# Worms

dokumentacja

Zespół 7:
Adrian Gliźniewicz
Jakub Szydełko
Dawid Hebda

## 1. Założenia wstępne przyjęte w realizacji projektu

1. Założenia podstawowe
Gra pozwala graczowi poruszać się postacią po terenie. Gracz za pomocą broni jest w stanie oddać strzał z ustaloną mocą i ustalonym kierunku. Po wystrzale na pocisk działa siła grawitacji.
2. Założenia rozszerzone
W wersji rozszerzonej przede wszystkim dostajemy możliwość dynamicznego niszczenia terenu. Drugorzędnym celem jest gra wspólna z innymi osobami oraz kontrolowanie więcej niż jednej postaci. W razie możliwości zostanie dodane wiele typów broni, losowe generowanie mapy(X), menu główne(X) czy też…
networking :) (X)

## 2. Specyfikacja danych wejściowych

Specyfikacja danych wejściowych opisuje sposób, w jaki gra przetwarza sygnały z klawiatury gracza, wykorzystując architekturę ECS do zarządzania entity i komponentami. Integracja z systemami zarządzającymi zdarzeniami umożliwia płynne sterowanie postaciami oraz kamerą, reagując dynamicznie na akcje gracza i zapewniając interaktywność świata gry.

Legenda:

1)Poruszanie Gracza:
- A - poruszanie w lewo,
- D - poruszanie w prawo,
- SPACE - skok.

2)Obsługa broni:
- Q - zmiana broni,
- W - zmiana kąta nachylenia broni przeciwnie do ruchu wskazówek zegara,
- S - zmiana kąta nachylenia broni zgodnie z ruchem wskazówek zegara,
- Shift - wystrzelenie broni z siłą zależną od długości przytrzymania.

3)Poruszanie kamerą:
- Strzałka w górę - poruszenie kamery w górę,
- Strzałka w dół - poruszenie kamery w górę,
- Strzałka w prawo - poruszenie kamery w prawo,
- Strzałka w lewo - poruszenie kamery w lewo.

