

STRESZCZENIE

ABSTRACT

SPIS TREŚCI

Streszczenie	1
Abstract	2
Spis treści.....	3
Wykaz ważniejszych oznaczeń i skrótów.....	5
1. Wstęp i cel pracy (Bogna Lew, Zofia Sosińska)	6
2. Przegląd strategii czasu rzeczywistego	7
2.1. System dialogów w grze Mass Effect 3 (Bartosz Strzelecki)	7
2.2. Model sztucznej inteligencji przeciwników w grach RTS (Bartosz Strzelecki)	8
2.3. Warhammer 40,000: Down of War (Bogna Lew).....	9
2.4. Mechanizm walki oraz zarządzania ekipunkiem w Kingdom Come: Deliverance (Bogna Lew)	9
2.5. Sterowanie jednostkami w grze Mount&Blade (Zofia Sosińska).....	10
2.6. Kompas w grze Skyrim (Zofia Sosińska)	10
2.7. Przedstawienie dostępnych pułapek do zbudowania w grze Orcs must die! (Zofia Sosińska) ..	10
3. Technologie, algorytmy i narzędzia	12
3.1. Przegląd silników (Bogna Lew).....	12
4. Projekt systemu	13
4.1. Organizacja (Bartosz Strzelecki).....	13
4.1.1. Główne etapy projektu	13
4.2. Skład zespołu projektowego	13
4.3. Analiza i specyfikacja wymagań [Bogna Lew]	13
4.4. Modelowanie terenu (Bogna Lew)	14
4.5. Interfejs użytkownika (Zofia Sosińska)	17
4.6. Nawigacja (Zofia Sosińska).....	21
4.7. Mechanizm budowania (Bogna Lew)	21
4.8. Sterowanie jednostkami, podążanie za główną postacią (Zofia Sosińska)	22
4.9. System dialogów (Bartosz Strzelecki)	22
4.10. Sztuczna inteligencja (Bartosz Strzelecki)	23
4.11. Widzenie przez ściany (Bartosz Strzelecki)	23
5. Implementacja	25
5.1. Kontroler postaci (Bogna Lew)	25
5.2. Zasady działania kompasu (Zofia Sosińska)	26
5.3. Mechanizm budowania (Bogna Lew)	29
5.4. System dialogów (Bartosz Strzelecki)	29
5.5. Sztuczna inteligencja (Bartosz Strzelecki)	30
5.5.1. Mechanika zachowań klas jednostek.....	30
5.5.2. Sztuczna inteligencja przyjaznych jednostek.....	32
5.6. Widzenie przez ściany (Bartosz Strzelecki)	32
6. Podsumowanie	34
Wykaz literatury	35
Wykaz rysunków.....	36

Wykaz tabel.....	37
------------------	----

WYKAZ WAŻNIEJSZYCH OZNACZEŃ I SKRÓTÓW

1. WSTĘP I CEL PRACY (BOGNA LEW, ZOFIA SOSIŃSKA)

Postrzeganie świata przed erą wielkich odkryć geograficznych znacząco różniło się od tego, które dominuje obecnie. Dawniej nie podejmowano decyzji strategicznych na podstawie precyzyjnych map, lecz ludzie zmuszeni byli budować przestrzenny obraz istniejącej sytuacji w toku dyskusji z naocznymi świadkami, takimi jak dowódcy czy podróżnicy.

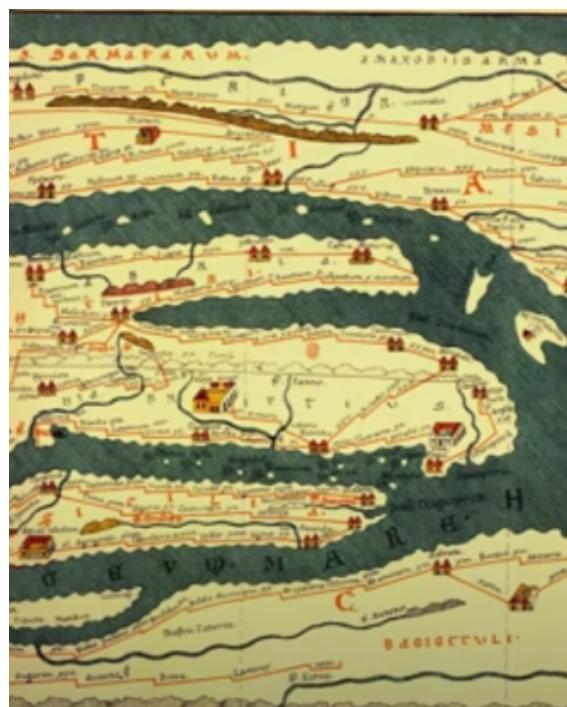
Współczesne gry komputerowe, których fabuła osadzona jest w tych czasach, stosuje wiele uproszczeń. Ma to na celu poprawienie jakości rozgrywki gracza, jednakże sprawia, że nie oddaje w pełni realiów. Często obraz dzisiejszych możliwości zasłania faktyczne dowodzenie w czasach, w których dzieje się gra. Technologia XXI wieku pozwala oficerom na wydawanie rozkazów na podstawie dokładnych map poprzez radio. Jednak w dawniejszych czasach było to niemożliwe. W większości gier historycznych grywalna postać momentami ma wręcz boskie umiejętności i wiedzę. Zwykły człowiek nie był w stanie w jednej chwili zobaczyć całego grywalnego świata i wydać komendę jednostkom, znajdującym się na drugim jego końcu, o przemieszczeniu się do punktu z dokładnością do jednego metra.

Celem projektu jest zaprojektowanie oraz zimplementowanie gry strategii czasu rzeczywistego RTS (ang. real-time strategy) osadzonej we wczesnym średniowieczu z trybem rozgrywki dla jednego gracza. Udostępniane przez nią mechaniki mają jak najlepiej oddawać realia tamtych czasów, przy jednoczesnym zachowaniu podstawowych cech tego gatunku. Omówione rozwiązanie powinno zbalansować narzucone ograniczenia tak, aby nie pogarszać jakości rozgrywki.

W rozdziale "Wprowadzenie do dziedziny" zostały przedstawione przykłady podstawowych mechanik w grach strategii czasu rzeczywistego. Kolejny rozdział skupia się na podstawowych narzędziach oraz technologiach wykorzystywanych do wytwarzania gier komputerowych. Dodatkowo precyzuje wybrane przez zespół środowiska, które zostaną wykorzystane do implementacji rozwiązań. W kolejnym rozdziale został przedstawiony projekt wytwarzanej gry. Skupia się on na wizji zespołu oraz oczekiwanych efektach. Na koniec, w rozdziale "Implementacja" zaprezentowane zostały osiągnięte rezultaty oraz w jaki sposób je opracowano.

2. PRZEGŁĄD STRATEGII CZASU RZECZYWISTEGO

"Starożytni świat widzieli inaczej, mniej płasko"[1]. Dobrze obrazującym ówczesne postrzeganie przestrzeni przykładem jest mapa Imperium Rzymskiego, pokazana na 2.1. Czytanie jej dosłownie miją się z celem. Nie są na niej zachowane ani proporcje, ani strony świata. Mimo tego, że basen Morza Śródziemnego został ówcześnie dosyć dokładnie oddany, "nie wydaje się, aby Rzymianom współczesna kartograficzna wierność była potrzebna"[1]. "Dowódcy opierali się na swojej wiedzy, wiedzy wynajętych przewodników oraz informacjach zwiadowców i tubylców"[1].



Rysunek 2.1: Mapa basenu Morza Śródziemnego z czasów Imperium Rzymskiego

Na odbiór gry wpływają zawarte w niej mechaniki. Twórcy bardzo częstoszą w nich uproszczenia, co ma na celu ułatwienie graczowi wykonywanie zadań. Ma to jednak ogromny wpływ na zachowanie realizmu w grach, zwłaszcza tych opartych na wydarzeniach historycznych oraz prawdziwym świecie.

W niniejszym rozdziale zostały przedstawione wybrane mechaniki wykorzystywane w grach typu RTS. Omówione zostało ich działanie na podstawie przykładów z najpopularniejszych gier. Celem tego jest obrazowe zaprezentowanie ich integralności w grach oraz wpływu na rozgrywkę.

2.1. System dialogów w grze Mass Effect 3 (Bartosz Strzelecki)

W Mass Effect 3 system dialogowy jest integralną częścią rozgrywki, pozwalając graczom na prowadzenie rozmów z różnymi postaciami w trakcie gry. System dialogów w Mass Effect 3 wykorzystuje interfejs oparty na kole dialogowym2.2. To koło przedstawia graczom wiele opcji odpowiedzi podczas rozmów, zwykle podzielonych na kategorie według ich ogólnego tonu lub intencji. Dostępne opcje często obejmują wybory, dyplomatyczne, agresywne, konfrontacyjne oraz opcje neutralne lub śledcze. Podczas niektórych rozmów lub przerywników filmowych gracze mogą przerwać trwającą rozmowę, szybko wy-

bierając określoną opcję dialogową. Te opcje przerywania pozwalają graczom podjąć natychmiastowe działania lub podjąć decyzje na miejscu, często wpływając na wynik sytuacji lub relacje postaci z innymi. Ogólnie rzecz biorąc, system dialogowy w Mass Effect 3 został zaprojektowany tak, aby zapewnić graczom bogate i wciągające doświadczenie w opowiadaniu historii, pozwalając im kształtować narrację poprzez wybory i interakcje z olbrzymią gamą postaci. System oferuje różnorodne opcje odpowiedzi, dynamiczne rozmowy i konsekwencje, przyczyniając się do fascynującej i rozgałęzionej narracji gry.



Rysunek 2.2: Przykład koła dialogowego w grze Mass Effect

2.2. Model sztucznej inteligencji przeciwników w grach RTS (Bartosz Strzelecki)

W sferze gier RTS sztuczna inteligencja (ang. artificial intelligence, AI) odgrywa kluczową rolę, wspierając doświadczenia z rozgrywki. W tym przypadku termin odnosi się do zbioru algorytmów i systemów zaprojektowanych w celu symulacji zachowań podobnych do tych gracza. Między innymi umiejętność podejmowania decyzji i rozwiązywania problemów.

Sztuczna inteligencja przeciwników w grach takich jak Warcraft III lub StarCraft II, przede wszystkim w trybie kampanii, jest odpowiedzialna za kontrolowanie wrogich jednostek w celu zaoferowania graczu wyzwania. Głównym zadaniem AI jest zasymulowanie strategicznych decyzji i wydajne zarządzanie zasobami. AI podejmuje decyzję na podstawie predefiniowanych zasad i algorytmów. Analizuje sytuację, w której się znajduje, biorąc pod uwagę siłę swojej własnej armii, siłę armii gracza oraz specjalne zdolności jednostek i środowisko, w którym toczy się gra. Ta analiza pozwala komputerowi na podejmowanie strategicznych decyzji jak na przykład, kiedy atakować, bronić się, eksplorować oraz rozszerzać swoje terytorium. W tych grach sztuczna inteligencja może przybrać jeden z kilku wariantów wynikających z poziomu trudności. Wyższe poziomy dają przeciwnikowi przewagę taką, jak wydajniejsze zbieranie zasobów lub szybsza produkcja jednostek.

