Dokumentacja do projektu na WDC pt. „Allegro Zelda Clone”

- Cezary Czubała

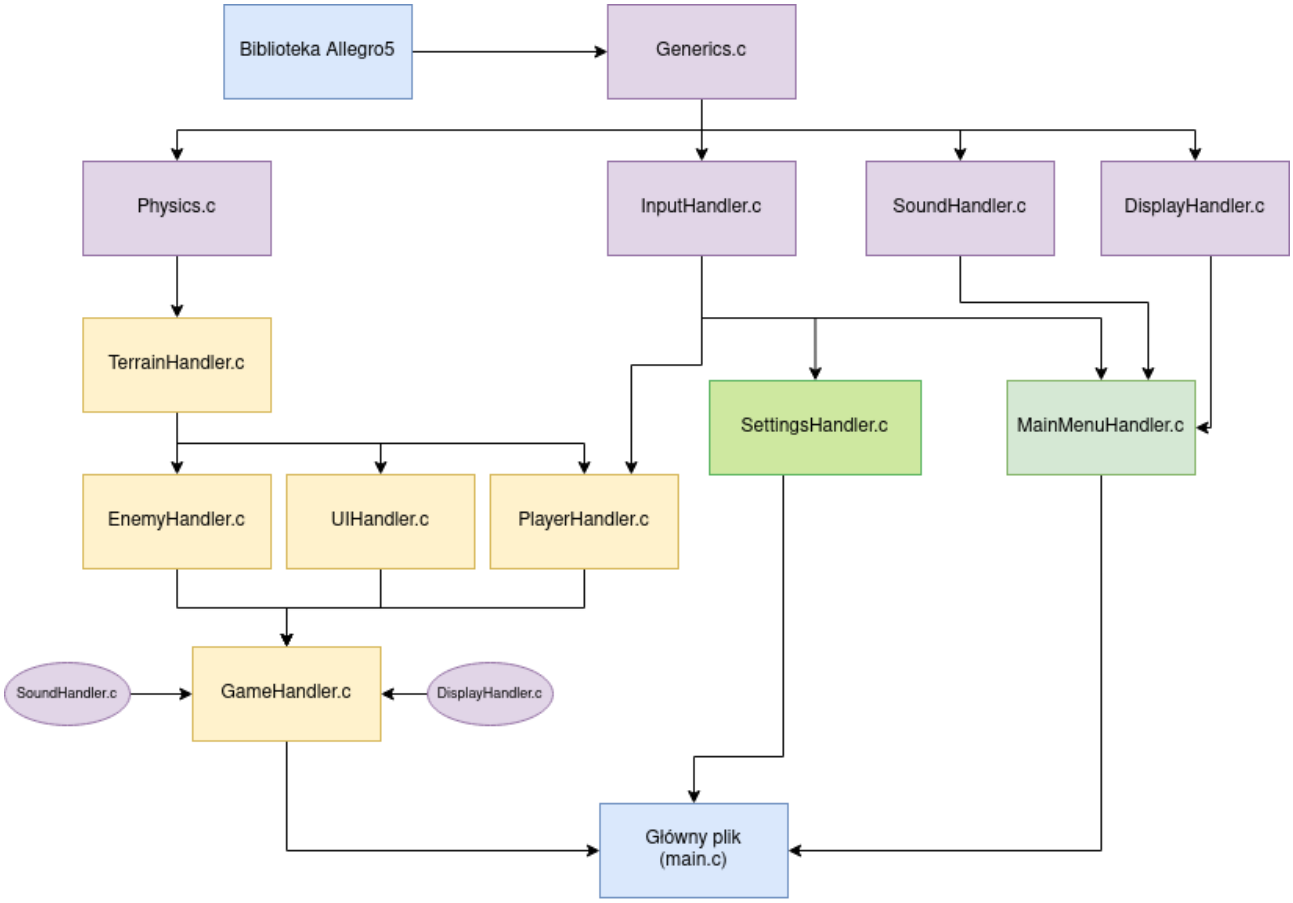
Spis treści

Wstęp

Niniejszy projekty został stworzony na potrzeby projektu zaliczeniowego na przedmiot „Wstęp do C”. Postanowiłem napisać grę inspirowaną grą „Legend of Zelda” która została wydana na konsolę NES w 1986r. Do napisania jej skorzystałem z biblioteki do pisania gier „Allegro5” (i oczywiście języka C), i została ona napisana na platformę Linux (Ubuntu).

