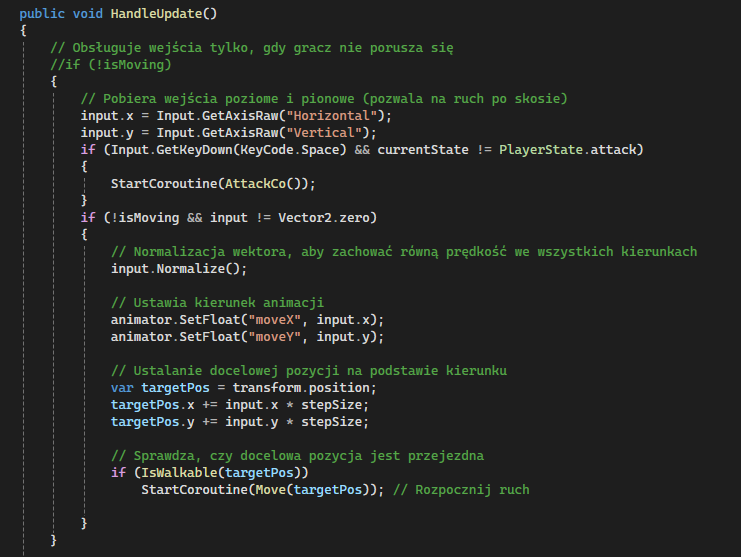
**README - Projekt gry 2D top-down w Unity**

**1. Problem z Ruchem Postaci na Skośnych Kierunkach**

W naszym projekcie gry 2D top-down w Unity napotkaliśmy problem, w którym postać poruszała się szybciej, gdy szła po skosie. Problem ten wynikał z faktu, że wektory ruchu dla osi X i Y były dodawane bez normalizacji, co powodowało, że przy jednoczesnym poruszaniu się w obu kierunkach (X i Y), prędkość postaci była większa niż w przypadku poruszania się tylko w jednej osi. Podczas testów zauważyliśmy, że postać poruszała się z wyższą prędkością, jeśli była w ruchu po skosie, co prowadziło do nienaturalnego przyspieszenia.

W odpowiedzi na ten problem podjęliśmy próbę różnych podejść, takich jak zmiana wartości prędkości w zależności od kierunku ruchu. Próby te, mimo że przyniosły pewne rezultaty, nie były w pełni satysfakcjonujące. Ostatecznie rozwiązaniem okazało się zastosowanie funkcji normalizującej wektor wejściowy. Normalizacja wektora ruchu sprawia, że bez względu na to, w którym kierunku postać się porusza (czy w osi X, Y, czy po skosie), jej prędkość jest zawsze jednolita. Zastosowanie tego rozwiązania poprawiło płynność ruchu postaci, eliminując problem niekontrolowanego przyspieszania w czasie poruszania się po przekątnej.  
  
 Obraz zawierający zrzut ekranu, tekst, diagram, kreskówka

Opis wygenerowany automatycznie

**2. Problemy z Interfejsem HUD w Różnych Proporcjach Ekranu**

W naszym projekcie gry 2D top-down w Unity napotkaliśmy problem z interfejsem HUD, który działał poprawnie tylko na proporcji ekranu 16:9. W innych proporcjach ekranu, takich jak 4:3 czy 21:9, elementy HUD były niewłaściwie rozmieszczone, co prowadziło do ich przesunięcia lub znikania z ekranu. Był to poważny problem, ponieważ gra miała być dostępna na różnych urządzeniach o różnych rozdzielczościach i proporcjach ekranów.

