Jakub Maliszewski,

Mateusz Lewczak,

Krzysztof Blankiewicz,

Konrad Polański

Raport – Projekt zespołowy

**Gra typu FPS(First-Person Shooter) w stylu gry Doom93**

# Wstęp

# Założenia projektowe

Nasz projekt polega na stworzeniu pierwszoosobowej gry typu FPS (First Person Shooter) w perspektywie 2,5D inspirowaną stylem klasycznej gry Doom z 1993 roku. Doom jest jednym z legendarnych tytułów w historii gier wideo, często uznawanym za pioniera gatunku FPS, który zdefiniował wiele podstawowych mechanik, które stały się podstawą tego typu gier do dzisiaj. W grze zastosowana została perspektywa gracza 2,5D, która łączy dwuwymiarowe tekstury z trójwymiarową przestrzenią, tworząc iluzję głębi i realizmu przy zachowaniu prostoty graficznej – bez skomplikowanych modeli 3D.

Celem projektu jest nie tylko odtworzenie kultowej mechaniki rozgrywki, w tym najważniejsze aspekty rozgrywki, takie jak ruch gracza po mapie, walka z przeciwnikami oraz zbieranie przedmiotów i interakcje tych mechanik ze sobą. Projekt dał nam również możliwość zapoznania się z technicznymi aspektami tworzenia silników gier z perspektywą 2,5D. Projekt pozwoli nam także zgłębić zagadnienia takie jak optymalizacja kodu pod kątem wydajności, efektywne zarządzanie pamięcią, renderowanie grafiki w czasie rzeczywistym oraz implementacja podstawowej sztucznej inteligencji przeciwników. Gra będzie naśladować klimat i działanie oryginalnej gry – szybka akcja połączona z prostym sterowaniem, walka z przeciwnikami. Kluczowym elementem będzie zachowanie estetyki retro, z uwzględnieniem oryginalnej grafiki, jednak z pewnymi uproszczeniami na potrzeby projektu zespołowego.

Oryginalnego Dooma93 napisano w języku C, a w pewnych krytycznych aspektach, np. do obsługi grafiki oraz dźwięku wykorzystano assemblera, co zapewniło dużą wydajność na komputerach z tego okresu []. My natomiast do skonstruowania naszego silnika gry sięgnęliśmy po język C++ w standardzie C++11, również ze względów wydajnościowych. Do utworzenia okna gry oraz rysowania elementów w grze użyliśmy popularnej i prostej w użytkowaniu biblioteki SFML 2.6.1. Do efektywnego zarządzania kodem źródłowym i umożliwienie wygodnej pracy zespołowej w projekcie wykorzystaliśmy system kontroli wersji Git, który pozwala na śledzenie zmian w kodzie i łatwe integrowanie pracy wszystkich członków zespołu.

W projekcie zastosowaliśmy wzorzec projektowy ECS (Entity-Component-System) do zarządzania logiką silnika gry. Wzorzec pozwoli na elastyczne tworzenie i zarządzanie obiektami gry poprzez podział na trzy główne typy elementów: Encje (Entities), Komponenty (Components) oraz Systemy (Systems). Takie podejście projektowe ułatwia dodawanie nowych elementów gry i modyfikację już istniejących co poprawia skalowalność i elastyczność projektu [].

Proces rozwoju gry będzie obejmował regularnie przeprowadzane testy kolejnych implementowanych modułów, aby zapewnić stabilność i jakość tworzonej gry. Końcowym wynikiem projektu będzie gotowa, grywalna wersja demo, prezentująca kluczowe mechaniki gry, dostępna do gry na komputerach PC z przynajmniej jednym poziomem.

# Harmonogram i spis zadań

Harmonogram projektu został rozpisany na dwa półsemestry, aby zapewnić wystarczająco dużo czasu na projektowanie, implementację, testowanie oraz dokończenie projektu. Daje to także margines na rozwiązanie problemów, które mogą pojawić się w trakcie realizacji projektu. Podczas przydziału zadań podzieliliśmy grę na odpowiednie moduły i systemy, tak aby w miarę możliwości każdy był odpowiedzialny za swoją autonomiczną część projektu, co na końcu pozwoli scalić wszystko w całość.