W grze Warcraft III w trybie kampanii zachowanie przeciwników jest zaprojektowane z myślą o zanurzeniu gracza w fabularnej opowieści, jednocześnie prezentując wciągające wyzwania związane z rozgrywką. Akcje wykonywane przez sztuczną inteligencję są dostosowane do celów danej misji, co pozwala na dopasowanie do obowiązującej narracji. Początkowo przeciwnik konstruuje i rozbudowuje swoją bazę, w celu zgromadzenia odpowiedniej liczby zasobów, szkolenia jednostek i prowadzenia badań. AI strategicznie rozmieszcza budynki i struktury obronne, aby ochronić swoją fortecję przed najazdami gracza. Misje kampanii często też zawierają oskryptowane wydarzenia lub walki, które dodają głębi rozgrywce. Podczas tych starć wroga sztuczna inteligencja może zachowywać się w specjalny sposób, kontrolując potężne jednostki, do których gracz normalnie nie ma dostępu lub inicjując działania, które popychają narrację do przodu. Te wyreżyserowane wydarzenia tworzą niezapomniane chwile i jeszcze

bardziej wciągają gracza w fabułę kampanii. Zachowanie wroga w kampanii jest zróżnicowane i obejmuje różnorodne cele misji i scenariusze. Gracze mogą napotkać wrogów, którzy preferują agresywne ataki, inni skupią się na strategiach obronnych lub specjalizują się w taktyce hit and run. Sztuczna inteligencja dostosowuje proces podejmowania decyzji do konkretnych wymagań misji, często wykorzystując ukształtowanie terenu, synergię jednostek i scenariusze wydarzeń, aby rzucić wyzwanie umiejętnościom gracza. Ogólnie rzecz biorąc, zachowanie wrogów w kampanii Warcraft III ma na celu zapewnienie dynamicznego i wciągającego doświadczenia. Gracze muszą wykorzystywać myślenie strategiczne, zarządzanie zasobami i efektywny skład jednostek, aby przewyciążyć różnorodne strategie stosowane przez wrogą sztuczną inteligencję.

2.3. Warhammer 40,000: Dawn of War (Bogna Lew)

Warhammer 40,000: Dawn of War jest grą typu RTS osadzoną w uniwersum gry bitowej Warhammer 40,000. Udostępnia ona tryb jednoosobowy oraz wieloosobowy dla maksymalnie sześciu graczy. W pierwszym wariantie gracz wciela się w postać dowódcy armii Space Marines z Blood Ravens i ma za zadanie zapobiec inwazji Orków. Gra Warhammer 40,000: Dawn of War bardzo szybko zyskała na popularności i oferowała wszystko, co było potrzebne dla tego gatunku. Z tego powodu warto się jej przyjrzeć, pomimo faktu, że jej realia znacznie odbiegających od tych, w których zostanie osadzona tworzona przez nas gra.

Warhammer 40,000: Dawn of War wyróżnia model pozyskiwania surowców. W grze dostępne są dwa rodzaje: Energia, która jest generowana przez dedykowane do tego budowle oraz Rekwizycja, której szybkość wytwarzania jest uzależniona od kontrolowanych przez gracza punktów strategicznych. Taka mechanika znacznie lepiej wpasowuje się w realia gry oraz wymusza na użytkowniku przyjęcie agresywniejszej strategii.

Dodatkowo Warhammer 40,000: Dawn of War posiada typowy dla gier RTS mechanizm tworzenia budowli. Gracz ma do dyspozycji jednostki, którym może zlecić budowę wybranego przez siebie obiektu po poniesieniu kosztów jego utworzenia. Zanim będzie możliwe rozpoczęcie budowania użytkownik musi wybrać miejsce, w którym budynek powstanie. Robi to, przesuwając jego podgląd po mapie. W tym czasie gra dokonuje walidacji miejsca i informuje gracza czy wybrany obszar jest poprawny, odpowiednio podświetlając widok budynku. Wybudowanie obiektu nie jest natychmiastowe, co sprawia, że gra lepiej oddaje realia, w których jest osadzona.

2.4. Mechanizm walki oraz zarządzania ekwipunkiem w Kingdom Come: Deliverance (Bogna Lew)

Kingdom Come: Deliverance to gra z gatunku RPG osadzoną w realiach Europy Środkowej na początku XV wieku. Chociaż nie jest to gra czasu rzeczywistego to jest to pozycja warta wymienienia ze względu na dbałość twórców o zachowanie realizmu epoki oraz staranność wykonania mechanizmów walki oraz zarządzania ekwipunkiem. Jest ona przeznaczona dla jednego gracza, a całość zaprezentowana jest z perspektywy pierwszoosobowej. W trakcie rozgrywki użytkownik rozwija swoją postać, bierze udział w starciach, prowadzi rozmowy z niezależnymi postaciami i wiele więcej.

Godny uwagi jest mechanizm walki. Twórcy skupili się na jak najdokładniejszym oddaniu średniowiecznego stylu walki. W tym celu skrupulatnie przestudiowali w jaki sposób władano mieczem w tamtych czasach, a następnie w pełni oddali to w grze. Wykorzystali do tego tysiące animacji oraz

starannie oddali fizykę pojedynków. W efekcie powstał realistyczny mechanizm walki, który umożliwia graczy parowanie, zadawanie ciosów oraz blokowanie.

Kolejnym elementem wartym wymienienia jest rozbudowany system zarządzania ekipunkiem. Przeznaczony do tego panel jest podzielony na dwie sekcje - jedna w której wyświetlona jest lista posiadanych rzeczy oraz druga przeznaczona na postać gracza. Może on dowolnie spersonalizować swoją postać, poprzez możliwość założenia wielu elementów ubioru naraz tworząc warstwy. Dodatkowo, każdy przedmiot posiada swoją wagę, a postać swój maksymalny udźwig. W przypadku przekroczenia limitu gracz zostaje ukarany poprzez spowolnienie ruchów w walce i uniemożliwieniu biegania. Jest to wzorowane na rzeczywistości, dzięki czemu gra jeszcze lepiej oddaje realia epoki.

2.5. Sterowanie jednostkami w grze Mount&Blade (Zofia Sosińska)

Mount&Blade jest to gra komputerowa z gatunku cRPG (komputerowa gra fabularna, ang. computer role-playing game) z elementami strategicznymi, stworzona przez turecką firmę TaleWorlds Entertainment i wydana przez Paradox Interactive. W otwartym świecie fikcyjnej krainy Calradia, stylizowanej na czasy średniowieczne, gracz ma pełną dowolność stylu rozgrywki. W jego mocy jest zarówno zbieranie armii i dążenie do zostania królem, jak i zostanie wasalem jednego z władców. Za pomocą dialogów i walk z postaciami gracz buduje unikalową historię. Jedną z wertych uwagi mechanik, zaimplementowaną w grze Mount&Blade, jest sterowanie jednostkami służącymi granemu charakterowi. Gracz bezpośrednio kieruje jedynie główną postacią. Podczas walki reszcie może wydawać rozkazy. Poprzez cyfry 0-4 wybiera grupę, do której się odnosi np. łuczników. Następnie przez klawisze F1-F11 wydaje konkretny rozkaz np. odwrót. Sztuczna inteligencja postaci zajmuje się już samym wykonaniem czynności. Gracz nie martwi się, czy jednostki znajdą optymalną drogę, będą celować w przeciwników, czy z nimi walczyć.

2.6. Kompas w grze Skyrim (Zofia Sosińska)

The Elder Scrolls V: Skyrim (skrótnie Skyrim) jest to fabularna gra akcji o otwartym świecie, wyprodukowana przez Bethesda Game Studios i wydana przez Bethesda Softworks. Skyrim jest piątym tytułem z serii The Elder Scrolls oraz kontynuacją gry The Elder Scrolls IV: Oblivion. Jest to jednak nowa historią osadzoną w uniwersum The Elder Scrolls, a nie kontynuacją poprzednika. Fabuła opiera się na powrocie smoków do krainy Tamriel. Bohater okazuje się posiadać moc Głosu, dzięki czemu jest w stanie posługiwać się zaklęciami tych starożytnych stworzeń. Z punktu tworzonej przez nas gry, szczególnie interesujące jest bardzo proste i sprytne rozwiązanie, jakim jest pasek przedstawiający pole widzenia gracza. Służy on między innymi jako kompas, ponieważ jedną z jego mechanik jest pokazanie użytkownikowi stron świata, znajdujących się w kierunku, w którym on patrzy. Pasek ułatwia także poruszanie się po świecie sygnalizując położenie wrogów, kompanów i ważnych dla rozgrywki lokalizacji.

2.7. Przedstawienie dostępnych pułapek do zbudowania w grze Orcs must die! (Zofia Sosińska)

Orcs must die! to strategiczna gra akcji stworzona i wydana przez studio Robot Entertainment. Akcja toczy się w krainie fantasy, w której największym zagrożeniem dla ludzkości są orkowie. W obronie świata przed tymi stworzeniami staje Zakon dowodzony przez Wojennych Magów. Wznieśli oni system fortec odgradzających ojczyznę orków od pozostałych ziem. Zadaniem gracza jest wcielenie się w

jednego z Wojennych Magów i mordowanie nadciągających grup orków za pomocą różnorodnych broni i mechanizmów, które może postawić. W przejrzysty sposób zostało rozwiązane samo wyświetlenie dostępnych do zbudowania pułapek. Graczowi pokazują się wizerunki mechanizmów, które może postawić. Naciskając odpowiedni numer na klawiaturze, gracz wybiera, co chce zbudować. Po zatwierdzeniu lewym przyciskiem myszki, budynek pojawia się w zaznaczonym miejscu.

3. TECHNOLOGIE, ALGORYTMY I NARZĘDZIA

3.1. Przegląd silników (Bogna Lew)

Podstawą tworzenia gier jest silnik graficzny. Stanowi serce kodu, odpowiadając za interakcję poszczególnych elementów. Dostarcza podstawowe narzędzia dzięki którym łatwiej można dokonywać zmian w grze. Wybranie odpowiedniego silnika przed rozpoczęciem pracy ma kluczowy wpływ na proces wytwórczy i efekt końcowy.

Obecnie dostępnych jest wiele silników, a każdy z nich posiada różne możliwości. W celu uproszczenia wyboru zdecydowaliśmy się zawęzić listę do trzech pozycji. Są to Godot, Unity oraz Unreal Engine.

Pierwszy z nich jest w pełni darmowym silnikiem open source. Posiada prosty i intuicyjny interfejs, a w internecie tworzonych jest przez społeczność wiele samouczków. Nie posiada on jednak oficjalnej dokumentacji oraz jest zdecydowanie mniej popularny od pozostałych dwóch.

Kolejny silnik, Unity jest określany jako przyjazny dla początkujących. Posiada bogatą dokumentację oraz jest dostępnych dużo samouczków stworzonych przez jego społeczność. Unity świetnie się nadaje do tworzenia gier 3D. Silnik ten jest dostępny w wersji bezpłatnej oraz oferującej więcej możliwości wersji płatnej. Co więcej, posiada możliwość rozszerzenia o dodatkowe narzędzia dostępne w Assets Store.

Ostatni z silników jest najbardziej kojarzony z grami AAA. Cechuje go zaawansowana grafika, która umożliwia wytwarzanie fotorealistycznych gier. Korzystanie z niego jest darmowe, a opłata w wysokości 5% jest naliczana jedynie gdy gra zarobi ponad milion USD.