Aby naprawić ten problem, musieliśmy przeanalizować ustawienia Canvas w Unity i dokonać odpowiednich zmian w konfiguracji. Postanowiliśmy dostosować ustawienia Canvas i rozdzielczości, co pozwoliło na poprawne skalowanie interfejsu użytkownika. Zastosowaliśmy tryb **Canvas Scaler** w ustawieniach UI, ustawiając go na **Scale With Screen Size**. Dodatkowo, odpowiednio dobraliśmy punkt odniesienia oraz wartości szerokości i wysokości, co pozwoliło na prawidłowe skalowanie i rozmieszczanie elementów HUD w różnych proporcjach ekranu. Dzięki temu rozwiązaniu, elementy interfejsu zaczęły się prawidłowo skalować i dostosowywać do różnych proporcji ekranu, co rozwiązało problem niewłaściwego rozmieszczenia elementów HUD w różnych rozdzielczościach.  
  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

**3. Uproszczenie Systemu Ekwipunku i Broni**

W naszym projekcie gry 2D top-down w Unity napotkaliśmy problem z systemem przedmiotów i broni, który początkowo był zaprojektowany jako bardziej złożony ekwipunek. System ten wymagał śledzenia wielu przedmiotów, ich przypisywania do odpowiednich slotów oraz zarządzania nimi w czasie gry, co stało się zbyt skomplikowane i powodowało trudności w implementacji oraz optymalizacji. Początkowo planowaliśmy stworzenie skomplikowanego systemu ekwipunku, który umożliwiłby graczowi przechowywanie i zarządzanie dużą ilością przedmiotów i broni.

Po kilku próbach implementacji skomplikowanego systemu okazało się, że zarządzanie ekwipunkiem stało się trudne w utrzymaniu, a jego optymalizacja sprawiała, że cała mechanika rozgrywki stała się bardziej złożona. Po analizie postanowiliśmy uprościć tę mechanikę i zamieniliśmy złożony system na prostsze rozwiązanie. Zamiast zarządzać ekwipunkiem, postać po prostu posiadała aktywny przedmiot lub broń w danym momencie, a zmiana uzbrojenia odbywała się poprzez prostą zamianę aktywnego przedmiotu. To rozwiązanie okazało się znacznie łatwiejsze do zaimplementowania i bardziej stabilne, eliminując problem skomplikowanego systemu ekwipunku, a także umożliwiając bardziej elastyczną obsługę broni i przedmiotów.  
  
  
Obraz zawierający kreskówka, piksel

Opis wygenerowany automatycznieObraz zawierający kreskówka, piksel, ścieg

Opis wygenerowany automatycznie  
Obraz zawierający piksel, kreskówka

Opis wygenerowany automatycznieObraz zawierający kreskówka, piksel

Opis wygenerowany automatycznie

**4. Porzucenie Systemu Teleportów na Rzecz Tradycyjnego Poruszania Się**

W naszym projekcie gry 2D top-down w Unity początkowo zaplanowaliśmy wprowadzenie systemu teleportów, które miały umożliwiać graczowi szybkie przemieszczanie się między różnymi miejscami na mapie. Teleporty miały być aktywowane przez specjalne punkty, które gracz mógłby odwiedzać i wykorzystywać do teleportacji. Początkowo wydawało się to atrakcyjnym rozwiązaniem, ponieważ miałoby to znacznie skrócić czas przemieszczania się po mapie i zwiększyć dynamikę rozgrywki.

Jednakże, po kilku próbach implementacji okazało się, że system teleportów wprowadzał zbyt wiele komplikacji, zwłaszcza w kwestii synchronizacji i logiki związanej z interakcją gracza z tymi punktami. Problemy pojawiły się również przy próbach implementacji płynnych przejść między lokalizacjami, co prowadziło do niepożądanych efektów wizualnych oraz błędów w rozgrywce, takich jak zacinanie się czy desynchronizacja postaci. W związku z tym, po dokładnym przemyśleniu, zdecydowaliśmy się porzucić pomysł teleportów na rzecz bardziej klasycznego sposobu poruszania się po mapie. Dzięki temu uniknęliśmy problemów z synchronizacją i uprościliśmy mechanikę gry, co pozwoliło na lepszą płynność rozgrywki, jednocześnie ułatwiając implementację i zarządzanie ruchem postaci.