W tym dokumencie opisaną są poszczególne moduło służące działaniu programu, ich funkcje, struktury itp.

Struktura gry



Krótki opis gry

W grze użytkownik porusza się graczem za pomocą klawiszy ruchów, a jego celem jest niszczenie przeciwników za których zdobywa się punkty i przechodzenie do kolejnego poziomu lochów. Każdy poziom lochów jest losowo generowany wg nasienia generowanego na początku gry. Przeciwnicy z czasem zyskuję maksymalną liczbę punktów życia i liczbę punktów ataku, czyniąc ich trudniejszych w zniszczeniu. Gracz natomiast co drugi poziom lochów zyskuję nowe maksymalne serce.

Niskopoziomowe moduły

Generics.h

Krótki opis modułu

Generic.h jest modułem który zawiera #include wszystkich potrzebnych bibliotek Allegro5 oraz parę innych podstawowych definicji używanych w reszcie modułów. Jest on zawarty we wszystkich podstawowych modułach.

Zmienne

ALLEGRO_FONT* GAME_FONT

Główna czcionka gry.

ALLEGRO_FONT* GAME_UI_FONT

Czcionka używania w interfejsie użytkownika.

ALLEGRO_TIMER* timer

Główny czasomierz gry.

ALLEGRO_EVENT_QUEUE* queue

Kolejka wydarzeń systemu gry.

long long GameSeed

Nasienie gry służące generowaniu plansz.

int Score

Licznik zdobytych punktów przez gracza.

char AssetDirectory[100]

Trzyma przedrostek ścieżki do katalogu z zasobami gry (grafiki i pliki audio).

LEVEL_TYPE CurrentLevel

Trzyma aktualną scenę gry.

bool ResetGameSettings

Zmienna określająca, czy należy zresetować zmienne growe (Score, generowanie przeciwników itd.)

Funkcje

void must_init(bool test, const char *description)

Służy sprawdzeniu czy plik/moduł został zainicjalizowany poprawnie. W razie błędnej inicjalizacji, drukuje do konsoli wiadomość o nie powodzeniu się inicjalizacji z nazwą modułu/pliku.

ALLEGRO_BITMAP* sprite_grab(ALLEGRO_BITMAP *sheet, int x, int y, int w, int h)

Bierze wskaźnik do załadowanej bitmapy, i zwraca fragment tej bitmapy jako osobną bitmapę zaczynający się w pozycji x,y i o wymiarach w,h.

void InitializeGenerics()

Wywoływana na początku, generuje nasienie, ustawia pewne inne zmienne, i ładuje moduły z biblioteki Allegro5.

int GetRandomDigit()

Zwraca losową cyfrę wg nasienia gry.

int Min(int x, int y)

Zwraca minimum z dwóch liczb typu int.

float Minf(float x, float y)

Zwraca minimum z dwóch liczb typu float.

float Maxf(float x, float y)

Zwraca maksimum z dwóch liczb typu float.

float Abs(float x)

Zwraca wartość bezwzględną liczby typu float.

Własne typy danych

struct Sprite

Struct trzymający wskaźnik do bitmapy.

enum LEVEL_TYPE

Możliwe wartości: Menu, Settings, Ingame. Określają aktualną scenę gry.

enum DIRECTION

Możliwe wartości: North, East, South, West. Określają kolejne kierunki kompasu.

DisplayHandler.h

Krótki opis modułu

Moduł odpowiadający za rysowanie grafik do ekranu i zmienianie trybu ekranu (pełnoekranowy/okienkowy).

Definicje

BUFFER_W

O wartości 256, definiuje szerokość buffer'a.

BUFFER_H

O wartości 176, definiuje wysokość buffer'a.

DISP_SCALE

O wartości 3, definiuje ile razy większy powinno być rzeczywiste okienko gry.

DISP_W

O wartości (BUFFER_W * DISP_SCALE), definiuje szerokość rzeczywistego okna gry.

DISP_H

O wartości (BUFFER_H * DISP_SCALE), definiuje wysokość rzeczywistego okna gry.

