Dokumentacja Projektu

1. Tytuł Projektu

Nazwa Gry: Grave Out

Ważna informacja: Aby zagrać w grę, konieczne jest posiadanie zalogowanego konta Steam, ponieważ Steam API jest zintegrowane w celu obsługi lobby. Gra wymaga więc stałego połączenia ze Steam.

Większość dalszych punktów tej dokumentacji zawiera linki do Gyazo, aby łatwo udostępniać przykłady i wizualne demonstracje dotyczące tego projektu. Bardzo proszę o uwzględnienie ich podczas czytania tej dokumentacji.

2. Opis Projektu

Krótki Opis: Grave Out to wieloosobowa gra akcji, w której bohaterowie budzą się w mrocznym świecie Limbo, aby rozwikłać zagadkę swojej śmierci oraz wielu innych tajemnic. Gra pozwala na grę w lobby do 4 osób, wykorzystując Steam API oraz Netcode for GameObjects. Gra oferuje zaawansowaną mechanikę walki, proceduralnie generowane środowisko oraz złożony system dialogów i zadań.

3. Twoja Rola w Projekcie

Rola: Junior Unity Game Developer (Programista / Projektant Poziomów)

Dodatkowo, moja rola obejmuje level design, programowanie oraz wszystkie aspekty z wyjątkiem tworzenia światopowiesci, muzyki i modelowania.

Kluczowe dokonania:

Implementacja lobby Steam oraz obsługa do 4 graczy.

Programowanie mechanik gry i synchronizacji w czasie rzeczywistym przy użyciu Steam API i Netcode for GameObjects.

Projektowanie poziomów gry i integracja zasobów.

Testowanie i optymalizacja sieciowych funkcji gry.

4. Kluczowe Funkcje i Mechaniki

4.1. Steam Lobby i Gra Wieloosobowa:

Implementacja lobby Steam, umożliwiającego tworzenie i dołączanie do gier wieloosobowych z wykorzystaniem Steam API.

Obsługa komunikacji sieciowej między graczami, w tym synchronizacja stanu gry, informacji o postaciach i innych istotnych danych.

Implementacja mechanizmów matchmakingu i wyszukiwania gier.

4.2. Proceduralna Generacja Świata

Projektowanie i implementacja proceduralnej generacji terenu, roślinności i obiektów w środowisku gry, zapewniając zróżnicowanie i unikalność poziomów.

Wykorzystanie zaawansowanych technik generacji proceduralnej, takich jak diagram Voronoi, Perlin Noise, Simplex Noise i inne algorytmy szumu, w celu tworzenia realistycznych i interesujących krajobrazów.

Stworzenie customowego, deterministycznego systemu losowości opartego na Linear Congruential Generator, umożliwiającego generowanie powtarzalnych światów na podstawie seeda.

4.3. System Walki

AI z nowatorskim systemem generowania waypointów i patrolowania w proceduralnie generowanym środowisku, omijającym przeszkody w czasie rzeczywistym.

System walki z różnymi animacjami i atakami dla każdej broni. Synchronizacja zmiennych animatora i interakcje za pomocą netcode.

https://i.gyazo.com/b0cdb3ee35a64c6c99d42dc7eedebd2a.mp4

Customowy lokalny system unikania (local avoidance) oraz zaawansowana wiedza z zakresu Unity NavMesh, w tym navigation path linking.

4.4. Zarządzanie Sejwami

Implementacja systemu zapisywania i wczytywania stanu gry, pozwalającego graczom kontynuować rozgrywkę od miejsca, w którym przerwali.

Wykorzystanie wzorców projektowych, takich jak async/await i ConcurrentQueue, do efektywnego zarządzania asynchronicznymi operacjami zapisu i odczytu danych.

4.5. System Dialogów, Zadań i Interakcji z NPC:

Zaawansowana integracja systemu dialogów i questów, umożliwiająca tworzenie rozbudowanych i

nieliniowych historii.

Synchronizacja stanu NPC (Non-Playable Characters) i ich interakcji z graczami w czasie rzeczywistym, zapewniając spójną rozgrywkę w środowisku wieloosobowym.

4.6. Multiplayer Kinematic Character Controller:

Implementacja customowego systemu kontroli ruchu postaci (Kinematic Character Controller) w środowisku gry wieloosobowej.

Synchronizacja ruchu i animacji postaci między klientem a serwerem za pomocą Netcode for GameObjects, zapewniając płynne i responsywne sterowanie postaciami w grze wieloosobowej.

Wykorzystanie interpolacji dla płynnego ruchu postaci na kliencie, minimalizując efekt "skakania" spowodowany opóźnieniami sieciowymi.

https://i.gyazo.com/8f2efe187ea8be305884af7320038c8f.mp4

Synchronizacja systemu interakcji między graczami za pomocą Netcode for GameObjects, umożliwiając wspólną interakcję z obiektami w świecie gry.

https://i.gyazo.com/509b7ca18e760dbb13180ce69e72a7b4.mp4

4.7. Ustawienia i Konfiguracja

Możliwość dowolnej konfiguracji ustawień graficznych gry oraz bindowania klawiszy, zapisywanie ustawień w Player Prefs.

Znajomość nowego i starego systemu input w Unity, niezależnie od kontrolerów.

4.8. Dźwięk

Podstawowa wiedza z zakresu FMOD i dźwięku.