**5. Rozwiązywanie Problemów z Kolizjami i Interakcjami Postaci**

W naszym projekcie gry 2D top-down w Unity napotkaliśmy problem z interaktywnością postaci i jej interakcją ze światem gry, szczególnie w kontekście kolizji i poruszania się po różnych warstwach. Początkowo postać mogła wchodzić w interakcje z obiektami na różnych warstwach, co prowadziło do błędów w poruszaniu się. Na przykład, postać przechodziła przez obiekty, które powinny blokować jej drogę, co sprawiało, że gra stała się nieintuicyjna i nieprzewidywalna.  
  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznieObraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Aby rozwiązać ten problem, postanowiliśmy podzielić wszystkie warstwy na cztery główne elementy: świat, tło, postać oraz warstwę interaktywną. Każdej z tych warstw przypisaliśmy odpowiednie właściwości dotyczące interakcji i kolizji. Dzięki temu, obiekty, które miały blokować drogę postaci, były odpowiednio oznaczone i reagowały na ruch postaci, uniemożliwiając jej przechodzenie przez nie. Równocześnie warstwy interaktywne, takie jak przedmioty, były oznaczone w taki sposób, by postać mogła wchodzić w interakcje z nimi. Dzięki temu systemowi udało się poprawić kolizje i interakcje postaci ze światem, zapewniając, że postać nie przechodzi przez obiekty, które nie powinny być dostępne, oraz że interaktywne elementy świata są odpowiednio oznaczone i reagują na działania gracza.|  
  
Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

**6. Problemy z Synchronizacją Projektu przy Używaniu Git**

W trakcie pracy nad projektem gry 2D top-down w Unity napotkaliśmy także problemy związane z synchronizacją silnika gry przy używaniu systemu kontroli wersji Git. Każda zmiana w projekcie, zwłaszcza w przypadku dużych plików binarnych, takich jak modele, tekstury czy pliki scen, powodowała trudności z synchronizacją i wprowadzaniem poprawek w zespole. Git nie radził sobie dobrze z rozbieżnościami w tych plikach, co prowadziło do konfliktów i problemów z prawidłowym załadowaniem projektu na różnych maszynach.

Aby rozwiązać te trudności, postanowiliśmy dokładnie przejrzeć ustawienia projektu w Unity i skonfigurować odpowiednie opcje dotyczące zapisywania plików w Gicie. Skupiliśmy się na odpowiedniej konfiguracji plików **.gitignore**, aby uniknąć dodawania niepotrzebnych plików binarnych do repozytorium, takich jak pliki cache, ustawienia specyficzne dla maszyn czy pliki generowane przez Unity, które nie powinny być śledzone. Dodatkowo, każdą zmianę w plikach projektu staraliśmy się testować indywidualnie, sprawdzając, czy wprowadzone zmiany nie powodują problemów przy synchronizacji i ładowaniu projektu na innych maszynach. Dzięki tej metodzie udało nam się uniknąć większości problemów z synchronizacją i zapewnić płynniejszą współpracę nad projektem, co znacznie ułatwiło pracę zespołową i ograniczyło liczbę błędów związanych z wersjonowaniem.

**7. Implementacja Funkcji Ataku**

W trakcie pracy nad projektem gry 2D top-down w Unity napotkaliśmy trudności związane z implementacją funkcji ataku. Początkowo, mieliśmy zaprojektowany szablon dla tej funkcji, jednak okazało się, że integracja różnych wartości, takich jak obrażenia, animacje ataku czy interakcje z wrogami, wymagała więcej pracy niż początkowo zakładaliśmy. Zamiast rozwijać tę funkcję na początku, postanowiliśmy pozostawić szablon prawie gotowy, który umożliwia implementację ataku, jednak bez pełnej logiki wrogów i wartości obrażeń. Dzięki temu stworzyliśmy fundament do dalszej pracy, tak by łatwo było dodać różne wartości dla każdego wroga, takie jak HP, obrażenia czy efekty animacji.  
  
