczytanej linii

    // Otwarcie pliku mapy w trybie tylko do odczytu
    fd = open(map->path, O_RDONLY);

    // Sprawdzenie, czy plik został poprawnie otwarty
    if (fd < 0) {
        map_error("Nie znaleziono mapy.");
    }

    // Inicjalizacja wymiarów mapy
    map->g_h = 0;   // Wysokość mapy (liczba wierszy)
    map->g_w = 0;   // Szerokość mapy (liczba kolumn)

    // Wczytanie pierwszej linii z pliku
    line = get_next_line(fd);

    // Pętla wczytująca kolejne linie mapy
    while (line) {
        // Zwiększenie licznika wysokości mapy
        map->g_h++;

        // Jeśli to pierwsza linia, ustal szerokość mapy
        if (map->g_h == 1) {
            map->g_w = ft_linelen(line);
        }

        // Sprawdzenie, czy aktualna linia ma taką samą długość jak pierwsza
        if (ft_linelen(line) != map->g_w) {
            map_error("Mapa nie jest prostokątna.");
        }

        // Wczytanie kolejnej linii
        line = get_next_line(fd);
    }

    // Sprawdzenie, czy mapa nie jest pusta
    if (map->g_h == 0) {
        map_error("Pusty plik mapy.");
    }

    // Zwrócenie 1, jeśli parsowanie przebiegło pomyślnie
    return (1);
}
```

```
int fill_grid(t_vars *vars) {
    t_point g_pos;      // Struktura przechowująca aktualną pozycję na siatce
    char *line;         // Zmienna do przechowywania wczytanej linii

    // Inicjalizacja zmiennych do wypełniania mapy
    initiate_map_filling(vars, &g_pos);

    // Wczytanie pierwszej linii z pliku
    line = get_next_line(vars->map.fd);

    // Pętla wczytująca kolejne linie mapy
    while (line) {
        // Alokacja pamięci dla nowego wiersza w siatce
        allocate_line(vars, g_pos);

        // Pętla wypełniająca aktualny wiersz
        while (g_pos.px_x < vars->map.g_w) {
            // Wypełnienie kafelka danymi z wczytanej linii
            fill_tiles(vars, line, g_pos);

            // Zliczenie elementów na mapie (start, wyjście, przedmioty)
            count_grid(vars, vars->map.grid[g_pos.px_y][g_pos.px_x], g_pos);

            // Przejście do następnej kolumny
            g_pos.px_x++;
        }

        // Przejście do początku następnego wiersza
        g_pos.px_x = 0;
        g_pos.px_y++;

        // Wczytanie kolejnej linii
        line = get_next_line(vars->map.fd);
    }

    // Zamknięcie pliku mapy
    close(vars->map.fd);

    // Sprawdzenie, czy mapa jest otoczona ścianami
    if (walls_error(vars)) {
        map_error("Mapa nie jest otoczona ścianami.");
    }

    // Sprawdzenie, czy istnieje ścieżka do wyjścia i wszystkich przedmiotów
    check_path(vars->player.pos, vars);

    // Końcowe sprawdzenie poprawności mapy
    check_map(vars);
```

```
    // Zwrócenie 1, jeśli wypełnianie siatki przebiegło pomyślnie
    return (1);

```

Renderowanie gry

Renderowanie gry odbywa się w funkcji `render`:

```c
int render(t_vars *vars) {
    // Rysowanie tła gry
    draw_background(vars);

    // Sprawdzenie, czy gra nie została jeszcze wygrana
    if (!vars->won) {
        // Rysowanie elementów mapy (ściany, przedmioty, wyjście)
        draw_map(vars);

        // Rysowanie gracza na jego aktualnej pozycji
        draw_player(vars);

        // Wyświetlanie liczby wykonanych ruchów
        draw_moves(vars);
    } else {
        // Jeśli gra została wygrana, wyświetl ekran zwycięstwa
        mlx_put_image_to_window(vars->mlx, vars->win, vars->yw_sp.img, 0, 0);

        // Wyświetlanie końcowej liczby ruchów
        mlx_put_image_to_window(vars->mlx, vars->win, vars->digits_sp[(vars->moves /
100)].img,
                                1 * SIZE + 10, 3 * SIZE + 10);
        mlx_put_image_to_window(vars->mlx, vars->win, vars->digits_sp[(vars->moves / 10) %
10].img,
                                2 * SIZE + 10, 3 * SIZE + 10);
        mlx_put_image_to_window(vars->mlx, vars->win, vars->digits_sp[(vars->moves % 100)
% 10].img,
                                3 * SIZE + 10, 3 * SIZE + 10);
    }

    // Zwrócenie 0, aby kontynuować pętlę renderowania
    return (0);
}

```

Obsługa zdarzeń

Obsługa klawiszy odbywa się w funkcji `key_handler`:

```c
int key_handler(int keycode, t_vars *vars) {
    // Obsługa klawisza ESC (kod 53 lub 65307)
    if (keycode == 53 || keycode == 65307) {
        // Zamknięcie okna i zakończenie programu
        close_window(vars);
    }
    // Obsługa ruchu gracza, jeśli gra nie została jeszcze wygrana
    else if (!vars->won) {
        // Ruch w prawo (klawisz D)
        if (keycode == 2) {
            update_player_position(vars, (t_point){vars->player.pos.px_x + 1,
vars->player.pos.px_y});
        }
        // Ruch w lewo (klawisz A)
        else if (keycode == 0) {
            update_player_position(vars, (t_point){vars->player.pos.px_x - 1,
vars->player.pos.px_y});
        }
        // Ruch w górę (klawisz W)
        else if (keycode == 13) {
            update_player_position(vars, (t_point){vars->player.pos.px_x, vars->player.pos.px_y
- 1});
        }
        // Ruch w dół (klawisz S)
        else if (keycode == 1) {
            update_player_position(vars, (t_point){vars->player.pos.px_x, vars->player.pos.px_y
+ 1});
        }
    }

    // Zwrócenie 0, aby kontynuować obsługę zdarzeń
    return (0);
}

```

Aktualizacja pozycji gracza

```c
void update_player_position(t_vars *vars, t_point np) {
    // Zwiększenie licznika ruchów i wyświetlenie aktualnej wartości
    ft_printf("Total moves: %d\n", ++vars->moves);

    // Sprawdzenie, czy nowa pozycja mieści się w granicach mapy
    if (np.px_x < vars->map.g_w && np.px_y < vars->map.g_h) {
        // Sprawdzenie, co znajduje się na nowej pozycji
        if (vars->map.grid[np.px_y][np.px_x] == COLLECT) {
            // Zebranie przedmiotu
            vars->collected++;
            // Zamiana przedmiotu na podłogę na mapie
            vars->map.grid[np.px_y][np.px_x] = FLOOR;

            // Sprawdzenie, czy zebrano wszystkie przedmioty
            if (vars->collected == vars->collectibles) {
                // Odblokowanie wyjścia
                vars->exit_unlocked = 1;
            }

            // Aktualizacja pozycji gracza
            vars->player.pos = np;
        }
        // Sprawdzenie, czy gracz dotarł do odblokowanego wyjścia
        else if (vars->map.grid[np.px_y][np.px_x] == EXIT && vars->exit_unlocked) {
            // Aktualizacja pozycji gracza
            vars->player.pos = np;
            // Ustawienie flagi wygranej
            vars->won = 1;
        }
        // Sprawdzenie, czy nowa pozycja nie jest ścianą
        else if (vars->map.grid[np.px_y][np.px_x] != WALL) {
            // Aktualizacja pozycji gracza
            vars->player.pos = np;
        }
    }
}
```

Zarządzanie zasobami

Ładowanie sprite'ów:

```c
void load_sprites(t_vars *vars) {
    load_map_sprites(vars);
    load_digits_sprites(vars);
}

void load_map_sprites(t_vars *vars) {
    vars->p_sp.img = mlx_xpm_file_to_image(vars->mlx, "img/p_sp.xpm", &vars->p_sp.px_w,
&vars->p_sp.px_h);
    vars->w_sp.img = mlx_xpm_file_to_image(vars->mlx, "img/w_sp.xpm",
&vars->w_sp.px_w, &vars->w_sp.px_h);
    // Ładowanie pozostałych sprite'ów...
}
```

Wskazówki i najlepsze praktyki

1. Zawsze sprawdzaj, czy alokacja pamięci się powiodła.
2. Używaj stałych (np. `WIN_NAME`, `SIZE`) zamiast "magicznych liczb".
3. Dziel kod na mniejsze, łatwe do zarządzania funkcje.
4. Używaj struktur do organizacji powiązanych danych.
5. Pamiętaj o zwalnianiu zaalokowanej pamięci.
6. Obsługuj błędy i nieprawidłowe dane wejściowe.
7. Komentuj kod, szczególnie skomplikowane fragmenty.
8. Używaj narzędzi do debugowania, takich jak Valgrind, aby wykryć wycieki pamięci.

Przykład komentowania kodu:

```c
void update_player_position(t_vars *vars, t_point np) {
    // Zwiększ licznik ruchów i wydrukuj aktualną wartość
    ft_printf("Total moves: %d\n", ++vars->moves);

    // Sprawdź, czy nowa pozycja jest w granicach mapy
    if (np.px_x < vars->map.g_w && np.px_y < vars->map.g_h) {
        // Sprawdź, co znajduje się na nowej pozycji
        if (vars->map.grid[np.px_y][np.px_x] == COLLECT) {
            // Zbierz przedmiot
            vars->collected++;
```

```
            vars->map.grid[np.px_y][np.px_x] = FLOOR;

            // Sprawdź, czy zebrano wszystkie przedmioty
            if (vars->collected == vars->collectibles) {
                vars->exit_unlocked = 1;
            }

            // Aktualizuj pozycję gracza
            vars->player.pos = np;
        } else if (vars->map.grid[np.px_y][np.px_x] == EXIT && vars->exit_unlocked) {
            // Gracz dotarł do wyjścia i zebrał wszystkie przedmioty
            vars->player.pos = np;
            vars->won = 1;
        } else if (vars->map.grid[np.px_y][np.px_x] != WALL) {
            // Przesuń gracza na nową pozycję, jeśli nie jest to ściana
            vars->player.pos = np;
        }
    }
}
```

Ulepszenia i rozszerzenia projektu so_long

1. Optymalizacja kodu

1. Użycie bufora do renderowania:
   Zamiast renderować każdy element osobno, możemy użyć bufora do przygotowania całej klatki, a następnie wyświetlić ją za jednym razem.