Integracja FMOD z projektem Unity: Implementacja efektów dźwiękowych, muzyki i dźwięków otoczenia przy użyciu FMOD.

Tworzenie i implementacja eventów FMOD: Dynamiczne dostosowywanie dźwięku do sytuacji w grze (np. zmiana muzyki w zależności od poziomu zdrowia postaci).

4.9. UI i UGUI

Pełna znajomość Unity UI i UGUI, w tym projektowanie i implementacja interfejsu użytkownika.

Zastosowanie elementów UI takich jak Canvas, Panel, Button, Text, Slider, Toggle, Dropdown oraz system eventów.

5. Techniczne Szczegóły Implementacji

5.1. Customowe Instancjonowanie Obiektów i Zarządzanie Zasobami:

Implementacja customowego instancjonowania obiektów dla optymalizacji wydajności.

Wykorzystanie puli obiektów (lokalnych i sieciowych) w połączeniu z Addressables dla efektywnego zarządzania zasobami i redukcji kosztów tworzenia/niszczenia obiektów. Zastosowanie **UniTask** (biblioteki asynchronicznej) do usprawnienia procesu ładowania i zarządzania zasobami **Addressables**.

5.2 Doświadczenie w tworzeniu gier wieloosobowych z wykorzystaniem Netcode for GameObjects (NGO).

Implementacja komunikacji klient-serwer za pomocą RPC (Remote Procedure Calls) i NetworkVariables.

Zarządzanie spawnowaniem i despawnowaniem obiektów sieciowych w NGO.

Synchronizacja transformacji, animacji i innych właściwości obiektów sieciowych.

Tworzenie niestandardowych komponentów sieciowych dla specyficznych potrzeb projektu.

Optymalizacja kodu sieciowego pod kątem wydajności i minimalizacji opóźnień.

5.3. Shadery i Universal Render Pipeline (URP):

Znajomość architektury shaderów i praktyczne wykorzystanie Shader Graph do wizualnego tworzenia oraz modyfikacji shaderów w URP.

Efektywne wykorzystanie Post Processing Stack v2 do implementacji i dostosowywania efektów post-processingu, takich jak bloom, ambient occlusion, color grading, depth of field i motion blur.

Doświadczenie w pracy z Render Features, umożliwiające rozszerzenie pipeline'u URP o niestandardowe efekty renderowania.

Praktyczne wykorzystanie Render Graph do analizy i optymalizacji procesu renderingu, identyfikacji wąskich gardeł oraz poprawy wydajności.

5.4. Unity 6 Preview i Nowe Technologie Renderingu:

Aktywnie korzystam z GPU Instancing i SRP Batcher dla optymalizacji renderingu.

Wykorzystanie Spatial-Temporal Post-Processing (STP) dla poprawy jakości wizualnej i wydajności renderingu.

Znajomość i implementacja ulepszeń Volume Framework dla lepszej kontroli nad efektami post-processingu, w szczególności w kontekście prototypowania wizualiów i efektów pogodowych w świecie gry Grave Out.

5.5. Entity Component System (ECS) i Data-Oriented Technology Stack (DOTS):

Znajomość architektury Entity Component System (ECS) i Data-Oriented Technology Stack (DOTS), umożliwiających wysokowydajne przetwarzanie danych i symulacje w grach.

Praktyczne wykorzystanie Burst Compiler i Jobs System do optymalizacji operacji na komponentach, systemach oraz generowaniu terenu w ECS, zwiększając wydajność i skalowalność gry.

Znajomość i implementacja rozwiązań ECS Physics dla Netcode, umożliwiających symulację fizyki po stronie serwera i synchronizację z klientami.

Zastosowanie ECS do zarządzania dużą liczbą obiektów w grze, takich jak drzewa, rośliny i inne elementy otoczenia, co przekłada się na lepszą wydajność i skalowalność.

Wykorzystanie DOTS do optymalizacji przetwarzania danych i poprawy wydajności gry, szczególnie w obszarach takich jak generowanie terenu i symulacje fizyczne.

5.6. Jobs System i Burst Compiler:

Głębokie zrozumienie Unity Job System:

Tworzenie i planowanie jobów (zadań) dla zrównoleglenia obliczeń.

Wykorzystanie IJobParallelFor do przetwarzania danych w sposób wielowątkowy.

Zapewnienie bezpieczeństwa wątków i unikanie race conditions.

Synchronizacja jobów za pomocą zależności (JobHandle).

Praktyczne wykorzystanie Burst Compiler:

Kompilacja kodu C# do wysoce zoptymalizowanego kodu maszynowego.

Wykorzystanie atrybutów Burst ([BurstCompile]) do kontroli kompilacji.

Znajomość ograniczeń Burst i umiejętność ich obejścia.

Znajomość Native Collections:

Wykorzystanie NativeArray, NativeList, itp. do efektywnego zarządzania danymi w środowisku wielowątkowym.

5.7. Cinemachine:

Wykorzystanie Cinemachine do tworzenia filmowych cutscenek i sekwencji przerywnikowych.

Dynamiczne śledzenie postaci, płynne przejścia między ujęciami i tworzenie efektów kamery (np. zoom, dolly).

Implementacja wirtualnych kamer dla uzyskania profesjonalnego wyglądu scen.

5.8. Deformacja Mesh i Synchronizacja:

Deformacja mesh w czasie rzeczywistym, synchronizowana przez netcode między klientem a serwerem, wykorzystując Jobs.