**Crystal Of Exchange**

****

**Projekt Kompetencyjny**

**Grupa:**

* **Dominik Rak**
* **Jakub Zawiasa**
* **Maciej Kopka**
* **Paweł Wolski**

**Założenia projektowe**

Celem projektu było stworzenie gry platformowej 2D z wykorzystaniem silnika gry UNITY oraz języka C#.

Zadanie to miało nam umożliwić naukę platformy UNITY, języka C# jako iż była to nasza pierwsza styczność z nim oraz obsługi programów graficznych potrzebnych nam do tworzenia tekstur oraz animacji do gry.

Gra nosi tytuł Crystal Of Exchange czyli Kryształ Zamiany. Wykorzystuje ona pomysł, w którym bohater potrafi przemieniać się w różne postacie z różnymi zestawami umiejętności.

W naszej grze bohater potrafi zamieniać się w maga, wojownika oraz łucznika.

Gra zawiera 3 poziomy kolejno: las, piekło oraz krainę lodu. Dodatkowo po przejściu każdego poziomu będziemy musieli się zmierzyć z mocniejszym przeciwnikiem, bossem.

**Wykorzystane narzędzia i technologie**

**Silnik gry UNITY**  
Pozwala na stworzenie gier 2D oraz 3D. Dostarcza narzędzia do implementowania fizyki, animacji, skryptów, interfejsu, interakcji między obiektami, zdarzeniami oraz wielu innych komponentów.

**Visual Studio Code / Microsoft Visual Studio**Edytory kodu umożliwiające nam pracę w języku C# wraz z synchronizacją UNITY API.

**Doxygen**Generowanie dokumentacji do skryptów wykorzystując adnotacje w kodzie w formacie JAVA DOC.

**Adobe Illustrator, Adobe After Effect, Adobe Photoshop**Oprogramowanie graficzne umożliwiające nam stworzenie grafik oraz animacji.

**UNITY Cloud Collaborate**Implementacja GIT wbudowana w UNITY pozwalająca na bezpośrednie zarządzanie projektem z poziomu aplikacji UNITY oraz na udostępnienie projektu pomiędzy zespołem.

**Proces tworzenia gry w aplikacji UNITY**

Grę składa się ze scen. Scena to plik z rozszerzeniem .unity. Za pomocą aplikacji UNITY do sceny dodajemy obiekty, rozmieszczamy je na scenie dodajemy do nich komponenty, zarządzamy ich animacjami, skryptami. Podczas tworzenia sceny możemy ją uruchomić bezpośrednio w aplikacji w celu sprawdzenia jak nasze obiekty zachowują się podczas rozgrywki.

Gotowe sceny dodajemy do Build’u, w którym ustalamy kolejność scen. Dodatkowo możemy w ustawieniach Build’u zarządzać ustawieniami graficznymi gry takimi jak rozdzielczość czy jakość grafik po jej zbudowaniu oraz wybrać platformę docelową, z dostępnych:

* PC/MaC/Linux
* iOS
* Android
* tvOS
* Xbox One
* PS4
* PS Vita
* WebGl
* Facebook

Możemy zarządzać również ustawieniami projektu takimi jak wprowadzanie danych, fizyka, które warstwy ze sobą kolidują czy na przykład dźwięku.

Gdy wszystkie ustawienia się skonfigurowane, sceny stworzone oraz dodane do Build’u możemy go zbudować, aplikacja zapyta się nas gdzie zapisać naszą grę, a następnie zacznie proces tworzenia. Po ukończeniu w wybranym przez nas miejscu znajdziemy plik .exe, którym uruchamiamy naszą grę oraz pozostałe pliki gry.

**Wykorzystane komponenty silnika gry UNITY**

**GameObject –** Obiekt posiadający atrybuty takie jak nazwa, tag oraz warstwa, można do niego dołączać komponenty oraz jako dzieci kolejne GameObject’y.

**Transform** – przetrzymuje informację o pozycji, rotacji i skali obiektu oraz pozwala na zmienianie każdej z tych właściwości.

**RigidBody 2D –** komponent odpowiedzialny za fizykę obiektu, pozwala na zarządzanie masą, działaniem grawitacji, sposób detekcji kolizji (dyskretnie lub ciągle), blokowanie rotacji na każdej z osi oraz na wiele pomniejszych opcji.