Tabela 3.1 przedstawia porównanie wymienionych silników w istotnych, z punktu widzenia projektu, aspektach.

Tabela 3.1: Porównanie silników.

Silnik	Unity	Unreal Engine	Godot
Popularność	duża	duża	mała
Elastyczność	duża	duża	duża
Obsługa	dobra	dobra	dobra
Język	C#	C++	C#, C++, GDScript
Baza wiedzy	dokumentacja, samouczki	dokumentacja, samouczki	samouczki, fora

Finalnie zdecydowaliśmy się na implementację gry w Unity, ponieważ jest to silnik, który najlepiej odpowiada wymaganiom projektu.

4. PROJEKT SYSTEMU

W tym rozdziale został przedstawiony zamysł członków zespołu na opracowywaną grę. Stanowi on opis pożądanych rezultatów, do których zespół będzie dążyć w trakcie implementacji. Kolejno zostały opisane oczekiwane działanie poszczególnych mechanik, z których będzie się składał ostateczny produkt.

4.1. Organizacja (*Bartosz Strzelecki*)

4.1.1. Główne etapy projektu

1. Wybór i analiza konkretnego kontekstu historycznego.
2. Syntetyczny opis modelu postrzegania przestrzeni na podstawie dzieł pisanych, architektury i sztuki.
3. Przegląd rozwiązań stosowanych w grach strategicznych z wybranego okresu oraz dodatkowo mechanizmów z innych gier, które mogłyby być zaadoptowane na potrzeby projektu.
4. Opracowanie fabuły, selekcja postaci i wydarzeń, a także określenie zakresu autonomii świata gry oraz możliwości modyfikowania go przez gracza.
5. Opracowanie szczegółowej koncepcji i projektu gry, w tym projekt mechanizmów zarówno tych pozwalających graczowi pozyskiwać informacje o świecie gry i zdarzeniach w nim zachodzących, jak również tych poprzez które gracz będzie wywierał wpływ.
6. Implementacja poszczególnych funkcjonalności gry.
7. Testowanie, weryfikacja założeń i validacja.
8. Stworzenie dokumentacji przeprowadzonych prac.

Przewidywany termin zakończenia prac nad projektem to grudzień 2023 roku.

4.2. Skład zespołu projektowego

Bogna Lew	184757
Zofia Sosińska	184896
Bartosz Strzelecki	184529

4.3. Analiza i specyfikacja wymagań [*Bogna Lew*]

Z punktu widzenia projektu kluczowe jest jak najdokładniejsze oddanie realiów przy jednoczesnym uwzględnieniu jakości rozgrywki gracza oraz cech charakterystycznych dla gier typu RTS. Co więcej, użytkownik będzie sterował wybraną przez siebie postacią, wydając polecenia zaprzyjaźnionym jednostkom oraz stawiając czoła swoim przeciwnikom. Z tego powodu kluczowe jest dobranie odpowiednich mechanik, które umożliwią osiągnięcie wymienionych założeń.

Podstawową funkcjonalnością, jaką gra musi oferować, jest sterowanie jednostkami. Gracz musi być w stanie poruszać się wybraną przez siebie postacią oraz dowodzić swoją drużyną przy zachowa-

niu realiów średniowiecznej komunikacji. Dlatego gra powinna udostępnić mechanizm przekazywania informacji zarówno pomiędzy jednostkami znajdującymi się obok siebie, jak i znacząco od siebie oddalonymi. Konieczne będzie wprowadzenie systemu dialogów, dzięki któremu gracz będzie mógł wchodzić w interakcję z innymi postaciami, pozyskiwać informację oraz wpływać na dynamikę świata.

Mechaniką, która ma najistotniejszy wpływ na postrzeganie świata w grze, jest sposób nawigacji. Musi ona jednocześnie umożliwiać graczy bez problemu zorientować się w terenie oraz nie odbiegać przy tym od realiów tamtych czasów. Z tego powodu, niemożliwe jest wykorzystanie najpopularniejszego rozwiązania stosowanego w większości gier, jakim jest minimapa.

Kolejną mechaniką typową dla gier RTS, którą gra powinna posiadać, jest budowanie budynków. Jest to charakterystyczna funkcjonalność, która dodatkowo urozmaica rozgrywkę. Istotną kwestią jest, aby obiekty były tworzone poprawnie oraz oddawały realia wybranej epoki. Dzięki tej mechanice gracz będzie w stanie tworzyć bazy o znaczeniu strategicznym, które dodatkowo pomogą mu we wzmacnianiu swoich postaci.

Dodatkowo gra musi udostępniać podstawową funkcjonalność swojego gatunku, czyli sterowanych przez sztuczną inteligencję przeciwników. Powinni oni wykonywać ataki typowe dla realiów epoki oraz potrafić poprawnie określić swój cel. Co więcej, powinni oni inicjować walkę z graczem, gdy go napotkają. Z tego powodu konieczne będzie utworzenie mechanizmu walki, który obsłuży zachowanie postaci biorących udział w starciu, jego przebieg oraz umożliwi użytkownikowi wykonywanie akcji swoją postacią.

Konieczne będzie utworzenie interfejsu graficznego, który w czytelny sposób wyświetli graczy aktualny stan gry, pomoże mu w nawigacji i poznawaniu świata oraz przedstawi możliwe do podjęcia akcje. Najważniejszymi jego cechami będą prostota i intuicyjność, aby nie przyłknąć użytkownika ogromem i skomplikowaniem informacji. Jego celem ma być pogłębienie immersji i przedstawienie świata z punktu widzenia granej postaci. Niezbędne będą informacje o posiadanych surowcach, czasie oraz położeniu. Interfejs użytkownika musi też jednak dostosowywać się do sytuacji i zmieniać się tak, aby gracz miał przedstawione możliwe opcje podczas dowodzenia walką, budowania budynków oraz prowadzenia dialogów. Kluczowe w przedstawieniu informacji będzie stylistyczne wpasowanie się w realia gry, aby umożliwić graczy poczucie się w epoce, w której jest osadzona fabuła.

4.4. Modelowanie terenu (*Bogna Lew*)

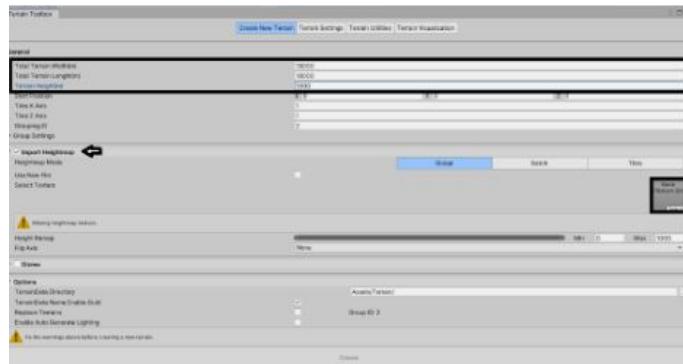
Do utworzenia terenu do gry wykorzystaliśmy zasób 3D World Building udostępniany przez Unity. Umożliwia on automatyczne generowanie terenu na podstawie heightmapy - monochromatycznego obrazu reprezentującego model wysokościowy. Kolor czarny reprezentuje najniższe punkty, natomiast kolor biały - najwyższe.

Wygenerowanie terenu umożliwia narzędzie Terrain Toolbox, które można uruchomić wybierając z menu Window -> Terrain -> Terrain Toolbox. Pozwala ono na ustawienie podstawowych parametrów takich jak długość, szerokość oraz wysokość terenu. Należy również zaznaczyć checkbox Import Heightmap oraz załączyć obraz z modelem wysokościowym. Na koniec trzeba nacisnąć przycisk Create, co spowoduje dodanie do sceny wygenerowanego terenu.

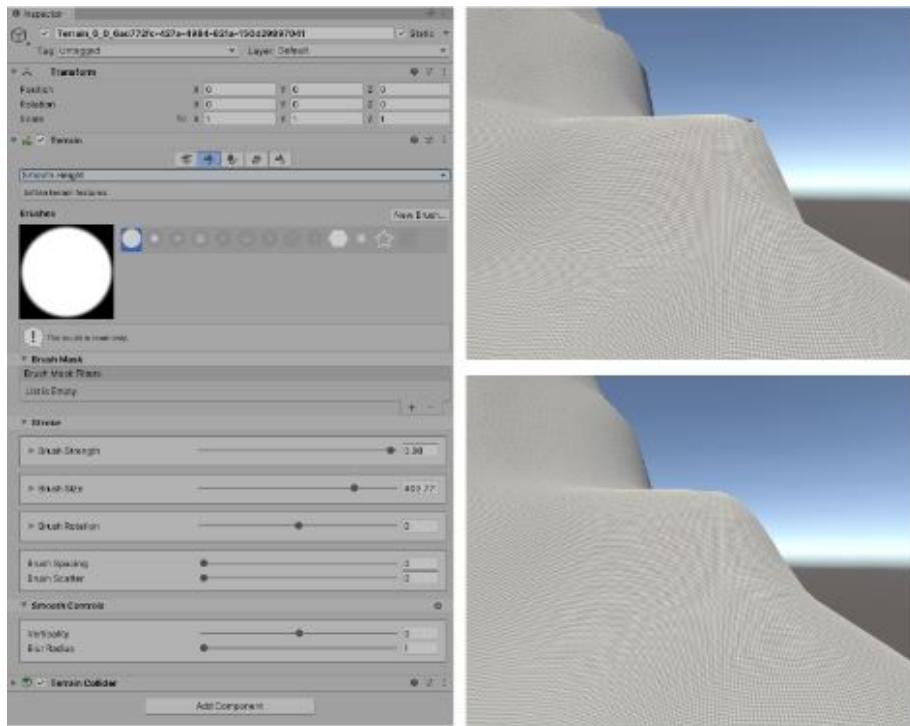
Tak utworzony teren, chociaż już jest grywalny, posiada ostre i postrzępione krawędzie, które nie wyglądają zbyt estetycznie. Wygładzenie ich poprawi wygląd terenu i sprawi, że będzie on bardziej realistyczny. Do tego służy narzędzie Smooth Height dostępne w inspektorze terenu. Powoduje ono uśrednienie pobliskich płaszczyzn, co pozwala na usunięcie nagłych zmian terenu i w rezultacie wygładzenie go.