## 3. Opis oczekiwanych danych wyjściowych

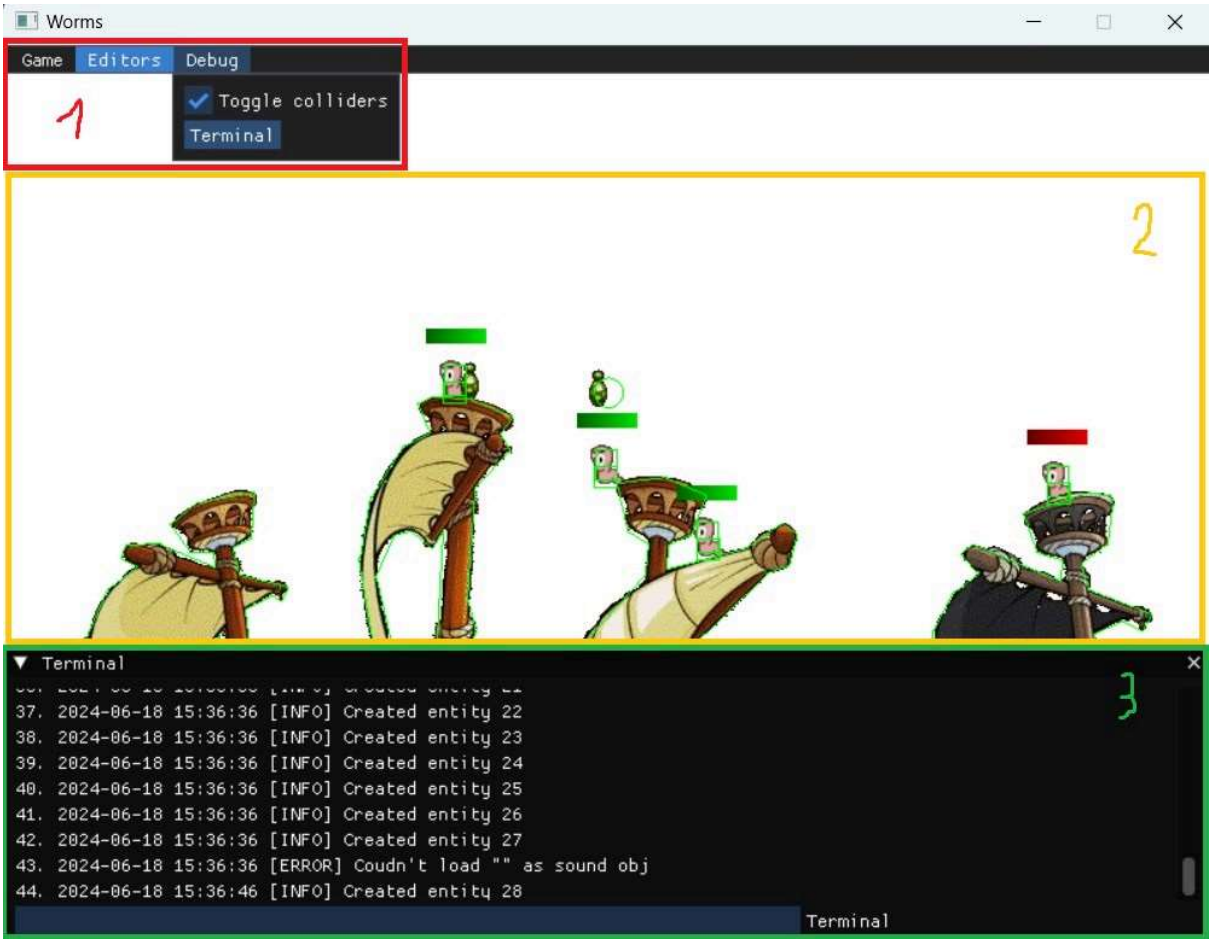
Gra pozwala graczowi na interaktywne działania takie jak poruszanie się postacią po terenie oraz wykonywanie skoków. Gracz może wybrać jedną z dwóch broni: granat, który wybucha po zadanym czasie oraz bazookę, której pocisk eksploduje natychmiast po trafieniu. Możliwe jest również oddawanie strzałów z określoną siłą. Pociski podlegają działaniu siły grawitacji, a ich eksplozja jest wizualnie reprezentowana. Po wybuchu pocisku teren w jego otoczeniu ulega zniszczeniu, a kolizje dostosowują się do aktualnego stanu.

Gra wspiera tryb wieloosobowy na jednym komputerze. Liczba drużyn jest stała i zdefiniowana bezpośrednio w kodzie. Każda drużyna składa się z wielu postaci, którymi gracze mogą naprzemiennie sterować. Kamera płynnie śledzi aktywnego gracza, przyspieszając, gdy znajduje się dalej od środka ekranu. Istnieje również możliwość ręcznego przesuwania kamery przez użytkownika.

## 4. Zdefiniowanie struktur danych

Struktury danych używane w tym projekcie obejmują dynamiczne tablice (std::vector), mapy asocjacyjne (std::unordered_map, std::map), opcjonalne wartości (std::optional), inteligentne wskaźniki (std::unique_ptr), funkcje (std::function), typy wyliczeniowe (enum), wszelkiego rodzaju klasu i wiele innych. Każda z tych struktur pełni kluczową rolę w zarządzaniu danymi i operacjami, zapewniając efektywność, bezpieczeństwo pamięci oraz czytelność kodu. Dodatkowo na potrzeby systemu ECS dodatkowo została stworzona struktura ComponentArray będąca wrapperem std::array. Rozszerzała ona jego funkcjonalność o zarządzanie komponentami aktywnych obiektów. Wśród nich znalazły się m.in. RigidBody, Position, Sprite. Wykorzystywane były także schematy programistyczne takie jak Singleton używany do tworzenia pojedynczych instancji klas oraz Factory wykorzystywane do łatwego i szybkiego tworzenia obiektów fizycznych.