```c
void render_to_buffer(t_vars *vars) {
    // Przygotuj bufor
    char *buffer = calloc(vars->map.g_w * vars->map.g_h * 4, sizeof(char));

    // Narysuj elementy do bufora
    draw_background_to_buffer(buffer, vars);
    draw_map_to_buffer(buffer, vars);
    draw_player_to_buffer(buffer, vars);

    // Wyświetl bufor na ekranie
    mlx_put_image_to_window(vars->mlx, vars->win, buffer, 0, 0);

    free(buffer);
}
```

2. Implementacja systemu cache'owania dla sprite'ów:
   Zamiast ładować sprite'y za każdym razem, gdy są potrzebne, możemy je załadować raz i przechowywać w pamięci.

```c
typedef struct s_sprite_cache {
    void *img;
    char *key;
} t_sprite_cache;

t_sprite_cache *sprite_cache;

void *get_sprite(char *key) {
    // Sprawdź, czy sprite jest w cache'u
    for (int i = 0; i < CACHE_SIZE; i++) {
        if (strcmp(sprite_cache[i].key, key) == 0) {
            return sprite_cache[i].img;
        }
    }

    // Jeśli nie, załaduj i dodaj do cache'u
    void *img = mlx_xpm_file_to_image(mlx, key, &width, &height);
    add_to_cache(key, img);
```

```
    return img;
}
```

2. Dodanie przeciwników z podstawowym AI

1. Dodaj nową strukturę dla przeciwników:

```c
typedef struct s_enemy {
    t_point pos;
    t_sprite sprite;
    int direction;  // 0: góra, 1: prawo, 2: dół, 3: lewo
} t_enemy;
```

2. Zaimplementuj funkcję do poruszania przeciwników:

```c
void move_enemies(t_vars *vars) {
    for (int i = 0; i < vars->enemy_count; i++) {
        t_point new_pos = vars->enemies[i].pos;

        // Losowy ruch
        int random_direction = rand() % 4;
        switch (random_direction) {
            case 0: new_pos.px_y--; break;  // góra
            case 1: new_pos.px_x++; break;  // prawo
            case 2: new_pos.px_y++; break;  // dół
            case 3: new_pos.px_x--; break;  // lewo
        }

        // Sprawdź kolizje
        if (is_valid_move(vars, new_pos)) {
            vars->enemies[i].pos = new_pos;
        }
    }
}
```

3. Dodaj sprawdzanie kolizji z przeciwnikami:

```c
void check_enemy_collision(t_vars *vars) {
    for (int i = 0; i < vars->enemy_count; i++) {
        if (vars->player.pos.px_x == vars->enemies[i].pos.px_x &&
            vars->player.pos.px_y == vars->enemies[i].pos.px_y) {
            vars->player.health--;
```

```
            if (vars->player.health <= 0) {
                game_over(vars);
            } else {
                reset_player_position(vars);
            }
        }
    }
}
```

3. Dodatkowe funkcjonalności

1. System poziomów:
   Dodaj możliwość przechodzenia do kolejnych poziomów po ukończeniu bieżącego.

```c
void load_next_level(t_vars *vars) {
    vars->current_level++;
    char level_path[50];
    sprintf(level_path, "maps/level_%d.ber", vars->current_level);

    // Zwolnij pamięć aktualnej mapy
    free_current_map(vars);

    // Załaduj nową mapę
    load_map(vars, level_path);

    // Zresetuj pozycję gracza i inne zmienne stanu
    reset_game_state(vars);
}
```

2. System punktacji:
   Dodaj system punktacji, który uwzględnia czas ukończenia poziomu i liczbę zebranych przedmiotów.

```c
void update_score(t_vars *vars) {
    int time_bonus = MAX_TIME - (get_current_time() - vars->level_start_time);
    vars->score += vars->collected * 100 + time_bonus;
}
```

3. Różne typy przedmiotów:
   Wprowadź różne typy przedmiotów do zbierania, każdy z inną wartością punktową lub efektem.

```c
```

```
enum item_type {
    COIN,
    POWERUP,
    KEY
};

typedef struct s_item {
    enum item_type type;
    int value;
    void (*effect)(t_vars *vars);
} t_item;

void collect_item(t_vars *vars, t_item item) {
    vars->score += item.value;
    if (item.effect != NULL) {
        item.effect(vars);
    }
}
```

4. Efekty dźwiękowe i muzyka:
   Dodaj proste efekty dźwiękowe dla akcji gracza i muzykę w tle.

```c
void play_sound(char *sound_file) {
    system(strcat("aplay ", sound_file));
}

// Użycie:
play_sound("sounds/collect.wav");
```