Zmienne

ALLEGRO_DISPLAY* Display

Wskaźnik do aktualnego okna gry.

bool Fullscreen

Zmienna pamiętająca czy gra jest aktualnie w trybie pełnoekranowym czy okienkowym.

Funkcje

void SwitchScreenMode()

Zmienia tryb ekranu na pełnoekranowy jeśli był okienkowy, i vice versa.

void disp_init()

Inicjalizuje okienko i moduły graficzne biblioteki Allegro5 przy rozpoczęciu gry.

void disp_deinit()

Niszczy okienko przy zamknięciu gry.

void disp_pre_draw()

Ustawia aktualne cel rysowania grafik na buffer który później zostanie ustawiony na rzeczywisty wynik okna.

void disp_post_draw()

Zamienia buffer i display, przedstawiając buffer z narysowanymi już grafikami.

InputHandler.h

Krótki opis modułu

Moduł zajmujący się wejściem gracza (klawiatura).

Definicje

KEY_SEEN

O wartości 1, służy rozstrzyganiu aktualnego stanu przycisku klawiatury

KEY_RELEASED

O wartości 2, służy rozstrzyganiu aktualnego stanu przycisku klawiatury (dokładnie gdy gracz puści aktualny przycisk).

Zmienne

unsigned char key[ALLEGRO_KEY_MAX]

Trzyma stany wszystkich dostępnych klawiszy klawiatury.

Funkcje

void keyboard_init()

Inicjalizuje klawiaturę i podpina ją do kolejki wydarzeń (aby wykrywać wciśnięcia klawiszy).

void keyboard_update(ALLEGRO_EVENT *event)

Jeśli aktualne wydarzenie jest wciśnięciem klawisza klawiatury, bądź jego puszczenie, to aktualizowany jest stan tego klawisza.

bool ButtonClicked(int keycode)

Zwraca prawdę jeśli klawisz został naciśnięty w danej klatce i fałsz w.p.p.

bool ButtonHeldDown(int keycode)

Zwraca prawdę jeśli klawisz jest trzymay i fałsz w.p.p.

SoundHandler.h

Krótki opis modułu

Moduł zajmujący się ładowaniem i puszczaniem muzyki i efektów dźwiękowych.

Zmienne

float MusicVolume

Trzyma aktualną głośność muzyki gry (w skali od 0 do 1)

float SFXVolume

Trzyma aktualną głośność efektów dźwiękowych gry (w skali od 0 do 1)

Funkcje

void init_audio()

Ładowanie poszczególnych plików audio i modułu dźwiękowego biblioteki Allegro5.

void ChangeMusicVolume(float newValue)

Zmień głośność muzyki.

void ChangeSFXVolume(float newValue)

Zmień głośność efektów dźwiękowych.

void PlaySound(enum Sounds sound)

Puść efekt dźwiękowy wg wartości enum Sounds sound

void PlayMusic(int MusicIndex)

Puść muzykę o danym indeksie (0 – Menu, 1 – W grze)

Własne typy danych

enum Sounds

O wartościach odpowiedających efektom dźwiękowym zawartych w grze.

Physics.h

Krótki opis modułu

Moduł zajmujący się fizyką gry. Trzyma definicje stałych fizycznych i sprawdza kolizje dwóch obiektów.

Funkcje

bool CheckCollision(BoxCollider A, BoxCollider B)

Sprawdza kolizję dwóch obiektów typu BoxCollider.

bool CheckCollisionF(BoxColliderF A, BoxColliderF B)

Sprawdza kolizję dwóch obiektów typu BoxColliderF.

void SetVector2(Vector2 *dest, int x, int y)

Ustawia wartości obiektu typu Vector2.

void SetVector2f(Vector2f *dest, float x, float y)

Ustawia wartości obiektu typu Vector2f.

Vector2 Vector2ToF(Vector2f src)

Zwróć nowy obiekt typu Vector2 będący zamienionym obiektem src (zaokrągla wartości w dół).