# 5. Specyfikacja interfejsu użytkownika




1. Pasek Menu z przyciskami:
    1.1. Game - jeszcze nie zaimplementowany, w domyśle rozpoczęcie nowej gry i zmiana parametrów gry taki jak rozmiar mapy i ilości wormsów.
    1.2. Editors - jeszcze nie zaimplementowany - w domyśle miał zawierać narzędzia potrzebne programiście.
    1.3. Debug - zawiera opcję wizualizacji colliderów oraz włączenia terminala.
2. Główny ekran na którym toczy się gra.
3. Ukrywany terminal na którym wypisywane są informacje potrzebne do debugowania.

## 6. Wyodrębnienie i zdefiniowanie zadań

1. Analiza projektu, zdefiniowanie celów projektu oraz zadań.
2. Utworzenie repozytorium na Github, oraz konfiguracja projektu w Visual Studio
3. Podpięcie potrzebnych bibliotek.
4. Stworzenie okna aplikacji.
5. Implementacja systemu ECS (Entity Control System), oraz systemów odpowiedzialnych za poruszanie, strzelanie i kontrolowania życia wormsów.
6. Stworzenie interfejsu graficznego użytkownika oraz narzędzi do debugowania.
7. Opracowanie algorytmu Douglas-Peuckera do aproksymacji konturów terenu.
8. Testowanie programu i naprawa błędów.
9. Sporządzenie dokumentacji.

## 7. Decyzja o wyborze narzędzi programistycznych

Dla efektywnego przechowywania i wersjonowania projektu użyliśmy systemu kontroli Git. Natomiast sam kod był pisany w środowisku programistycznym Visual Studio 2022. W projekcie korzystaliśmy z trzech głównych bibliotek: SDL - do stworzenia okna aplikacji oraz rysowania na nim obrazów, Box2D - do implementacji i obliczeń fizycznych, oraz Dear ImGui - do stworzenia graficznego interfejsu użytkownika.

## 8. Podział pracy i analiza czasowa

1. tydzień: Analiza projektu i nauka potrzebnych bibliotek.
2. tydzień: System ECS.
3. tydzień: Stworzenie testów.
4. tydzień: Renderowanie terenu, implementacja systemów inputów i poruszania kamery oraz stworzenie pierwszego pocisku.
5. tydzień: Wizualizacja colliderów, oraz system drużyn.
6. tydzień: Pierwsza broń, paski życia oraz collider factory.
7. tydzień: Stworzenie obiektu broni.
8. tydzień: Systemy obiektów gry oraz odświeżania fizyki. Możliwość strzelania pociskami z broni.
9. tydzień: Dodanie paska siły wystrzału z broni, oraz terminalu do debugowania. Implementacja w pełni zniszczalnego terenu.
10. tydzień: Możliwość zmiany broni oraz parametrów pocisków. Dodanie systemu życia wormsów oraz systemu cząsteczek.
11. tydzień: Dodanie dźwięków oraz możliwości zniszczenia wormsa. Płynny ruch kamery.

# 9. Opracowanie i opis niezbędnych algorytmów

W grze **Worms Armagedon** można zauważyć wiele ciekawych algorytmów. Najbardziej charakterystycznym z nich jest ten odpowiadający za dynamiczne niszczenie terenu. Inspiracją do użytych algorytmów stanowiła gra Noita gdzie cały świat stworzony z pixeli ulega symulacji fizycznej zintegrowanej z pomocą Box2d. Gra ta osiągała efekt ten efekt używając kolejno algorytmów:

- **marching squares** - do przekonwertowania obrazka na jego kontur za pomocą kanału alfa. Metoda ta polegała na zapisaniu dla każdego pixela wartości zależnej od kanału alfa jego, jego prawego, dolnego i prawego dolnego sąsiada. Następnie z pomocą tej informacji dało się określić w którą stronę powinniśmy się kierować by dalej podążać wzdłuż konturu.
- **Douglas–Peucker** - do uproszczenia konturu do prostszej formy którą następnie można łatwo przekazać do Box2d jako statyczną fixturę. Metoda ta polega na sprawdzeniu odległości punktami a prostą wyznaczoną między pierwszym a ostatnim punktem. Dla zbyt dużej odległości działamy rekurencyjnie tylko tym razem na przedziałach [pierwszy punkt, odległy punkt] i [odległy punkt, ostatni punkt]. Dzięki czemu pomijamy punkty które są skierowane w tą samą stronę.

# 10. Kodowanie

Cały projekt podzielono na 3 główne części: silnik, grę, własną bibliotekę realizującą Entity Component System (dodatkowo projekt zawierał testy do niektórych elementów silnika i ECS).

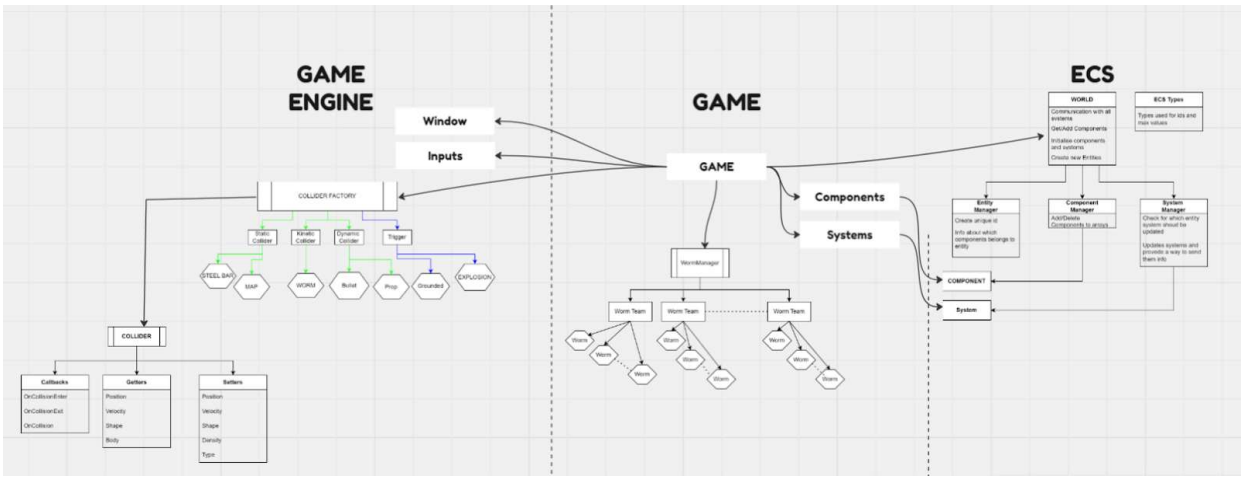
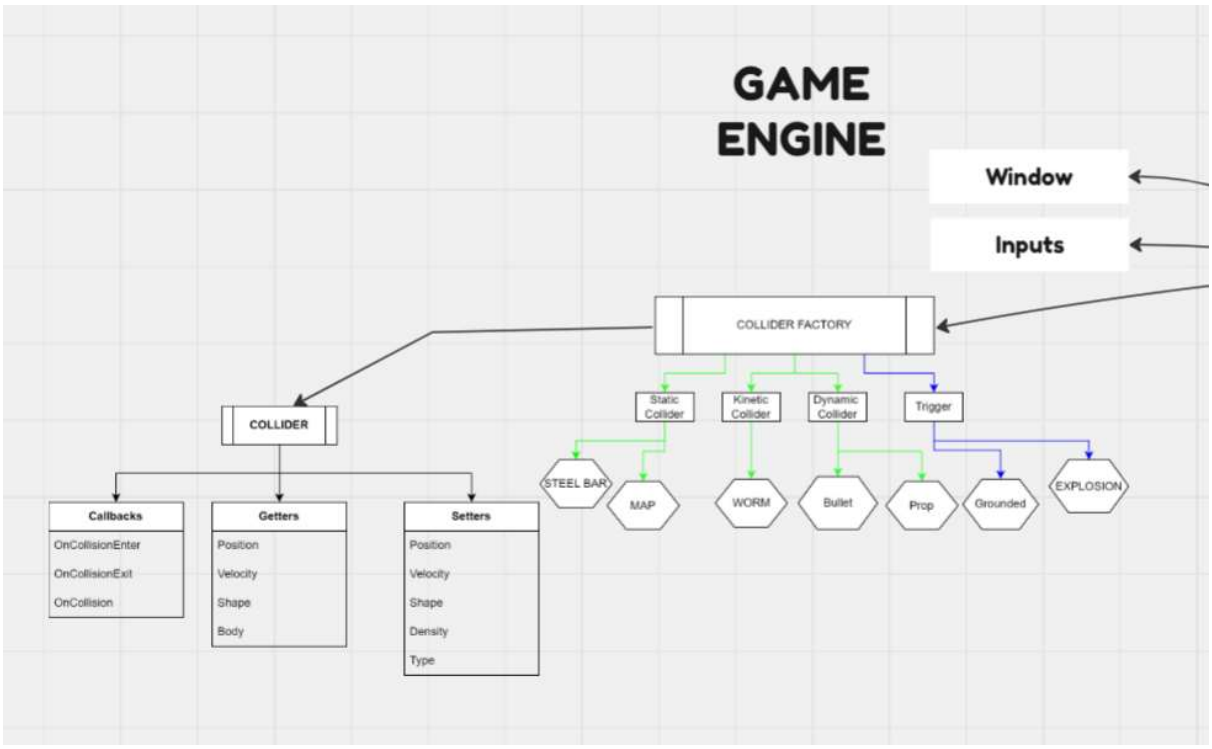
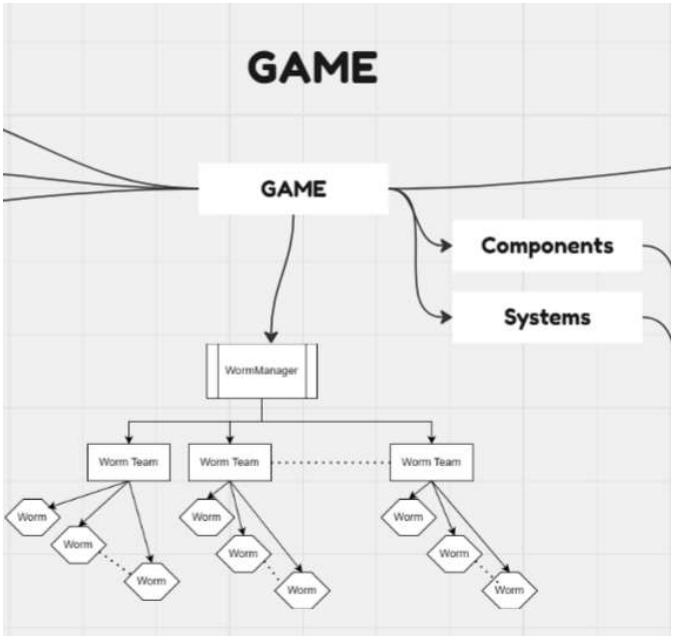