Rysunek 4.1: Przykładowe heightmapy



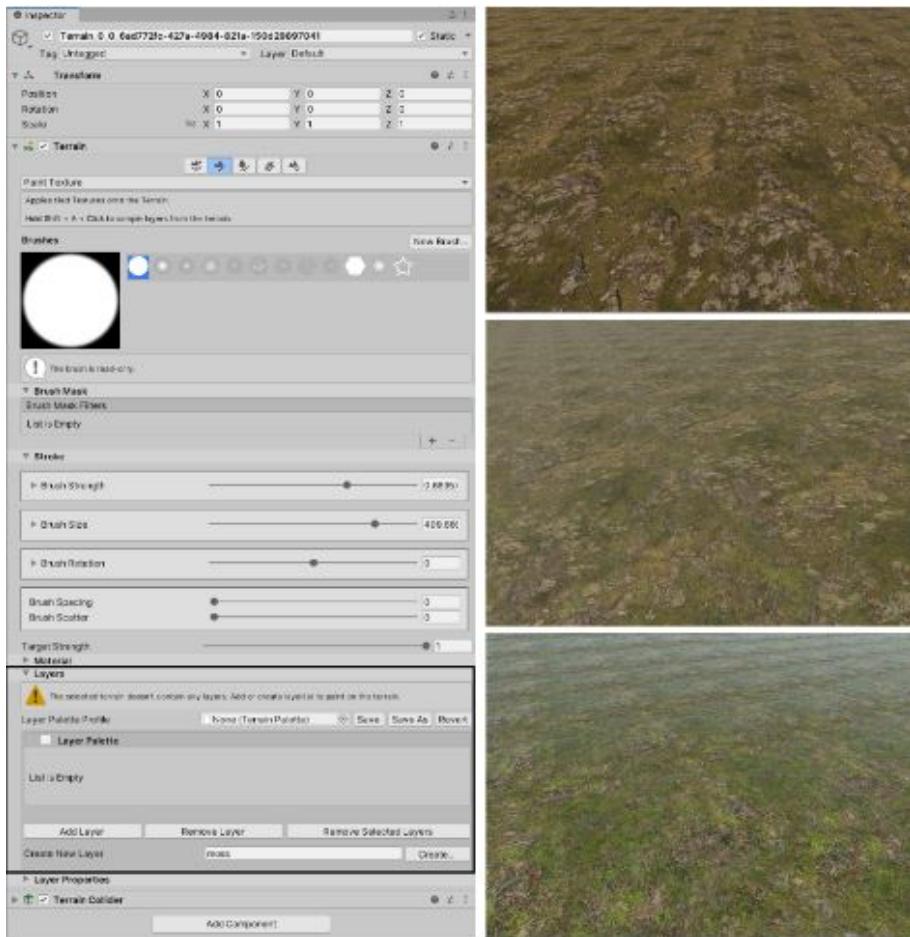
Rysunek 4.2: Widok na panel narzędziwa Terrain Toolbox z zaznaczonymi wymienionymi sekcjami.



Rysunek 4.3: Widok panelu inspektora oraz terenu przed (górny) i po (dolny) zastosowaniu narzędzia Smooth Terrain.

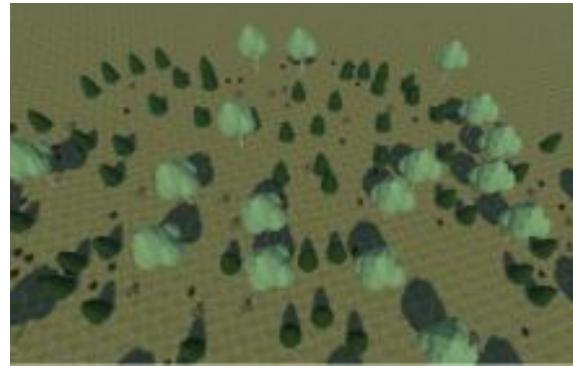
Kolejnym krokiem jest nałożenie tekstur. Służy do tego narzędzie Paint Texture. Umożliwia ono dodanie warstw, którymi będzie można pokolorować teren. Warstwa znajdująca się najwyższej jest uznana

wana za domyślną i jej tekstura zostanie nałożona na cały teren. Pozostałe warstwy natomiast stanowią swego rodzaju paletę kolorów, którymi można pomalować teren za pomocą pędzla, który można wybrać w zakładce Brushes. W zakładce Stroke można ustawić podstawowe parametry pędzla takie jak Brush Size oraz Brush Strength. Pierwszy parametr odnosi się do rozmiaru pędzla, a co za tym idzie obszaru na który dana tekstura zostanie nałożona, natomiast drugi pozwala na określenie w jakim stopniu nakładany materiał zakryje już nałożony.

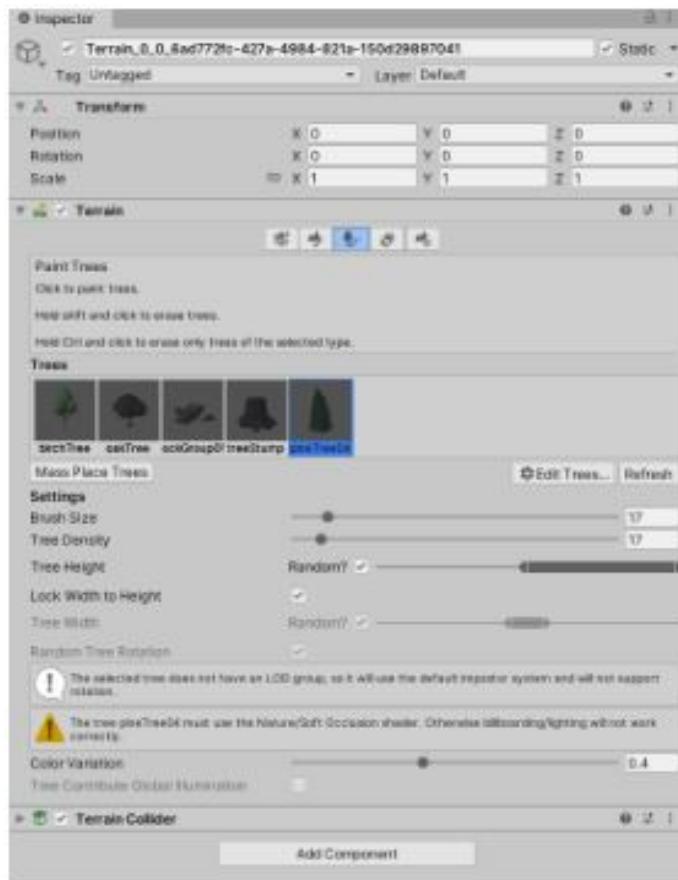


Rysunek 4.4: Widok panelu inspektora z wybranym narzędziem Paint Texture oraz efektu przed nałożeniem 2 tekstyury (górny zrzut), po nałożeniu tekstyury gdy Brush Strength wynosi 0.26 (środkowe zdjęcie) oraz gdy wynosi 0.67 (dolne).

Kolejną opcją udostępnianą przez Unity jest możliwość automatycznego ustawienia obiektów na mapie w losowy sposób, dzięki narzędziu Paint Trees. Do udostępnianych przez nie parametrów należą między innymi Brush Size, działający analogicznie jak poprzednio, oraz Tree Density, które definiuje średnią liczbę drzew umieszczanych na zdefiniowany obszar. Obok przedstawiono przykładowy rezultat. Wykorzystano do tego paczkę LowPoly Trees and Rocks dostępną w Unity Assets Store.



Rysunek 4.5: Widok na teren z drzewami.



Rysunek 4.6: Widok na inspektor z włączonym narzędziem Paint Trees.

4.5. Interfejs użytkownika (Zofia Sosińska)

Interfejs użytkownika (ang. user interface, UI) jest to zbiór najważniejszych informacji przedstawiony graczowi w sposób czytelny. Może się to odbywać za pomocą na przykład obrazków, tekstów, czy wskaźników. Dzięki UI możliwy jest wgląd w aktualny stan wiedzy grywalnej postaci. Projekt interfejsu użytkownika przewiduje trzy tryby: zwykły, budowania oraz walki. Zadaniem każdego z nich będzie odzwierciedlenie aktualnej wiedzy granej postaci z naciskiem na najpotrzebniejsze w danej chwili informacje.

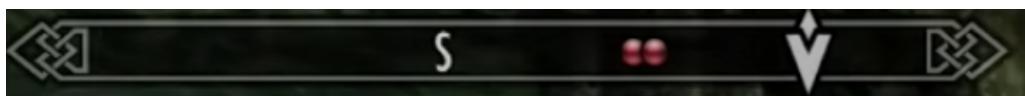
Tryb zwykły

Tryb zwykły przewiduje podstawowe funkcje, takie jak pokazanie: surowców i funduszy, aktualnego czasu w grze, kompasu, ikony postaci, na które gracz może się przełączyć; Inspiracją dla górnego paska z informacjami oraz dla ikon dostępnych postaci jest ten użyty w grze Warcraft 3. Prostota i surowość użytego stylu będą współgrać z klimatem gry.



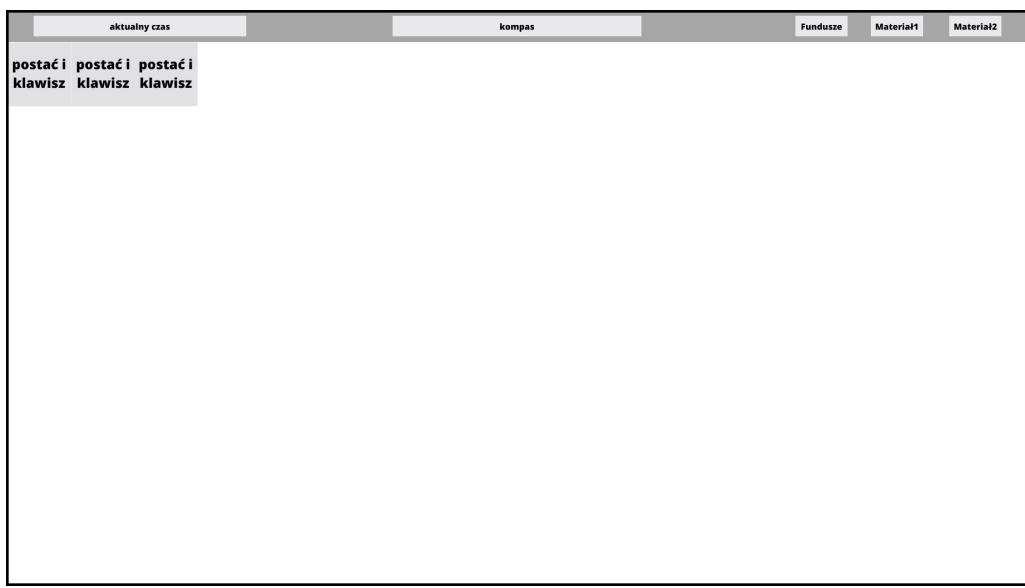
Rysunek 4.7: Pasek z informacjami oraz dla ikony dostępnych postaci w grze Warcraft 3.

W naszej grze skupimy się jednak na tym, aby interfejs użytkownika zabierał jak najmniej miejsca. Dlatego też projekt zakłada, że poszczególne obiekty nie będą ze sobą połączone, a jedynie “dryfować” w przestrzeni. Jako ważny element tej części UI zawarty zostanie kompas, wzorowany na tym z gry The Elder Scrolls V: Skyrim.



Rysunek 4.8: Kompas z gry Skyrim

Szacowany projekt trybu zwykłego UI wyświetlono na rysunku 4.9 .



Rysunek 4.9: Projekt trybu zwykłego UI.