BoxCollider BCFtoBC(BoxColliderF src)

Zwróć nowy obiekt typu BoxCollider będący zamienionym obiektem src (zaokrągla wartości w dół).

void SetBoxCollider(BoxCollider *dest, int x, int y, int w, int h)

Ustawia wartości obiektu typu BoxCollider.

void SetBoxColliderF(BoxColliderF *dest, float x, float y, float w, float h)

Ustawia wartości obiektu typu BoxColliderF.

Vector2 GetDirectionVector(DIRECTION dir)

Zwraca jednostkowy Vector2 odpowiadający danemu kierunku typu DIRECTION.

bool EqualVectors(Vector2f a, Vector2f b, float MarginOfError)

Sprawdza czy dwa obiekty typu Vector2f są równe (uwzględniając pewny błąd o wartości MarginOfError).

bool EqualVectorsInt(Vector2 a, Vector2 b, int MarginOfError);

Sprawdza czy dwa obiekty typu Vector2 są równe (uwzględniając pewny błąd o wartości MarginOfError).

Własne typy danych

struct Vector2

Obiekt trzymający dwie wartości x,y typu int.

struct Vector2f

Obiekt trzymający dwie wartości x,y typu float.

struct BoxCollider

Obiekt trzymający dwie wartości Origin,Dimensions typu Vector2.

struct BoxColliderF

Obiekt trzymający dwie wartości Origin,Dimensions typu Vector2f.

Moduły Ingame (growe)

PlayerHandler.h

Krótki opis modułu

Moduł zajmujący się poruszaniem się gracza, jego animacjami, statystykami itp.

Zmienne

BoxColliderF PlayerCollider

Hitbox gracza

BoxColliderF SwordCollider

Hitbox miecza gracza

DIRECTION PlayerDirection

Aktualny kierunek w który gracz jest skierowany.

bool PlayerDead

Określa czy gracz nie żyje (ma zerową liczbę punktów życia)

Funkcje

void DrawPlayer()

Rysuje gracza na ekranie.

void HandlePlayer(ALLEGRO_EVENT *event, bool *done, bool *redraw)

Zajmuje się mechaniką gracza (sprawdza czy gracz się rusza lub atakuje).

void InitPlayer()

Przygotowuje gracza podstawową pozycję, ładuje jego grafiki.

void DeinitPlayer()

Usuwa grafiki gracza z pamięci (wywoływana pod koniec gry).

void DamagePlayer(int damage, Vector2f velocity)

Zadaj graczowi „damage” punktów obrażeń i ustawia jego prędkość na „velocity”.

void GivePlayerHealth(int health)

Daj graczowi „health” punktów życia.

bool IsPlayerDamaged()

Sprawdza, czy gracz jest aktualnie w stanie bycia obrażonym.

PlayerStats GetPlayerStats()

Otrzymaj statystyki gracza.

Vector2f GetPlayerPosition()

Otrzymaj pozycję gracza.

Własne typy danych

struct PlayerStats

Obiekt trzymający aktualną liczbę punktów życia gracza i maksymalną jaką może trzymać.

TerrainHandler.h

Krótki opis modułu

Moduł zajmujący się generowaniem terenu (lochów), kolizjami terenu i trzymający grafiki służące rysowaniu poszczególnych pokoi.

Zmienne

int CurrentFloor

Aktualny poziom lochów na którym gracz się znajduje.

bool ChangingFloors

Czy gra aktualnie zmienia poziom lochów.

Funkcje

void DrawTerrain()

Rysuje aktualny pokój w którym gracz przebywa.

void InitTerrain()

Inicjalizuje grafiki terenu i generuje poziom lochów.

void MoveLevel(Vector2 directionVec, DIRECTION dir)

Zmienia aktualny pokój na ten wskazany przez podany kierunek.

void EnteringNewRoom(BoxColliderF *PlayerCollider, Vector2f *PlayerPosition)

Sprawdza czy gracz wchodzi do nowego pokoju.

void UpdateColliders()

Aktualizuje hitboxy ścian w nowym pokoju.

bool CollidedWithWalls(BoxColliderF entity)

Sprawdza, czy dany obiekt jest w kolizji ze ścianami pokoju.

bool CollidedWithNewFloorTile(BoxColliderF player)

Sprawdza, czy gracz jest w kolizji z wejściem na kolejny poziom lochów.

Level *GetRoom(int x, int y)

Otrzymaj wskaźnik do pokoju o danych współrzędnych w aktualnych lochach.