1. Przygotowanie środowiska programistycznego (Wiosna)
   1. Wybór narzędzi programistycznych (marzec):

* Decyzja o wyborze narzędzi programistycznych, takich jak kompilator, biblioteki, system kontroli wersji oraz środowiska IDE (cały zespół),
* Utworzenie repozytorium i konfiguracja systemu kontroli wersji dla całego zespołu (cały zespół),
  1. Instalacja i konfiguracja narzędzi (marzec – kwiecień)
* Instalacja wybranych narzędzi na komputerach roboczych każdego z członków zespołu (cały zespół),
* Konfiguracja środowiska programistycznego, ustawienia projektu w IDE (cały zespół),
* Stworzenie początkowej struktury katalogów w repozytorium (Mateusz Lewczak),
* Przetestowanie instalacji poprzez uruchomienie przykładowego projektu SFML (cały zespół)

1. Implementacja podstawowej struktury silnika (Wiosna)
   1. Struktura klasy GameEngine (kwiecień) (Jakub Maliszewski),
   2. Implementacja wzorca projektowego ECS (Entity – Component – System) (kwiecień – maj) (Mateusz Lewczak),
   3. Implementacja klasy WADLoader wczytującej dane z pliku o rozszerzeniu .WAD z poziomem (kwiecień – maj) (Mateusz Lewczak),
2. Implementacja głównych systemów logiki gry oraz potrzebnych komponentów (Lato) (do połowy sierpnia)
   1. System kontroli gracza i komponent stanu gracza (Jakub Maliszewski),
   2. System ruchu gracza i komponent ruchu (Jakub Maliszewski),
   3. System broni i komponent broni (Krzysztof Blankiewicz),
   4. System obrażeń i komponenty życia oraz obrażeń (Krzysztof Blankiewicz),
   5. System przeciwników i komponent przeciwnika (Jakub Maliszewski),
3. Implementacja systemów odpowiedzialnych za renderowanie grafiki (Lato)
   1. System renderowania minimapy i komponentu spritu minimapy (Mateusz Lewczak),
   2. System renderowania grafiki i komponentu spritu w grze (Mateusz Lewczak),
   3. System renderowania interfejsu gracza (Jakub Maliszewski).
4. Testowanie, optymalizacja i dokumentacja (Lato)
   1. Testowanie i optymalizacja (sierpień)
      * Przeprowadzenie pełnych testów funkcjonalnych i wydajnościowych, aby upewnić się, że gra działa zgodnie z założeniami projektowymi (cały zespół),
      * Poprawa ewentualnych błędów i optymalizacja kodu w krytycznych obszarach silnika (cały zespół),
   2. Opracowanie dokumentacji i końcowego raportu (sierpień – wrzesień)

* Poprawienie komentarzy w kodzie opisujących strukturę projektu (cały zespół),
* Sporządzenie raportu końcowego, podsumowującego wyniki projektu, napotkane problemy i ich rozwiązania oraz wnioski z realizacji projektu (cały zespół).

# Podstawy teoretyczne

# Doom93 (Jakub Maliszewski)

Doom93 stworzony przez firmę id Software i wydany w 1993 roku, zrewolucjonizował rynek gier komputerowych. Unikalny styl i dynamiczna rozgrywka połączona z zastosowaniem nowatorskich jak na tamte czasy technik programistycznych spowodowały, że Doom stał się legendarnym tytułem rozpoznawanym praktycznie przez każdego gracza. Głównymi programistami byli John Carmack i John Romero, którzy wspólnymi siłami stworzyli silnik zdolny do renderowania złożonych scen 2,5D w czasie rzeczywistym na ówczesnych komputerach klasy PC o bardzo ograniczonych zasobach sprzętowych. Powszechnie dostępne komputery w tamtym czasie nie posiadały dedykowanych układów graficznych (ang. GPU – Graphics Processing Unit) co spowodowało, że twórcy musieli się posunąć do wielu nietypowych i innowacyjnych jak na te czasy rozwiązań, aby gra mogła w ogóle powstać.