**Collider 2D –** komponent, który wyznacza sferę podlegającą kolizji z innymi obiektami zawierającymi Collider. Kształt Collidera oraz jego rozmiar możemy ustalić z poziomu aplikacji UNITY.

**Sprite Renderer –** komponent wyświetlający grafikę przypisaną do obiektu oraz pozwalający na zmienianie sposobu wyświetlania

**HingeJoint2D –** komponent umożliwiający łączenie zawiasowe obiektów. Może służyć jako zawiasy drzwi, jako ogniwa łańcucha lub tak jak w naszym przypadku jako punkty w linie oraz zawiasy w zapadniach. Posiada również opcję motor umożliwiającą napędzanie obiektów (np. drzwi obrotowe), u nas wykorzystane do powrotu zapadni do stanu pierwotnego po tym jak się opuści.

**LineRenderer –** komponent pozwalający na renderowanie linii o zadanej szerokości z wykorzystaniem swoich tekstur pomiędzy listą obiektów.

**Animator –** komponent zarządzający animacjami, w którym definiujemy stany oraz momenty przejścia pomiędzy nimi. Można w nim utworzyć parametry, które będą warunkami przejść oraz którymi możemy zarządzać z poziomu skryptów.

**Animation –** animacja, którą możemy utworzyć za pomocą serii grafik, nagrać ją z poziomu tworzenia sceny bądź zdefiniować klatka po klatce wykorzystując każdy z dostępnych komponentów podpiętych pod animowany obiekt. Na przykład animacja obiektu posiadająca komponent Transform, może zmieniać parametr Scale powodując zmniejszanie się lub zwiększanie się obiektów.

Animacja pozwala również na wywoływanie zdarzeń w konkretnych klatkach, które mogą uruchamiać funkcję ze skryptów.

**Audio Source –** źródło dźwięku, do którego podpinamy plik dźwięku. Umożliwia uruchomienie dźwięku w momencie gdy obiekt zostaje uruchomiony, w dowolnym momencie działania gry z poziomu gry, na zapętlenie dźwięku, głośności, efektów dźwiękowych, zasięgu dźwięku i tym podobnych.

**Tilemap –** komponent tworzony z grafik, za pomocą którego można „rysować” otoczenie wykorzystując narzędzie Tile Pallete.

**Canvas –** obiekt pozwalający na tworzenie interfejsu użytkownika z wykorzystaniem komponentów takich jak : **Button, Text , Image, Slider, Toggle, Panel, Rect Transform.** Może być interaktywna lub nie.

**Canvas Group –** komponent podpinany pod Canvas pozwalający na ustalanie przezroczystości Canvas.

**Camera –** Obiekt, który jest naszym oknem na grę, bez niego nie możemy widzieć rozgrywki. Możemy go umieścić statycznie w jednym miejscu bądź za pomocą skryptu podążać za innym obiektem.

**Audio Listener –** komponent podpinany pod kamerę aby ta mogła przechwytywać i odtwarzać dźwięki z obiektów z Audio Source.

**Prefab –** jest to zapisany GameObject utworzony przez twórcę tak aby można go było wykorzystać ponownie w innym miejscu sceny bądź na innych scenach. W naszym przypadku często wykorzystywany w celu tworzenia pocisków, które pojawiają się na scenie w trakcie działania gry i są odtwarzane z wcześniej zdefiniowanych obiektów prefab.

**Sposób działania skryptów**

UNITY podczas działania gry uruchamia każdy skrypt podpięty pod aktywne obiekty na scenie.

Każdy skrypt, który działa w UNITY musi dziedziczyć z MonoBehaviour. Jest to obiekt posiadający w sobie funkcje wywoływane automatycznie w odpowiednich momentach.  
Wykorzystywane przez nas funkcję, które nadpisywaliśmy w naszych skryptach to:

* **Start()** – uruchamia się w momencie, gdy obiekt pojawia się na scenie
* **Awake() –** uruchamia się, gdy obiekt staje się aktywny
* **Update() –** wywoływana jest w każdej klatce podczas działania gry
* **FixedUpdate() –** podobnie do Update(), jednak nie wykonuje się z czasem zależnym od wygenerowania klatki, a ze stałym odstępem czasu 0.2s (możliwa jest zmiana odstępu). Używana głównie do operacji związanych z fizyką gry.
* **LateUpdate() -**  identycznie jak Update() ale dopiero, gdy Update() zakończy działanie.