Vector2 GetCurrentRoom()

Otrzymaj współrzędne aktualnego pokoju w którym gracz przebywa.

Własne typy danych

enum LevelType

Możliwe wartości: ROOM_TEST, ROOM_SPAWN, ROOM_ENEMY, ROOM_NEXT_FLOOR. Służy określeniu typu pokoju przy generacji lochów.

struct Level

Trzyma dane odnośnie aktualnego pokoju w lochach, tak jak indeksy kafelków i drzwi w danym pokoju.

EnemyHandler.h

Krótki opis modułu

Moduł zajmuje się generowaniem i mechanikami przeciwników w grze.

Funkcje

void LoadEnemyAssets()

Załaduj grafiki przeciwników.

void DrawEnemies()

Rysuj przeciwników na ekranie.

Void HandleEnemies()

Sprawdź kolizje przeciwników z graczem, mieczem gracza, ścianami i pomiędzy sobą, jak i rusz ich wg ich prędkości lub obierz dla danego przeciwnika nowy kierunek.

Void init_enemies()

Generuj nowych przeciwników na aktualnym poziomie lochów.

Własne typy danych

enum EnemyType

Możliwe wartości: KnightEnemy, Heart. Służy określeniu typu przeciwnika. Heart jest szczególnym typem przeciwnika który nie zadaje obrażeń graczowi, a daje przy kolizji z nim.

struct Enemy

Trzyma podstawowe informacje o danym przeciwniku, takie jak jego prędkość, hitbox, animacje, ile punktów obrażeń zadaje graczowi, jego punkty życia itp.

UIHandler.h

Krótki opis modułu

Rysuje interfejs użytkownika w grze (ilość jego punktów i ile zdrowia posiada gracz).

Funkcje

void init_ui()

Ładuje grafiki do UI i ustawia pewne podstawowe zmienne.

void DrawUI()

Rysuje interfejs użytkownika.

void DrawGameDeath()

Rysuje ekran śmierci po tym jak gracz zginął.

GameHandler.h

Krótki opis modułu

Skleja wszystkie moduły Ingame w jedną całość i wywołuje ich odpowiednie funkcje aby główna część gry działała.

Funkcje

void HandleGame(ALLEGRO_EVENT *event, bool *done, bool *redraw)

Zajmuje się głównymi mechanikami gry: zajmuje się graczem, przeciwnikami i kolizjami poprzez moduły niżej kompilowane.

void DrawGame()

Rysuje grę poprzez funkcje modułów niżej kompilowanych.

void InitiateGame()

Inicjalizuje wszystkie potrzebne zasoby do głównej części gry.

void DeinitGame()

Kasuje zasoby używane w grze (wywoływane na koniec gry).

void DrawNewFloorText()

Rysuje numer aktualnego poziomu lochów przy zmienianiu poziomu lochów.

Moduły MainMenu

MainMenuHandler.h

Krótki opis modułu

Moduł zajmujący się rysowaniem i mechanikami menu głównego gry, który wita gracza na początku gry.

Funkcje

void init_main_menu()

Ładuje zasoby głównego menu i ustawia podstawowe zmienne.

void HandleMainMenu(ALLEGRO_EVENT *event, bool *done, bool *redraw)

Zajmuje się głównymi mechanikami menu głównego.

void DrawMainMenu()

Rysuje menu główne gry.

void UpdateMainMenuInfo()

Aktualizuje opcje i informacje menu głównego.

Moduły Settings

SettingsHandler.h

Krótki opis modułu

Moduł zajmujący się rysowaniem i głównymi mechanika okna ustawień gry.

Funkcje

void init_settings_menu()

Ładuje zasoby okna ustawień.

void HandleSettings(ALLEGRO_EVENT *event, bool *done, bool *redraw)

Zajmuje się głównymi mechanikami okna ustawień (np. zmiana opcji przy naciśnięciu klawisza.

void DrawSettings()

Rysuje okno ustawień gry.

Główny plik programu

main.c

Main.c oczywiście skleja wszystkie niżej kompilowane moduły w jedną całość jaką jest gra. Moduły jakie main.c implementuje i wykonuje są przedstawione na drzewku dziedziczenia na początku dokumentu.