Jednym z kluczowych rozwiązań zastosowanych w Doomie była technika raycastingu. Raycasting to metoda, która polega dosłownie na „rzucaniu promieni” z punktu widzenia gracza w różnych kierunkach, aby określić co gracz widzi na ekranie. Każdy promień przeszukuje przestrzeń gry w poszukiwaniu najbliższej powierzchni i na tej podstawie renderuje odpowiednią odpowiednią teksturą w odpowiedniej skali. Technika ta była znacznie mniej zajmująca zasoby w porównaniu do tradycyjnych technik renderowania 3D (w tamtych czasach był to głównie mało wydajny algorytm malarski używany w programach modelowania inżynierskiego CAD). Pozwalało to na stworzenie płynnej i realistycznej iluzji trójwymiarowej przestrzeni przy jednoczesnym małym zapotrzebowaniu w moc obliczeniową co umożliwiało wykorzystanie ówczesnych procesorów.

Kolejnym znacznie bardziej nowatorskim i przełomowym rozwiązaniem zastosowanym w Doomie było wykorzystanie drzew BSP (Binary Space Partitioning) do zarządzania geometrią poziomów i znajdowaniem obiektów najbliżej gracza. Drzewa BSP umożliwiły efektywne dzielenie przestrzeni gry na mniejsze elementy i szybkie ich odnajdywanie, gdy były potrzebne np. do określenia trafienia podczas strzału. To podejście znacznie przyspieszyło proces renderowania, gdyż silnik gry był w stanie bardzo szybko określić, które obszary powinny być widoczne z danej pozycji gracza i w jakiej kolejności powinny być rysowane. Ta struktura danych okazała się krytyczna dla osiągnięcia płynnej rozgrywki, umożliwiając szybkie i efektywne renderowanie nawet na słabszym sprzęcie. To pionierskie rozwiązanie zdefiniowało przyszłość tego typu gier, zostało dopracowane w kolejnej produkcji Carmack’a, czyli równie legendarnym Quake’u oraz w innym tytule legendarnym dla gier FPS – Counter Strike na silniku Source.

Doom wprowadził również innowacyjne podejście do zarządzania zasobami gry. Wszystkie dane poziomu takie jak dźwięki, odnośniki do używanych tekstur, struktura poziomu, rozmieszczenie przeciwników, gracza oraz innych obiektów na mapie były przechowywane w plikach o rozszerzeniu .WAD (skrót od Where’s All the Data). Format ten ułatwił dystrybucję oraz modyfikację gry co znacząco przyczyniło się do rozwoju popularności tak zwanego „modowania” gry przez samych graczy. Układ pliku umożliwiał również szybkie ładowanie zasobów w trakcie rozgrywki, optymalizując czas ładowania.

Kolejną nowością było pierwsze w historii zastosowanie techniki dynamicznego oświetlenie. Silnik pozwalał na zmienianie jasności obiektów w czasie rzeczywistym, co pozwalało na tworzenie bardziej realistycznych scen z efektami świetlnymi i migającymi światłami, które potęgowały klimat horroru. W połączeniu z unikalnym stylem graficznym, te efekty wizualne dodawały immersji i pozwalały graczom „zanurzyć się” w atmosferze gry.

Jakby tego było mało Doom był również jednym z pierwszych tytułów, które wprowadziły tryb gry sieciowej (multiplayer), znany jako „deathmatch”. Gracze mogli łączyć się ze sobą lokalnie wykorzystując protokół IPX i rywalizując na specjalnie zaprojektowanych mapach. Stało się to podstawą do dalszego rozwoju gier wieloosobowych i podłożyło fundamenty pod przyszłe duże sukcesy gatunku FPS.

Wszystkie te innowacje przyczyniły się do niesamowitego sukcesu gry czyniąc ją rozpoznawalną praktycznie dla każdego laika gier komputerowych. Doom był nie tylko technologicznym arcydziełem swoich czasów, ale również przetarł drogę dla branży kier komputerowych jaką znamy dzisiaj. Pewne jest, że gra była w pewnym sensie kamieniem milowym w historii gier komputerowych i zainspirowała wielu twórców do poszukiwania nowych, bardziej wydajnych metod tworzenia gier.