Tryb budowania

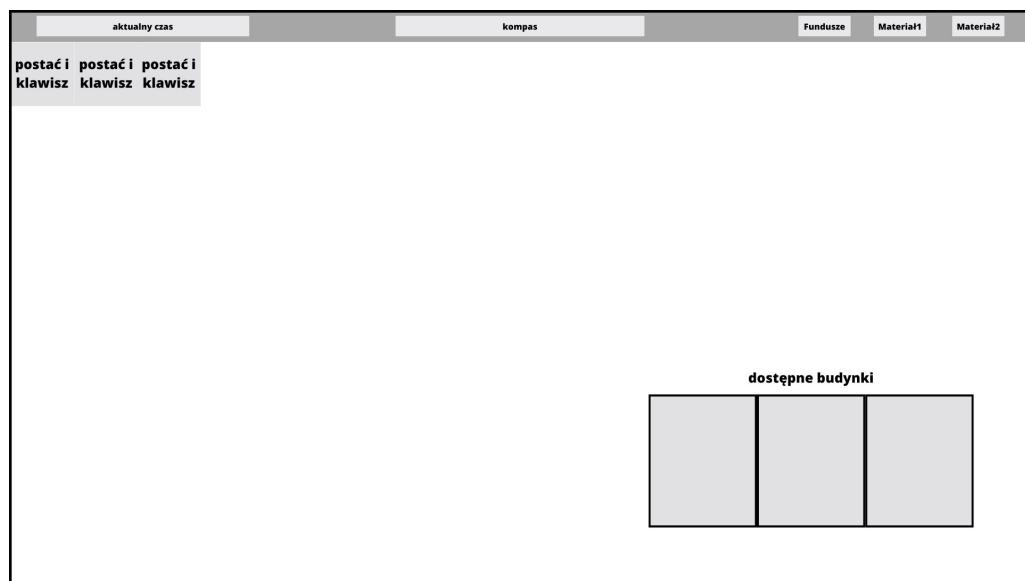
W trybie budowania informacje wcześniej przedstawione zostaną na ekranie. Dodatkowo pokażą nam się dostępne do zbudowania budynki, a po wybraniu pojawią się przed nami. Po zatwierdzeniu

budynek zostanie wybudowany. Inspiracją do przedstawienia dostępnych budowli jest rozwiązanie gry Orcs must die!



Rysunek 4.10: Wyświetlenie dostępnych pułapek w Orcs must die!

Szacowany projekt trybu budowania UI wyświetlono na rysunku 4.11 .

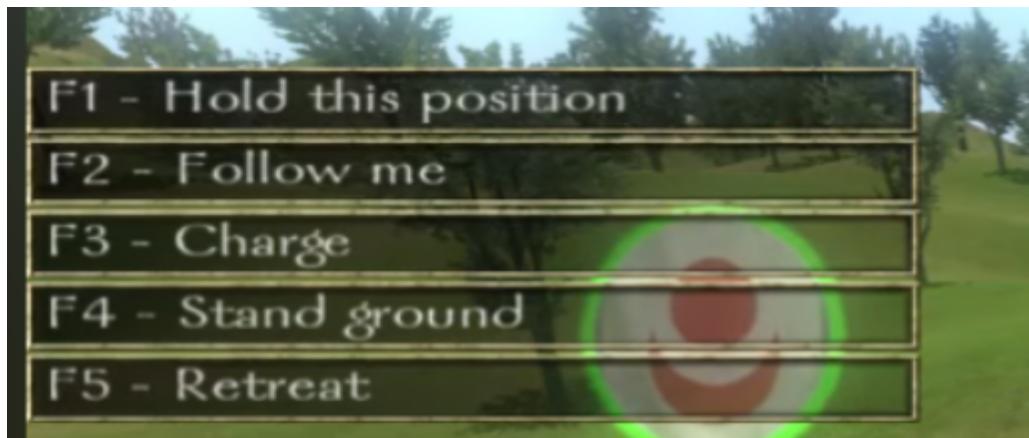


Rysunek 4.11: Projekt trybu budowania UI.

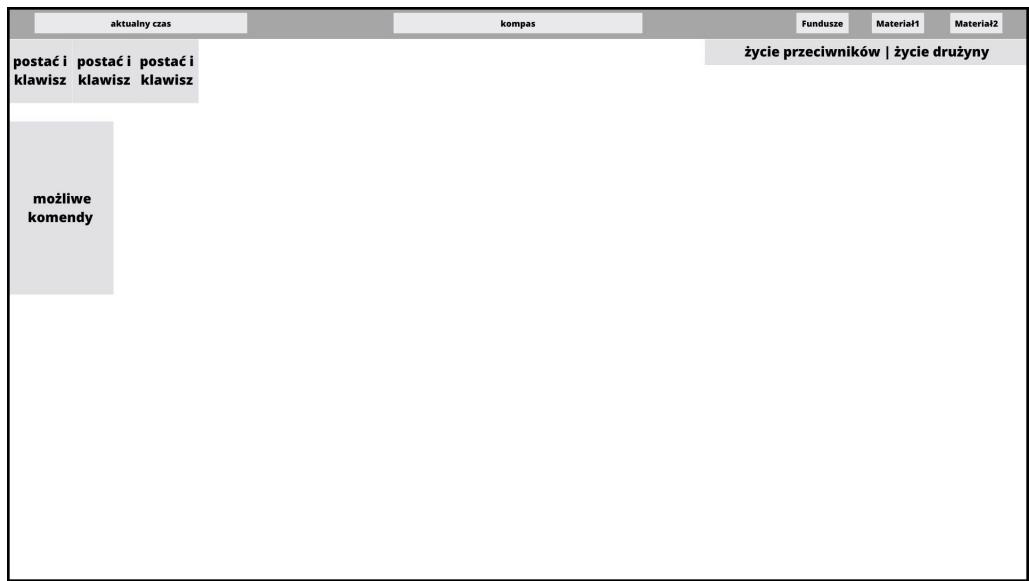
Tryb walki

Bliźniaczko do trybu budowania, gdy rozpocznie się walka, podstawowe informacje zostają na ekranie, a dodatkowo gracz dostaje informacje o dostępnych rozkazach do wydania. Nasze rozwiązanie będzie podobne do pomysłu z gry Mount&Blade.

Szacowany projekt trybu budowania UI wyświetlono na rysunku 4.13 .



Rysunek 4.12: Wykaz dostępnych rozkazów z gry Mount&Blade.



Rysunek 4.13: Projekt trybu walki UI.

4.6. Nawigacja (Zofia Sosińska)

Kluczowym dla gry założeniem jest ułatwienie graczowi wczucia się w realia świata, w którym się znajduje. Jako jeden z głównych warunków pogłębienia immersji uwypuklono brak implementacji mapy, na której gracz widziałby świat. W ten sposób nie upraszczamy mu poruszania się i odnajdywania lokacji tak, jak i człowiek w realnym świecie w czasach średniowiecznych nie kierował się zapisanymi na kartce kartograficznymi obrazami, ale własną i zdobytą od innych wiedzą o otaczającym go terenie. Jedyną pomocą, jaką otrzyma gracz, będzie pasek obrazujący pole widzenia granej postaci. Pierwszą rolą narzędzia będzie pokazanie kierunku świata, który znajduje się w polu widzenia gracza. Zakładamy, że grana postać potrafi sama taką informację odczytać chociażby z położenia Słońca. Nie jest to więc sztucznie upraszczająca rozgrywkę mechanika, a jedynie odciążenie użytkownika kompasem. Kolejną informacją na omawianym polu będzie miejsce w którym znajduje się przeciwnik. Dotyczy to antagonistów widocznych w polu widzenia, jak i ukrytych za ścianą. Druga część będzie logicznie możliwa dzięki specjalnej umiejętności widzenia wrogów za przeszkodą zaimplementowanej dla postaci druida. Po przyłączeniu takiej osoby do drużyny użytkownik może poprosić przyjaciela o użyczenie mu swej mocy.

4.7. Mechanizm budowania (Bogna Lew)

Jednym z istotnych elementów w grach real-time strategy jest tworzenie baz i budowanie fortyfikacji. Mechanizm ten stanowi urozmaicenie rozgrywki i wprowadza dodatkowe aspekty możliwe do uwzględnienia w planowaniu strategii. Dla wielu gier RTS jest wręcz nieodłącznym elementem, który umożliwia graczwowi tworzenie i rozwój nowych jednostek, produkcję zasobów, umacnianie swojej pozycji oraz zwiększanie swojej potęgi.

Rozpatrując ten mechanizm konieczne jest uwzględnienie podstawowych ograniczeń związanych z ukształtowaniem terenu oraz rozmieszczeniem już istniejących obiektów i elementów scenerii. Czynniki te mają ogromne znaczenie, gdyż zignorowanie ich może doprowadzić do błędów oraz niepożądanych efektów i w rezultacie do obniżenia jakości rozgrywki. Dodatkowo kluczową kwestią są zasoby wymagane do wybudowania danego obiektu. Ich istnienie powoduje, że gracz jest w pewien sposób ograniczony i nie może tworzyć budowli w dowolnej ilości.

Jednym z błędów, którym należy przeciwdziałać jest nakładanie się na siebie obiektów. Taka sytuacja może powstać w wyniku braku rozpatrywania przez grę położenia elementów terenu i istniejących budowli w trakcie umieszczania na mapie nowych budynków przez gracza. W efekcie może dojść do sytuacji, że obiekt zostanie wybudowany w miejscu zajmowanym przez inną część scenerii. Gra powinna przeciwdziałać temu zjawisku, uniemożliwiając graczwowi tworzenie nowych budynków w miejscu, które w nawet najmniejszym stopniu narusza obszar zajęty przez już istniejący element.

Inną sytuacją, której należy unikać jest umożliwienie graczwowi wybudowanie budynku w sposób fizycznie niemożliwy. Przykładem może być ulokowanie obiektu na zbyt stromym zboczu albo na nieodpowiednim gruncie, co w rzeczywistym świecie byłoby niedopuszczalne. Również w tym przypadku gra nie powinna umożliwić graczwowi wykonanie takiej akcji. W przeciwnym razie obiekt mógłby zostać umieszczony w miejscu do którego postać gracza nie będzie mieć dostępu i nie będzie mógł z niego korzystać. W efekcie zużyje on bezsensownie swoje zasoby tracąc je bezpowrotnie, co może negatywnie wpływać na jego opinię o grze.

Inspiracją do implementacji tego mechanizmu jest gra Warhammer 40,000: Dawn of War. Chociaż realia tej gry znacząco różnią się od tych w których zostanie osadzona fabuła tworzonej przez nas gry,

to stanowi świetny przykład pożądanej efektu. Udostępnia ona możliwość podglądu z trzeciej osoby przed wybudowaniem wraz z walidacją położenia. Jeśli miejsce w którym gracz chce postawić budynek jest poprawne tzn. nie nachodzi na inne obiekty oraz teren jest odpowiedni to pokazywany widok jest podświetlany na zielono, w przeciwnym razie - na czerwono. Jest to zbliżony efekt, który chcielibyśmy uzyskać dla widoku pierwszoosobowego.

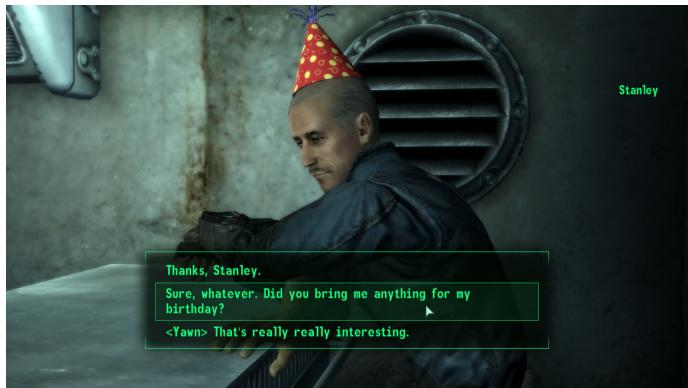
4.8. Sterowanie jednostkami, podążanie za główną postacią (Zofia Sosińska)

Na samym początku gry stworzona przez gracza postać pojawia się sama. W tym momencie jest to jedyny obiekt, którym gracz może sterować. Kontroluje, gdzie postać idzie, jak walczy oraz z kim rozmawia. Z biegiem czasu gra będzie jednak naciskać na formowanie drużyny, ponieważ pokonywanie wielu przeciwników w pojedynkę będzie się stawało zbyt trudne. Pojawia się w takim momencie problem. Stworzenie mechaniki poruszania się i oddziaływania na otaczający świat dla jednej postaci wydaje się być proste i intuicyjne, ale kierowanie wieloma osobami już nie. ez wprowadzenia zmian, używając jednego sposobu dyrygowania wszystkimi jednostkami tak samo, gra będzie prędko męczyć gracza. Dla czynności mało znaczących, takich jak przemieszczenie drużyny w konkretne miejsce wprowadzi monotonię i czasochłonność. Każdą postać należy wybrać i przemieścić ją w konkretne miejsce. Kilka kliknięć przy jednej postaci jest akceptalne, ale przy kilku wprowadzi to ogromne opóźnienia. Gracz byłby już wielokrotnie dalej. Jeszcze gorsze skutki pokazałyby się podczas walki. Szybkie przeskakiwanie pomiędzy postaciami podniosłoby zauważalnie trudność gry. Poruszanie się jedną postacią i zabijanie przeciwników nie ma sensu, gdy reszta drużyny jest bita i nie może się obronić, ponieważ gracz musi się przełączyć na inną postać, aby ta wykonała ruch. Stąd potrzeby jest algorytm odpowiadający za właściwe poruszanie się pobocznych postaci. Tworzona gra będzie dopuszczała małe, kilkunastoosobowe drużyny z przywódcą - postacią grywalną przez użytkownika - na czele. Podczas walki graczowi pokażą się możliwe do wydania polecenia oraz specyfikacja, jakiej grupy mają one dotyczyć. Za pomocą określonych klawiszy klawiatury będzie on mógł kontrolować zachowanie kompanów. Po rozwiązaniu problemu mechaniki sterowania jednostkami w walce, nie można przeoczyć samego poruszania się oraz interakcji ze światem. Najprostszym rozwiązaniem będzie implementacja mechanizmu, według którego drużyna, po wykryciu znacznego przemieszczenia się przywódcy, sama będzie za nim podążać. Kompani nie będą też mieli opcji samodzielnej interakcji ze światem. Poza walką zostaną jedynie biernymi obserwatorami.

4.9. System dialogów (Bartosz Strzelecki)

System dialogów jest podstawową metodą, którą gracz będzie wykorzystywał, aby pozyskać informacje o świecie oraz celach misji. Gracz może inicjować konwersacje z postaciami niezależnymi, po czym zostaną mu zaproponowane opcje sposobu prowadzenia rozmowy. W zależności od wybranych opcji dialogowych gracz może się spodziewać różnych konsekwencji.

Dialogue Editor autorstwa Grasshop Dev jest prostym narzędziem pozwalającym na szybkie dodawanie i modyfikację dialogów. Zawiera zestaw elementów ułatwiających wdrożenie systemu do projektu oraz udostępnia struktury danych wykorzystywanych do tworzenia interfejsu użytkownika. Podczas rozmowy z postaciami niezależnymi gracz będzie mógł pozyskać informację o geografii świata, możliwych zagrożeniach oraz zadaniach do wykonania. Podobne systemy występują w grach takich jak Pillars of Eternity oraz w grach z serii Mass Effect.



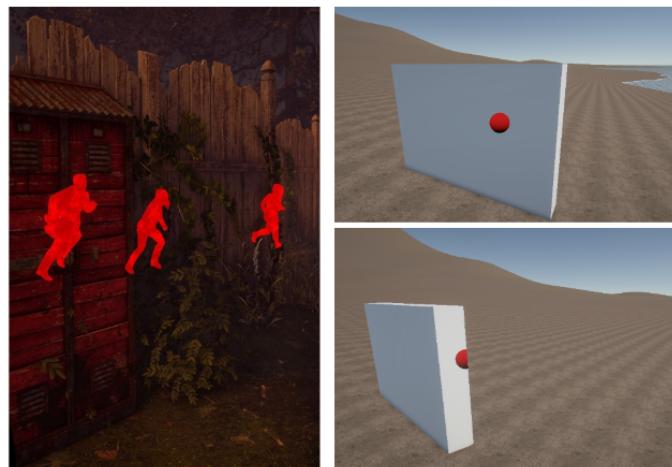
Rysunek 4.14: Kadr z gry Fallout 3 przedstawiający przykładowy dialog

4.10. Sztuczna inteligencja (*Bartosz Strzelecki*)

W przypadku implementacji mechanizmów sztucznej inteligencji przeciwników będziemy się inspirować trybami kampanii w grach Warcraft III oraz Starcraft II. Na mapie będą rozsiane punkty, w których będą pojawiać się przeciwnicy. Tak długo, jak drużyna gracza jest poza zasięgiem, wrogowie pozostają nieaktywni. Aktywni przeciwnicy zachowują się zgodnie z ich archetypem (klasą postaci) oraz pozostają aktywni tak długo, jak drużyna gracza jest w zasięgu. Zdezaktywowani przeciwnicy wracają do swojego oryginalnego stanu. Gracz będzie napotykał tego typu obozowiska przede wszystkim w trakcie eksploracji świata. Innym planowanym przykładem implementacji sztucznej inteligencji przeciwników jest model, w którym jednostki wroga poruszają się z punktu początkowego w stronę bazy gracza. Jeśli podczas swojej podróży napotkają drużynę gracza, wtedy niezwłocznie zmieniają swój cel ataku. Będziemy wyróżniać trzy archetypy jednostek w zależności od sposobu walki (bliski zasięg, średni zasięg, daleki zasięg). Postacie walczące na bliski zasięg mają na celu podejście w stronę najbliższego przeciwnika i wykonać atak. Jednostki średnio zasięgowe w momencie, w którym najbliższy przeciwnik jest odpowiednio daleko, wykonują atak dystansowy, w przeciwnym wypadku zachowują się tak jak jednostki walczące w zwarciu. Postacie dalekodystansowe dokonują ataków dystansowych w kierunku najbliższego przeciwnika, natomiast uciekają, gdy ten podejdzie zbyt blisko.

4.11. Widzenie przez ściany (*Bartosz Strzelecki*)

Do umiejętności wykorzystywanych przez gracza będzie należeć zdolność widzenia przeciwników oraz innych istotnych obiektów przez przeszkody. Gracz po wcisnięciu przycisku przez krótki okres będzie w stanie zobaczyć sylwetki przeciwników znajdujących się w jego polu widzenia. Grafika przedstawia rozwiązanie zawarte w grze Dead by Daylight. Markery nie poruszają się za celem lecz pojawiają się i pozostają w tym samym miejscu przez czas trwania animacji.



Rysunek 4.15: Po lewej stronie przykład z gry Dead by Daylight. Po prawej zachowanie shadera w przypadku przysłanianiu markera przez przeszkode.

5. IMPLEMENTACJA

Ten rozdział skupia się na zaprezentowaniu implementacji projektu w silniku Unity. Przedstawia on wykorzystane metodyki i narzędzia wykorzystane do osiągnięcia określonych wymagań. Jak również zawiera fragmenty wykorzystanych algorytmów i zrzuty ekranu reprezentujące efekty pracy.

5.1. Kontroler postaci (*Bogna Lew*)

W trakcie rozgrywki istotnym aspektem wpływającym na jakość jest mechanizm sterowania postacią. Z tą mechaniką gracz ma bezpośredni kontakt, ponieważ to właśnie za jej pomocą może eksplorować świat.

Do implementacji tego mechanizmu zainspirowaliśmy się grą Skyrim. Postać jest sterowana za pomocą klawiszy "w", "a", "s" oraz "d", natomiast jej rotacja oraz obrót kamery jest kontrolowany przez mysz. Całość została podzielona na cztery części.

Pierwsza z nich odpowiada za przemieszczanie się postaci. Do tego wykorzystuje Input Manager, w którym zostały zmapowane odpowiednie klawisze dla każdej z osi, wzdłuż której gracz może się przemieszczać. W rezultacie powstały dwa kierunki - przód/tył oraz prawo/lewo. Naciśnięcie odpowiedniego przycisku skutkuje zwróceniem wartości 1 bądź -1, które symbolizują zwrot wektora przemieszczenia wzdłuż osi. Na tej podstawie wyznaczane jest faktyczne przesunięcie postaci względem kierunku, w którym jest zwrócona oraz modyfikowane jest jej położenie. Dodatkowo ten komponent odpowiada za wyznaczenie prędkości, z jaką gracz się porusza. Domyslnie postać przemieszcza się tempem chodu, jednakże jeśli gracz przytrzyma lewy klawisz Shift, to zacznie się poruszać biegiem.

Druga część kontrolera odpowiada za ustawienie odpowiedniej animacji. Do tego wykorzystuje ona wartości zwrotu wzdłuż obu osi oraz prędkość, które są wyznaczane w poprzednim komponencie. Na ich podstawie wyznacza kolejne części nazwy animacji, po czym, jeśli jest inna niż aktualnie wyświetlana, uruchamia ją.

Kolejny komponent nadzoruje rotację postaci oraz co za tym idzie - kamery. Analogicznie wykorzystywany jest Input Manager, jednak w tym przypadku przechwytywane jest przesunięcie myszy. Komponent udostępnia graczu możliwość sterowania kierunkiem, w którym postać patrzy, a co za tym idzie - względem którego się przemieszcza. Jednocześnie następuje dostosowanie położenia kamery tak, aby zawsze patrzyła w tym samym kierunku co postać gracza. Ponadto komponent umożliwia przesunięcie kamery po łuku do góry bądź do dołu. Dzięki temu gracz może dokładniej zobaczyć, co znajduje się nad oraz tuż przed nim.

Ostatni komponent odpowiada za kontrolowanie przybliżania kamery. W tym celu w Input Managerze zostało zmapowane kółko myszy. Zwrócona przez system wartość jest wykorzystywana do obliczenia odległości kamery od postaci przy zachowaniu ustalonych przez grę ograniczeń. Dodatkowo komponent dokonuje również automatycznego przybliżania, jeśli wskutek przemieszczania się postaci, pomiędzy nią a kamerą miałaby się znaleźć przeszkoda. Dzięki temu nie zostanie zasłonięty widok tego co się dzieje wokół postaci. Odsunięcie się od przeszkode zaskutkuje oddaleniem kamery do poprzedniej odległości.