Skrypty posiadające zmienne z poziomem protekcji public umożliwiają na zdefiniowanie wartości tych zmiennych z poziomu aplikacji UNITY w zakładce Inspector. Pozwala to na utworzenie referencji do obiektów na scenie oraz na przypisanie wartości dla standardowych typów zmiennych.

**Skrypty utworzone do zarządzania obiektami w grze**

Szczegółowe informacje o zmiennych oraz metodach w skryptach znajdują się w dokumentacji wygenerowanej za pomocą Doxygen.

**Skrypty przeciwników**

**Audio –** Skrypt, wykorzystywany przez animację do uruchomienia dźwięku podpiętego do skryptu w momencie uruchomienia event’u z animacji.

**BigFootController –** Skrypt zarządzający bossem dla Krainy Lodu, Wielką Stopą. Boss będzie atakował pięściami, wykonywał atak specjalny w skoku, rzucał kulą śnieżną, oraz chronił się przed strzałami po utracie połowy zdrowia.

**CollisionDetect –** Skrypt, który sprawdza czy kolizja, która nastąpiła z obiektem była związana z graczem, jeśli tak odpycha gracza o ile ten nie używa tarczy. Jeśli używa przeciwnik się od niej odbija.

**EnemyInteraction –** Skrypt odpowiadający za interakcję przeciwnika z graczem

**EnemyRangedAttack –** Skrypt umożliwiający przeciwnikowi wystrzeliwanie pocisków. Pocisk jako obiekt Prefab ze skryptem EnemyRangedWeapon podpinamy pod skrypt.

**EnemyRangedWeapon -**Skrypt odpowiedzialny za zachowywanie się wystrzeliwanych pocisków. Nadaje im prędkość oraz definiuje jakie zadają obrażenia graczowi oraz odpowiada za niszczenie pocisku po zderzeniu z innym obiektem.

**EntMove –** Skrypt zarządzający bossem dla Lasu, Entem. Boss będzie atakował pięściami ,wykonywał atak specjalny w skoku, strzelał serią pocisków, drewnianymi palami , oraz chronił się przed strzałami po utracie połowy zdrowia.

**FollowPatrolAI –** Skrypt odpowiedzialny za zachowanie się przeciwników patrolujących i śledzących. Dziedziczy z PatrolAI. Przeciwnik będzie patrolował dostępny dla siebie teren. Jeśli w zasięgu wzroku pojawi się gracz zacznie się do niego szybciej przemieszczać w celu zaatakowania go. Dodatkowo podczas stanu patrolu, w zdefiniowanym interwale czasu poprzez zmienną lookBehindInterval będzie zatrzymywał się, odwracał i sprawdzał czy gracz nie jest za jego plecami, a po upływie czasu opisanego za pomocą lookBehindTime odwróci się ponownie i wróci do patrolowania opisanego w PatrolAI.

**GemsAfterKill -**  Skrypt, który po śmierci przeciwnika wyrzuca obiekty Gems dające dodatkowe punkty.

**InfernoBossMove –** Skrypt odpowiedzialny za zarządzanie bossem Piekła, Inferno. Dziedziczy z EntMove. Boss potrafi atakować, tworzyć słupy płomieni rozchodzące się po jego bokach, unieść się w powietrze, a następnie wystrzelić kilkadziesiąt kul ognia wokół siebie po czym musi odpocząć i jest bezbronny oraz po utracie połowy zdrowia „spalić cięciwę łuku” gracza czyli zablokować możliwość strzelania.

**JumpingEnemy**

**PatrolAI**

**SimpleEnemyInfo**

**Skrypty przedmiotów i otoczenia**

**AidKit**

**Arrow**

**Coin**

**hidingTrap**

**MovingPlatform**

**MovingSaw**

**Saw**

**SpikeController**

**Skrypty Gracza**

**Archer**

**BowShooting**

**CameraFollow**

**CharacterController2D**

**PlayerInfo**

**PlayerMovement**

**RopeScript**

**ThrowHook**

**ThrowQubic**

**Warrior**

**WarriorAttack**

**Wizard**

**Skypty Interfejsu oraz zarządzania grą**

**backgroundMove**

**BackgroundScroller**

**EndLevel**

**Fade**

**GameOverScreen**

**LevelsManager**

**MainMenu**

**SceneLoader**

**Scrolling**

**UIController**