

Dokumentacja techniczna

**Planet of war ZTWN-6182**

**Michał “Szorader” Górecki**

SPIS TREŚCI

1.Wprowadzenie....................................................................................................................................3

Opis gry, Zawartość dokumentacji

2. Gameplay, Mechaniki gry, Komponenty gry......................................................................................4

Tutorial, Zasady gry, Sterowanie, Interface, Postacie, Poziomy

3.Skrypty...............................................................................................................................................164.Grafika i Dźwięk................................................................................................................................46 5.Silnik i Technologie...........................................................................................................................47 Edytor, Wykorzystane biblioteki, Funkcje, Języki programowania, Wymagania sprzętowe

**1.Wprowadzenie**

**Opis gry:**

* 1. *Gatunek*: Tower Defense
  2. *Temat*: Przenosimy się na pustynne wzgórza i śnieżne czubki gór z gęstymi lasami planety ZTWN-6182, wśród górniczego ludu ztrowegów doszło do buntu przeciwko władzy. Gracz ma za zadanie pomóc królowi w obronie bazy poprzez stawianie wież obronnych i powstrzymywanie buntowników w ich pojazdach.
  3. *Cechy*:
     1. Unikalny świat: Gra przenosi gracza na tajemniczą planetę ZTWN-6182, która jest charakteryzowana przez pustynne wzgórza, śnieżne czubki gór i gęste lasy. Ten niezwykły świat stanowi tło dla rozgrywki.
     2. Tower Defense: Gra należy do gatunku Tower Defense, w którym gracz musi strategicznie rozmieszczać wieże obronne w celu powstrzymania fal wrogów przed dotarciem do bazy.
     3. Obronny aspekt: Celem gry jest obrona bazy przed buntownikami. Gracz musi budować i ulepszać wieże obronne, aby skutecznie odpierać ataki wrogów.
     4. Różnorodność wrogów: Wrogowie w grze reprezentują różne pojazdy używane przez buntowników. Każdy z nich ma swoje unikalne cechy i wymaga dostosowania strategii obronnej.
     5. Różnorodność wież: Gra oferuje różne rodzaje wież obronnych, z różnymi umiejętnościami i atakami. Gracz może wybierać spośród różnych strategii obronnych, dostosowując je do rodzaju wrogów i terenu.
     6. Rozwój i ulepszanie: Gracz może rozwijać swoje wieże, odblokowywać nowe umiejętności i ulepszać istniejące, co pozwala na bardziej efektywną obronę bazy.

**Zawartość dokumentacji:**

1. *Opis gry*: Szczegółowy opis gry, jej mechaniki i celu rozgrywki.
2. *Instrukcje*: Przewodnik, który wyjaśnia zasady gry, sposób sterowania, budowania wież, ulepszania itp.
3. *Mechaniki gry*: Dokładne omówienie różnych aspektów gry, takich jak system wież obronnych, fal wrogów, zarządzanie zasobami, rozwój postaci itp.
4. *Opis wrogów*: Charakterystyka różnych rodzajów wrogów, ich zachowanie, mocne i słabe strony.
5. *Opis wież*: Informacje o różnych rodzajach wież obronnych, ich zdolnościach, kosztach, możliwościach ulepszeń itp.
6. *Strategie i taktyki*: Porady i sugestie dotyczące skutecznych

**2.Gameplay, mechaniki gry, komponenty gry**

**Tutorial**

1. Sterowanie jest przedstawione poniżej.
2. Wejście na dany poziom nie powinno sprawić trudności ufam tobie, że ci się uda.
3. Stawianie wieży czyli musisz wybrać najpierw jaka cię interesuje i masz na nią zasoby
4. Następnie najeżdzasz na białe koło (zmienia kolor zależnie czy stać na wieżę czy nie) i klikając PPM stawiasz
5. Twoim zadaniem jest bronić bazy
6. Teraz planujesz własne taktyki i strategie aż pokonasz wszystkich wrogów
7. Powodzenia!!!!

**Zasady gry:**

1. Gracz musi wybudować wieże obronne w strategicznych punktach, aby powstrzymać nadchodzące fale wrogów. Każda wieża posiada unikalne umiejętności i cechy, takie jak szybkostrzelność, zasięg lub obrażenia.
2. Gracz ma ograniczone zasoby, które musi wykorzystać w sposób strategiczny. Niektóre wieże są droższe niż inne, co wymaga od gracza dokładnego zaplanowania ich rozmieszczenia.
3. Wrogowie poruszają się po ustalonych ścieżkach, zabicie ich pozwala na uzyskanie większej ilości zasobów
4. Gracz przegrywa, gdy przeciwnicy zniszczą jego bazę. Gracz wygrywa, gdy pokona wszystkie fale wrogów.

**Sterowanie**

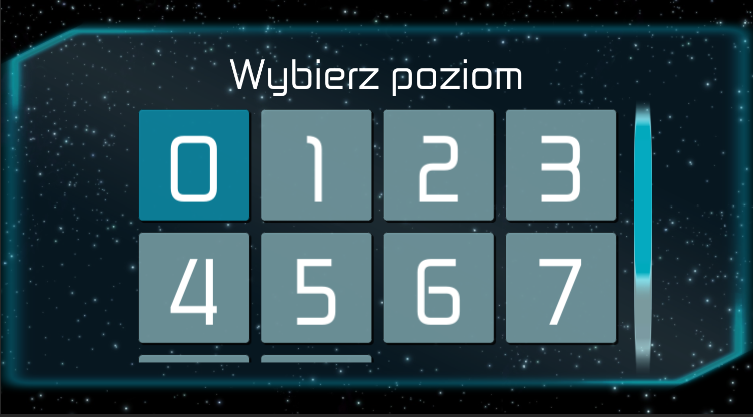
1. *Sterowanie klawiszami WSAD:*
   * 1. Klawisz W: Poruszanie się do przodu.
     2. Klawisz S: Poruszanie się do tyłu.
     3. Klawisz A: Poruszanie się w lewo.
     4. Klawisz D: Poruszanie się w prawo.
2. *Poruszanie przy zbliżeniu do krawędzi ekranu:*
3. Jeśli gracz zbliży się do krawędzi ekranu, kamera będzie automatycznie poruszać się w tym kierunku, aby zapewnić lepszą widoczność i umożliwić eksplorację świata gry.
4. *Przycisk Esc i przycisk P- Menu pauzy:*
5. Po naciśnięciu przycisku Esc, zostanie wywołane menu pauzy, które zawiera opcje takie jak: powrót do głównego menu lub kontynuacje.
6. *Interakcja za pomocą prawego przycisku myszy (PPM):*
7. Prawy przycisk myszy służy do interakcji z interfejsem użytkownika. Może to obejmować klikanie na przyciski, przeciąganie elementów, wybieranie opcji z rozwijanych menu itp.

**Interface**

1. W menu głównym gry znajduje się interfejs, który prezentuje użytkownikowi ważne opcje i informacje. Interfejs ten składa się z dwóch przycisków: "Graj" i "Wyjdź". Przycisk "Graj" umożliwia rozpoczęcie gry i przeniesienie gracza do wyboru poziomu. Przycisk "Wyjdź" pozwala użytkownikowi na bezpieczne opuszczenie gry. Jednak oprócz tych przycisków, istnieje również bardzo istotna informacja, która jest wyświetlana w menu głównym, której nie należy pomijać.



1. W menu wyboru poziomu gracze mają możliwość dokonania wyboru spośród dostępnych poziomów gry. Jest to miejsce, w którym mogą zdecydować, na którym poziomie chcą rozpocząć swoją przygodę. Wybór poziomu odbywa się poprzez interakcję z odpowiednimi przyciskami reprezentującymi każdy poziom. Co ważne, nie wszystkie poziomy są od razu dostępne dla gracza. Niektóre z nich mogą być zablokowane na początku gry, a ich odblokowanie wymaga ukończenia wcześniejszych poziomów.



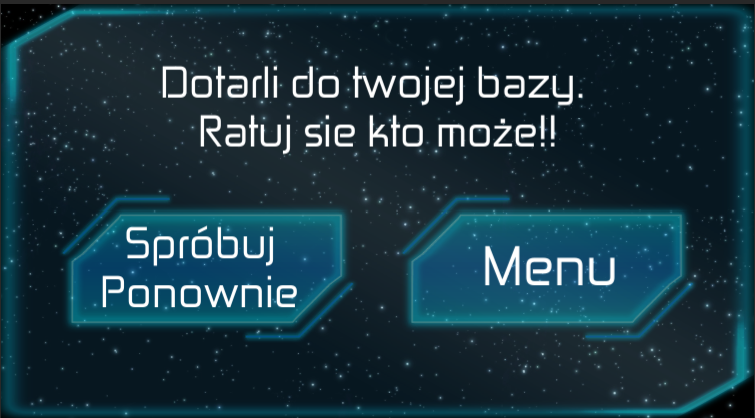
1. Interface w czasie rozgrywki składa się z trzech zasadniczych sekcji, które mają istotne znaczenie dla przebiegu gry i dostarczają gracza niezbędnych informacji. Każda sekcja ma swoje określone miejsce na ekranie i pełni ważne funkcje, aby zapewnić płynność i zrozumiałość rozgrywki. Pierwsza sekcja to odliczanie czasu do rozpoczęcia kolejnej rundy po zniszczeniu wszystkich przeciwników. Jest to kluczowy element, który dostarcza graczowi informację o zbliżającym się wyzwaniu. Czas do rozpoczęcia kolejnej rundy może mieć wpływ na strategię gracza, motywując go do szybkiego przygotowania się na następne starcie. Zawiera również informacje dotyczące ilości aktualnych zasobów. Druga sekcja składa się z ikony wieży oraz jej kosztu. Ta sekcja umożliwia graczowi wybór i budowę wież w odpowiednich miejscach na planszy. Ikona wieży reprezentuje rodzaj wieży, który gracz może postawić, natomiast podana obok cena wieży informuje gracza o koszcie zbudowania danej wieży. Jest to istotne, ponieważ gracze muszą zarządzać swoim budżetem i decydować, na jakie wieże warto zainwestować, aby skutecznie bronić się przed przeciwnikami. Trzecia sekcja zawiera aktualną liczbę posiadanych żyć. Życia są kluczowym wskaźnikiem w grze, ponieważ reprezentują zdolność gracza do przetrwania. Ta sekcja informuje gracza o aktualnej liczbie pozostałych żyć i stanowi istotne wyzwanie, które gracze muszą chronić i utrzymać na jak najwyższym poziomie. W przypadku utraty wszystkich żyć, gra może dojść do końca, więc gracze muszą być świadomi i podejmować odpowiednie decyzje, aby uniknąć porażki.



1. Pierwszą opcją w Pause Menu jest "Kontynuuj". Ta opcja umożliwia graczowi powrót do gry w miejscu, w którym ją zatrzymał. Dzięki temu gracze mogą natychmiast wznowić swoją przygodę bez konieczności ponownego rozpoczynania poziomu. To pozwala na płynne przejście z przerwy w grze do kontynuacji rozgrywki. Drugą opcją w Pause Menu jest "Menu". Ta opcja otwiera główne menu gry, gdzie gracze mogą mieć dostęp do różnych funkcji, takich jak wybór poziomu. Trzecią opcją w Pause Menu jest "Spróbuj ponownie". Ta opcja pozwala graczowi na powtórzenie obecnego poziomu od początku. Jest to przydatna opcja dla graczy, którzy chcą poprawić swoje wyniki, zmienić strategię lub po prostu spróbować jeszcze raz. Dzięki tej funkcji gracze mają kontrolę nad swoim postępem i mogą dążyć do doskonałości w każdej rozgrywce.