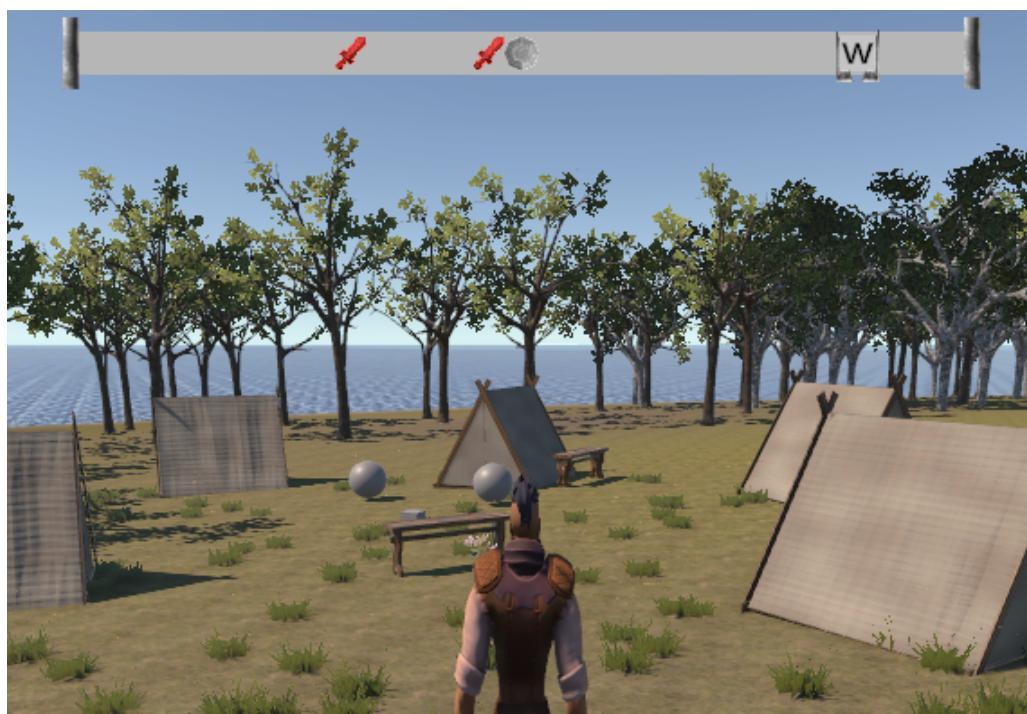
5.2. Zasady działania kompasu (Zofia Sosińska)

Projekt kompasu jest na tyle prosty, aby nie przytłoczyć gracza nadmierną liczbą bodźców. Składa się z horyzontalnego, jednolitego paska, na którym wyświetlane będą najważniejsze informacje o otoczeniu, symbol ośmiokąta, wskazujący na przestrzeń znajdującą się centralnie przed bohaterem oraz z bocznych pasków, wyróżniających końce narzędzia.



Rysunek 5.1: Rozpiska elementów: a. główny pasek, b. symbol środka, c1., c2. paski końców kompasu.

Ikony wyświetlane na kompasie będą przedstawiać najważniejsze informacje w polu widzenia gracza, czyli stronę świata, w kierunku której jest on zwrócony, oraz przeciwników. Poniżej przedstawiono kod odpowiedzialny za wyznaczenie pozycji symbolu na omawianym narzędziu. Wejściem jest komponent RectTransform symbolu przypisanemu danemu obiektemu oraz jego położenie. Po obliczeniu wektora prowadzącego do uzyskania np. pozycji wroga, wyznaczany jest kąt, o jaki musi się obrócić. To przeliczamy na pozycję na kompasie i jeśli obiekt jest w polu widzenia, wyświetlamy symbol.



Rysunek 5.2: Wizualizacja przypadku, w którym gracz patrzy centralnie na obozowisko wrogów. Na przeciwnie oraz po lewej stronie znajdują się przeciwnicy, co jest zasygnalizowane na kompasie za pomocą symboli mieczy. Znajduje się na nim także informacja, że bohater jest lekko odchylony od Wschodu.

Listing 5.1: Fragment kodu odpowiedzialny za ustawienie symbolu na pasku kompasu

```
void SetMarkerPosition(RectTransform markerTransform, Vector3 worldPosition)
{
    Vector3 dirToTarget = worldPosition - CameraTransform.position;
```

```

        float angle = Vector2.SignedAngle(
            new Vector2(dirToTarget.x, dirToTarget.z),
            new Vector2(CameraTransform.transform.forward.x,
            CameraTransform.transform.forward.z));
        float compassPositionX = Mathf.Clamp(
            2 * angle / Camera.main.fieldOfView, -1, 1);
        if (compassPositionX == 1 || compassPositionX == (-1))
        {
            markerTransform.anchoredPosition = new Vector2(0, 100);
        }
        else
        {
            markerTransform.anchoredPosition = new Vector2(
                compassBarTransform.rect.width / 2 * compassPositionX, 0);
        }
    }
}

```

Problem lokalizacji stron świata pojawił się, gdy nieprawidłowo wyświetlały się symbole po użyciu najprostszego rozwiązania. Na przykład dla Północy było to pobranie wartości od Vector3.forward, co jest skrótnym zapisem Vector3(0, 0, 1). Lewy dolny róg mapy jest położony w punkcie (0, 0, 0), co oznacza, że wymagane jest przesunięcie, aby prawidłowo zasymulować strony świata. Północ i Południe przesunęliśmy do połowy szerokości, a Wschód i Zachód - długości mapy. Każde z nich oddaliliśmy o 60000 jednostek, co pozwala na wiarygodną simulację stron świata na kompasie.

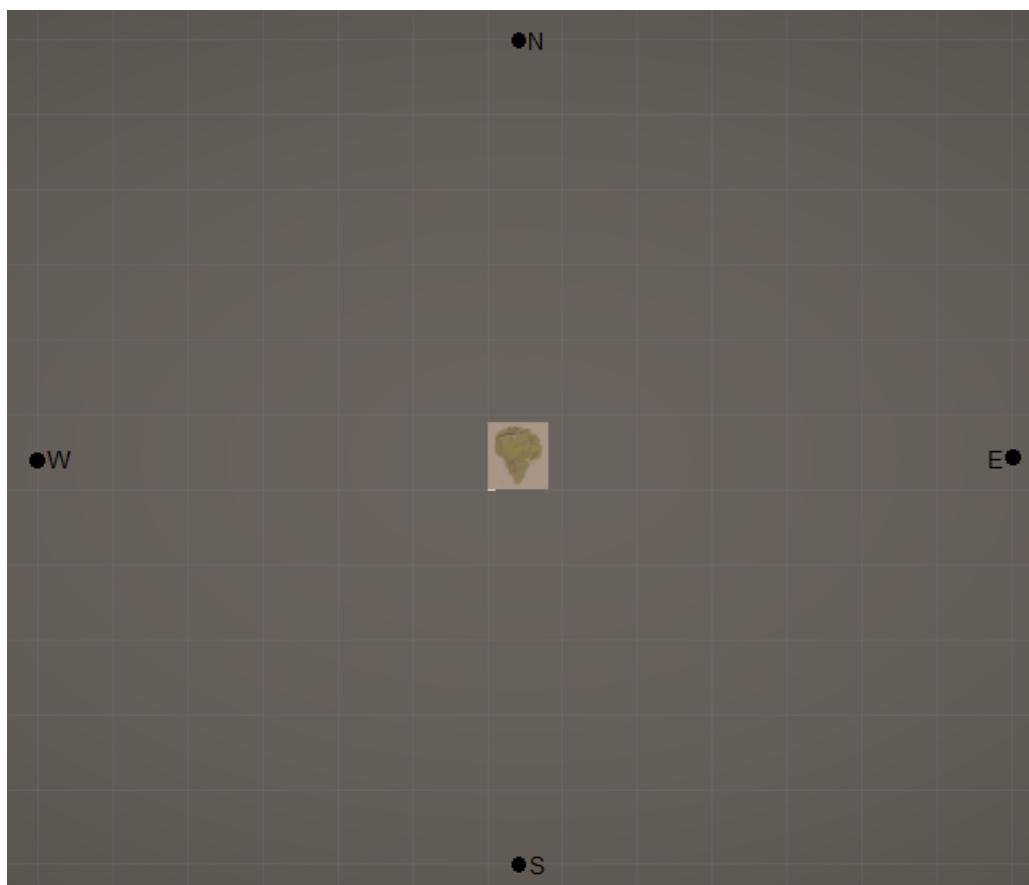
Wykrycie wrogich jednostek znajduje się w funkcji Start. Każdemu obiekowi jest przypisywany symbol czerwonego miecza i w zależności od położenia wroga, obrazek wyświetlany jest odpowiednio na kompasie.

Listing 5.2: Fragment kodu odpowiedzialny za połączenie wrogich obiektów na mapie z symbolami wyświetlonymi na kompasie

```

void SetPositionOfEnemies()
{
    foreach (
        var e in enemiesOnMap.Zip(
            enemiesOnUI, (x, y) => new {
                enemyOnMap = x, enemyOnUI = y }))
    {
        SetMarkerPosition(
            e.enemyOnUI.GetComponent<RectTransform>(),
            e.enemyOnMap.transform.position);
    }
}

```



Rysunek 5.3: Rozmieszczenie zasymulowanych stron świata.

5.3. Mechanizm budowania (Bogna Lew)

Opracowany mechanizm umożliwia graczowi na przełączenie się w tryb budowania poprzez naciśnięcie klawisza Tab, który od razu wyświetli podgląd bazowego obiektu. Widok budynku przemieszcza się przed postacią oraz odpowiednio obraca się razem z nią. Efekt poruszania się podglądu został uzyskany za pomocą poniższych wzorów:

$$x = r \times \sin(\alpha)$$

$$y = 0$$

$$z = r \times \cos(\alpha)$$

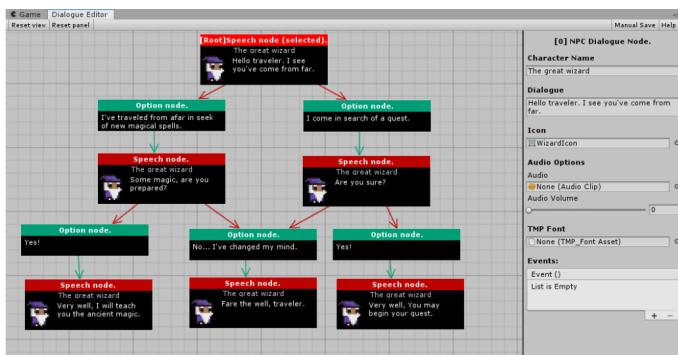
gdzie r to odległość środka obiektu od postaci gracza, natomiast α to kąt o jaki jest on obrócony względem osi y.

Podgląd budynku będzie widoczny do czasu, aż gracz go umieści naciskając prawy przycisk myszy, bądź wychodząc z trybu edycji naciskając klawisz Escape. Jeżeli widok budowli znajduje się w poprawnym do umieszczenia miejscu to jest on podświetlany na zielono, w przeciwnym razie - na czerwono. Wybudowanie powoduje przywrócenie bazowych kolorów obiektu oraz włącza wykrywanie kolizji z nim, dzięki czemu budynek poprawnie oddziałuje z otaczającym go środowiskiem.