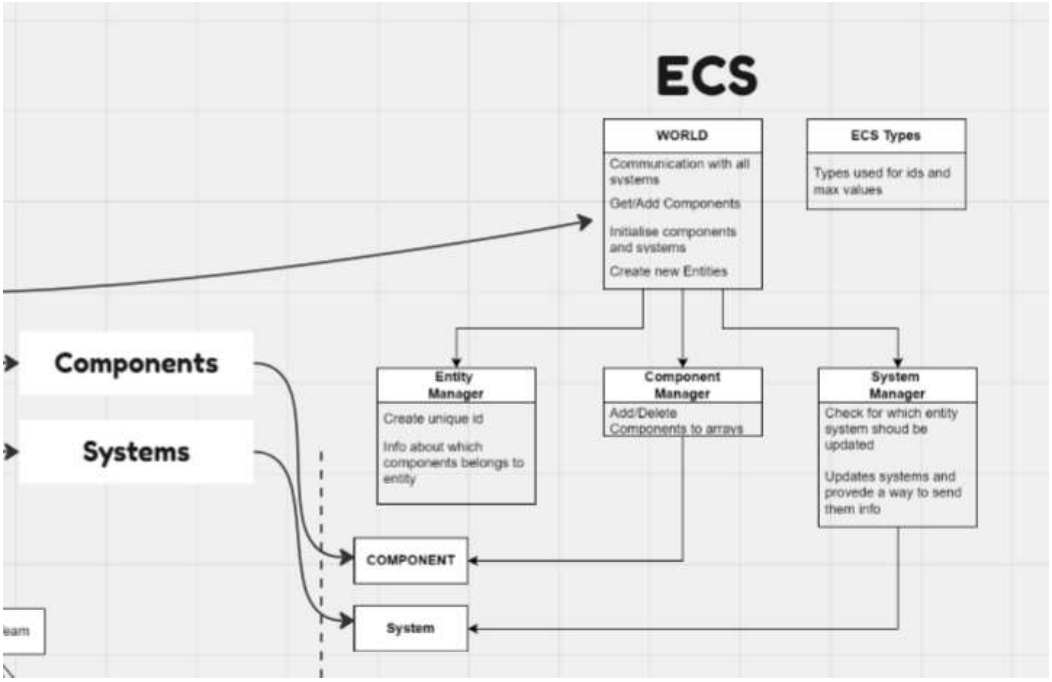

Diagram przedstawiający cały projekt




Część odpowiedzialna za silnik była odpowiedzialna przede wszystkim za wrapowanie bibliotek: SDL, Box2d, SDL_image, SDL_mixer jak i za inicjalizację całego programu, obsługę inputów, obsługę terminalu, error handlingu, dodawanie i usuwanie komponentów a także komponentów należących do nich.




Część odpowiedzialna za grę miała 3 główne zadania, zarządzanie wormsami, zarządzanie brońmi, implementacja potrzebnych do działania gry komponentów i systemów współgrających z ECS.



Prosta implementacja systemu ECS, w celu wydajnego tworzenia wielu obiektów.

## 11. Testowanie

Testy którym poddana została gra i jej silnik można podzielić na ręczne i automatyczne. Ręcznie testowany był gameplay oraz niestabilności gry i silnika. Automatycznie natomiast system **ECS** stworzony na potrzeby gry oraz algorytm do tworzenia i upraszczania konturów. Pomogło to uniknąć wielu przeoczeń.

## 12. Wdrożenie, raport i wnioski

Projekt został zrealizowany na poziomie podstawowym i częściowo na poziomie rozszerzonym. W stosunkowo krótkim czasie udało się dobrze odwzorować oryginał i przygotować grę pod dalszy jej rozwój. Dodawanie kolejnych elementów takich jak bronie, dźwięki, particle, efekty pocisków to rzecz stosunkowo łatwa. Co do pozostałych rozszerzonych wymagania takich jak np networking, również one byłyby łatwe do wdrożenia z obecnym systemem ECS.

Największym problemem w czasie produkcji były niedopatrzenia związane z pamięcią, które okazywały się często ciężkie do wychwycenia. Kolejną problematyczną rzeczą było nadużywanie schematu singletona w różnych miejscach w kodzie. Mimo, że nie spowodowało to bezpośrednio jakichś błędów, trzeba pamiętać że nie jest to dobra praktyka.

Podsumowując projekt został zrealizowany pomyślnie i pomógł uzyskać wiele różnorodnych doświadczeń w tworzeniu aplikacji używając języka c++