1. Pierwszą opcją w Menu Game Over jest "Menu". Ta opcja przenosi gracza z powrotem do głównego menu gry, gdzie mogą uzyskać dostęp do różnych funkcji, takich jak wybór poziomu, ustawienia czy informacje o grze. Dzięki tej opcji gracze mogą w dowolnym momencie przenieść się do menu i eksplorować inne elementy gry. Drugą opcją w Menu Game Over jest "Spróbuj ponownie". Ta opcja daje graczowi możliwość natychmiastowego rozpoczęcia nowej rozgrywki na tym samym poziomie, który doprowadził do przegranej. Jest to świetne rozwiązanie dla graczy, którzy chcą poprawić swoje umiejętności, zmienić strategię lub po prostu spróbować ponownie. Dzięki temu gracze mogą szybko wrócić do akcji i podjąć kolejne wyzwanie. Dodatkowo, Menu Game Over zawiera wyraźny napis z informacją o przegranej. Ten napis jest wyświetlany w sposób czytelny i zauważalny, aby gracze od razu odczytali, że zostali pokonani. Dzięki tej informacji gracze mają jasne zrozumienie sytuacji i mogą podjąć odpowiednie działania, takie jak powrót do menu lub próba ponownego przejścia poziomu.



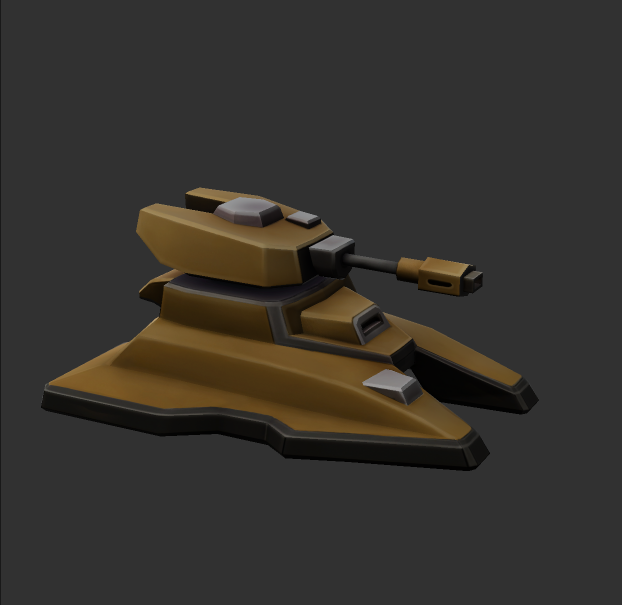
1. Pierwszą opcją w Menu Level Complete jest przycisk "Menu". Ta opcja przenosi gracza z powrotem do głównego menu gry, gdzie mogą eksplorować inne funkcje, takie jak wybór poziomu, ustawienia czy informacje o grze. Dzięki tej opcji gracze mogą wybrać kolejne wyzwania lub dostosować swoje ustawienia zgodnie z preferencjami. Drugą opcją w Menu Level Complete jest przycisk "Start". Ta opcja pozwala graczowi na natychmiastowe rozpoczęcie następnego poziomu gry, aby kontynuować swoją przygodę. Jest to doskonałe rozwiązanie dla graczy, którzy chcą kontynuować rozwijanie swoich umiejętności i zdobywać kolejne wyzwania. Dzięki tej opcji gracze mogą bezpośrednio przejść do następnego poziomu, nie tracąc czasu na powtarzanie już ukończonych etapów. Dodatkowo, Menu Level Complete zawiera wyraźny napis z informacją o wygranej. Ten napis jest wyświetlany w sposób czytelny i zauważalny, aby gracze od razu odczytali, że osiągnęli sukces na danym poziomie. Dzięki tej informacji gracze mogą cieszyć się swoimi osiągnięciami i mieć motywację do dalszego postępu w grze.



1. Opuszczenie granic mapy. Komunikat informuje gracza, że wykroczył poza granice mapy. Ta informacja jest istotna, ponieważ pomaga graczowi zorientować się, że przemieszczenie poza dopuszczalny obszar może mieć negatywne konsekwencje dla rozgrywki. Napis dostarcza jednoznacznej i precyzyjnej informacji, która umożliwia graczowi podjęcie odpowiednich działań w celu uniknięcia dalszych naruszeń granic.



**Postacie**

1. MachinGun: Wieża MachinGun to podstawowa jednostka obronna. Posiada szybkostrzelne karabiny maszynowe, które zadają umiarkowane obrażenia wrogim jednostkom na krótkim dystansie. Jest skuteczna przeciwko słabszym wrogom i może zatrzymać ich na początkowych etapach gry. Zadaje 30 obrażeń na sekundę, a jej szybkostrzelność to 2 pociski na sekundę, a jej koszt to 100$ a jej życie to 90 punktów.
2. LaserTower: Wieża Laserowa wykorzystuje potężny wiąz laserowy do rażenia wrogów. Zadaje 80 obrażeń na sekundę a jej koszt to 300$ a jej życie to 300 punktów.
3. RocketTower: Wieża Rakietowa wystrzeliwuje rakiety, które wybuchają na wrogu, powodując obrażenia. Zadaje obrażenia równe 100, a jej szybkostrzelność to 1 rakieta na dwie sekundy, a jej koszt to 200$ a jej życie to 180 punktów.
4. SuperTower: Lepsza Wieża Rakietowa wystrzeliwuje rakiety, które wybuchają na wrogu, powodując obrażenia. Zadaje obrażenia równe 300, a jej szybkostrzelność to 1 rakieta na 4 sekundy, a jej koszt to 400$ a jej życie to 300 punktów.
5. EnemyTank: Enemy Tank pojazd przeciwnika, który atakuje wieże i ma na celu dotarcie do naszej bazy. Zadaje obrażenia 10 a 20 obrażeń na sekundę, po zabiciu uzyskamy 75$ zasobów a posiada 105 punktów życia.
6. Buggy: Buggy szybka pozornie niegroźna jednostka nie strzela ale szybko jeździ, po zabiciu uzyskamy 50$ zasobów a posiada 200 punktów życia.
7. BossTankEnemy: BossTankEnemy potężniejsza wersja EnemyTanka. Zadaje 100 obrażeń a łącznie 200 obrażeń na sekundę, po zabiciu uzyskamy 200$ zasobów a posiada 500 punktów życia.

**Poziomy**

1. Ogólna zasada polega na tym, że wrogie jednostki pojawiają się w określonym punkcie, który jest oznaczony symbolem gwiazdki. Może działać jeden punkt albo wszystkie jednocześnie. Pozycja pojawienia się wrogów może się różnić dla różnych poziomów gry. Każdy poziom da się przejść!!.

**3. Skrypty**(Opis niektórych skryptów)

**Skrypt AutoDestroy**

Opis: Skrypt AutoDestroy jest odpowiedzialny za automatyczne niszczenie obiektu, do którego jest dołączony, po określonym czasie. Jest to przydatne w przypadku tymczasowych obiektów, które nie są już potrzebne w dalszej części gry, takich jak efekty wizualne czy dźwięki.

**Używane komponenty Unity:**

* Brak używanych komponentów.

**Zmienne:**

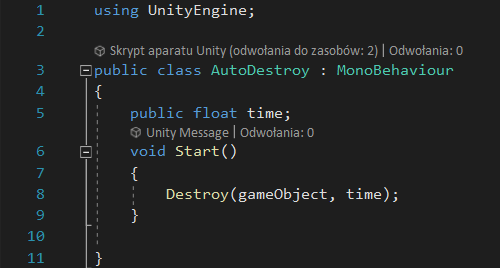
* time (float): Określa czas w sekundach, po którym obiekt ma zostać zniszczony.

**Metody:**

* Start(): Metoda ta jest wywoływana raz, przy starcie gry. Wewnątrz tej metody jest wywoływana funkcja Destroy(), która przyjmuje jako argument obiekt, do którego jest dołączony skrypt (gameObject), oraz wartość zmiennej time. Powoduje to zniszczenie obiektu po upływie określonego czasu.

**Użycie:**

1. Dołącz skrypt AutoDestroy do obiektu, który chcesz automatycznie zniszczyć.
2. W polu "time" ustaw wartość czasu w sekundach, po którym obiekt ma zostać zniszczony.



**Skrypt Base**

**Opis**: Skrypt **Base** jest odpowiedzialny za obsługę kolizji z wrogimi obiektami, które przychodzą w kontakt z bazą gracza. Po zderzeniu z wrogiem, skrypt zmniejsza liczbę dostępnych żyć gracza i sprawdza, czy liczba żyć wynosi zero, co powoduje zakończenie gry.

**Używane komponenty Unity**:

* + Brak używanych komponentów.

**Zmienne**:

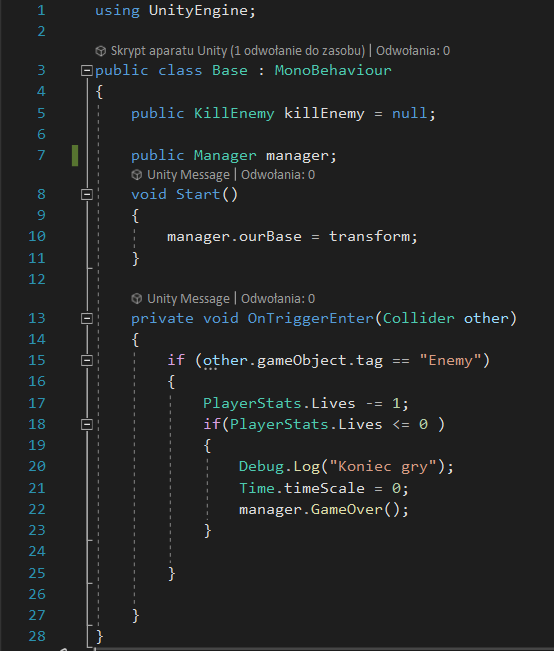
* + **killEnemy** (*KillEnemy*): Obiekt klasy **KillEnemy**, który odwołuje się do innego skryptu odpowiedzialnego za niszczenie wrogich obiektów.
  + **manager** (*Manager*): Obiekt klasy **Manager**, który zarządza różnymi aspektami gry.

**Metody**:

* + **Start()**: Metoda wywoływana na początku gry. Przypisuje bieżący obiekt transformacji (czyli bazę gracza) do zmiennej **ourBase** w skrypcie **manager**, co umożliwia innym elementom gry dostęp do bazy.
  + **OnTriggerEnter(Collider other)**: Metoda wywoływana, gdy inny obiekt wchodzi w strefę kolizji tego obiektu. Sprawdza, czy kolidujący obiekt ma tag "Enemy" (wrogowie). Jeśli tak, zmniejsza liczbę dostępnych żyć gracza o 1. Następnie sprawdza, czy liczba żyć gracza wynosi zero. Jeśli tak, wyświetla informację o zakończeniu gry, zatrzymuje czas gry (Time.timeScale = 0) i wywołuje metodę **GameOver()** w skrypcie **manager**.

**Użycie**:

1. Dołącz ten skrypt do obiektu reprezentującego bazę gracza w scenie.
2. Upewnij się, że masz skrypt **KillEnemy** podpięty do innego odpowiedniego obiektu w grze.
3. Przypisz obiekty **KillEnemy** i **Manager** do odpowiednich zmiennych w inspektorze skryptu **Base**.



**Skrypt BuildManager**

**Opis**: Skrypt BuildManager jest odpowiedzialny za zarządzanie budową wież obronnych w grze. Umożliwia wybieranie i budowanie różnych typów wież na odpowiednich pozycjach.

**Używane komponenty Unity**:

* + Brak używanych komponentów.

**Zmienne**:

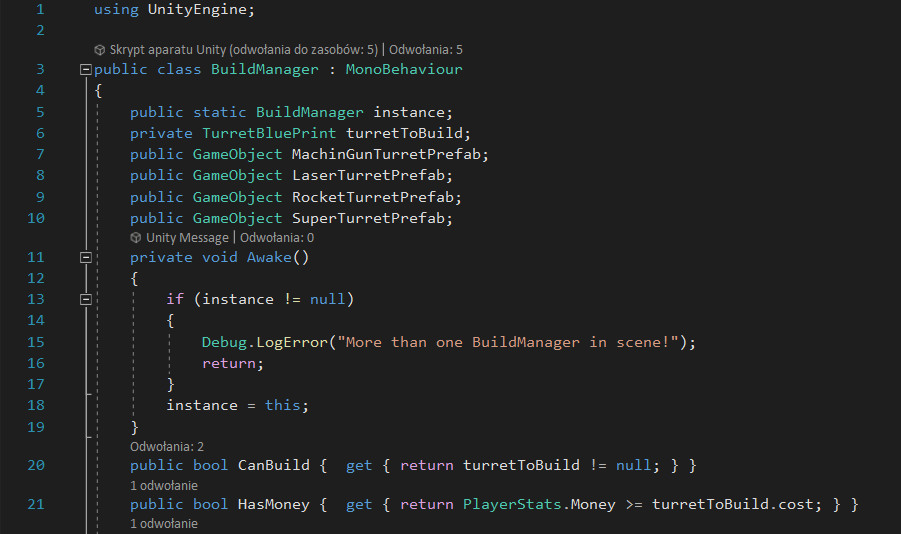
* + **instance** (*BuildManager*): Referencja do jedynego istniejącego obiektu BuildManager w scenie. Umożliwia dostęp do tego skryptu z innych skryptów.
  + **turretToBuild** (*TurretBluePrint*): Obiekt przechowujący informacje o wybranym typie wieży, która ma zostać zbudowana.
  + **MachinGunTurretPrefab** (*GameObject*): Prefab dla wieży maszynowej.
  + **LaserTurretPrefab** (*GameObject*): Prefab dla wieży laserowej.
  + **RocketTurretPrefab** (*GameObject*): Prefab dla wieży rakietowej.
  + **SuperTurretPrefab** (*GameObject*): Prefab dla superwieży.

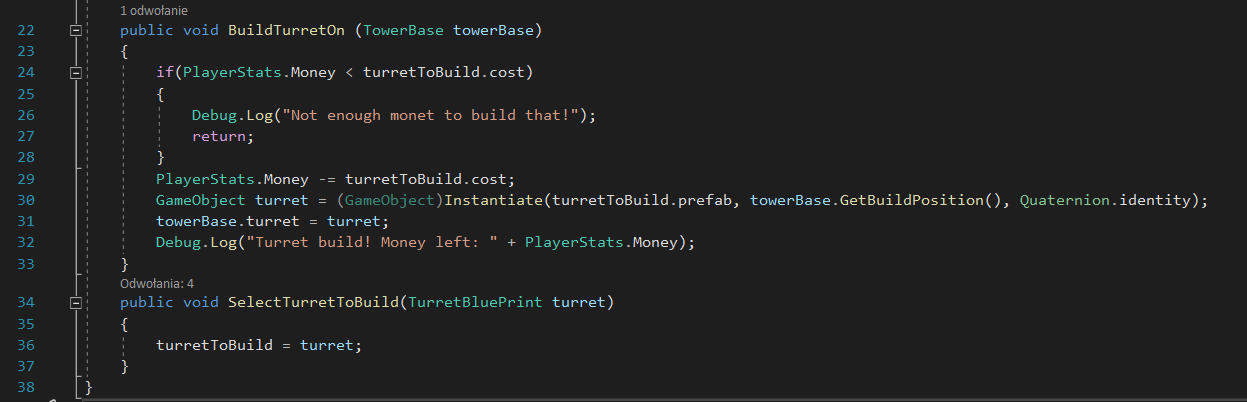
**Metody**:

* + **Awake()**: Metoda wywoływana zaraz po utworzeniu obiektu. Sprawdza, czy istnieje już inny obiekt BuildManager w scenie. Jeśli tak, wyświetla błąd w konsoli i zatrzymuje tworzenie obecnego obiektu.
  + **CanBuild** (*bool*): Właściwość tylko do odczytu zwracająca informację, czy jest wybrany typ wieży do zbudowania.
  + **HasMoney** (*bool*): Właściwość tylko do odczytu zwracająca informację, czy gracz ma wystarczającą ilość pieniędzy na zbudowanie wybranej wieży.
  + **BuildTurretOn(TowerBase towerBase)**: Metoda budująca wieżę na podanej pozycji (towerBase). Sprawdza, czy gracz ma wystarczającą ilość pieniędzy na zbudowanie wybranej wieży. Jeśli nie, wyświetla odpowiednie powiadomienie. Jeśli tak, obniża ilość pieniędzy gracza, tworzy prefab wieży na odpowiedniej pozycji i przypisuje go do zmiennej turret w skrypcie TowerBase.
  + **SelectTurretToBuild(TurretBluePrint turret)**: Metoda wybierająca typ wieży do zbudowania na podstawie przekazanego obiektu TurretBluePrint.

**Użycie**:

1. Dołącz ten skrypt do obiektu BuildManager w scenie.
2. Przypisz odpowiednie prefabry wież do zmiennych MachinGunTurretPrefab, LaserTurretPrefab, RocketTurretPrefab i SuperTurretPrefab w inspektorze skryptu.
3. Aby zbudować wieżę, wywołaj metodę BuildTurretOn, przekazując obiekt TowerBase jako argument.
4. Aby wybrać typ wieży do zbudowania, wywołaj metodę SelectTurretToBuild, przekaz





**Skrypt BulletTower**

**Opis**: Skrypt **BulletTower** odpowiada za poruszające się pociski związane z wieżą. Pociski mają określoną prędkość i zadają pewną ilość obrażeń wrogim jednostkom. Po trafieniu w wroga, pocisk jest niszczony.

**Używane komponenty Unity**:

* + Transform: do manipulacji pozycją i rotacją pocisku.
  + AudioSource: do odtwarzania dźwięku eksplozji.
  + Collider: do wykrywania kolizji z innymi obiektami.

**Zmienne**:

* + **speed** (float): określa prędkość poruszania się pocisku.
  + **amountOfDamage** (int): określa ilość obrażeń zadawanych wrogim jednostkom.
  + **audioExplosion** (AudioClip): dźwięk eksplozji, który zostanie odtworzony po starcie pocisku.

**Metody**:

* + **Start()**: metoda wywoływana raz, przy starcie gry. Odtwarza dźwięk eksplozji na pozycji pocisku.
  + **Update()**: metoda wywoływana co klatkę. Porusza pocisk w przód na podstawie wartości zmiennej **speed** i czasu delta.
  + **OnTriggerEnter(Collider other)**: metoda wywoływana, gdy pocisk wejdzie w kolizję z innym obiektem. Jeśli kolizją jest wrogie jednostka (o tagu "Enemy"), to zadaje jej obrażenia i niszczy pocisk.

**Użycie**:

1. Dołącz ten skrypt do obiektu reprezentującego pocisk.
2. Ustaw odpowiednie wartości dla zmiennych **speed**, **amountOfDamage** i **audioExplosion** w inspektorze skryptu.
3. Pocisk będzie automatycznie poruszał się do przodu i niszczył się po zderzeniu z wrogiem lub po upływie 1 sekundy.



**Skrypt CameraMovement**

**Opis**: Skrypt **CameraMovement** odpowiada za poruszanie kamerą w grze. Umożliwia przesuwanie kamery w różnych kierunkach przy użyciu klawiszy WSAD lub przesuwania kursora myszy do krawędzi ekranu. Pozwala również na przybliżanie i oddalanie kamery za pomocą kółka myszy.

**Używane komponenty Unity**:

* + Transform: do manipulacji pozycją kamery.

**Zmienne**:

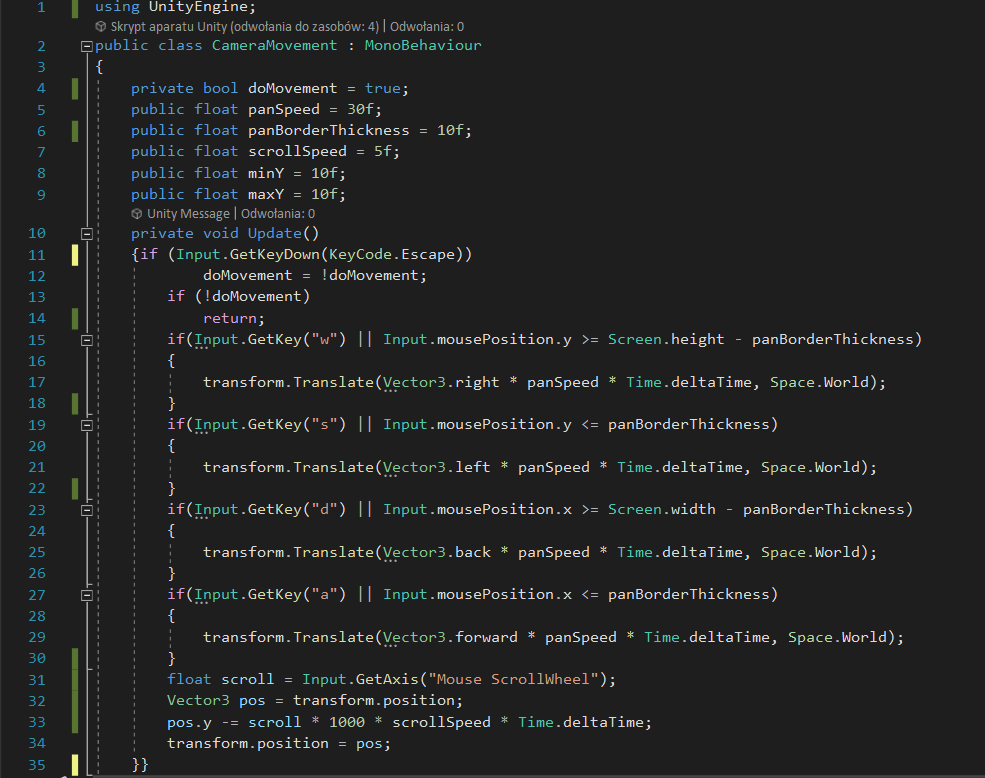
* + **doMovement** (bool): flaga określająca, czy ruch kamery jest włączony.
  + **panSpeed** (float): prędkość przesuwania kamery.
  + **panBorderThickness** (float): odległość od krawędzi ekranu, przy której kamera zacznie się poruszać.
  + **scrollSpeed** (float): prędkość przybliżania i oddalania kamery.
  + **minY** (float): minimalna wysokość kamery.
  + **maxY** (float): maksymalna wysokość kamery.

**Metody**:

* + **Update()**: metoda wywoływana co klatkę. Odczytuje wejście od użytkownika i wykonuje odpowiednie operacje przesunięcia kamery oraz przybliżania/oddalania.

**Użycie**:

1. Dołącz ten skrypt do obiektu reprezentującego kamerę.
2. Ustaw odpowiednie wartości dla zmiennych **panSpeed**, **panBorderThickness**, **scrollSpeed**, **minY** i **maxY** w inspektorze skryptu.
3. Uruchom grę i używaj klawiszy WSAD do przesuwania kamery w odpowiednie kierunki lub przesuwaj kursor myszy do krawędzi ekranu.



**Skrypt Enemy**

**Opis**: Skrypt **Enemy** reprezentuje wroga w grze. Po otrzymaniu obrażeń wywołuje metodę **Damaging()** z obiektu **KillEnemy**, aby zmniejszyć jego ilość życia.

**Używane komponenty Unity**:

* + Żadne.

**Zmienne**:

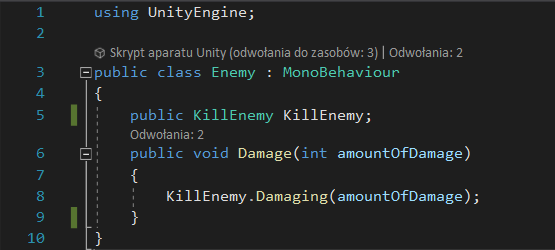
* + **KillEnemy** (KillEnemy): referencja do obiektu **KillEnemy**, który zarządza ilością życia wrogów.

**Metody**:

* + **Damage(int amountOfDamage)**: metoda wywoływana, gdy wróg otrzymuje obrażenia. Przekazuje ilość obrażeń do obiektu **KillEnemy**, który zarządza ilością życia wrogów.

**Użycie**:

1. Dołącz ten skrypt do obiektu reprezentującego wroga.
2. Przypisz odpowiedni obiekt **KillEnemy** do zmiennej **KillEnemy** w inspektorze skryptu.
3. Podczas gry, gdy wróg otrzymuje obrażenia, zostanie wywołana metoda **Damage()**, która przekaże ilość obrażeń do obiektu **KillEnemy**.



**Skrypt EnemyMovement**

**Opis**: Skrypt **EnemyMovement** jest odpowiedzialny za poruszanie się wroga w kierunku bazy gracza. Wykorzystuje komponent **NavMeshAgent** do nawigacji po środowisku.

**Używane komponenty Unity**:

* + **NavMeshAgent**: komponent umożliwiający poruszanie się obiektu wzdłuż nawigowalnej siatki.

**Zmienne**:

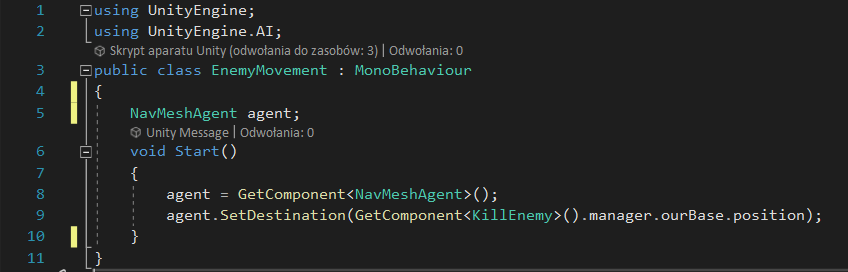
* + **agent** (NavMeshAgent): referencja do komponentu **NavMeshAgent** przypisanego do tego samego obiektu.

**Metody**:

* + **Start()**: metoda wywoływana przy starcie gry. Pobiera komponent **NavMeshAgent** przypisany do tego obiektu i ustawia docelowe miejsce poruszania się na pozycję bazy gracza, pobraną z komponentu **KillEnemy**.

**Użycie**:

1. Dołącz ten skrypt do obiektu reprezentującego wroga.
2. Upewnij się, że obiekt posiada komponent **NavMeshAgent**.
3. Przypisz obiekt **KillEnemy** zawierający referencję do bazy gracza do odpowiedniego komponentu **KillEnemy**.
4. Podczas gry, wróg będzie poruszał się w kierunku bazy gracza, korzystając z komponentu **NavMeshAgent** i ustawionego celu.



**Skrypt EnemySpawner**

**Opis**: Skrypt **EnemySpawner** jest odpowiedzialny za generowanie fal wrogów w określonych interwałach czasowych. Tworzy wrogów na podstawie danych zdefiniowanych w tablicy fal.

**Używane komponenty Unity**:

* + **Text**: komponent umożliwiający wyświetlanie tekstu.

**Zmienne**:

* + **EnemiesAlive** (int): liczba żywych wrogów.
  + **waves** (Wave[]): tablica fal wrogów.
  + **spawnPoint** (Transform): pozycja, na której będą pojawiać się wrogowie.
  + **timeBetweenWaves** (float): czas oczekiwania między falami.
  + **countdown** (float): odliczanie do kolejnej fali.
  + **waveCountdownText** (Text): komponent tekstowy wyświetlający odliczanie.
  + **manager** (Manager): referencja do obiektu zarządzającego.
  + **waveIndex** (int): indeks aktualnej fali.

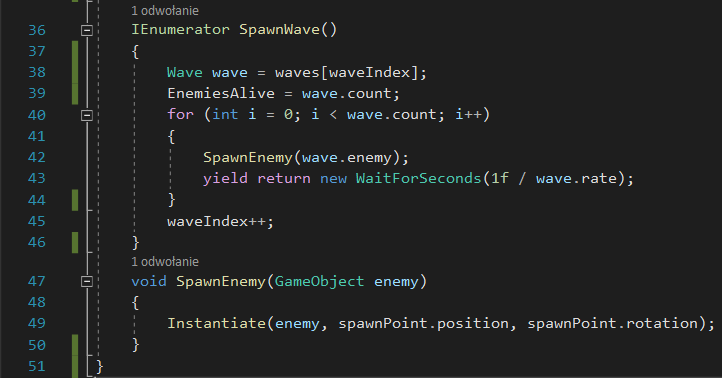
**Metody**:

* + **Update()**: metoda wywoływana co klatkę. Sprawdza warunki zakończenia fali wrogów oraz odliczanie czasu do następnej fali. Jeśli nie ma żywych wrogów, rozpoczyna nową falę.
  + **SpawnWave()**: metoda asynchroniczna generująca falę wrogów. Tworzy wrogów na podstawie danych z tablicy fal.
  + **SpawnEnemy()**: metoda tworząca pojedynczego wroga na podstawie podanego obiektu wroga.

**Użycie**:

1. Dołącz ten skrypt do obiektu, który będzie odpowiedzialny za generowanie wrogów.
2. Zdefiniuj tablicę **waves** zawierającą informacje o kolejnych falach wrogów.





**Skrypt EnemyTowerMovement**

**Opis**: Skrypt **EnemyTowerMovement** odpowiada za ruch wieży wroga. Wieża obraca się w kierunku najbliższej wieży gracza, gdy ta znajduje się w zasięgu. Skrypt również obsługuje przełączanie się pomiędzy wieżą gracza jako obecnym celem.

**Używane komponenty Unity**:

* + Żaden dodatkowy komponent nie jest używany w tym skrypcie.

**Zmienne**:

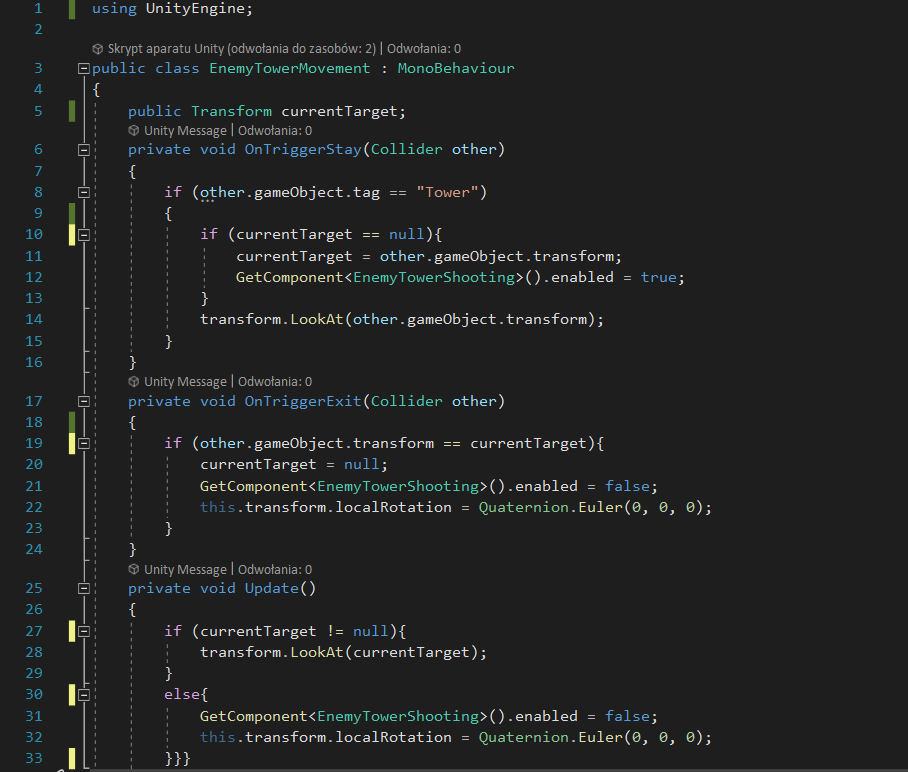
* + **currentTarget** (Transform): aktualnie wybrany cel, czyli wieża gracza.

**Metody**:

* + **OnTriggerStay(Collider other)**: wywoływana, gdy inny obiekt wchodzi w obszar kolizji. Jeśli to wieża gracza, ustawia ją jako obecny cel i włącza skrypt strzelania wroga.
  + **OnTriggerExit(Collider other)**: wywoływana, gdy inny obiekt opuszcza obszar kolizji. Jeśli to aktualny cel, resetuje cel, wyłącza skrypt strzelania wroga i przywraca rotację wieży wroga do pierwotnej wartości.
  + **Update()**: metoda wywoływana co klatkę. Jeśli jest ustawiony aktualny cel, wieża obraca się w jego kierunku. W przeciwnym razie wyłącza skrypt strzelania wroga i przywraca rotację wieży wroga do pierwotnej wartości.

**Użycie**:

1. Dołącz ten skrypt do obiektu reprezentującego wieżę wroga.
2. Ustaw odpowiednie tagi i kolizje, aby wieże gracza i wroga mogły wejść w obszar kolizji.
3. Skrypt automatycznie obsługuje ruch i obrót wieży w kierunku wieży gracza w zasięgu.
4. Skrypt również włącza i wyłącza skrypt strzelania wroga w zależności od obecności celu.
5. Możesz dostosować zasięg, rotację i inne parametry w zależności od potrzeb.



**Skrypt KillEnemy**

Opis: Skrypt KillEnemy jest odpowiedzialny za obsługę zdrowia i usunięcie wrogich jednostek w grze Tower Defense. Monitoruje aktualne zdrowie wroga, aktualizuje pasek zdrowia na interfejsie użytkownika, obsługuje zadawane obrażenia oraz usuwa jednostkę w przypadku zniszczenia. Dodatkowo, przy usunięciu wroga, skrypt generuje efekt eksplozji oraz odtwarza dźwięk eksplozji, zmniejsza liczbę aktywnych wrogów i zwiększa wartość pieniędzy gracza.

**Zmienne:**

* + public int worth: Określa wartość pieniężną, jaką gracz otrzymuje za zniszczenie wroga.
  + public float health: Określa aktualne zdrowie wroga.
  + public float startHealth: Określa początkowe zdrowie wroga.
  + public Manager manager: Referencja do obiektu klasy Manager.
  + public GameObject explosionPrefab: Prefab efektu eksplozji.
  + public AudioClip audioExplosion: Dźwięk eksplozji.
  + public Image healthBar: Komponent obrazu paska zdrowia na interfejsie użytkownika.

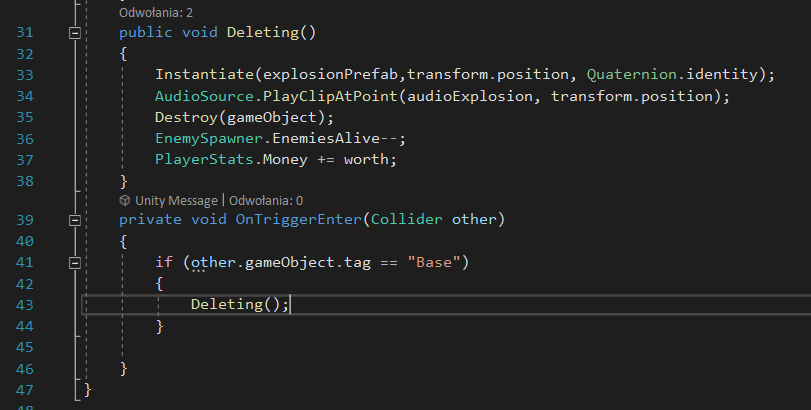
**Metody:**

* + void Awake(): Metoda inicjalizuje zdrowie wroga na wartość startową oraz wyszukuje obiekt klasy Manager w scenie.
  + public void Damaging(int amountOfDamage): Metoda obsługuje zadawanie obrażeń wrogowi. Redukuje zdrowie wroga o podaną wartość amountOfDamage. Aktualizuje pasek zdrowia na interfejsie użytkownika. Jeżeli zdrowie wroga spadnie do zera, wywołuje metodę Deleting().
  + public void Deleting(): Metoda usuwa wroga z gry. Tworzy efekt eksplozji z użyciem prefabu explosionPrefab i odtwarza dźwięk eksplozji audioExplosion. Niszczy obiekt wroga. Zmniejsza liczbę aktywnych wrogów w klasie EnemySpawner. Zwiększa wartość pieniędzy gracza w klasie PlayerStats.
  + private void OnTriggerEnter(Collider other): Metoda wywoływana, gdy wroga dotknie inny obiekt. Jeżeli dotknięty obiekt ma tag "Base", wywołuje metodę Deleting().

**Użycie:**

1. Do obiektu reprezentującego wroga należy dołączyć skrypt KillEnemy.
2. Na skrypcie można ustawić wartość pola worth, która określa ilość pieniędzy, jaką gracz otrzymuje za zniszczenie wroga.
3. Pola health i startHealth można ustawić na początkowe zdrowie wroga.
4. Skrypt wymaga obiektu klasy Manager w scenie. Należy przypisać referencję do tego obiektu do pola manager w skrypcie.
5. Do pola explosionPrefab należy przypisać prefab efektu eksplozji, który zostanie wygenerowany przy usunięciu wroga.
6. Do pola audioExplosion należy przypisać dźwięk eksplozji, który zostanie odtworzony przy usunięciu wroga.
7. Pole healthBar reprezentuje komponent obrazu paska zdrowia na interfejsie użytkownika. Skrypt automatycznie aktualizuje ten pasek w zależności od aktualnego zdrowia wroga.





**Skrypt KillTower**

**Opis**: Skrypt **KillTower** odpowiada za obsługę zdrowia wieży gracza i jej usunięcie po zniszczeniu. Gdy zdrowie wieży spada poniżej zera, inicjowane jest zniszczenie, które obejmuje efekt eksplozji, odtworzenie dźwięku eksplozji oraz wyłączenie komponentów wieży.

**Używane komponenty Unity**:

* + **Manager**: odwołanie do skryptu Manager, który zarządza grą.
  + **GameObject**: referencje do obiektów w Unity.

**Zmienne**:

* + **healthTower** (int): aktualne zdrowie wieży.
  + **manager** (Manager): odwołanie do skryptu Manager.
  + **TowerBase** (GameObject): referencja do wieży bazowej, do której wieża gracza jest przypisana.
  + **explosionPrefab** (GameObject): prefabrykat efektu eksplozji.
  + **audioExplosion** (AudioClip): dźwięk eksplozji.

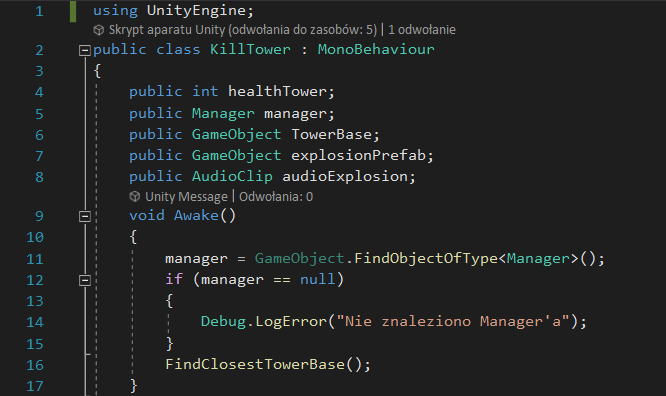
**Metody**:

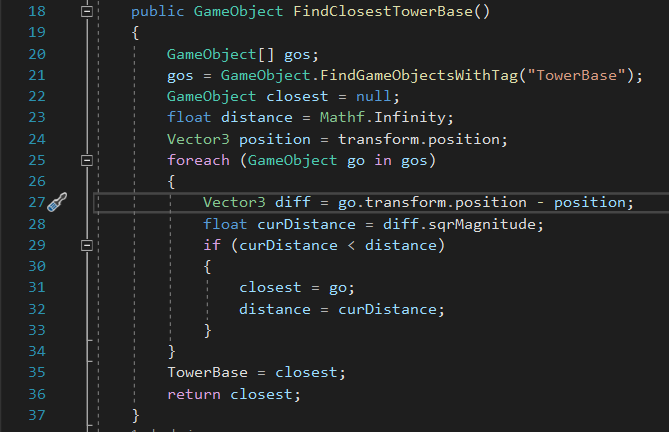
* + **Awake()**: wywoływana przy inicjalizacji skryptu. Wyszukuje obiekt Manager w scenie i znajduje najbliższą wieżę bazową dla wieży gracza.
  + **FindClosestTowerBase()**: znajduje najbliższą wieżę bazową na podstawie tagu "TowerBase".
  + **Damaging(int amountOfDamage)**: redukuje zdrowie wieży o określoną wartość. Jeśli zdrowie spada poniżej zera, inicjowane jest zniszczenie wieży.
  + **Deleting()**: inicjuje zniszczenie wieży, obejmujące efekt eksplozji, odtworzenie dźwięku eksplozji oraz wyłączenie komponentów wieży. Wieża gracza zostaje usunięta z gry, a komponenty wieży bazowej są włączane, aby umożliwić ponowne zbudowanie wieży na tym miejscu.

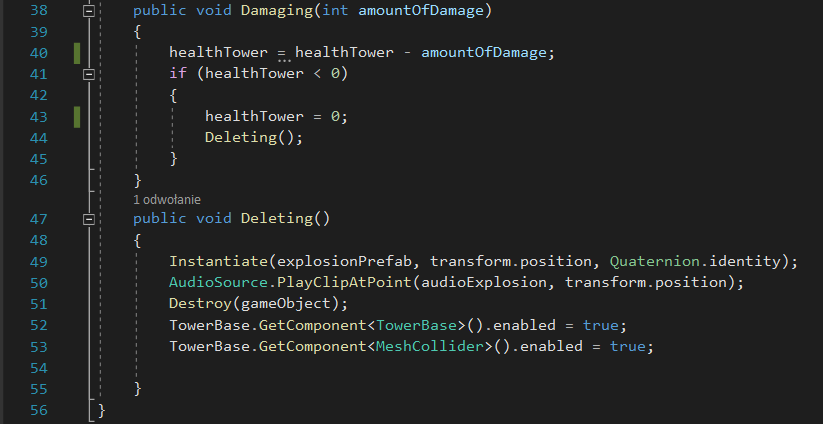
**Metody Unity**:

* + **Instantiate()**: tworzy instancję prefabrykatu efektu eksplozji na określonej pozycji i zadanym obrocie.
  + **Destroy()**: niszczy obiekt wieży gracza.
  + **GetComponent()**: pobiera referencję do komponentu TowerBase i MeshCollider wieży bazowej.

**Użycie**: Aby skrypt działał poprawnie, upewnij się, że przypisane są odpowiednie prefabrykaty i komponenty w Unity. Skrypt można podłączyć do obiektu reprezentującego wieżę gracza.







**Skrypt LeavingMap**

**Opis**: Skrypt "LeavingMap" jest odpowiedzialny za obsługę sytuacji, gdy obiekt opuszcza obszar wyznaczony przez trigger. W momencie, gdy obiekt opuszcza trigger, kamera zostaje przesunięta do określonej pozycji, a tekst zostaje aktywowany. Po pewnym czasie, tekst zostaje automatycznie ukryty.

**Używane komponenty Unity**:

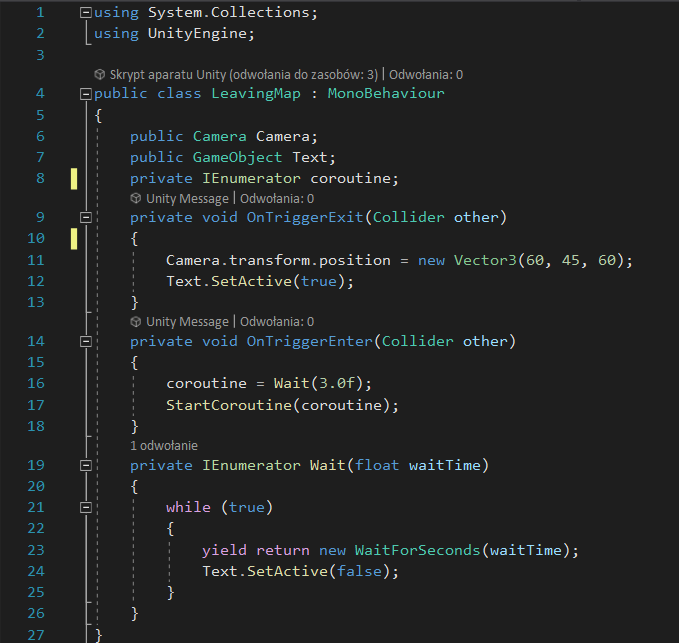
* + **Camera**: referencja do kamery, która zostanie przesunięta.
  + **Text**: referencja do obiektu tekstu, który będzie aktywowany/ukryty.

**Metody Unity**:

* + **OnTriggerExit(Collider other)**: wywoływana, gdy obiekt opuszcza trigger.
  + **OnTriggerEnter(Collider other)**: wywoływana, gdy obiekt wchodzi w trigger.
  + **StartCoroutine(IEnumerator routine)**: uruchamia korutynę (czasowo opóźnione wykonywanie kodu).
  + **WaitForSeconds(float seconds)**: tworzy obiekt oczekujący przez określony czas.
  + **SetActive(bool value)**: aktywuje/ukrywa obiekt.

**Użycie**:

1. Dodaj skrypt do obiektu posiadającego trigger.
2. Przypisz odpowiednie komponenty i obiekty w Unity.
3. Dostosuj pozycję kamery oraz czas oczekiwania na tekst według potrzeb.



**Skrypt LevelSelector**

**Opis**: Skrypt "LevelSelector" jest odpowiedzialny za wybieranie poziomu gry. W zależności od postępu gracza, przyciski poziomów są aktywne lub nieaktywne. Po wybraniu poziomu, skrypt uruchamia efekt przejścia do wybranego poziomu.

**Używane komponenty Unity**:

* + **SceneFader**: referencja do skryptu obsługującego efekt przejścia między scenami.
  + **Button[] levelButtons**: tablica przycisków reprezentujących poziomy gry.

**Metody Unity**:

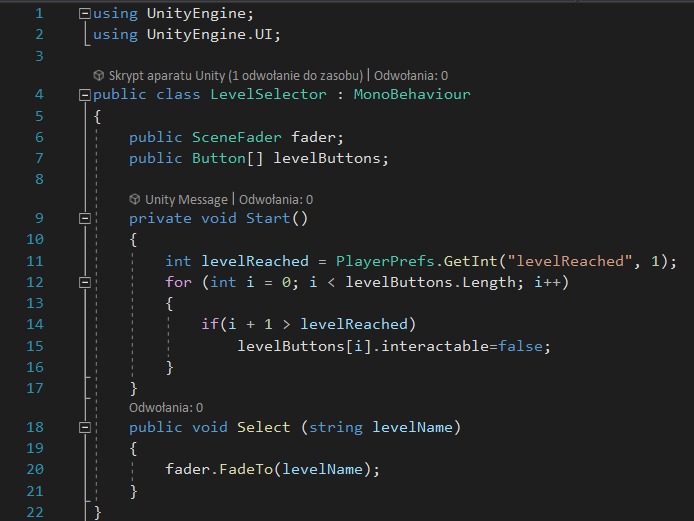
* + **Start()**: wywoływana raz podczas inicjalizacji skryptu.
  + **Select(string levelName)**: wywoływana po wybraniu konkretnego poziomu.

**Metody:**

* + **PlayerPrefs.GetInt(string key, int defaultValue)**: pobiera wartość zapisaną w PlayerPrefs o określonym kluczu. Jeśli wartość nie istnieje, zwraca wartość domyślną.
  + **Button.interactable**: określa, czy przycisk jest interaktywny (klikalny) czy nie.

**Użycie**:

1. Dodaj skrypt do obiektu będącego selektorem poziomów.
2. Przypisz referencje do SceneFader oraz przycisków poziomów w Unity.
3. Dostosuj kod w metodzie Start() w celu określenia dostępności przycisków poziomów w zależności od postępu gracza.
4. Dostosuj kod w metodzie Select(string levelName) w celu uruchomienia efektu przejścia do wybranego poziomu.



**Skrypt LivesUI**

**Opis**: Skrypt "LivesUI" jest odpowiedzialny za aktualizację wyświetlanego tekstu dotyczącego liczby żyć gracza. Tekst jest pobierany z komponentu Text i aktualizowany w każdej klatce gry na podstawie wartości przechowywanej w PlayerStats.Lives.

**Używane komponenty Unity**:

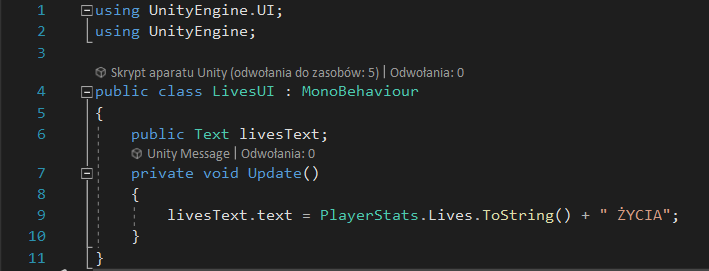
* + **Text livesText**: referencja do komponentu Text, w którym będzie wyświetlana liczba żyć.

**Metody Unity**:

* + **Update()**: wywoływana co klatkę gry.

**Użycie**:

1. Dodaj skrypt do obiektu zawierającego komponent Text, w którym ma być wyświetlana liczba żyć.
2. Przypisz referencję do komponentu Text w Unity.
3. W PlayerStats lub innej klasie odpowiedzialnej za przechowywanie liczby żyć gracza, utwórz publiczne statyczne pole "Lives", które będzie przechowywać aktualną wartość liczby żyć.
4. Skrypt "LivesUI" automatycznie aktualizuje tekst na podstawie wartości przechowywanej w PlayerStats.Lives.



**Skrypt MainMenu**

**Opis**: Skrypt "MainMenu" odpowiada za obsługę głównego menu gry. Udostępnia trzy metody:

* + **Play()**: Przełącza do wybranego poziomu za pomocą obiektu SceneFader.
  + **Settings()**: Wywołuje logowanie informacji o wybraniu ustawień.
  + **Quit()**: Wywołuje logowanie informacji o zamknięciu aplikacji.

**Używane komponenty Unity**:

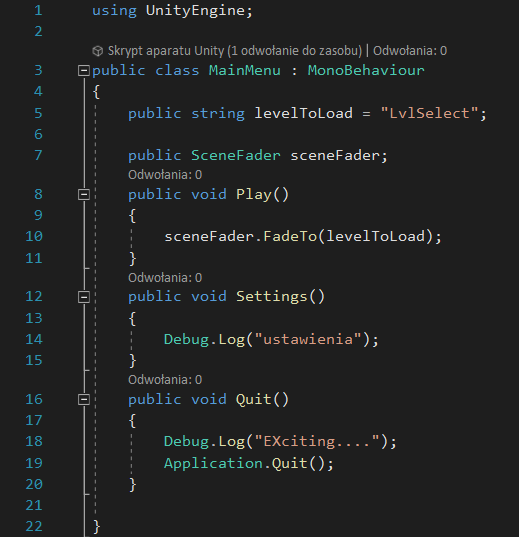
* + **string levelToLoad**: nazwa poziomu, do którego ma zostać przełączona gra.
  + **SceneFader sceneFader**: referencja do obiektu SceneFader, który będzie używany do przejścia do wybranego poziomu.

**Metody Unity**:

* + **Play()**: wywoływana po kliknięciu przycisku "Play".
  + **Settings()**: wywoływana po kliknięciu przycisku "Settings".
  + **Quit()**: wywoływana po kliknięciu przycisku "Quit".

**Użycie**:

1. Dodaj skrypt do obiektu reprezentującego główne menu gry.
2. Przypisz odpowiednie wartości do pól levelToLoad i sceneFader w Unity.
3. Skrypt jest gotowy do obsługi kliknięć przycisków Play, Settings i Quit w głównym menu gry.



**Skrypt MoneyUI**

**Opis**: Skrypt "MoneyUI" odpowiada za aktualizację wyświetlanego stanu pieniędzy w interfejsie użytkownika. Wykorzystuje komponent Text do wyświetlania aktualnej ilości pieniędzy gracza.

**Używane komponenty Unity**:

* + **Text moneyText**: referencja do komponentu Text, który wyświetla stan pieniędzy.

**Metody Unity**:

* + **Update()**: wywoływana co klatkę, aktualizuje wyświetlaną wartość pieniędzy na podstawie wartości z PlayerStats.Money.

**Użycie**:

1. Dodaj skrypt do obiektu, który ma wyświetlać aktualny stan pieniędzy.
2. Przypisz komponent Text, który będzie wyświetlał wartość pieniędzy, do pola moneyText w Unity.



**Skrypt PauseMenu**

**Opis**: Skrypt "PauseMenu" odpowiada za obsługę menu pauzy w grze. Umożliwia włączanie i wyłączanie menu pauzy poprzez naciśnięcie klawisza Escape lub P. Dodatkowo, umożliwia powrót do menu głównego.

**Używane komponenty Unity**:

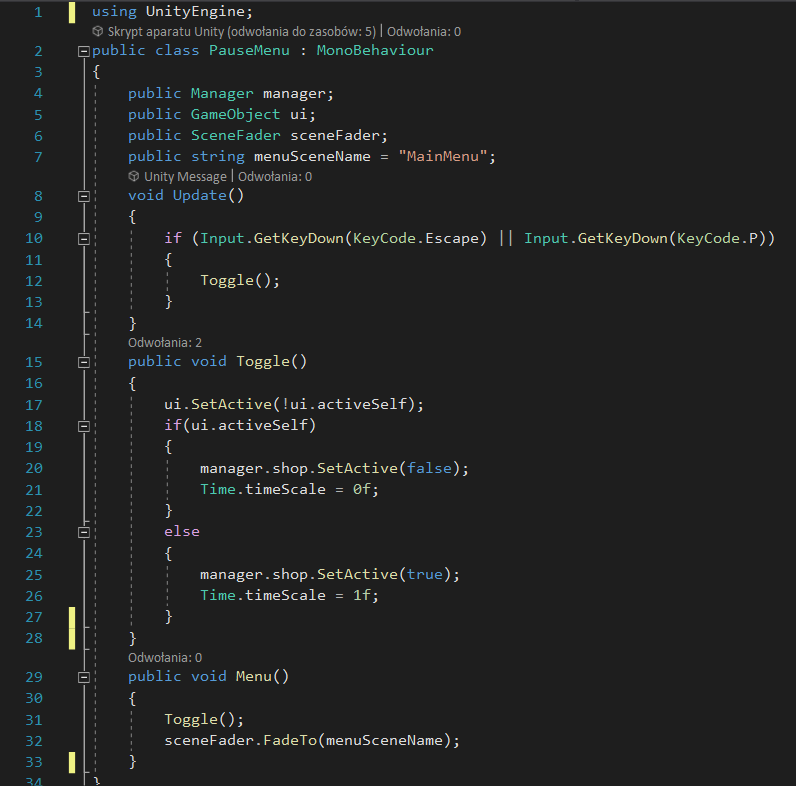
* + **GameObject ui**: referencja do obiektu zawierającego menu pauzy.
  + **Manager manager**: referencja do komponentu Manager, który kontroluje inny aspekt gry.
  + **SceneFader sceneFader**: referencja do komponentu SceneFader, który obsługuje przejście do innej sceny.
  + **string menuSceneName**: nazwa sceny menu głównego.

**Metody Unity**:

* + **Update()**: wywoływana co klatkę, sprawdza naciśnięcie klawisza Escape lub P i wywołuje metodę Toggle().
  + **Toggle()**: przełącza stan menu pauzy (włączanie/wyłączanie) oraz kontroluje czas gry i stan sklepu w zależności od aktywności menu pauzy.
  + **Menu()**: wywoływana po naciśnięciu przycisku powrotu do menu, wyłącza menu pauzy i inicjuje przejście do menu głównego za pomocą komponentu SceneFader.

**Użycie**:

1. Dodaj skrypt do obiektu, który ma obsługiwać menu pauzy.
2. Przypisz odpowiednie obiekty i komponenty do pól w Unity, takie jak GameObject ui, Manager manager, SceneFader sceneFader i nazwę sceny menu głównego menuSceneName.



**Skrypt PlayerStats**

**Opis**: Skrypt "PlayerStats" przechowuje statystyki gracza, takie jak ilość pieniędzy (Money) i ilość żyć (Lives). Po uruchomieniu gry, wartości początkowe są przypisywane do odpowiednich zmiennych.

**Używane komponenty Unity**:

* + **int startMoney**: wartość początkowa dla ilości pieniędzy.
  + **int startLives**: wartość początkowa dla ilości żyć.

**Zmienne statyczne**:

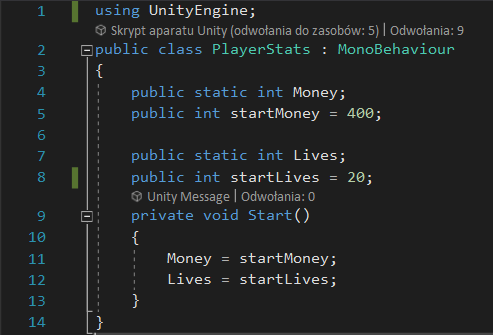
* + **int Money**: aktualna ilość pieniędzy, dostępna jako zmienna statyczna.
  + **int Lives**: aktualna ilość żyć, dostępna jako zmienna statyczna.

**Metody Unity**:

* + **Start()**: wywoływana przy uruchomieniu gry, przypisuje wartości początkowe do zmiennych Money i Lives.

**Użycie**:

1. Dodaj skrypt do obiektu gracza lub innego odpowiedniego obiektu.
2. Przypisz odpowiednie wartości początkowe do pól startMoney i startLives w Unity.



**Skrypt SceneFader**

**Opis**: Skrypt "SceneFader" odpowiada za płynne przenikanie (przezroczystość) między scenami gry. Wykorzystuje komponent "Image" do kontrolowania widoczności obrazu, który jest stosowany na pełnym ekranie.

**Używane komponenty Unity**:

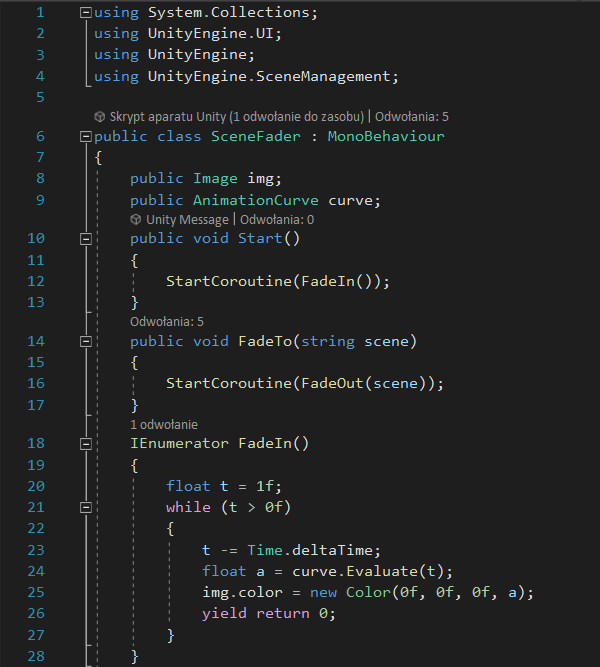
* + **Image img**: obrazek używany do przenikania między scenami.
  + **AnimationCurve curve**: krzywa animacji kontrolująca przejście między przezroczystością.

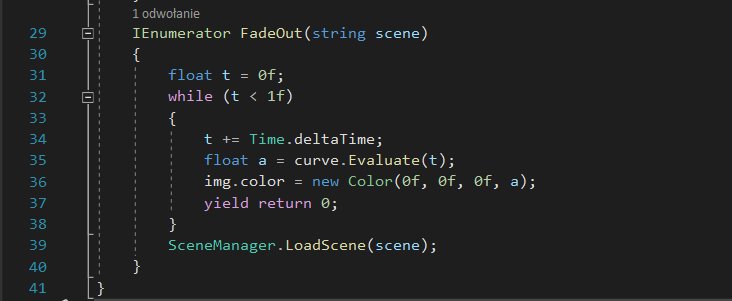
**Metody**:

* + **Start()**: wywoływana przy uruchomieniu skryptu, rozpoczyna płynne przenikanie do sceny.
  + **FadeTo(string scene)**: rozpoczyna płynne przenikanie do określonej sceny.
  + **FadeIn()**: realizuje płynne przenikanie na początku, zaciemniając ekran.
  + **FadeOut(string scene)**: realizuje płynne przenikanie na końcu, rozjaśniając ekran, a następnie ładowanie nowej sceny.

**Użycie**:

1. Dołącz skrypt "SceneFader" do obiektu w scenie.
2. Przypisz obrazek (komponent "Image") do pola "img" w Unity.
3. Dostosuj krzywą animacji "curve" w zależności od preferencji.





**Skrypt Shop**

**Opis**: Skrypt "Shop" reprezentuje sklep, w którym gracz może wybierać i kupować różne typy wież obronnych. Każdy rodzaj wieży ma swoje parametry i koszt. Skrypt pozwala na wybór odpowiedniej wieży i przekazuje ją do menedżera budowy (BuildManager), który zajmuje się budową wież.

**Używane komponenty Unity**:

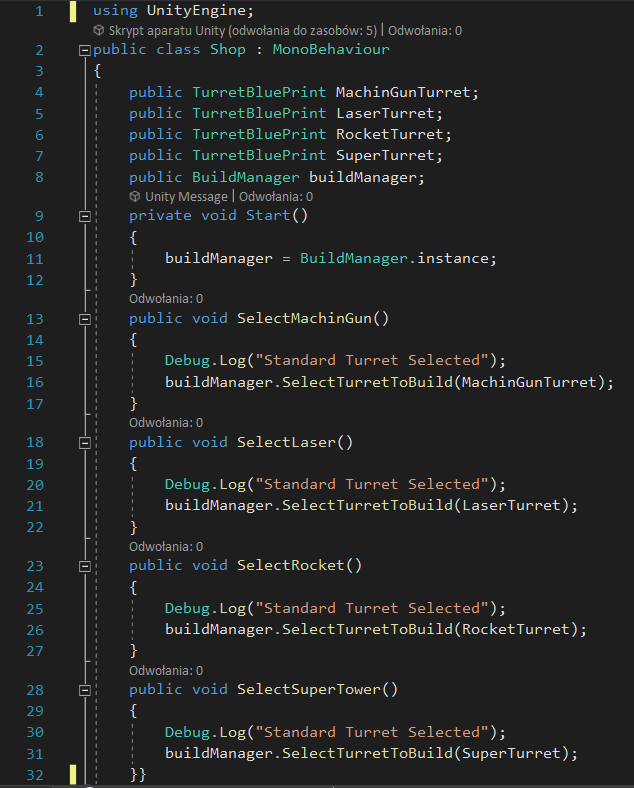
* + **TurretBluePrint machineGunTurret**: wzorzec (blueprint) wieży typu Machine Gun.
  + **TurretBluePrint laserTurret**: wzorzec (blueprint) wieży typu Laser.
  + **TurretBluePrint rocketTurret**: wzorzec (blueprint) wieży typu Rocket.
  + **TurretBluePrint superTurret**: wzorzec (blueprint) wieży typu Super.
  + **BuildManager buildManager**: referencja do menedżera budowy.

**Metody**:

* + **Start()**: wywoływana przy uruchomieniu skryptu, inicjalizuje referencję do menedżera budowy.
  + **SelectMachineGun()**: wywoływana po wybraniu wieży typu Machine Gun, przekazuje wzorzec tej wieży do menedżera budowy.
  + **SelectLaser()**: wywoływana po wybraniu wieży typu Laser, przekazuje wzorzec tej wieży do menedżera budowy.
  + **SelectRocket()**: wywoływana po wybraniu wieży typu Rocket, przekazuje wzorzec tej wieży do menedżera budowy.
  + **SelectSuperTower()**: wywoływana po wybraniu wieży typu Super, przekazuje wzorzec tej wieży do menedżera budowy.

**Użycie**:

1. Dołącz skrypt "Shop" do obiektu reprezentującego sklep w scenie.
2. Przypisz wzorce (blueprinty) wież obronnych do odpowiednich pól w Unity.
3. Upewnij się, że w scenie istnieje również obiekt z komponentem "BuildManager".
4. Skonfiguruj przyciski lub inne elementy interfejsu użytkownika w celu wywołania odpowiednich metod wyboru wież.



**Skrypt Tower**

**Opis**: Skrypt "Tower" reprezentuje wieżę obronną. Po otrzymaniu obrażeń, wieża przekazuje je do skryptu "KillTower" w celu zaktualizowania jej stanu zdrowia.

**Używane komponenty Unity**:

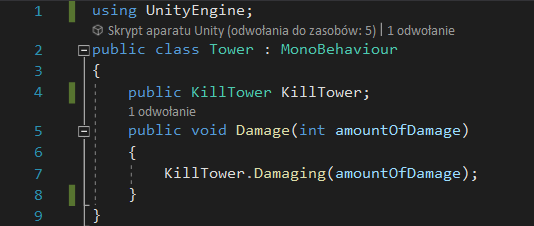
* + **KillTower killTower**: skrypt odpowiedzialny za zarządzanie zdrowiem wieży.

**Metody**:

* + **Damage(int amountOfDamage)**: wywoływana w przypadku otrzymania obrażeń. Przekazuje ilość zadanych obrażeń do skryptu "KillTower".

**Użycie**:

1. Dołącz skrypt "Tower" do obiektu reprezentującego wieżę obronną w scenie.
2. Przypisz skrypt "KillTower" do pola "killTower" w Unity.
3. Wywołaj metodę "Damage" na obiekcie "Tower", przekazując jako argument ilość zadanych obrażeń.



**Skrypt TowerBase**

**Opis**: Skrypt "TowerBase" reprezentuje podstawę wieży, na której można zbudować wieżę obronną. Pozwala na interakcję z graczem, taką jak umieszczanie wież na podstawie.

**Używane komponenty Unity**:

* + **Color hoverColor**: kolor podświetlenia podstawy wieży, gdy gracz najedzie na nią kursorem.
  + **Color notEnoughMoneyColor**: kolor podświetlenia podstawy wieży, gdy gracz nie ma wystarczającej ilości pieniędzy na zbudowanie wieży.
  + **Vector3 positionOffset**: przesunięcie pozycji budowania wieży względem pozycji podstawy wieży.
  + **GameObject turret** (opcjonalny): obecna wieża na podstawie, jeśli istnieje.

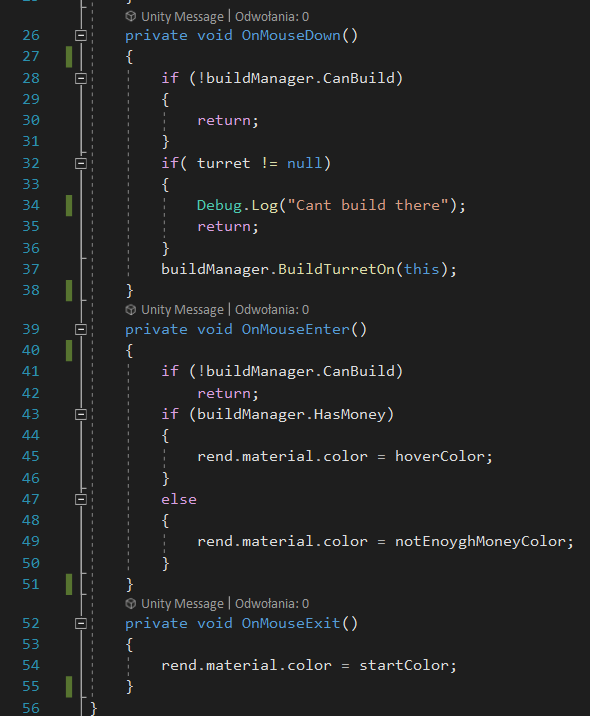
**Metody**:

* + **GetBuildPosition()**: zwraca pozycję budowania wieży na podstawie, uwzględniając przesunięcie.
  + **OnMouseDown()**: wywoływana po kliknięciu na podstawę wieży. Sprawdza, czy można zbudować wieżę i czy na podstawie już się nie znajduje inna wieża. Jeśli tak, wywołuje metodę "BuildTurretOn" z obiektu "BuildManager".
  + **OnMouseEnter()**: wywoływana, gdy kursor najedzie na podstawę wieży. Sprawdza, czy można zbudować wieżę i zmienia kolor podświetlenia na odpowiedni.
  + **OnMouseExit()**: wywoływana, gdy kursor opuści podstawę wieży. Przywraca domyślny kolor podstawy.

**Użycie**:

1. Dołącz skrypt "TowerBase" do obiektu reprezentującego podstawę wieży w scenie.
2. W polach "hoverColor" i "notEnoughMoneyColor" ustal odpowiednie kolory.
3. Opcjonalnie, przypisz obiekt wieży (jeśli istnieje) do pola "turret" w skrypcie "TowerBase".
4. Upewnij się, że skrypt "BuildManager" jest poprawnie skonfigurowany i dostępny w scenie.
5. Uruchom grę i przetestuj interakcję z podstawą wieży. Sprawdź, czy podświetlenie zmienia się odpowiednio w zależności od możliwości budowy i dostępności środków finansowych. Upewnij się, że wieże są budowane tylko wtedy, gdy jest to możliwe, a na podstawie nie znajduje się już inna wieża.
6. Możesz dostosować skrypt "TowerBase" poprzez dodanie dodatkowych funkcjonalności, takich jak animacje, dźwięki czy efekty wizualne, w zależności od potrzeb i projektu gry.





**Skrypt TowerMovement**

**Opis**: Skrypt “TowerMovement” odpowiada za poruszanie się wieży w stronę wroga. Jeśli wrogie obiekty znajdują się w zasięgu wieży, ta obraca się w kierunku najbliższego wroga i aktywuje skrypt "TowerShooting" do strzelania.

**Używane komponenty Unity**:

* + **Transform currentTarget**: aktualny cel wieży.

**Metody**:

* + **OnTriggerStay(Collider other)**: wywoływana, gdy obiekt wchodzi w obszar kolidera wieży. Jeśli obiekt ma tag "Enemy" i nie ma jeszcze przypisanego aktualnego celu, ustawia go jako aktualny cel i aktywuje skrypt "TowerShooting". Wieża obraca się w kierunku wroga.
  + **OnTriggerExit(Collider other)**: wywoływana, gdy obiekt opuszcza obszar kolidera wieży. Jeśli obiekt jest aktualnym celem, usuwa cel i wyłącza skrypt "TowerShooting".
  + **Update()**: wywoływana co klatkę. Jeśli istnieje aktualny cel, wieża obraca się w jego kierunku. W przeciwnym razie wyłącza skrypt "TowerShooting".

**Użycie**:

1. Dołącz skrypt "TowerMovement" do obiektu reprezentującego wieżę obronną w scenie.
2. Przypisz odpowiednie wartości do pól w Unity.
3. Upewnij się, że skrypt "TowerShooting" jest również dołączony do wieży.





**Skrypt TowerShooting**

**Opis**: Skrypt "TowerShooting" odpowiada za strzelanie wieży. Wieża cyklicznie wywołuje funkcję "Fire", która tworzy instancję prefabrykatu pocisku w pozycji i rotacji określonej przez "bulletPosition".

**Używane komponenty Unity**:

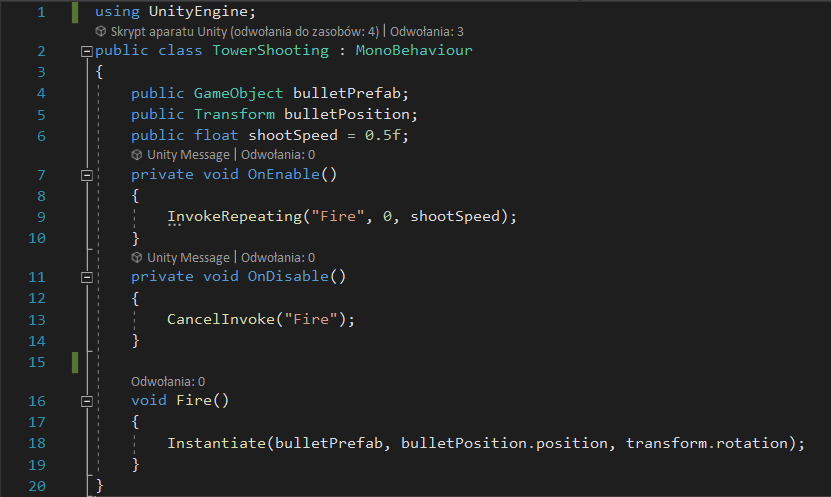
* + **GameObject bulletPrefab**: prefabrykat pocisku, który zostanie wystrzelony.
  + **Transform bulletPosition**: pozycja, z której zostanie wystrzelony pocisk.
  + **float shootSpeed**: czas między kolejnymi strzałami.

**Metody**:

* + **OnEnable()**: wywoływana, gdy skrypt zostaje włączony. Powoduje cykliczne wywoływanie funkcji "Fire" z określoną częstotliwością, zaczynając od zera.
  + **OnDisable()**: wywoływana, gdy skrypt zostaje wyłączony. Anuluje cykliczne wywoływanie funkcji "Fire".
  + **Fire()**: wywoływana cyklicznie z określoną częstotliwością. Tworzy instancję prefabrykatu pocisku w pozycji i rotacji określonej przez "bulletPosition".

**Użycie**:

1. Dołącz skrypt "TowerShooting" do obiektu reprezentującego wieżę obronną w scenie.
2. Przypisz odpowiednie prefabrykaty i transformacje do pól w Unity.
3. Ustaw wartość "shootSpeed" dla określenia częstotliwości strzałów.



**Skrypt TurretBluePrint**

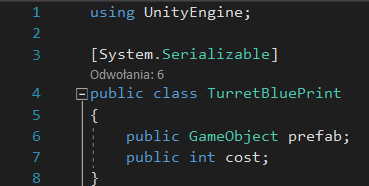
**Opis**: Klasa "TurretBluePrint" reprezentuje konfigurację wieży (turret) w grze. Zawiera informacje o prefabrykacie (prefab) wieży oraz jej koszcie (cost).

**Pola**:

* + **GameObject prefab**: prefabrykat (prefab) wieży, który będzie budowany w grze.
  + **int cost**: koszt budowy wieży.

**Użycie**:

1. Twórz obiekty typu "TurretBluePrint" w celu zdefiniowania różnych konfiguracji wież w grze.
2. Przypisz odpowiednie prefabrykaty i koszty do pól w Unity dla każdej konfiguracji.
3. Wykorzystuj obiekty "TurretBluePrint" w innych skryptach, takich jak "Shop" lub "BuildManager", do budowy wież w grze.



**Skrypt Wave**

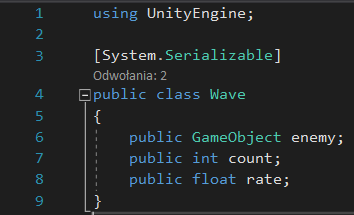
**Opis**: Klasa "Wave" reprezentuje falę w grze, która składa się z określonej liczby wrogów (enemy) i częstości ich generowania (rate).

**Pola**:

* + **GameObject enemy**: prefabrykat (prefab) wroga, który będzie generowany w danej fali.
  + **int count**: liczba wrogów w danej fali.
  + **float rate**: częstość generowania wrogów w danej fali (liczba wrogów generowanych na sekundę).

**Użycie**:

1. Twórz obiekty typu "Wave" w celu zdefiniowania kolejnych fal w grze.
2. Przypisz odpowiednie prefabrykaty wrogów, liczbę wrogów i częstość generowania do pól w Unity dla każdej fali.
3. Wykorzystuj obiekty "Wave" w innych skryptach, takich jak "EnemySpawner", do generowania kolejnych fal wrogów w grze.



**4.Grafika i dźwięk**

**Grafika:**

1. Grafiki wykorzystane w grze zostały pozyskane ze strony Asset Store (assetstore.unity.com). Zasoby graficzne były dostępne zgodnie z licencją i zostały wykorzystane w grze zgodnie z ich warunkami użytkowania. Grafiki zostały wykonane w taki sposób, aby przenosiły nas chociaż częściowo do świata innych planet. Styl graficzny został starannie dopasowany, aby tworzyć unikalną atmosferę i wizualne doświadczenie dla gracza.

**Dźwięk:**

1. Muzyka wykorzystana w grze również pochodzi z Asset Store (assetstore.unity.com) i została użyta zgodnie z jej licencją. Dźwięki zostały dobrane tak, aby współgrały z klimatem gry i wzbogacały jej doświadczenie dźwiękowe. Wszystkie pliki dźwiękowe zostały zapisane w formacie WAV, aby zapewnić wysoką jakość dźwięku i kompatybilność z platformami, na których gra jest uruchamiana.

**5.Silniki i technologie**

**Wersja Edytora**

1. 2021.3.18f1

**Wykorzystane biblioteki i frameworki:**

1. *UnityEngine:* Jest to biblioteka wchodząca w skład silnika Unity, który jest popularnym środowiskiem do tworzenia gier. Biblioteka ta zawiera wiele kluczowych klas i funkcji potrzebnych do tworzenia, zarządzania i renderowania obiektów w grze, zarządzania fizyką, dźwiękiem, animacją, grafiką itp.
2. *System.Collections:* Ta biblioteka zawiera klasy i interfejsy do obsługi kolekcji danych, takich jak listy, tablice, słowniki itp. Udostępnia różne algorytmy i metody manipulacji danymi, takie jak sortowanie, przeszukiwanie, filtrowanie, iteracja itp.
3. *UnityEngine.UI:* Jest to biblioteka, która zawiera klasy i narzędzia do tworzenia interfejsu użytkownika (UI) w grze. Udostępnia komponenty UI, takie jak przyciski, tekst, obrazy, suwaki, pola tekstowe itp., które można używać do projektowania interaktywnych i responsywnych elementów UI.
4. *UnityEngine.SceneManagement:* Ta biblioteka oferuje klasy i metody do zarządzania scenami w grze. Umożliwia przełączanie się między różnymi scenami, ładowanie i usuwanie scen, kontrolowanie kolejności ładowania itp. Jest używana do organizowania i zarządzania różnymi ekranami i poziomami w grze.
5. *UnityEngine.AI:* Biblioteka ta dostarcza narzędzia i funkcje związane z sztuczną inteligencją (AI) w grze. Oferuje algorytmy nawigacji, takie jak nawigacja agenta, systemy kolizji, algorytmy przeszukiwania ścieżki itp., które pozwalają na tworzenie inteligentnego zachowania i poruszania się wirtualnych postaci w grze.
6. *[System.Serializable]:* Jest to atrybut/system dostarczony przez .NET Framework, który informuje środowisko, że klasa lub struktura może być serializowana. Pozwala na przechowywanie i odtwarzanie danych obiektu w formacie tekstowym lub binarnym. Jest często stosowany w kontekście tworzenia i obsługi zapisu stanu gry, zarządzania danymi konfiguracyjnymi itp.

**Funkcje**

1. Funkcja "Navigation Bake" w Unity jest narzędziem, które umożliwia generowanie danych nawigacyjnych dla systemu nawigacji w grach. Pozwala na automatyczne tworzenie ścieżek i obszarów dostępnych dla agentów nawigacyjnych w trójwymiarowym środowisku gry.

**Języki programowania**

1. C#