5.4. System dialogów (Bartosz Strzelecki)

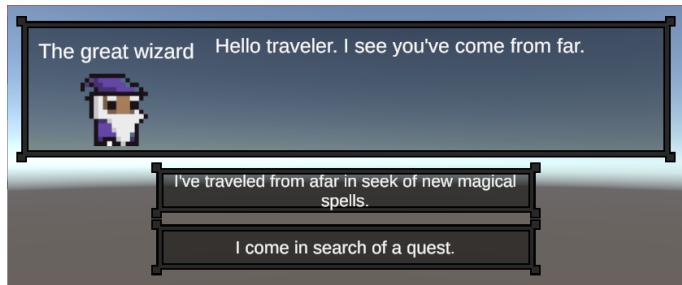
W naszej implementacji konwersacje składają się z dwóch rodzajów węzłów. Jednego odpowiedzialnego za wypowiedzi postaci niezależnych oraz drugiego pozwalającego na podjęcie przez gracza decyzji. Dodanie nowej konwersacji odbywa się poprzez stworzenie obiektu z przypisanym komponentem NPC Conversation. Wywołanie dialogu można osiągnąć poprzez wykorzystanie metody

```
ConversationManager.Instance.StartConversation(/*NPCConversation*/);
```



Rysunek 5.4: Przykładowa konwersacja z wykorzystaniem Dialogue Editor

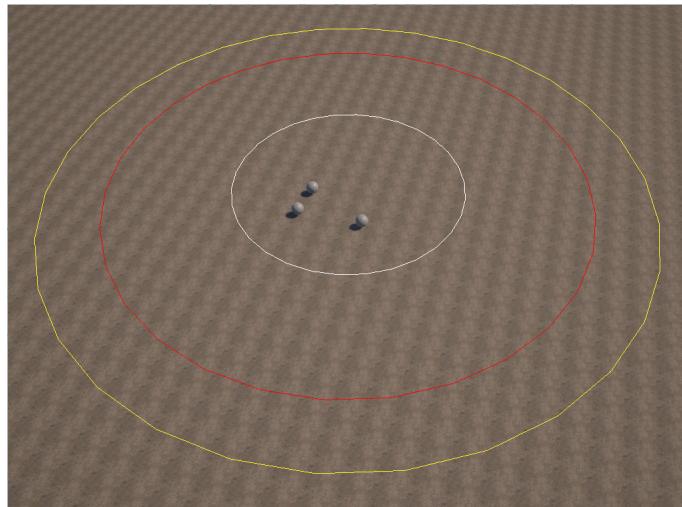
System pozwala na łatwe dostosowanie elementów interfejsu użytkownika odpowiedzialne za wyświetlenie dialogów, aby jak najlepiej wpasować się w styl graficzny gry.



Rysunek 5.5: Przykładowe okno dialogowe widziane z perspektywy gracza.

5.5. Sztuczna inteligencja (*Bartosz Strzelecki*)

Nawigacja przeciwników została zrealizowana poprzez wbudowany w silnik Unity system NavMesh. Pozwala nam on na łatwe wyznaczenie powierzchni, po której mogą poruszać się postacie niekontrolowane przez gracza oraz realizuje zadanie wyznaczania ścieżki dla tych postaci.



Rysunek 5.6: Obraz przedstawia zasięgi odpowiednich regionów.

Przeciwnicy są kontrolowani poprzez jeden obiekt przydzielający cele każdemu przypisanemu wrogowi. W normalnym trybie wrogowie poruszają się w sposób losowy w obrębie wyznaczonej przestrzeni (biały okrąg). Kiedy przyjazne jednostki znajdą się w wystarczającej odległości (czerwony okrąg), przeciwnicy obiorą sobie za cel jedną z nich. Po opuszczeniu przez drużynę gracza wyznaczonego obszaru (żółty okrąg) wrogowie wracają do poruszania się w sposób losowy w obrębie białego okręgu.

5.5.1. Mechanika zachowań klas jednostek

Jednostki bliskozasięgowe

Zachowanie jednostek bliskozasięgowych polega na wybraniu przeciwnika będącego w czerwonym obszarze, który ma najwyższą wartość priorytetu obliczonego między innymi na podstawie odległości, ilości innych atakujących przeciwników oraz klasy celu. Jednostka posiadająca cel ataku zmierza w jego kierunku, obierając najkrótszą drogę. Po dotarciu do celu podróży jednostki biorące udział w walce losują naprzemiennie liczby, od których zależy wynik walki, jak i aktualnie wykorzystywana animacja.

Jednostki zasięgowe

Takie jednostki wystrzeliwują w stronę przeciwników pociski, których celność jest reprezentowana przez dwie strefy. Jedną oznaczającą 50% celności, czyli rozrzut, który będzie posiadała połowa pocisków, oraz drugą oznaczającą maksymalny rozrzut. Ten mechanizm pozwala na łatwe zamodelowanie celności ze względu na odległość, wielkość celu oraz na osłony, za którymi może stać wroga jednostka.



Rysunek 5.7: Reprezentacja graficzna celowania widziana z perspektywy jednostki dalekozasięgowej. Czerwony okrąg reprezentuje obszar, w którym znajdzie się 50% pocisków. Żółty obszar pokazuje maksymalny rozrzut.



Rysunek 5.8: System celowania występujący w grze Phoenix Point.

Jednostki hybrydowe

Obejmują zachowanie dwóch powyższych klas jednostek w zależności od odległości od przeciwnika. W sytuacji, w której wroga jednostka znajduje się w pobliskim otoczeniu, wykorzystywany jest kod

przeznaczony dla bliskozasięgowych jednostek, w przeciwnym przypadku wybierana jest logika jednostek zasięgowych.

5.5.2. Sztuczna inteligencja przyjaznych jednostek.

Kontrola jednostek przez gracza odbywa się za pomocą wybory jednej z opcji w dwóch fazach

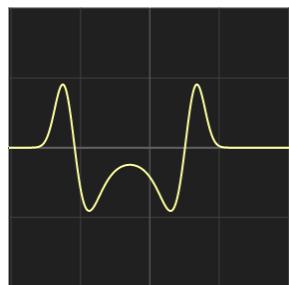
- Faza wybrania jednostek
 - Wszyscy
 - Jednostki bliskozasięgowe
 - Jednostki hybrydowe
 - Jednostki dalekozasięgowe
 - Opcja anulowania wyboru
- Faza wydania rozkazu
 - Podążanie za graczem
 - Zatrzymanie
 - Atak
 - Podejście do wskazanego przez gracza miejsca
 - Ucieczka
 - Opcja anulowania rozkazu

Wydanie rozkazu polega na kliknięciu przycisku odpowiedzialnego za wejście w tryb wydawania poleceń, a następnie wybraniu numeru opcji reprezentującej grupę jednostek, której komenda ma dotyczyć. Ostatecznie należy podać numer rozkazu, który zostanie wydany. Przykładowa reprezentacja graficzna systemu jest widoczna na rysunku 4.12. Wykorzystanie tego modelu pozwala na łatwe rozwinięcie systemu o dodanie nowych metod wyboru grup jednostek oraz możliwych instrukcji dla przyjaznych agentów.

5.6. Widzenie przez ściany (Bartosz Strzelecki)

Efekt został osiągnięty poprzez zmodyfikowanie potoku renderowania w taki sposób, że w zależności od wartości w buforze głębi jest wykorzystywany inny shader. W tym przypadku, jeżeli sfera jest przesłonięta przez ścianę jest ona narysowana w przeciwnym wypadku jest uruchamiany pusty shader.

Po naciśnięciu przycisku E następuje zagranię animacji opisanej wzorami $w(t, offset) = 1.1 \times 2.1 - \left(\frac{(\sin(t) + 1 - 0.4 - offset)^2}{0.02} \right)$ oraz $w(t, 0) - w(t, -0.2) + w(t, -1) - w(t, -1.2)$. Od podanych funkcji zależy przeźroczystość, jak i natężenie efektu Fresnela.



Rysunek 5.9: Wykres przedstawiający funkcję opisującą zachowanie efektu Fresnela w animacji markera

Listing 5.3: Fragment shadera odpowiedzialny za animacje

```
fixed4 frag (v2f i) : SV_Target
{
    float t = 6.2 * _Progress - 0.6;
    fixed4 pattern = tex2D(_PatternTex, i.uv + _Speed *t);
    float fresnellInfluence = dot(i.worldPos, i.viewDir);
    float saturatedFresnel = saturate(1 - fresnellInfluence);

    float g = w(t, 0) - w(t, -0.2) + w(t, -1) - w(t, -1.2);
    float4 color = pow(saturatedFresnel, g * _FresnelPow) * (_Color * _ColorIntensity);
    color.a *= dot(i.worldPos, i.viewDir);
    return color;
}
```

6. PODSUMOWANIE

WYKAZ LITERATURY

- [1] G. Bobrek. "Historyczne bzdury, które na stałe trafiły do gier." (), [Online]. Available: <https://www.youtube.com/watch?v=-eySMANZ80k&t=2s>. (accessed: 24.06.2023).

WYKAZ RYSUNKÓW

2.1	Mapa basenu Morza Śródziemnego z czasów Imperium Rzymskiego	7
2.2	Przykład koła dialogowego w grze Mass Effect	8
4.1	Przykładowe heightmapy	15
4.2	Widok na panel narzędzia Terrain Toolbox z zaznaczonymi wymienionymi sekcjami.	15
4.3	Widok panelu inspektora oraz terenu przed (górny) i po (dolny) zastosowaniu narzędzia Smooth Terrain.	15
4.4	Widok panelu inspektora z wybranym narzędziem Paint Texture oraz efektu przed nałożeniem 2 tekstury (górnego zrzutu), po nałożeniu tekstury gdy Brush Strength wynosi 0.26 (środkowe zdjęcie) oraz gdy wynosi 0.67 (dolne).	16
4.5	Widok na teren z drzewami.	17
4.6	Widok na inspektor z włączonym narzędziem Paint Trees.	17
4.7	Pasek z informacjami oraz dla ikony dostępnych postaci w grze Warcraft 3.	18
4.8	Kompas z gry Skyrim	18
4.9	Projekt trybu zwykłego UI.	18
4.10	Wyświetlenie dostępnych pułapek w Orcs must die!	19
4.11	Projekt trybu budowania UI.	19
4.12	Wykaz dostępnych rozkazów z gry Mount&Blade.	20
4.13	Projekt trybu walki UI.	20
4.14	Kadr z gry Fallout 3 przedstawiający przykładowy dialog	23
4.15	Po lewej stronie przykład z gry Dead by Daylight. Po prawej zachowanie shadera w przypadku przysłaniania markera przez przeszkode.	24
5.1	Rozpiska elementów: a. główny pasek, b. symbol środka, c1., c2. paski końców kompasu.	26
5.2	Wizualizacja przypadku, w którym gracz patrzy centralnie na obozowisko wrogów. Na przeciwnieko oraz po lewej stronie znajdują się przeciwnicy, co jest zasygnalizowane na kompasie za pomocą symboli mieczy. Znajduje się na nim także informacja, że bohater jest lekko odchylony od Wschodu.	26
5.3	Rozmieszczenie zasymulowanych stron świata.	28
5.4	Przykładowa konwersacja z wykorzystaniem Dialogue Editor	29
5.5	Przykładowe okno dialogowe widziane z perspektywy gracza.	30
5.6	Obraz przedstawia zasięgi odpowiednich regionów.	30
5.7	Reprezentacja graficzna celowania widziana z perspektywy jednostki dalekozasięgowej. Czerwony okrąg reprezentuje obszar, w którym znajdzie się 50% pocisków. Żółty obszar pokazuje maksymalny rozrzut.	31
5.8	System celowania występujący w grze Phoenix Point.	31
5.9	Wykres przedstawiający funkcję opisującą zachowanie efektu Fresnela w animacji markera	32

WYKAZ TABEL

3.1 Porównanie silników.	12
----------------------------------	----