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 oraz GIMP**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.

**Discord**Komunikator internetowy pozwalający nam na szybką wymianę wiadomości oraz plików

**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.

**Trudności podczas wykonywania projektu oraz jak je rozwiązaliśmy**

**Niski poziom dokumentacji –** Dokumentacja API UNITY jak i komponentów możliwych do wykorzystania jest mało dokładna co często zmuszało nas do szukania nieoficjalnych źródeł informacji o konkretnych podzespołach i opcjach w aplikacji. Powodowało to częstą utratę czasu. Z czasem jednak udało nam się znaleźć kilka źródeł, które często nam pomagały. Bardzo pomocny okazał się kanał w serwisie Youtube poświęcony UNITY o nazwie Brackeys.

**Wysoki próg wejścia –** Mnogość opcji oraz komponentów jak i wcześniej wymieniony niski poziom dokumentacji sprawił, że w początkowej fazie projektu stworzenie jakiegokolwiek działającego obiektu było dla nas wielkim problem wymagającym bardzo dużej ilości czasu. Później jednak w raz z naszą nauką, sprawność tworzenia znacząco wzrosła.

**Środowisko mało przyjazne użytkownikowi –** Sposób rozmieszczenia elementów interfejsu był dla nas mało wygodny, a ilość paneli , które musiały być otwarte do sprawnej pracy sprawiał, że główny obszar aplikacji Scena , w którym umieszczamy i projektujemy obiekty stawał się mały i mało wygodny do pracy. Niestety tego problemu nie byliśmy w stanie rozwiązać i musieliśmy do niego przywyknąć. Odbiło się to jednak na naszym komforcie i szybkości pracy.

**Wysokie wymagania sprzętowe aplikacji UNITY –** Unity pozwala na testowanie sceny bezpośrednio w aplikacji bez tworzenia gry za każdym razem, gdy coś zmienimy. Jednakże jest to proces bardzo wymagający sprzętowo i podczas testów często doświadczaliśmy zamrożeni aplikacji.

**Praca na wspólnym projekcie UNITY –** Na początku zamiast usługi **UNITY Cloud Collaborate** korzystaliśmy z serwisu GitHub. Jednak szybko pojawiły się Merge Conflicty ponieważ nie znając dokładnej struktury plików tworzonych przez UNITY dla projektu wybraliśmy cały folder projektu jako repozytorium. Jak się okazało UNITY w folderze projektu przechowuje pliki zależne od maszyny, i które są tylko potrzebne dla aplikacji UNITY, a nie dla projektu jako takiego. Ponieważ dla każdego pliki te były różne powstawały Merge Conflicty. Z tego powodu zaczęliśmy korzystać z **UNITY Cloud Collaborate** co całkowicie rozwiązało problem. Jednakże trzeba zaznaczyć, że korzystanie z serwisów takich jak GitHub jest możliwe potrzeba jednak dodać odpowiednie pliki i foldery do pliku gitignore.

**Współbieżność pracy -** Gdy każdy z nas pracował w tym samym czasie na tej samej scenie często zmienialiśmy sobie nawzajem elementy obiektów co powodowało zamieszanie. Rozwiązaliśmy to tworząc dla każdej osoby oddzielną scenę, na której tworzył obiekty i zapisywał jako prefab, a następnie przenosiliśmy je do scen głównych.

**Współpraca –** Z powodu braku doświadczenia w pracy zespołowej oraz spraw i obowiązków osobistych nasza współpraca była utrudniona. Często występowały różnice zdań, które jednak ostatecznie zawsze udawało nam się rozwiązać. Dodatkowe obowiązki sprawiały, że problematyczne było znalezienie wspólnego czasu gdy jakiś element wymagał pracy kilku osób jednocześnie.

**Współbieżność działania skryptów –** Ponieważ każdy ze skryptów posiadający metodę Update() (została ona wytłumaczona w dalszej części wraz ze sposobem działania skryptów w UNITY) uruchamiał ją jednocześnie w każdej klatce problematyczna była synchronizacja działań pomiędzy nimi oraz uniemożliwiało to użycie np. pętli w wielu sytuacjach co wymagało przemyślenia sposobu działania naszych algorytmów.

**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 gdzie są widoczne ( można je ukryć za pomocą adnotacji [HideInInspector] jeśli chcemy aby zmienna była widoczna dla każdego skryptu ale nie z poziomu aplikacji). 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 -** Skrypt odpowiedzialny za zachowanie się przeciwników patrolujących. Dziedziczy z **PatrolAI.** Przeciwnik będzie patrolował dostępny dla siebie teren oraz przeskakiwał nad strzałami.

**PatrolAI** ­- Skrypt odpowiedzialny za zachowanie się przeciwników patrolujących. Przeciwnik będzie patrolował dostępny dla siebie teren. Po natrafieniu na przeszkodę lub przepaść zawróci.

**SimpleEnemyInfo –** Skrypt przechowujący zdrowie przeciwnika, punkty za jego zabicie oraz pozwalający na zranienie go oraz zarządzający jego śmiercią.

**Skrypty przedmiotów i otoczenia**

**AidKit** – Skrypt przypinany do przedmiotów, który w momencie wystąpienia eventu spowodowanego z kolizją z graczem, uzdrawia go. Pozwala to na stworzenie apteczek lub mikstur leczących.

**Arrow –** Skrypt przypinany do pocisków wystrzeliwanych przez gracza, pozwala na zdefiniowanie, prędkości, obrażeń oraz czasu życia pocisku

**Coin –** Skrypt przypinany do obiektów możliwych do zebrania takich jak monety oraz kryształy. Odpowiada za przypisanie zebranego przedmiotu do gracza oraz za usunięcie go ze sceny po zebraniu.

**hidingTrap –** Skrypt przypinany do obiektów, które pokazują się, a następnie chowają i powtarzają to ciągle. W naszym przypadku użyte głównie do chowających się pułapek, stąd nazwa skryptu.

**MovingPlatform –** Skrypt zarządzający poruszającą się platformą. Wyznaczamy dla niej listę punktów pomiędzy, którymi będzie się poruszała, punkt startowy oraz prędkość. Następnie platforma będzie przemieszczała się ze stałym tempem pomiędzy kolejnymi punktami.

**MovingSaw –** Skrypt dziedziczący z **Saw** i wprowadzający możliwość przemieszczania się piły pomiędzy dwoma punktami.

**Saw –** Skrypt dziedziczący z **hidingTrap**, dodatkowo wprowadzający obrót pułapki, w naszym przypadku piły.

**SpikeController –** Skrypt przypinany do obiektów, które po kolizji z graczem ranią go, umożliwia zdefiniowanie obrażeń oraz siły odepchnięcie po kolizji.

**Skrypty Gracza**

**Archer –** Skrypt zarządzający dostępnością umiejętności łucznika **(BowShooting i ThrowHook),** które stają się aktywne gdy gracz jest w stanie archer.

**BowShooting** – Skrypt umożliwiający wystrzeliwanie pocisków przez łucznika. Umożliwia wybór pocisku jako prefab oraz zdefiniowanie czasu pomiędzy kolejnymi strzałami. Zarządza odtwarzaniem animacji oraz dźwięków wystrzału.

**CameraFollow** – Skrypt podpinany pod kamerę, który powoduję, że kamera podąża za graczem.

**CharacterController2D** – Skrypt, który zarządza fizyką podczas poruszania się. Umożliwia ruch, skakanie oraz kucanie. Definiuje się w nim, wysokość skoku, procent prędkości podczas kucania oraz możliwość kontroli podczas skoku.

**PlayerInfo** – Skrypt przechowujący podstawowe informację o graczu takie jak zdrowie aktualne i maksymalne, zebrane przedmioty, punkty, zarządzający aktualnie wybraną postacią i pozwalający na jej zmianę klawiszami 1 2 3 , zarządzający otrzymywaniem obrażeń, paskiem zdrowia w interfejsie, odtwarzający dźwięki podczas zranienia oraz przypisujący gracza jako dziecko platformy, na której aktualnie stoi aby w razie gdy ta się porusza, gracz poruszał się razem z nią. Pozwala również na pauzę klawiszem ESC. W momencie śmierci aktywuje ekran Game Over.

**PlayerMovement** – Skrypt, który przechwytuje wciskane przez gracza przyciski sterowania aby użyć odpowiednich funkcji skryptu **CharacterController2D** do poruszenia się postacią oraz odtwarza odpowiednie animacje. Pozwala też na ustalenie prędkości z jaką gracz się porusza.

**RopeScript** – Skrypt odpowiedzialny za tworzenie liny pomiędzy graczem, a hakiem wystrzeliwanym za pomocą skryptu **ThrowHook.** Gdy hak zostanie wystrzelony, skrypt tworzy pomiędzy nim, a graczem węzły zdefiniowane jako obiekty prefab posiadające w sobie komponenty **Line Renderer**  oraz **HingeJoint**. Następnie łączy każdy węzeł ze sobą, pierwszy z hakiem, a ostatni z graczem, a komponent **Line Renderer**  renderuje pomiędzy nimi linię. Pozwala na zdefiniowanie odległości pomiędzy węzłami, maksymalnej i minimalnej długości liny oraz za pomocą funkcji makeRopeShorter oraz makeRopeLonger zmieniać jej długość.

**ThrowHook -**  Skrypt umożliwia wystrzeliwanie haka z liną generowaną przez skrypt **RopeScript**, przyczepianego do platform. Wystrzeliwuje hak za pomocą prawego klawisza myszy w kierunku wskazanym przez kursor. Hak musi być przypisany jako obiekt prefab. Skrypt pozwala na zdefiniowanie prędkości wystrzeliwanego haka oraz warstwy do której hak może się przyczepiać. Pozwala też na zmianę jej długości klawiszami Q oraz E.

**ThrowQubic –** Skrypt pozwalający na tworzenie kostek terenu po lewej bądź prawej stronie postaci za pomocą sekwencji klawiszy C X lub C V. Kostka jest obiektem prefab. Skrypt pozwala na zdefiniowanie odległości kostki od gracza podczas tworzenia.

**Warrior -** Skrypt zarządzający dostępnością umiejętności wojownika (**WarriorAttack**), które stają się aktywne gdy gracz jest w stanie warrior.

**WarriorAttack –** Skrypt pozwalający na atakowanie mieczem i blokowanie ataków tarczą gdy gracz jest w stanie warrior. Pozwala na zdefiniowanie obrażeń ataku i zasięgu. Odtwarza również odpowiednie animacje.

**Wizard -** Skrypt zarządzający dostępnością umiejętności maga (**ThrowQubic**), które stają się aktywne gdy gracz jest w stanie wizard.

**Skypty Interfejsu oraz zarządzania grą**

**EndLevel –** Skrypt przypinany do obiektu, który kończy poziom. W naszym przypadku jaskini. Gdy gracz wejdzie w collider znajdujący się w obiekcie aktywuję się trigger, który blokuje sterowanie, pauzuje wszystkie obiekty oraz wyświetla ekran podsumowujący poziom i pozwalający na powrót do menu lub rozpoczęcie następnego poziomu.

**Fade –** Skrypt z metodą statyczną FadeCanvas pozwalający na stworzenie animacji znikania i pojawiania się obiektów Canvas.

**GameOverScreen –** Skrypt zarządzający przyciskami na obiekcie Canvas, który jest ekranem Game Over.

**LevelsManager –** Skrypt zarządzający przyciskami poziomów w menu głównym, aktywujący te które dla których poziomy są już odblokowane i przypisujący im odpowiednią funkcję ładującą poziom.

**MainMenu –** Skrypt zarządzający przyciskami głównego menu oraz przejściem pomiędzy odpowiednimi sekcjami menu.

**SceneLoader –** Skrypt ładujący poziomy oraz wyświetlający ekran ładowania.

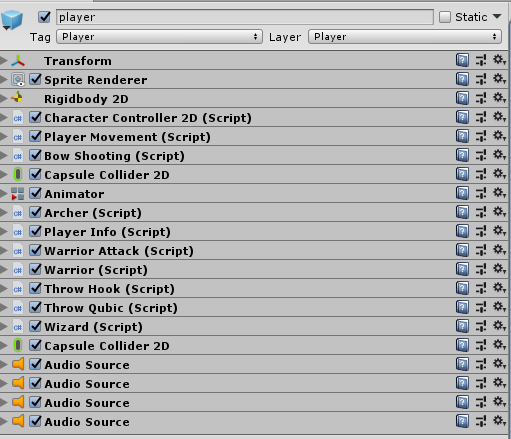
**UIController –** Skrypt zarządzający interfejsem podczas rozgrywki. Wyświetla stan życia, punktów, zebranych przedmiotów oraz postaci wybranej przez gracza.

**Obiekty utworzone w grze**

**Gracz**

****

**Komponenty**

****

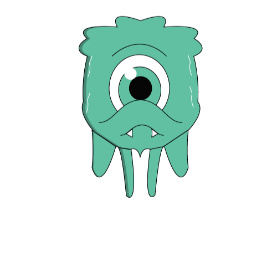
**Umiejętności**

**Łucznik -**  strzelanie z łuku oraz lina umożliwiająca zaczepianie się o elementy terenu

**Mag –** tworzenie ograniczonej liczby kostek terenu oraz zmniejszanie się

**Wojownik –** atak mieczem i blok ataków tarczą

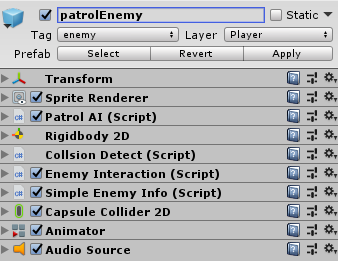
**Przeciwnicy patrolujący**

****

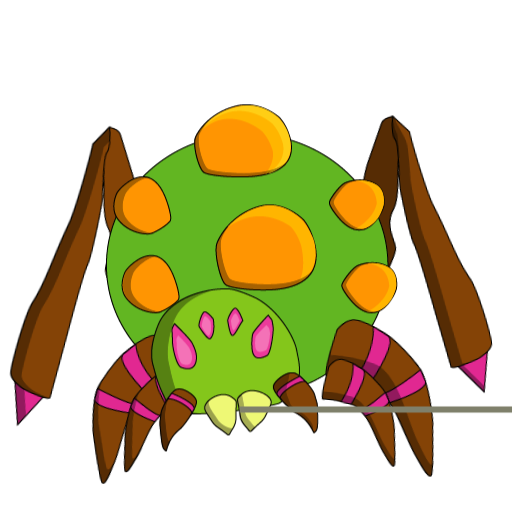
|  |  |  |
| --- | --- | --- |
| Muchomor | Imp | Snot |
| Las | Inferno | Kraina Lodu |

Przeciwnicy poruszający się cały czas od jednej krawędzi platformy, na której się znajdują do drugiej.

**Komponenty**

****

**Przeciwnicy skaczący**

****

|  |  |  |
| --- | --- | --- |
| Pająk | Płomień | Gremlin |
| Las | Inferno | Kraina Lodu |

Przeciwnicy poruszający się cały czas od jednej krawędzi platformy, na której się znajdują do drugiej i potrafiący przeskakiwać nad strzałami.

**Komponenty**

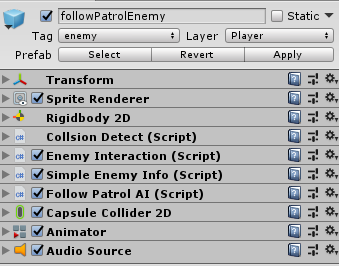
****

**Przeciwnicy ścigający**

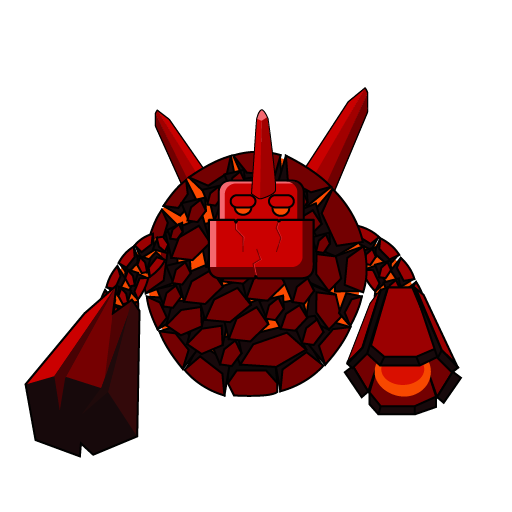
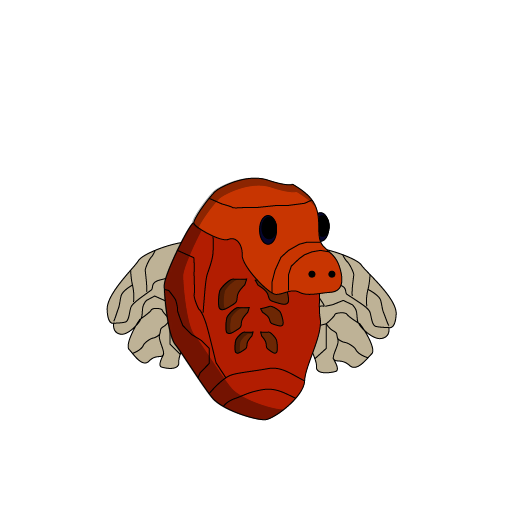
****

|  |  |  |
| --- | --- | --- |
| Wąż | Diabeł | Dżin |
| Las | Inferno | Kraina Lodu |

Przeciwnicy poruszający się cały czas od jednej krawędzi platformy, na której się znajdują do drugiej. W momencie pojawienia się gracza w zasięgu wzroku, zaczynają go gonić. Co ustalony czas sprawdzają czy gracz nie jest za ich plecami.

****

**Przeciwnicy strzelający**

****

|  |  |  |
| --- | --- | --- |
| Mucha | Hellrock | Zły mag |
| Las | Inferno | Kraina Lodu |

Przeciwnicy poruszający się cały czas od jednej krawędzi platformy, na której się znajdują do drugiej. Co ustalony odcinek czasu wystrzeliwują pocisk w kierunku, w którym się znajdują. Mogą również stać cały czas w jednym miejscu i wyłącznie strzelać.



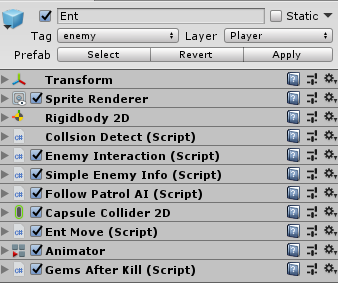
**Boss Lasu – Ent**

****

****

Boss Lasu Ent patroluje swój teren, co wyznaczony czas będzie wykonywał specjalny atak, w którym wystrzeliwuje serię pocisków. Gdy gracz zbliży się zbytnio wykona atak z wyskoku po czym będzie atakował normalnie. Po stracie połowy życia będzie blokował każdą strzałę.

**Komponenty**

****

**Boss Inferno**

****

**** ****

Boss Inferno będzie patrolował swoją okolicę wykonując dwa specjalne ataki co wyznaczony czas. Pierwszy z nich to uniesienie się w powietrze i wystrzelenie dookoła wielu kul ognia. Kule po zderzeniu się z terenem znikają. Po powrocie z powietrza Inferno staje się bezpronny przez krótki czas. Drugi atak to wytworzenie rozchodzących się płomieni po obu stronach Inferno. Po utracie połowy zdrowia Inferno „spala cięciwę łuku gracza” blokując całkowicie możliwość strzału.

**Komponenty**

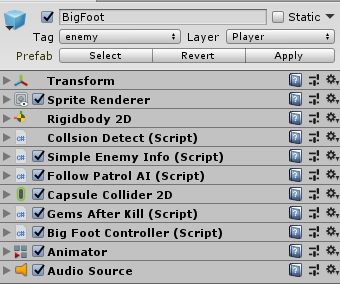
****

**Boss Krainy Lodu Wielka Stopa**

****

Wielka stopa będzie patrolować wyznaczony teren oraz wykonywać atak z wyskoku oraz rzut kulą śnieżną.

**Komponenty**



**Przedmioty**

|  |  |  |
| --- | --- | --- |
| **Moneta** |  |  |
| Można ją zebrać, po przejściu poziomu jest przeliczana na 100 punktów | | |
| **Klejnot** |  |  |
| Można go zebrać, po przejściu poziomu jest przeliczany na 500 punktów | | |
| **Jaskinia** |  |  |
| Po wejściu w jaskinie kończy ona poziom wyświetlając ekran podsumowujący | | |
| **Mikstura Duża** |  |  |
| Po zebraniu ulecza całe zdrowie | | |
| **Mikstura Mała** |  |  |
| Po zebraniu leczy fragment zdrowia | | |

**Pułapki i przeszkody**

|  |  |  |
| --- | --- | --- |
| **Kolce** |  |  |
| Po wejściu na nie gracz otrzymuje obrażenia i zostaje odepchnięty | | |
| **Piła** |  |  |
| Podobnie jak kolce po zetknięciu się z piłą gracz otrzymuje obrażenia i zostaje odepchnięty.  Występuje w trzech wersjach w zależności od wersji skryptu Saw. Obracającej się w miejscu, obracającej się w miejscu i chowającej się oraz poruszającej się pomiędzy dwoma punktami. | | |
| **Zapadnia** |  |  |
| Po wejściu na nią lewa bądź prawa część w zależności od ustawienia, zapadnie się pod ciężarem gracza po czym powróci do pierwotnej pozycji. | | |

|  |  |  |
| --- | --- | --- |
| **Kula na łańcuchu** | Składa się z trzech elementów, mocowania, łańcucha oraz kuli. Napięty łańcuch wraz z kulą kręci się wokół mocowania, a po zderzeniu z graczem odpycha go i zadaje obrażenia. Łańcuch jest połączony z mocowaniem i kulą za pomocą komponentu HingeJoint2D | |
|  | |  |
|  | |  |
|  | |  |

**Budowa Interfejsu**

**Ekran Powitalny**

****

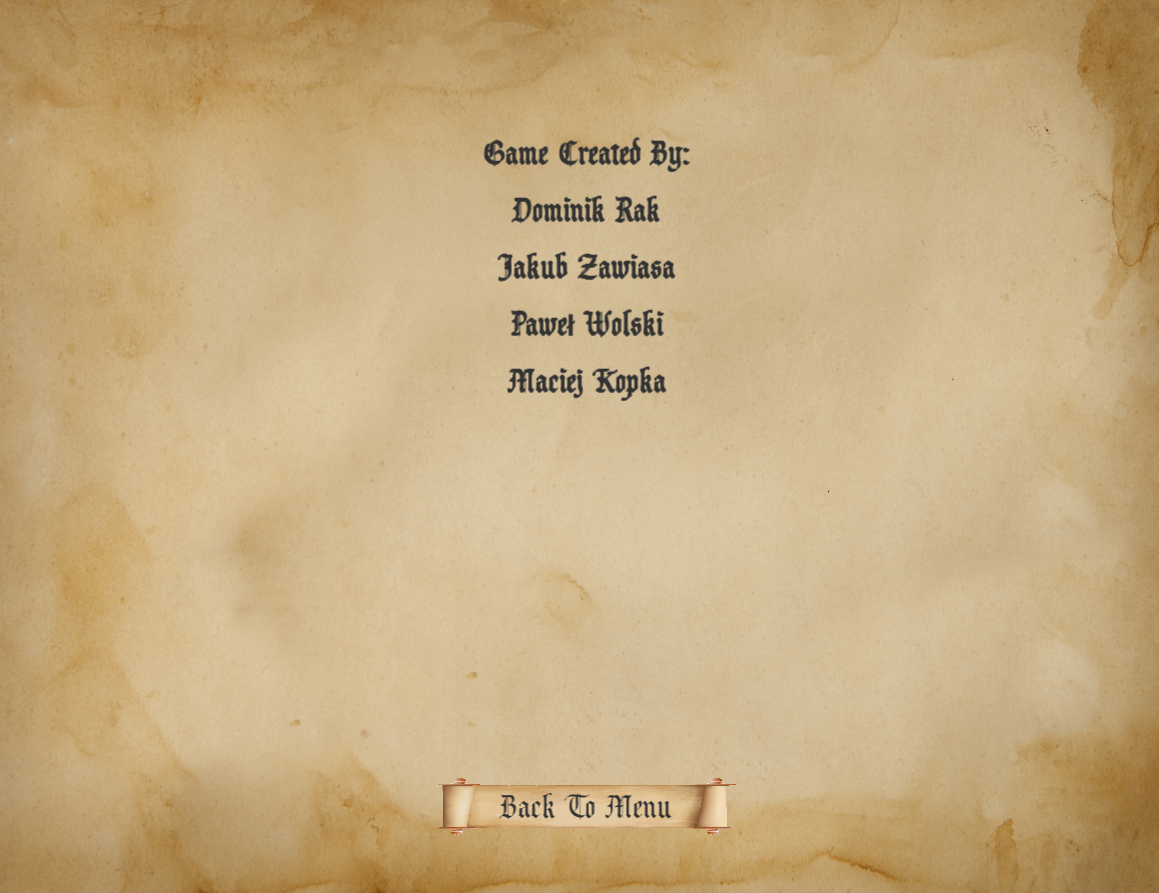
**Menu Główne**

|  |  |
| --- | --- |
|  | Rozpoczęcie nowej gry  Przejście do ekranu wyboru poziomu  Ekran z Informacjami o twórcach  Wyjście z gry |

**Ekran Wyboru poziomu**

|  |  |
| --- | --- |
|  | Przyciski przenoszące do konkretnego poziomu wraz ze sliderem  Powrót do menu |

**Ekran Credits**

****

**Ekran Ładowania**

Obrazek kryształu wypełnia się procentowo w takim stopniu w jakim jest załadowana scena

****

****

**Ekran Game Over**

Przycisk Restart ponawia poziom, przycisk Back to menu wraca do Menu.

****

**Ekran końca poziomu**

Wyświetla ilość zebranych monet i kryształów oraz sumę zdobytych punktów. Można z niego przenieść się do Menu lub następnego poziomu.

****

**Interfejs gracza podczas gry**

|  |  |  |
| --- | --- | --- |
| Ikona aktualnej postaci  Ilość zebranych monet i klejnotów  Ilość dostępnych kostek |  | Aktualny wynik |