Rys.1 Screenshot z gry Doom93 – Początek kultowej pierwszej misji E1M1

# Silniki gier (Jakub Maliszewski)

Silnik gry, to zestaw podstawowych modułów i mechanizmów programistycznych, które umożliwiają realizację kluczowych funkcji potrzebnych do działania gry. Stanowi on główną część kodu gry, która zajmuje się interakcją pomiędzy poszczególnymi obiektami w grze. Struktura silników gier nie jest z góry zdefiniowana i mogą się one między sobą znacząco różnić, jednak istnieją pewne podstawowe komponenty, które występują w nich praktycznie zawsze.

Jednym z podstawowych zadań silnika gry jest generowanie i wyświetlanie grafiki na ekranie. Zależnie od projektu mogą wspierać zarówno renderowanie 2D, jak i 3D, pozwalając na tworzenie różnych rodzajów wizualizacji. Dobór techniki renderowania ma znaczący wpływ na wygląd i realizm grafiki w grze.

Innym aspektem silników gry jest zdefiniowanie zasad fizyki panujących w grze. Moduły silnika odpowiedzialne za fizykę dotyczą ruchu postaci, obiektów, grawitacji, kolizji oraz interakcji obiektów między sobą. Moduły fizyki mogą być bardzo rozbudowane pozwalając na skomplikowane symulacje mechaniki płynów na potrzeby gry, odkształcania się ciał oraz ich reakcji na siły zewnętrzne – np. niszczenie otoczenia przez akcje gracza.

W grach kluczową rolę odgrywa zarządzanie pamięcią i optymalizacja wydajności. Odpowiednie zarządzanie zasobami, minimalizacja zużycia pamięci oraz obciążenia procesora to jedne z głównych zadań silnika zapewniającego płynność działania gry bez opóźnień i z odpowiednią ilością klatek na sekundę.

Kolejnym istotnym elementem silnika jest sztuczna inteligencja w kontekście interaktywnych postaci niezależnych (NPC) oraz systemów zachowań tych postaci lub wrogów w grze. Moduły AI mogą zawierać algorytmy śledzenia gracza, podejmowania decyzji, wyznaczania ścieżek oraz adaptacji do zmieniających się stanów w grze.

Oprócz tego silniki gier obsługują generowanie i przetwarzanie dźwięku, w tym efektów dźwiękowych, muzyki oraz dialogów. Wbrew pozorom jest to jeden z ważniejszych elementów silnika, który wpływa znacząco na odbiór gry.

Wiemy już, że bez silnika gra nie może istnieć, jednakże nie przy każdej produkcji silniki są pisane od zera. Generalnie tworzenie silników jest bardzo czasochłonne i przy profesjonalnych produkcjach wymaga współpracy wielu wyspecjalizowanych w konkretnych aspektach programistów i wiąże się z gigantyczną inwestycją. Aby uniknąć takich wydatków studia produkcyjne bardzo często wielokrotnie korzystają z kolejnych iteracji tego samego silnika, który powstał wcześniej, przy tworzeniu kolejnych gier. Przykładem może być seria Crysis wykorzystująca silnik CryEngine, który ewoluował, z każdą kolejną częścią gry []. Alternatywą dla pisania własnych, oryginalnych silników mogą być również gotowe silniki takie jak Unreal Engine, Unity lub Godot Engine, których wykorzystanie znacząco skraca czas produkcji i jest opcją, z której korzystają zarówno małe jak i duże studia w zależności od technicznych wymagań prowadzonego projektu oraz posiadanych zasobów (budżetu oraz czasu).

My oczywiście nie będziemy korzystać z gotowych rozwiązań i, chcąc zgłębić dogłębnie techniczne aspekty tworzenia gry, spróbujemy napisać nasz własny silnik od zera.

# Opis realizowanego projektu

Może taka główna struktura silnika.

System Kontroli gracza,

System ruchu gracza,

System Przeciwników,

- opis zrealizowanego projektu, dane techniczne, wykonane testy, omówienie  
uzyskanych wyników, ...

# Podsumowanie

-- dyskusja otrzymanych wyników,  
-- co się udało zrobić a co nie, proponowane dalsze prace