Bullet Hell

Projekt zespołowy

Autorzy:

Igor Cybulski

Jakub Owczarek

Konrad Kruszelnicki

Maciej Świstak

# Wstęp

## Cel projektu.

Celem projektu było stworzenie gry, typu bullet hell, używając silnika Unity.

## Założenia

Głównymi założeniami dotyczącymi projektu były:

1. Stworzenie broni
2. Stworzenie przeciwników
3. Stworzenie systemu poruszania się oraz kolizji z przeciwnikami
4. Stworzenie wroga typu „boss”
5. Optymalizacja gry

## Problemy

Podczas tworzenia powyższych ficzerów, głównym problemem było stworzenie ich w taki sposób, aby działały poprawnie wraz z pozostałymi. Kolejnym problemem było napisanie takiego kodu, aby zmiany w nim były łatwe, oraz pozostali programiści mogli bez problemu zrozumieć w jaki sposób on działa, z pomocą przyszły wzorce projektowe oraz interfejsy.

# Podział obowiązków

Igor Cybulski – stworzenie systemu broni, system poziomów i doświadczenia, system ulepszania broni, 3 bronie, bazowe poruszanie się gracza (input system), bazowy przeciwnik, stworzenie pomocniczych skryptów które pomagają kontrolować przebieg gry, optymalizacja

Jakub Owczarek – stworzenie przeciwnika typu Slime, broni o nazwie ShootingBow, licznika czasu, systemu pozwalającego na „kiwanie” się przeciwników, systemu Knockback pozwalającego na odpychanie przeciwników przez pociski gracza

Maciej Świstak – zaprojektowanie i zaimplementowanie przeciwnika typu „boss” w sposób aby był on zbalansowany w porównaniu do pozostałych przeciwników

Konrad Kruszelnicki – stworzenie systemu kolizji postaci, dodanie zdrowia oraz paska zdrowia postaci, zaprojektowanie prototypowej broni „shuriken”, zaprojektowanie i dodanie mikstur życia oraz ich spawnera, spowolnienie czasu na zakończenie gry, zapętlenie rozrywki

# Opis gry

## Sterowanie

Za pomocą klawiszy W A S D, nasza postać porusza się, broń throwing knives (bazowa broń gracza), rzuca noże w kierunku kursora myszki.

## Gameplay

Pokonując przeciwników, możemy otrzymać punkty doświadczenia, gry zbierzemy wystarczającą ilość, postać zdobędzie kolejny poziom. Wraz z kolejnymi poziomami, gracz dostaje wybór, którą broń ulepszyć bądź czy dodać nową do arsenału. Gracz może posiadać maksymalnie <?> broni, z czego każda broń może mieć maksymalnie 6 poziom.

Z czasem przeciwnicy stają się silniejsi (posiadają więcej punktów zdrowia). Gracz po zetknięciu się z przeciwnikiem bądź jego pociskiem, traci zdrowie, gdy zdrowie spadnie do zera zostanie zagrana animacja śmierci, a plansza zrestartuje się. Zdrowie można zregenerować podnosząc mikstury zdrowia które pojawiają się losowo na mapie.

# Działanie programu

# broń

Każda broń dziedziczy po klasie weapon, zawiera on bazowe statystyki które broń powinna zawierać, przykładowo obrażenia oraz częstość używania. Zawiera także bazowe definicje podstawowych funkcjonalności, takich jak ulepszenie broni.

Przykład działania na podstawie broni throwing knives.

new protected void Start(){

        base.Start();

        cooldown=1;

        timer.Init(cooldown);

        timer.AddTimerEndEvent(SpawnProjectiles);

        timer.StartTimer();

        damage = 5;

        pierce = 1;

        speed = 7;

        knife = Resources.Load("Projectiles/Knife") as GameObject;

        playerPos = Player.Instance.gameObject.transform;

    }

protected override void SpawnProjectiles(){

        Vector2 v = IgorUtils.Vectors.GetMousePosNormalized();

        var spawnedKnife = Instantiate(knife, playerPos.position,

        IgorUtils.Rotation.GetRotation(Vector2.right,v),parent);

        spawnedKnife.GetComponent<StraightProjectile>().Init(v, damage, pierce, speed,knifeLifeSpan);

    }

Metoda void Start(), jest metodą używaną przez każdy obiekt silnika Unity. Uruchamia się ona w pierwszej klatce od razu gdy obiekt zostanie wczytany. W niej należy ustawić bazowe parametry broni, dodatkowo, jest wczytywany pocisk który dana broń wykorzystuje, przypisywana pozycja gracza do zmiennej oraz tworzony timer(o nich niżej). Metoda SpawnProjectiles(), jest metodą która definiuje w jaki sposób dana broń ma atakować, w przypadku ThrowingKnives, jest brana pozycja myszki względem gracza, następnie jest tworzony pocisk za pomocą instantiate. Po odpowiednim ustawieniu rotacji i pozycji pocisku, jest on inicjalizowany za pomocą parametrów broni.

## Projectile:

Projectile jest klasą definiującą zachowanie pocisków, posiada on interfejs IDamager, definiujący zachowanie obiektów które zadają obrażenia. Aby zoptymalizować grę należało także zaimplementować „długość życia pocisku”: wystrzelony pocisk po x sekundach, zostanie zniszczony, w przeciwnym wypadku, pociski mogłyby lecieć w nieskończoność a tym samym zmniejszając ilość klatek na sekundę. W przypadku ThrowingKnives są używane StraightProjectile’e, są to pociski które poruszają się po linii prostej we wskazanym kierunku.

Kod poruszający pociskiem. Należy tutaj zauważyć że każdy kod który jest wykonywany co klatkę, musi być pomnożony przez Time.fixedDeltaTime, gdy jest w on użyty w FixedUpdate(), oraz Time.DeltaTime, gdy jest on użyty w Update(). Powodem tego jest to aby każdy gracz bez względu na ilość klatek na sekundę miał takie same działanie programu.

protected override void FixedUpdate()

    {

        transform.position += direction \* speed \* Time.fixedDeltaTime;

    }

## 4.3 Timer:

IEnumerator TimerLoop(){

        while(true){

            timer-=tickSpeed;

            yield return new WaitForSeconds(tickSpeed);

            if(timer<=0){

                timer=timerBase;

                onTimerEndEvents.Invoke();

            }

        }

    }

Timer jest klasą która pozwala na łatwe tworzenie eventów, które mają się powtarzać co x sekund. Tworząc i uruchamiając timer, wywoływany jest „coroutine” który co x sekund wywołuje jakąś metode lub zestaw metod z innego skryptu, w przypadku broni będzie to użycie broni. Coroutine na polski oznacza współprogram, jest to specyficzny typ funkcji która po wyjściu może wrócić do tego samego miejsca w którym ostatnio się skończyła, co pozwala na tworzenie wydarzeń które mogą się dziać przez dłuższy czas, lub okresowo. W przypadku zwykłej funkcji, gdy program nie jest wielowątkowy, program rozpoczynając wykonywanie funkcji, nie wykona żadnych innych operacji póki nie skończy wykonywania aktualnej funkcji.

Dodatkowo timery posiadają inne metody pomagające kontrolować ich przebieg, takie jak

UpdateTimer() - > wstawienie nowych wartości timera, innymi słowy przyśpieszenie lub zwolnienie timera bez zatrzymywania go

StopTimer() -> zatrzymanie timera

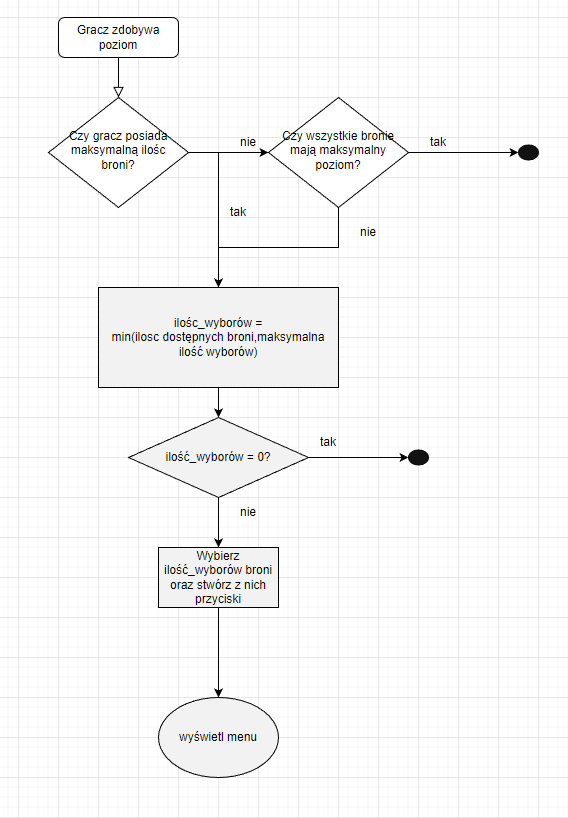
## 4.3 System doświadczenia:

Po pokonaniu przeciwnika upuszcza on klejnot doświadczenia, gdy gracz się zbliży do niego na pewną odległość poleci on w stronę gracza, dając mu punkty doświadczenia. Gdy ilość punktów doświadczenia przekroczy próg do następnego poziomu, otworzy się menu z wyborem broni.



Gracz może wybrać jedną z maksymalnie 3 propozycji, ulepszając już posiadaną broń, lub dodając kolejną do swojego arsenału.

Schemat blokowy kodu który zarządza powyższym menu.



Ilość dostępnych broni oznacza, ilość tych broni które nie są w arsenale gracza, lub są w arsenale ale nie posiadają maksymalnego poziomu.

## 4.4 Przeciwnicy:

Każdy z przeciwników dziedziczy po klasie BaseEnemy. Zawiera ona bazowe statystyki, które przeciwnik powinien zawierać takie jak np. życie, prędkość, ilość zadawanych obrażeń oraz informacje z których korzysta gra takie jak np. RigidBody2D(hitbox), facingright(kierunek w którym patrzy się przeciwnik)

Obraz zawierający tekst

Opis wygenerowany automatycznie

Metoda Start() ustala poprzez zmienną Target, cel dla przeciwnika jakim jest gracz, rb w którym przechowuje hitbox danego przeciwnika oraz jego hp , które zwiększa się wraz z upływem czasu.

Metoda FixedUpdate(), która wykonuje się w każdej klatce programu i spełnia takie zadania jak: sprawdzenie czy jest możliwość ruchu, kiwanie przeciwnika ,poruszanie przeciwnikiem, obracaniem przeciwnika w kierunku gracza poprzez funkcję Flip(). Klasa ta posiada także funkcję TakeDamage(int dmg) która pozwala przeciwnikowi otrzymywać obrażenia, zaaplikować Knockack i w razie potrzeby usunąć przeciwnika z gry metodą Destroy(gameObject) wraz ze zostawieniem kuli doświadczenia czym zajmuje się metoda DropLoot(). Ostatnią metodą pozwalającą zadawać obrażenia graczowi jest DealDamage(Idamagable damagable).

1)Skeleton

Najprostszy typ przeciwnika, porusza się w kierunku gracza i zadaje mu obrażenia gdy się z nim zderzy. Jest obiektem klasy BaseEnemy

2)Slime

Slime porusza się w kierunku przeciwnika aż osiągnie pewną ustaloną od niego odległość, wtedy zatrzymuje się i zaczyna strzelać pociskami w kierunku gracza. Zadaje obrażenia poprzez swoje pociski i zderzenie z graczem.

W metodzie Start() oprócz wykonania poleceń Start() z klasy dziedziczącej pobierany jest obrazek pocisku oraz tworzony timer, który odpowiada za częstotliwość strzelania. W każdej klatce następuje decyzja czy Slime powinien poruszać się dalej w kierunku gracza czy może zacząć strzelać.

3)Tammy

Tammy jest wrogiem, który z założenia miał być „bossem”, trudniejszym przeciwnikiem, ale zarówno zbalansowanym, tak aby nie był zbyt mocny, ale również żeby nie był zbyt słaby. Porusza się on w stronę gracza ciągle strzelając dookoła siebie. Ze względu na dużą ilość pocisków, które wystrzeliwuje nie zadają one bardzo dużej ilości obrażeń, natomiast pociski są częściej używane. Każda kolejna seria pocisków jest obracana o drobną ilość, aby ataków nie dało się unikać poprzez stanie w jednym punkcie. Ma on również zwiększoną ilość życia w porównaniu z podstawowymi wrogami. Jako, że jest to bardziej wymagający przeciwnik w porównaniu do pozostałych pozostawia on również więcej punktów doświadczenia.

## 4.4a Knockback:

Knockback odpowiada za lekkie odepchnięcie przeciwników w zetknięciu z pociskiem gracza. Obraz zawierający tekst

Opis wygenerowany automatycznie

Zmienna delay przechowuje czas po jakim Knockback znów może nastąpić, rb przechowuje hitbox, a baseEnemy odpowiedni komponent gry czyli przeciwnika. Metoda PlayFeedback(GameObject sender, float strength) przyjmująca dwa parametry: obiekt który wyśle Feedback oraz jego siłę. Funkcja zmienia wartość can\_move z obiektu BaseEnemy aby KnockBack był widoczny, następnie ustala kierunek i aplikuje odpowiednią siłę na pobrany hitbox. Funkcja Reset() pilnuje aby Feedback uruchamiał się co odpowiedni czas i pozwala na ponowny ruch przeciwnika po przesunięciu.

## 4.5 Zdrowie postaci gracza

Postać gracza posiada i zaczyna z góry określoną wartością punktów życia (health points - HP). Przeciwnicy w zależności od ich rodzaju mogą zadać postaci gracza obrażenia redukując wartość jego obecnego zdrowia. W przypadku gdy zdrowie postaci spadnie do zera – postać gracza przestaje wchodzić w interakcję z otoczeniem, następuje animacja śmierci i rozrywka zaczyna się od nowa. Gracz może przywrócić swojej postaci określoną wartość HP poprzez zebranie mikstur życia pojawiających się na mapie. W celu zbalansowania poziomu trudności istnieje maksymalna wartość równoczesnych mikstur życia na mapie. Aby umożliwić graczowi śledzenie na bieżąco stanu zdrowia postaci dodano pasek zdrowia. Utworzony został za pomocą dwóch nachodzących na siebie sprite’ów – jednego w kolorze czarnym wizualizującym wartość zdrowia maksymalnego oraz drugiego w kolorze czerwonym indykującym obecną wartość HP.

Obraz zawierający tekst

Opis wygenerowany automatycznie

Rysunek Tuż pod postacią gracza znajduje się pasek zdrowia. Na prawo od postaci widać miksturę życia.