  
Obraz zawierający zrzut ekranu, piksel, linia

Opis wygenerowany automatycznieObraz zawierający zrzut ekranu, tekst, Czcionka, design

Opis wygenerowany automatycznie

Zdecydowaliśmy się na implementację podstawowych elementów, takich jak pasek HP oraz animacje ataku, które zostały już zaimplementowane, co pozwoliło nam na łatwe testowanie animacji i stanu zdrowia w trakcie rozgrywki. Ostatecznie, szablon umożliwia płynne dodawanie nowych wrogów i dostosowywanie wartości ataku, co pozwoli na elastyczną i efektywną implementację funkcji ataku w grze. Takie podejście pozwala na szybkie sterowanie ,eliminowanie błędów i modyfikowanie wartości w trakcie rozwoju gry.  
Obraz zawierający zrzut ekranu, Oprogramowanie graficzne, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

**8. Rozwój Świata Gry i Rozszerzanie Zawartości**

W trakcie rozwoju naszego projektu gry 2D top-down w Unity napotkaliśmy wyzwanie związane z ogromnym potencjałem rozwoju świata gry. Początkowo mieliśmy zamiar skupić się na detalach, ale szybko zauważyliśmy, że zbyt duża koncentracja na szczegółach może opóźnić cały proces i sprawić, że rozwój gry stanie się zbyt czasochłonny. Zamiast tworzyć skomplikowane, dopracowane detale, które mogłyby zajmować zbyt wiele czasu, zdecydowaliśmy się na stworzenie szerokiego wachlarza obiektów i elementów świata, które można łatwo dodawać i modyfikować w przyszłości.

Zaprojektowaliśmy elastyczny system, który pozwala na szybkie wstawianie nowych elementów do gry, takich jak obiekty, budynki, tereny czy interaktywne przedmioty, bez konieczności głębokiej ingerencji w kod. Dzięki temu stworzyliśmy solidną podstawę, która umożliwia dalszy rozwój i rozszerzanie świata gry bez nadmiernego komplikowania procesu. Taki system pozwala na szybkie i efektywne dodawanie nowych obiektów, co znacznie ułatwia dalszy rozwój gry, jednocześnie dając dużą swobodę w rozszerzaniu świata i jego elementów.  
  
Obraz zawierający Plan, diagram, mapa

Opis wygenerowany automatycznie

**9. Problemy z Podążaniem Przeciwników za Bohaterem**

Na początku pracy nad projektem gry 2D top-down w Unity napotkaliśmy problem z podążaniem przeciwników za bohaterem, co było trudne do zaimplementowania w sposób naturalny i płynny. Przeciwnicy początkowo mieli problemy z prawidłowym śledzeniem postaci, zwłaszcza gdy bohater poruszał się w bardziej złożony sposób. W początkowej wersji, wrogowie często gubili ślad bohatera lub poruszali się w sposób nienaturalny, co psuło dynamikę rozgrywki.

Aby rozwiązać ten problem, postanowiliśmy wykorzystać gotową funkcję opartą na sztucznej inteligencji, która pozwalała na prostą implementację mechaniki podążania za bohaterem. Jednak funkcja ta działała tylko wtedy, gdy stan wroga został ustawiony na "podążanie za bohaterem". Aby usprawnić ten proces, połączyliśmy ją z systemem wykrywania bohatera za pomocą obszaru detekcji o okrągłym kształcie, który wykrywał postać w jego zasięgu. Dzięki temu, kiedy wrogowie znajdą się w odpowiednim promieniu od bohatera, aktywujemy stan podążania za nim, co pozwala im śledzić bohatera bez problemu. To rozwiązanie umożliwiło nam płynne i realistyczne śledzenie postaci przez kilku przeciwników jednocześnie, zapewniając im odpowiednią reakcję, gdy skupimy na sobie większą liczbę wrogów.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie