SPRAWOZDANIE Z UKOŃCZONEGO PROJEKTU

(2020-2021)

Temat projektu: Gra stworzona w silniku UNITY

Wydział: EEIA

Kierunek: Informatyka

Semestr: 5

Grupa dziekańska: TI04

Zajęcia: Projekt Kompetencyjny

Skład grupy:

Krzysztof Pijanowski, 222540

Łukasz Szuligowski, 222589

Tomasz Wysocki, 222619

Spis treści

[1. Cele projektu 2](#_Toc61972778)

[2. Założenia i zawarte mechaniki w grze 2](#_Toc61972779)

[2.1. MAPA 3](#_Toc61972780)

[2.2. PRZECIWNICY 8](#_Toc61972781)

[2.3. Spawnpoint/Checkpoint 13](#_Toc61972782)

[2.4. Dynamiczny świat 14](#_Toc61972783)

[2.5. Główna postać i mechanika ruchu 19](#_Toc61972784)

[2.6. Animacje postaci, monet i innych elementów 20](#_Toc61972785)

[2.7. Dodanie dźwięków i dialogów 21](#_Toc61972786)

[2.8. Napisy pomagające graczowi 21](#_Toc61972787)

[2.9. Poboczna postać (ptak) 22](#_Toc61972788)

[2.10. Menu przed grą 23](#_Toc61972789)

[2.11. Kamera i tło 24](#_Toc61972790)

[3. Przykładowe skrypty gry 25](#_Toc61972791)

[3.1. Skrypt dźwiękowy 25](#_Toc61972792)

[3.2. Skrypt teleportujący 27](#_Toc61972793)

[3.3. Skrypt wyświetlający tekst 28](#_Toc61972794)

[3.4. Skrypt zliczający monety 29](#_Toc61972795)

[3.5. Skrypt platformy 30](#_Toc61972796)

[3.6. Skrypt spadających kolców 32](#_Toc61972797)

[3.7. Skrypt dla drzwi 34](#_Toc61972798)

[3.8. Skrypt menu 36](#_Toc61972799)

[3.9. Rotacja obiektu 37](#_Toc61972800)

[3.10. Skrypt do poruszania graczem 38](#_Toc61972801)

[4. Problemy, plany rozbudowy, opinia oraz podsumowanie projektu 41](#_Toc61972802)

# 1. Cele projektu

Głównym celem naszego projektu było stworzenie gry (platformówki) na silniku Unity. Gra ta powinna zawierać podstawowe mechaniki, które charakteryzują ww. gatunek gier. Skupiliśmy się na stworzeniu jednego poziomu gry oraz zawarciu w nim jak największej ilości różnorodnych elementów. Dopilnowaliśmy też, by każdy gracz, był kierowany i otrzymał podpowiedzi (komunikaty co klikać itp.).

# 2. Założenia i zawarte mechaniki w grze

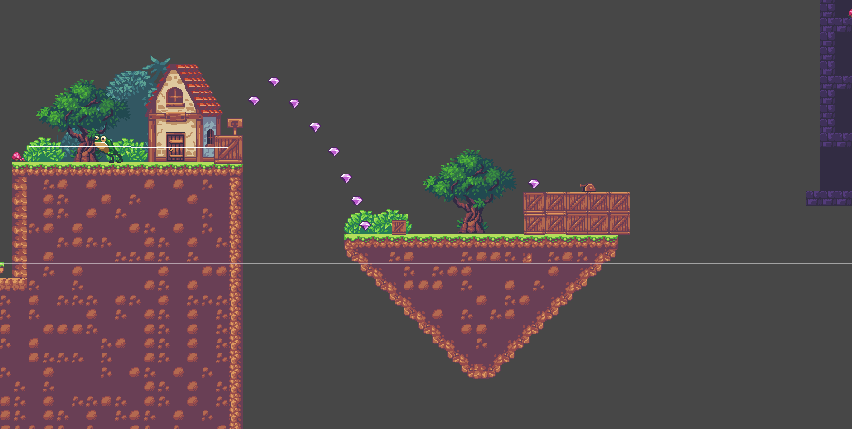
Poniżej będzie podana lista elementów, które zostały zawarte w grze. Później zostaną one opisane w bardziej szczegółowy sposób.

* Mapa
* 2 formy przeciwników oraz kolce
* Spawnpointy, Checkpointy
* Dynamiczny świat (ruchome, przesuwane obiekty, dźwignie)
* Główna postać i jej mechanika ruchu
* Animacje postaci, monet i innych elementów
* Dodanie dźwięków i dialogów
* Napisy pomagające graczowi
* Poboczna postać (ptak)
* Proste menu pojawiające się przed grą
* Kamera oraz tło.

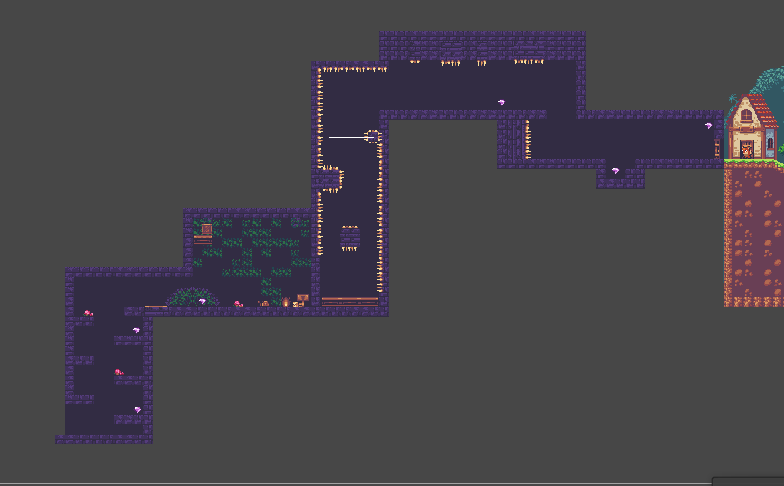
## 2.1. MAPA



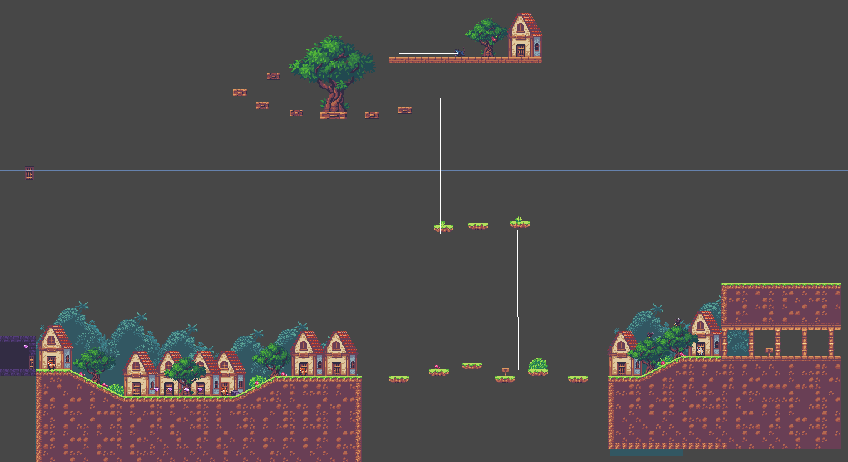
Fot. 2.1. Część mapy użytej w grze (tzw. Spawn)



Fot. 2.2. Część mapy użytej w grze (fragment z przepaścią, przesuwaną skrzynią i wysuwanym mostem)



Fot. 2.3. Część mapy użytej w grze (fragment zawierający skoki, płytka naciskowa, unikanie kolców i otwierane drzwi)

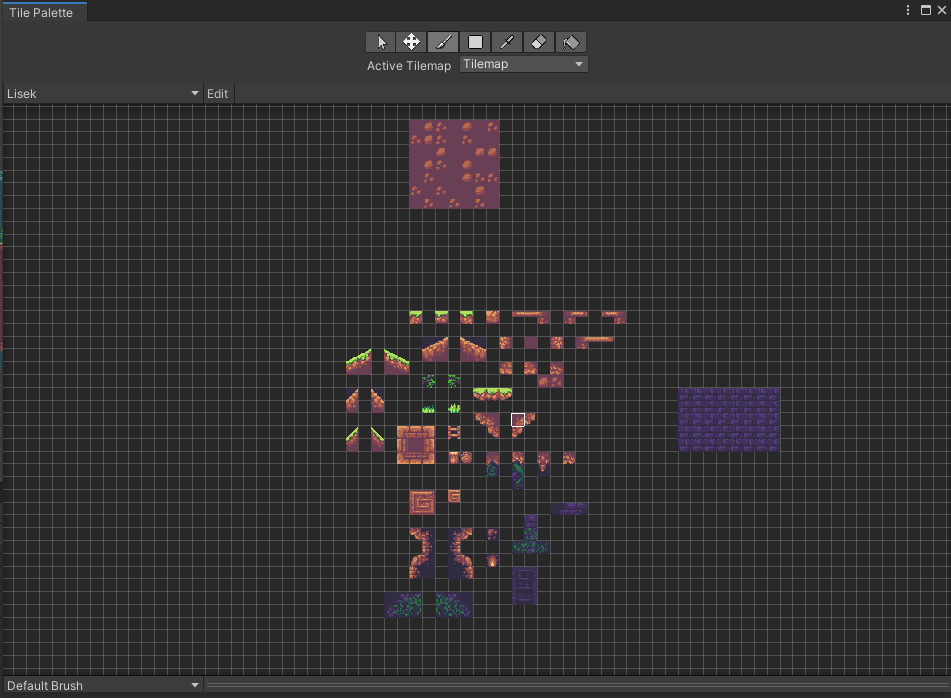


Fot. 2.4. Cześć mapy, skoki, odbieranie misji oraz sam koniec gry

Powyższe obrazki starają się w jak najbardziej zwięzły sposób pokazać wizualnie całą mapę (widok z silnika, nie jest to widok jaki widzi gracz). Zachęcamy do pobrania gry z poniższego linku w celu na własne oczy zobaczenia jak wygląda gra oraz jej mechaniki.

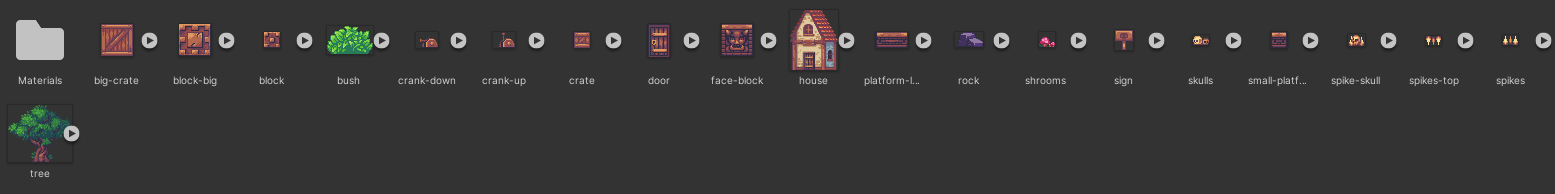
<https://drive.google.com/file/d/1qNpeNMy0LSGONGufAyGxI4WGvVJ6Fx3f/view?usp=sharing>

Tworząc mapę (elementy, które są pierwszoplanowe), korzystaliśmy z TilePalette (coś na kształt palety malarskiej, tylko zamiast kolorów mamy gotowe tekstury, które nakłada się na główne okno).



Fot. 2.5. Wygląd oknaTilePallete

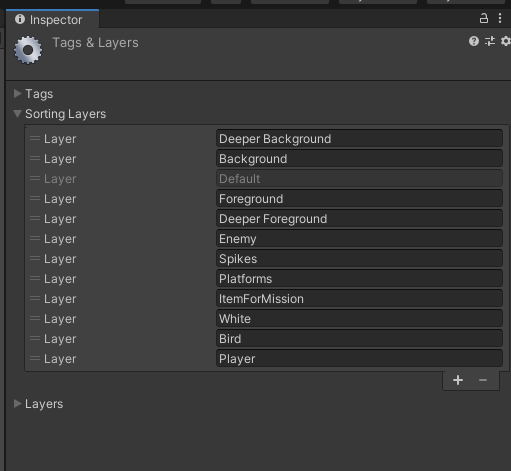
Do elementów mapy (tła), głównie w celu upiększenia, dodania szczegółów, używaliśmy tekstur bardziej swobodnych (współrzędne tych obiektów nie musza się mieścić w specjalnej siatce, można dowolnie je ustawiać).



Fot. 2.6.Przykłady obiektów, które nie korzystały z TilePalette są zamieszczone wyżej

Warto dodać, że każdy blok z TilePalette może zostać zamieszczony w grze w sposób swobodny, jednak paleta przyspiesza znacząco budowanie mapy.

Unity posiada trzy osie współrzędnych, ale my w naszej grze korzystamy tylko z dwóch (nie poruszamy się postacią w „głębię gry”). Jednak gra musi dla każdego obiektu (w celu dobrego wyświetlenia) rozróżniać tzw.: sortinglayer (sortownik warstwy).



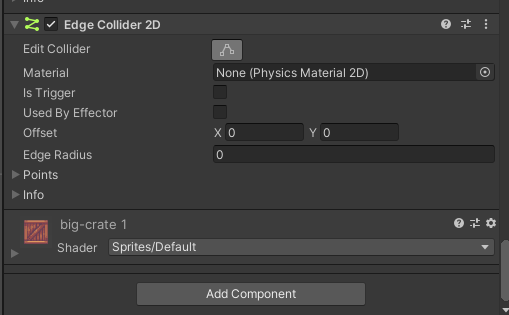
Fot. 2.7. Wygląd okna umożliwiającego sortowanie warstw

Ilość warstw jakie można dodać jest nieograniczona. Kolejność jest bardzo ważna. Ona decyduje w jakiej kolejności coś jest wyświetlane przed czymś. Możemy w ten sposób wyświetlać na samym początku (najbliżej) np.: postać, a za nią w tle drzewa.



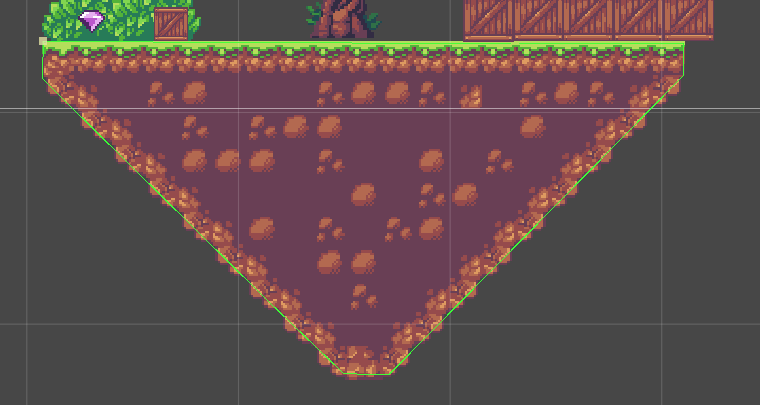
Fot. 2.8. Efekt ustawienia kolejności warstw

Ostatnią rzeczą, ale bardzo ważną w mapie jest dodanie do niej kolizji (Collidera). Najprościej mówiąc jest totekstura po której będziemy chodzić. Dodaje się nowy komponent z prawej części ekranu o nazwie (Box lub Edge, zależnie od przeznaczenia).



Fot. 2.9. Okno Edge Colidera dla obiektu big-crate 1

Ważne, by dodać go do naszego obiektu tworzącego mapę. Sam Collider objawia się zieloną linią, która w przypadku Edge można dowolnie modyfikować, a Box układać w różne kombinacje kwadratów i prostokątów.

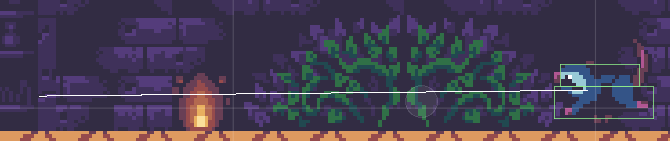


Fot. 2.10. Wizualne przedstawienie Collidera

## 2.2. PRZECIWNICY

W naszej grze wyróżniamy dwa typy przeciwników, „groźne” oposy i neutralne żaby. Warto zaznaczyć, że obiekty te posiadają animacje, ale trudno je zamieścić w sprawozdaniu.

Zaczynając od oposów:



Fot. 2.11. Opos

Ten typ przeciwnika posiada dwa Collidery (górny i dolny). W ten sposób potrafimy rozróżnić wskoczenie na przeciwnika oraz po prostu wbiegnięcie w niego. Daje nam to możliwość zabicia go (w przypadku wskoczenia na niego – zetknięcia Collidera postaci z Colliderem oposa) lub sprawienia, że zabije on naszego gracza (zetknięcie dolnego Collidera). Sam opos porusza się po strefie wyznaczonej białą linią na obrazku.

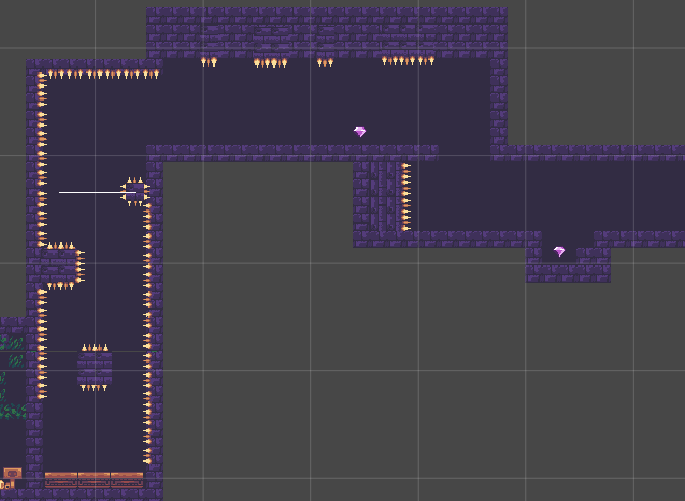
Przejdźmy do żaby:



Fot. 2.12. Żaba

Żaba posiada tylko jeden Collider. Nie zabija ona gracza, ale potrafi go przepychać. Tak samo jak opos zmienia swoje położenie w kierunkach zaznaczonych białą linią. Jest ona animowana.

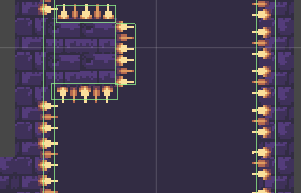
Wliczmy do kategorii przeciwników kolce:



Fot. 2.13. Scena z kolcami

W naszej grze wyróżniamy trzy rodzaje kolców: statyczne, dynamiczne oraz bodźcowe (OnTrigger).

Statyczne to prawie nasz opos (Collider, który podczas zetknięcia zabija gracza).



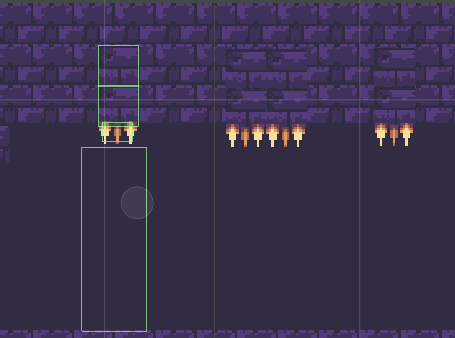
Fot. 2.14. Kolce i ich Colider

Dynamiczne, które poruszają się od punktu A do B (tak samo jak opos czy żaba).Posiadają Collidery zabijające gracza.



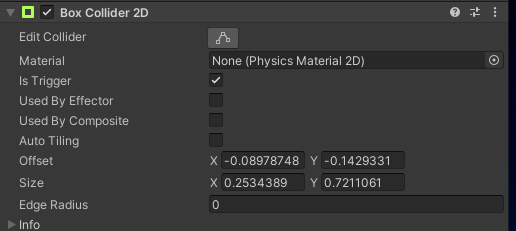
Fot. 2.15. Kolce ze ścieżką poruszania

Ostatnim wymienionym typem były kolce bodźcowe. Opiszmy je w prostych słowach, jeśli nasz gracz wejdzie w strefę (TriggerArea) to coś się wydarzy. W tym przypadku pójdzie w ruch platforma z kolcami.



Fot. 2.16. Kolce ze strefą TriggerArea

Dolna część to wspomniana TriggerArea. Nasuwa się pytanie czym się różni od Collidera. Trigger nie wpływa na ruchy gracza. Zaznaczamy to w komponentach, by uaktywnić w skryptach tą funkcję (isTrigger).



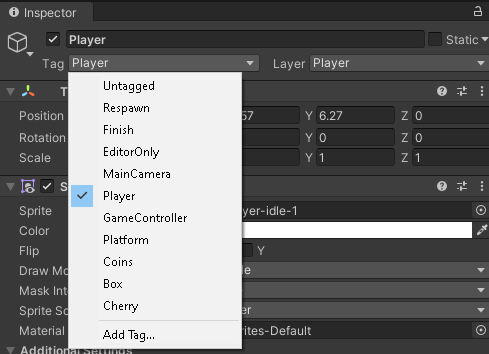
Fot. 2.17. BoxColider i funkcja Is Trigger

Wracając do obrazka, dwie tekstury na górze to zwykłe Collidery, a ten mały jeden pod nimi to Collider zabijający. WspomianaTriggeArea uaktywnia cały skrypt i cała ta platforma spada, a po chwili, wraca na pozycję wyjściową.



Fot. 2.18. Ruchome kolce w trakcie rozgrywki

Opiszmy w tym dziale też Tagi obiektów. Co one nam dają? Dzięki nim możemy rozróżniać obiekty w skryptach, by np.: przeciwnik nie startował skryptu dla gracza. Zmieniamy go w górnym prawym rogu obiektu (lub dodajemy nowy).

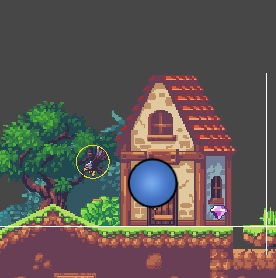


Fot. 2.19. Dodawanie tagów

Nie każdy obiekt musi mieć swój Tag, ale warto by te, które coś wyjątkowego robią go miały.

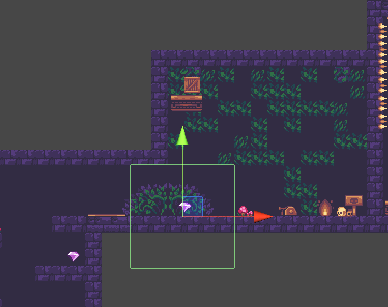
## 2.3. Spawnpoint/Checkpoint

Prawidłowe używanie tych elementów gry zakłada zapisywanie pozycji, w których ma odradzać się gracz (w raz z postępem w przejściu gry).



Fot. 2.20. Zaznaczony spawnpoint

Mechanizm działania checkpointu w naszej grze polega na wejściu w strefę TriggerArea, która uruchamia skrypt nadpisujący koordynaty obiektu spawn (w nim są obecne koordynaty miejsca odrodzenia).

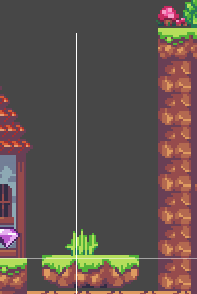


Fot. 2.21. TriggerArea dla checkpointu

## 2.4. Dynamiczny świat

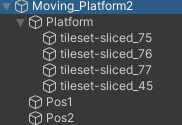
W tym dziale opisaliśmy obiekty, które mogą wchodzić w interakcję z graczem lub są to ruchome platformy

Zaczynając od platformy:



Fot. 2.22. Jedna z wielu ruchomych platform w grze

Platforma to prosty obiekt naszej gry, taki sam jak opos czy żaba. Jest to zbitek tekstur, które posiadają Collider i odpowiedni skrypt.



Fot. 2.23. Cały obiekt powyższej platformy

W folderze Platform zawierają się tekstury. Pos1 oraz Pos2 to współrzędne dwóch punktów pomiędzy którymi lata platforma. W Unity, by taki zbitek tekstur podróżował wspólnie wystarczy dodać skrypt wykonywalny (jako komponent) do folderu rodzica (w tym przypadku Platform).

Najprostszą rzeczą, do której nie trzeba nic pisać jest zaprogramowanie przesuwania bloku.



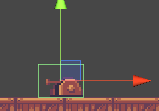
Fot. 2.24. Przesuwana skrzynia

Wystarczy dodać trzy gotowe komponenty podane przez Unity. Jest to już poznany Collider (obojętnie, który byle 2D), Rigidbody 2D (funkcja odpowiedzialna za fizykę na obiekcie) oraz ostatni Fixed Joint 2D. Dodanie ich powoduje automatycznie, że nasz obiekt będzie już przesuwany przez inny obiekt posiadający Collider2D.

Przejdźmy dalej do dźwigni.



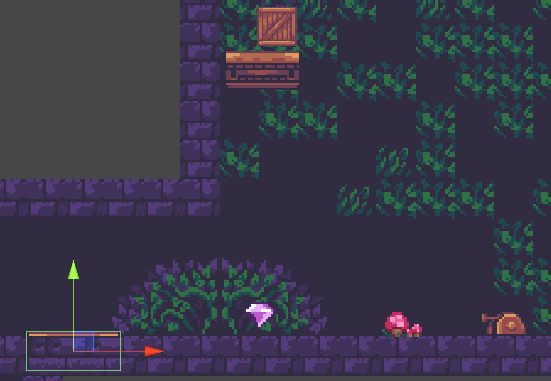
Fot. 2.25. Wysuwany most za pomocą wciśnięcia klawisza



Fot. 2.26. TriggerArea naszej dźwigni

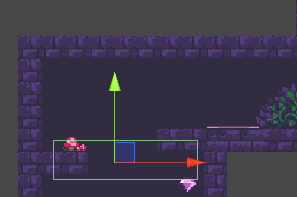
Odnośnie naszej dźwigni jest to prosty skrypt, który sprawdza czy nasza postać (sprawdza Tag), jest w TriggerArea dźwigni i po wciśnięciu odpowiedniego klawisza uaktywnia akcję skryptu (w tym przypadku robi ruch elementami mostu na odpowiednie pozycje).

Element kolejny to płytka naciskowa (dość prymitywna). Aby uaktywnić akcję należy przesunąć skrzynię do TriggerArea. Po raz kolejny liczy się tu bardzo odpowiedni Tag.



Fot. 2.27. TriggerArea płytki naciskowej

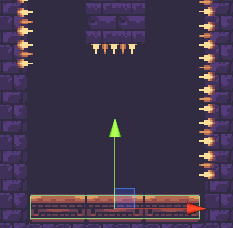
Zadbaliśmy też o odpowiednie zabezpieczenia odnośnie gry. Zakładając sytuację, że gracz zrzuci kostkę po za mapę lub w miejsce uniemożliwiające misję.



Fot. 2.28. TriggerArea do debugowania

W momencie wpadnięcia kostki do TriggerArea zamieszczonej wyżej powoduje start skryptu, który „oddaje kostkę”, następuje przeniesienie jej na odpowiednie współrzędne.

Jednym z ostatnich typów dynamicznego świata jest platforma, która wykrywa kolizję z graczem i w ten sposób porusza się do góry. W przypadku kiedy gracz nie stoi na niej, powoduje to powrót na pozycję wyjściową.



Fot. 2.29. Platforma, tylko na nacisk gracza

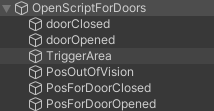
Ostatni element naszego dynamicznego świata to drzwi (otwierane?). Wykorzystujemy tutaj efekt podmiany.



Fot. 2.30. Drzwi i ich collidery oraz triggery

Na początku warto napisać, że drzwi na górze są w zupełnie innym miejscu normalnie, ale na potrzebę zrobienia fotografii zostały przeniesione.

Mechanizm jest podobny do dźwigni. Po wejściu w TriggerArea możemy wcisnąć klawisz. W tym przypadku skrypt musi pamiętać o tym czy drzwi są otwarte czy zamknięte i odpowiednio je podmieniać. Widzimy jasno, że zamknięte drzwi posiadają Collider, a te otwarte nie.



Fot. 2.31. Folder z obiektami do drzwi

GameObjecty (po prostu obiekty) doorClosed i doorOpened to są nasze tekstury drzwi. Następnie mamy TriggerArea (wyżej był opisany mechanizm działania). Kończymy na pozycjach OutOfVision, czyli pozycja gdzie nasz gracz nie będzie widział podmiany. Pozostałe dwa obiekty to pozycję dla drzwi, potrzeba ich dwóch, aby zawiasy drzwi nie przemieszczały się w sposób absurdalny (by całość prezentowała się lepiej).

## 2.5. Główna postać i mechanika ruchu

To zagadnienie było jednym z najtrudniejszych do napisania. Sterowanie postacią działa na przyciskach W,A,D. Odpowiednio skok, lewo, prawo. Samo napisanie skryptu oraz dodanie odpowiedniej animacji do naszej postaci zależnej od akcji jaka się dzieje było najtrudniejszym do zaimplementowania elementem naszej gry. Więcej omówione zostanie na temat tego w dziale poświeconym skryptom.

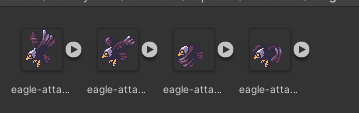


Fot. 2.32. Nasza postać

Trzeba zauważyć, że nasza postać posiada dwa Collidery (okrągły i kwadratowy). Jest to związane z poruszaniem się (koło sprawia, że movement jest bardziej płynny). Pamiętajmy o tym, że nasza postać będzie podlegać fizyce (grawitacji), więc musimy dodać RigidBody2D <layer i Tag też>.

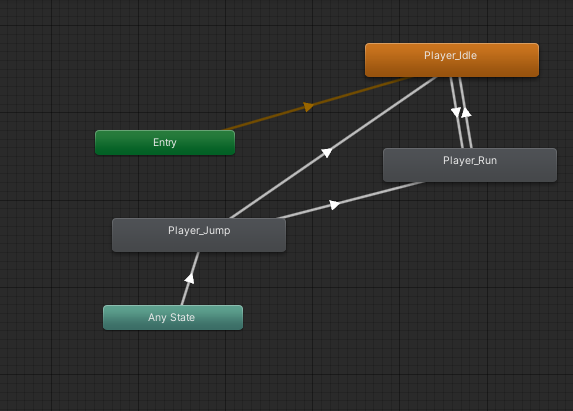
## 2.6. Animacje postaci, monet i innych elementów

Animacje w naszej grze odgrywają rolę czysto estetyczną. Jest ona jednak pewnym standardem (bez niego gra traci urok). By stworzyć animację potrzebuje paru tekstur.



Fot. 2.33. Tekstury ptaka

By uzyskać animację należy zaznaczyć obrazki i przeciągnąć je na scenę (trzeba też ją zapisać w odpowiednim folderze). Potem musimy jeszcze ustawić parametry, które będą wpływać na szybkość przejść pomiędzy kolejnymi klatkami. Przedmiot animowany otrzymuje zawszę tzw.: Animator. Najprościej mówiąc jest to rzecz, która będzie odpowiadać za to co pokazać.



Fot. 2.34. Animator gracza

Robiąc animację musimy pamiętać o tym, by nasza postać była animowana w lewo jak i w prawo (jedna linijka dodatkowa w kodzie).

## 2.7. Dodanie dźwięków i dialogów

Nie można przedstawić dźwięków w sprawozdaniu, jednak możemy kolejny raz zachęcić do odpalenia naszej gry i sprawdzeniu na własne uszy tego. Dźwięki i dialogi nie są wszechobecne w naszej grze (zaimplementowaliśmy je tylko by się nauczyć to robić). W sumie mamy dźwięk zbierania monety, otwierania oraz zamykania drzwi. Co do dialogu to został on nagrany i przetworzony za pomocą programu Audacity. Pojawia się on podczas odbierania oraz oddawania misji, która jest obecna w grze. Do specjalnego GameObjectu trzeba dodać komponent AudioSource oraz specjalny skrypt.

## 2.8. Napisy pomagające graczowi



Fot. 2.35. Przykładowa klatka z gry z napisami

Wszystkie napisy są dostępne (napisy to też licznik gemów) już na początku gry. Tylko każdy z nich jest ukryty odpowiednio wchodząc w daną strefę (TriggerArea) uaktywniamy konkretny napis przekazany w parametrze skryptu.



Fot. 2.36. TriggerArea dla napisu wyżej

Po opuszczeniu danej strefy napis znika.

Wszystkie napisy jak i elementy UI (User Interface) powinny znajdować się w folderze Canvas (specjalny przeznaczony na to rodzic). W przypadku naszych napisów korzystamy z tzw.: kotwicy (Anchora). Przytwierdzamy napis do dolnej części ekranu.



Fot. 2.37. Canvas naszej gry

## 2.9. Poboczna postać (ptak)

Początkowo zakładaliśmy, że ambitny plan stworzenia ptaka, który będzie za nami latać, a po użyciu pewnego przycisku (zmieniający postać na latające stworzenie) będzie bardzo prosty do zaimplementowania. Niestety przerosło to nas i zostało odłożone to na później. Plan został wyrzucony do śmietnika. Sam ptak został w grze jako drobny estetyczny dodatkowy element. Posiada on tzw.: pathfinding, szuka najbliższej drogi do celu (naszego bohatera), śledzi go wszędzie.



Fot. 2.38. Ptak i gracz

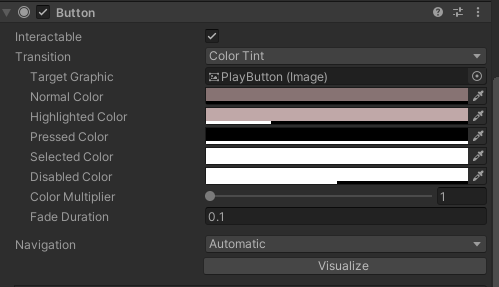
Warto dodać, że ptak jest też animowaną postacią.

## 2.10. Menu przed grą



Fot. 2.39. Prezentacja menu

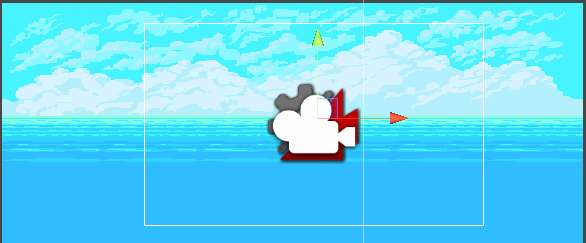
Nasze menu to jest osobna scena. Posiada ona nie wiele obiektów, ale są one bardzo ważne. Pojawia się tutaj Canvas, który był przed chwilą wspomniany. W tym obiekcie znajdują się dwa przyciski, dostosowane graficzne do naszego menu. Ustawienie koloru, ale też barwy po kliknięciu i najechaniu. Trzeba było też sprawić, aby jedyne co było widoczne to napis (ustawianie kanału Alpha).



Fot. 2.40. Ustawienia dla przycisków

Samo menu posiada jeszcze background image. Napisy są przymocowane do środka ekranu za pomocą Anchorów.

## 2.11. Kamera i tło

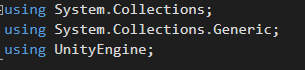


Fot. 2.41. Kamera w grze

Kamera przedstawiona na obrazku wyżej to nasz gracz. Przytwierdzona jest ona do naszej postaci. Gotowy komponent nazywa się CinemachineVirtualCamera. Ma ogromną ilość parametrów, których nie warto w tym sprawozdaniu wymieniać. Najważniejszym elementem jej jest to, że można ją przyczepić do gracza oraz to, że można przyczepić do niej tło. W ten sposób nie musimy na naszej całej mapie tworzyć niczego z tła głębokiego, bo gracz nawet nie zauważy, że tam czegoś brakuje (kamera i tło do niej przyczepione zrobi za nas wszystko).

# 3. Przykładowe skrypty gry

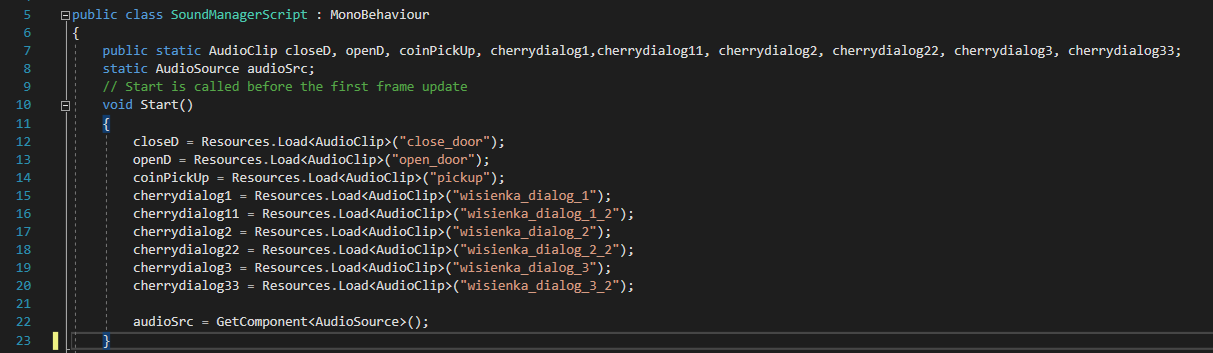
W tym dziale zostaną zamieszczone przykładowe ciekawsze skrypty, które warto opisać. Wszystkie pliki będą zawierać include’y. Są one podane niżej.



Fot. 3.1. Include’y występujące u nas w każdym skrypcie

Warto dodać, że każdy skrypt opierać się może na dwóch metodach/funkcjach. Jest to „void Start()” oraz „void Update()”. Pierwszy to zwykły init, na początku gry, metoda ta jest wykonywana. Drugi z kolej to metoda, która ponawia się co każdy frame (klatkę) gry. Gra jest stworzona w 30FPS (klatek na sek.), więc 30 razy (w każdej sekundzie).

## 3.1. Skrypt dźwiękowy

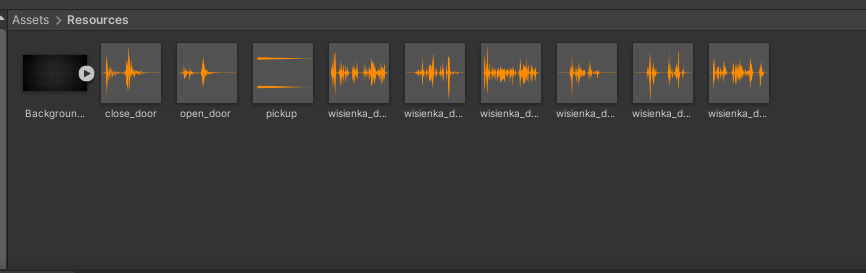


Fot. 3.2. Linie skryptu dźwiękowego

Zaczynając od linii 7: Definicja zmiennych, do których zostaną „zapisane” klipy.

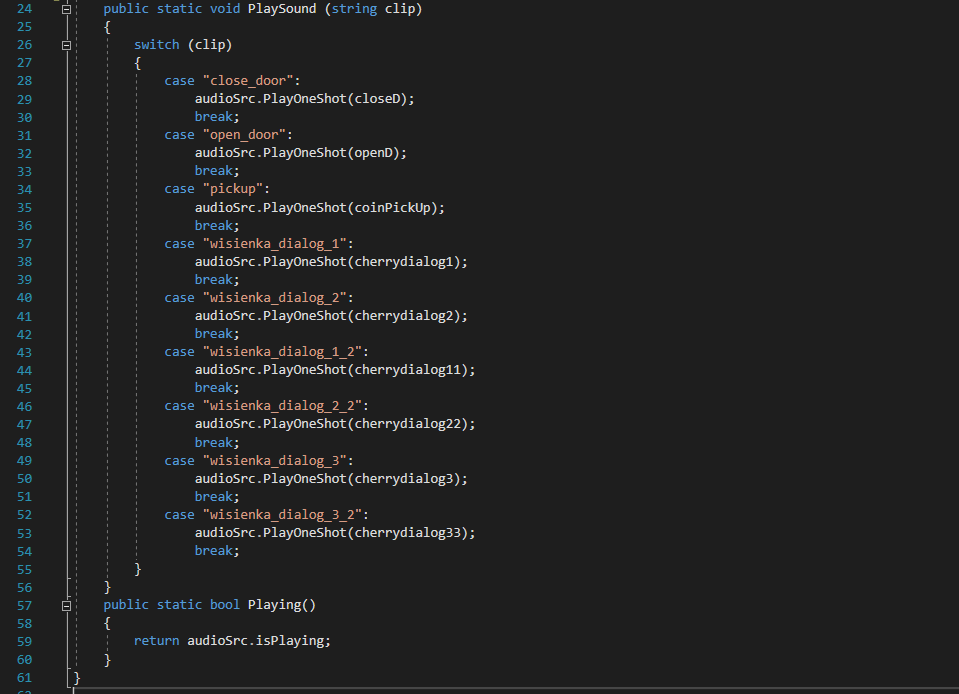
Linia 8: Zmienna odpowiedzialna za używanie dźwięku w Unity.

Linie 12-20: Wspomniana metoda „void Start()”, ładowanie klipów z folderu Resources do zmiennych. Czerwone nazwy są nazwami plików.



Fot. 3.3. Folder z dźwiękami

Linia: 22: Charakterystyczny zabieg dla AudioSource, inicjalizacja zmiennej tego typu.



Fot. 3.4. Kolejne linie kodu dźwiękowego

Linie 24-56: Funkcja służąca do odtworzenia konkretnego dźwięku po za ramami całego tego skryptu. Typowy switch/case dla stringa. Linie kodu „audioSrc.PlayOneShot” to odtworzenia konkretnego dźwięku w postaci OneShotu (pojedynczego klipu).

Linia 57-60: Funkcja zwracająca stan AudioSource (czy gra czy nie).

Wygląd linii kodu po za naszym skryptem, który wywołuje konkretny dźwięk wygląda tak:

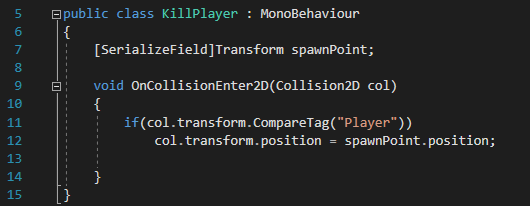


Fot. 3.5. Linia kodu dźwiękowego po za głównym skryptem

## 3.2. Skrypt teleportujący

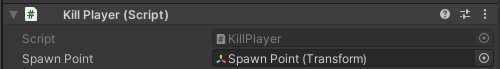
Skrypty teleportujące to linie kodu zmieniające położenie danej rzeczy w przypadku jakiegoś zdarzenia (np.: wejścia w TriggerArea). Skrypty te wykorzystujemy do wszystkiego w naszej grze i jest to podstawa podstawy.

Od zabijania gracza po oddawanie kostki w przypadku wypadnięcia jej.



Fot. 3.6. Linie kodu zabijającego (teleportującego)

Linia 7: [SerializeField] to rodzaj prywatnej zmiennej (niedostępna po za tym skryptem). Typ zmiennej Transform to zmienna trzymające pozycję, stopień nachylenia itp. W tym przypadku chcemy zabić gracza, więc w parametrze komponentu przekazujemy GameObject spawnpointu.



Fot. 3.7. Przekazywanie GameObjectu

W tym momencie nasz skrypt widzi ten obiekt jako współrzędne i inne cechy położenia.

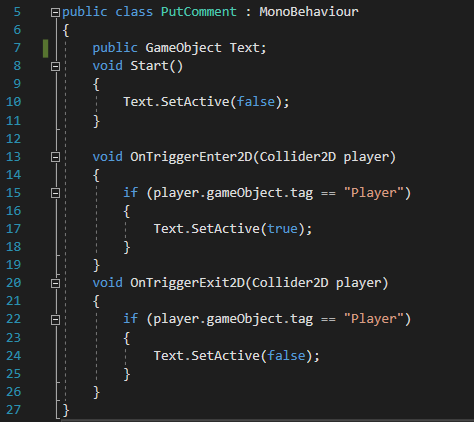
Linie 9-14: Są to funkcje opierające się na zdarzeniach wyróżniamy Collision oraz Trigger z tych podstawowych (wspomniano było dużo wcześniej czym się to różni). Trzeba jeszcze dopisać stan: Enter, Exit lub Stay. Opisują one moment kiedy dana linia kodu ma być wykonywana. W tym przypadku mamy przykład zetknięcia się dwóch Colliderów. W parametrze funkcji (zmienna col) jest przechowywany Collider, który powoduje dane zdarzenie. Patrzymy na Tag obiektu powodującego zdarzenie, jeśli jest to Tag gracza, zmieniamy współrzędne naszej postaci.

Warto pamiętać, że wszystkie operacje jakie wykonujemy są dla 2D.

## 3.3. Skrypt wyświetlający tekst

Wspomniane było w dziale z napisami, że każdy napis, który jest obecny w grze, jest w niej od samego początku. Jedynie wszystkie te napisy są dobrze ukryte.

Aby je pokazać potrzebny jest uniwersalny skrypt, który będzie obsługiwał wszystkie napisy (lepsze jest pisanie 1 skryptu dla wszystkich niż 20 dla każdego osobno).



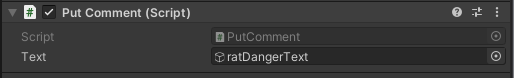
Fot. 3.8. Kod do tekstów

Linia 7: Stworzenie zmiennej, która będzie naszym tekstem (przekazana w parametrze).

Linie 8-11: Ukrycie tekstu przekazanego w parametrze na początku gry.

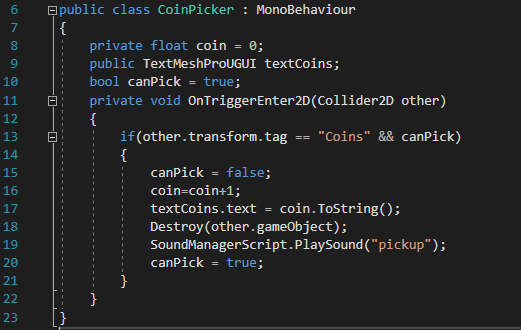
Linie 13-19: Wejście w TriggerArea przez gracza powoduje wyświetlenie tego konkretnego tekstu.

Linie 20-26: To samo co w liniach 13-19, ale ukrywa napis.



Fot. 3.9. Przykładowy tekst przekazany w parametrze

## 3.4. Skrypt zliczający monety



Fot. 3.10. Linie kodu zliczającego zdobyte monety

Linia 8: Zmienna przechowująca ilość zebranych monet.

Linia 9: Zmienna przez, która można zmieniać tekst w skrypcie.

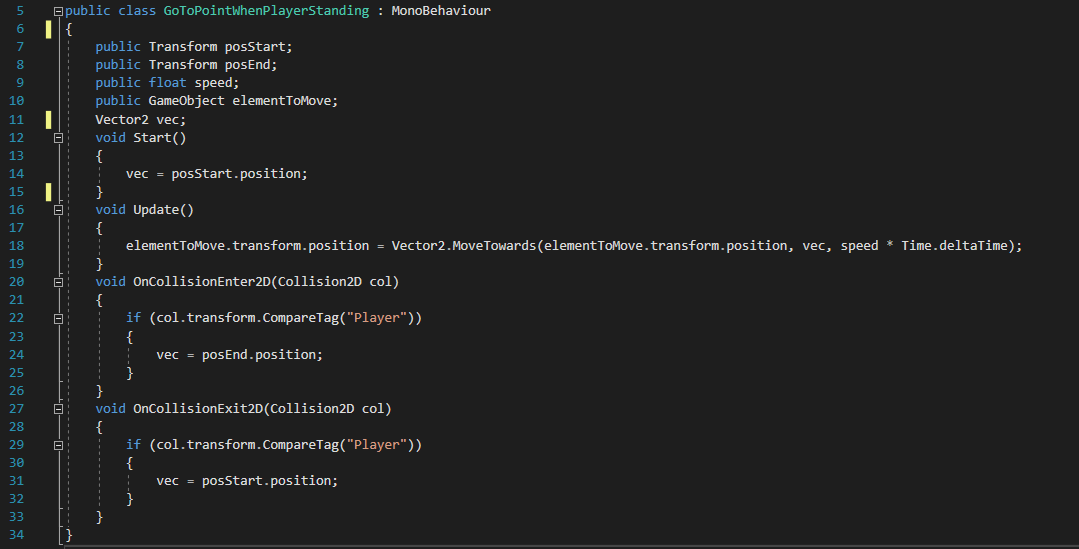
Linia 10: Pewnego rodzaju zmienna dla bezpieczeństwa.

Linie 11-22: Podczas wejścia w TriggerArea monety i możliwości podniesienia jej, inkrementujemy zmienną. W linii 17 zmieniamy napis ilości monet. W 18 usuwamy monetę z mapy, a w 19 odpalamy dźwięk zbierania monety.

## 3.5. Skrypt platformy

Przechodząc od razu do bardziej zaawansowanego skryptu platformy, idziemy do skryptu patrzącego czy stoi na niej gracz (nie ma sensu opisywać obu).

Warto tylko tutaj wspomnieć o deltaTime. Jest to czas jaki upłynął pomiędzy dwoma klatkami (frame’ami).



Fot. 3.11. Linie kodu platformy

Linie 7-8: Transformy (pozycje), końcowej i początkowej pozycji.

Linia 9: Prędkość ustawiana w parametrze.

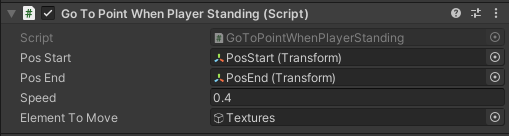
Linia 10: GameObject z teksturami, które będą się poruszać.

Linia 11: Zmienna typu Vector (posiada współrzędne X oraz Y).

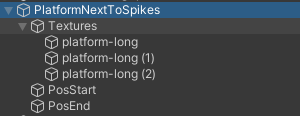
Linie 12-15: Przypisanie pod nasz vec wartości początkowej (platforma jest na dole).

Linie 16-19: Funkcja Update, która powoduje, że zgodnie ze zmienną vec (do góry lub do dołu) platforma się porusza w naturalnym tempie.

Linie 20-26 (oraz 27-33): Jeśli gracz dotyka Collidera platformy to zmienna vec się zmienia na współrzędne wyższej pozycji. W przeciwnym wypadku ta zmienna ma współrzędne niższej pozycji.



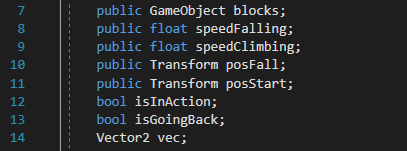
Fot. 3.12. Parametry tej funkcji



Fot. 3.13. Folder GameObject’ów wymienionego skryptu

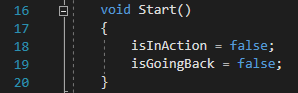
## 3.6. Skrypt spadających kolców

Dość skomplikowany skrypt, który powoduje ,że kolce na wejście gracza w TriggerArea lecą w dół (z konkretną prędkością), a później podnoszą się (z inna). Dopóki nie wrócą na pozycję startową nie mogą spaść.



Fot. 3.14. Zmienne skryptu

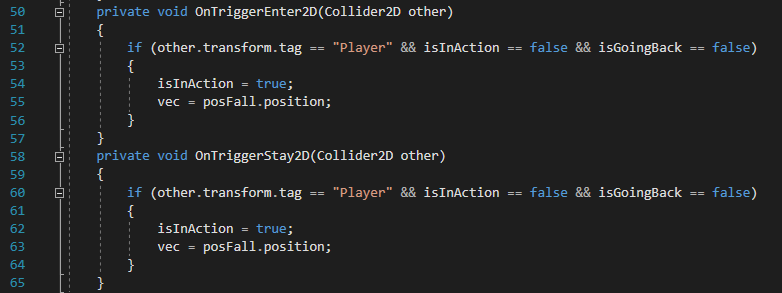
Chyba nazwy zmiennych w połączeniu z poprzednimi skryptami dają nam wystarczająco informacji.



Fot. 3.15. Inicjalizacja zmiennych

Na starcie zmieniamy typ zmiennych (stan oczekiwania).

Przejdźmy teraz do końca skryptu, który odpowiada za wykrywanie bodźca.

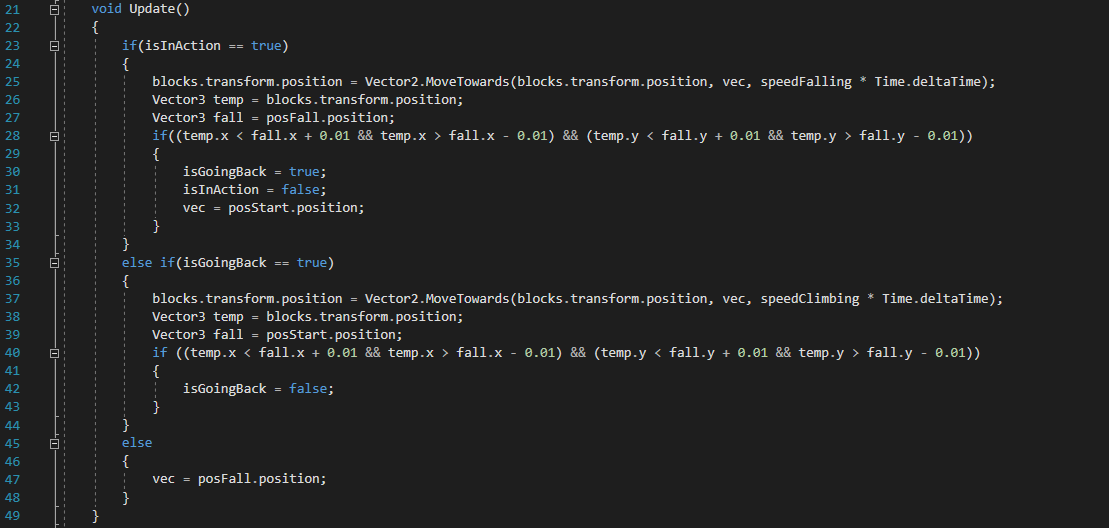


Fot. 3.16. Linie kodu odpowiadające za wykrywanie bodźca

Mamy tu przykład wykorzystania Enter oraz Stay (Enter działa tylko raz, na wejściu w TriggerArea, stąd funkcja Stay, aby wykonywać parę razy to zdarzenie).

Linie 52 oraz 60: Sprawdzenie Tagów oraz tego czy już nie jest wykonywane zdarzenie

Linie 55 oraz 63: Przypisanie do wektora pozycji, do której ma lecieć (spadać) platforma z kolcami. Podobne zdarzenia do przypadku z platformą opisane w dziale 3.5.

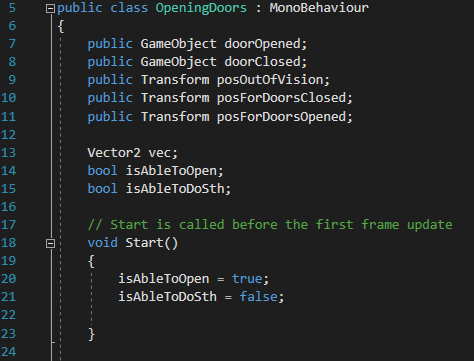


Fot. 3.17. Linie kodu odpowiadające za poruszanie kolcami

Zaczynając od znanych linii kodu poruszających blokami z daną prędkością w danym kierunku w liniach 25 oraz 37. Przechodzimy do linii, które sprawdzają czy pozycja naszej platformy pokrywa się już w zakresie błędu, by zmienić akcję.

Skrypt chodź długi po zapoznaniu się z podstawami wydaje się bardzo prosty.

## 3.7. Skrypt dla drzwi

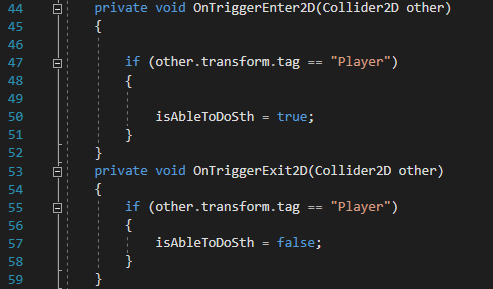


Fot. 3.18. Linie początkowe skryptu

Linie 7-11 były opisane w dziale poświeconym drzwiom (2.4). Przechodząc dalej typowa zmienna Vector oraz dwie zmienne true or false.

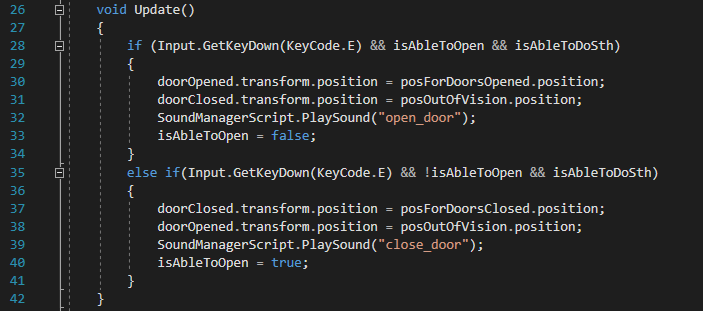
isAbleToOpen -> pokazuje czy drzwi są zamknięte czy otwarte (na starcie są zamknięte).

isAbleToDoSth-> pokazuje czy gracz jest w strefie, w której może już używać drzwi.



Fot. 3.19. Linie początkowe skryptu

Krótki komentarz, zmienna, która odpowiada za uzyskanie pozwolenia na wykonanie wydarzenia ulega zmianie w momencie wejścia w TriggerArea (w jej), gracza.



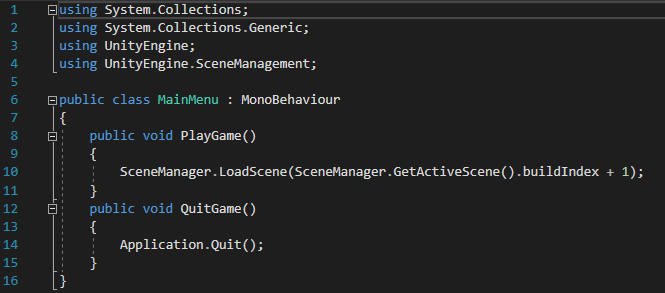
Fot. 3.20. Linie skryptu drzwi

W liniach 28-35 odbierany jest sygnał z klawiatury (przycisk E) oraz walidacja zmiennymi true or false sama akcja. Następnie następuje prosta podmiana pozycji obu par drzwi (patrz 2.4, jak to działa), otworzony jest dźwięk i zmiana stanu drzwi na przeciwny.

Tego typu skrypty będą też wykorzystywane w przypadku dźwigni i innych akcji pod przycisk.

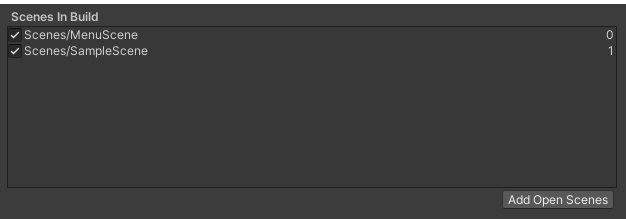
## 3.8. Skrypt menu

Samo menu nie posiada bardzo skomplikowanego skryptu.



Fot. 3.21. Linie skryptu drzwi

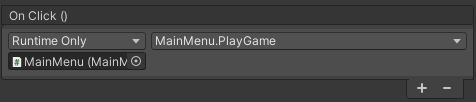
Tutaj liczy się bardzo konfiguracja Unity. Odnośnie linii 10, po wciśnięciu przycisku PLAY, nasza scena ulega zmianie na kolejną w liście ustawień Unity (ręczna konfiguracja).



Fot. 3.22. Sceny naszej gry

Łatwo zobaczyć, że skoro MENU to scena 0 to dodaniu jedynki przejdziemy do SampleScene, czyli naszej gry. W przypadku wciśnięcia przycisku QUIT wyjdziemy z gry (linia 14).

W przypadku przycisków dodajemy (bez skryptowo) akcję pod daną skryptową funkcję (OnClickEvent).

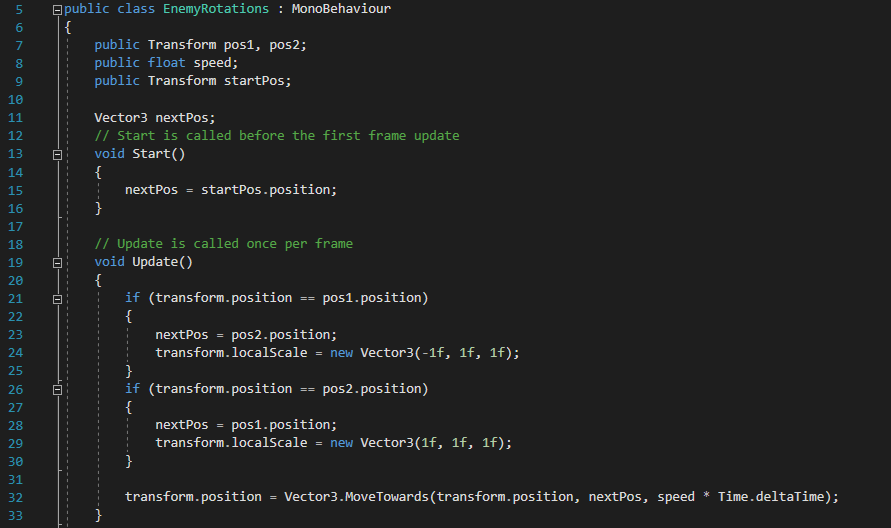


Fot. 3.23. OnClickEvent, komponent przycisku PLAY

Wszystkie elementy tak jak opisane w punkcie 2.10 są w komponentach obiektu typu Button.

## 3.9. Rotacja obiektu

Jest to bardzo drobna rzecz, która pozwala na obrót o 180 stopni danego obiektu, aby np.: ptak leciał w druga stronę, a nie tyłem.



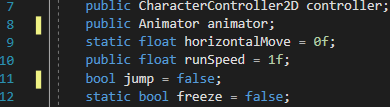
Fot. 3.24. Linie kodu poruszające od pozycji do pozycji naszym oposem

W liniach zaimplementowane obracanie obiektu jest w liniach 24 oraz 29. Dostajemy się do pól klasy, które odpowiedzialne są za rotacje obiektu i zmieniamy pierwszy wymiar na wartość przeciwną (zależnie od pozycji, w która idzie nasz opos). Reszta linii kodu to poruszanie się od punktu do punktu tym przeciwnikiem.

## 3.10. Skrypt do poruszania graczem

Spędziliśmy parę godzin na napisaniu dobrego skryptu do poruszania graczem, bez skutków. Uprościliśmy, więc sobie ten element gry skorzystaliśmy z gotowego skryptu zaproponowanego dla początkowych twórców gier. Plik CharacterController2D.cs w plikach projektowych jest zmienioną wersją gotowe kodu. Natomiast plik PlayerMovement.cs jest w pełni przez nas napisaną wersją. Oba te pliki są dołączone do naszej postaci i odpowiadają za prawidłową animację, chodzenie i skakanie. Dodatkowo gotowe rozwiązanie oferuje rzeczy takie jak kucanie, flip oraz tzw.: movement smoothing parametr może sprawić, że nasza postać będzie chodzić twardo po ziemi lub ślizgać się jak na lodzie.

Poniżej przedstawimy napisany przez nas wspomagacz.



Fot. 3.25. Zmienne naszego kodu

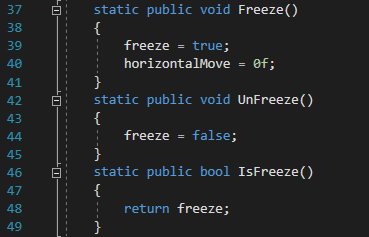
Linia 7: Zmienna naszego gotowe skrypty, który odpowiada za prawidłowy movement.

Linia 8: To animator, będziemy zmieniać wyświetlaną animację w zależności od stanu postaci.

Linia 9: Opowiada za ruch horyzontalny (na starcie będzie 0 bo nasza postać się nie rusza).

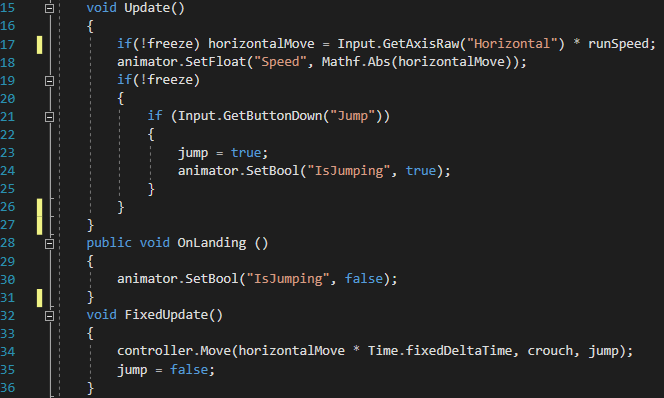
Linia 10: Prędkość poruszania się

Linia 11 oraz 12: Zmienna pokazująca czy nasza postać akurat skacze oraz zmienna do zamrażania postaci (potrzebna np.: przy odbieraniu misji, zabieranie możliwości poruszania się graczowi).



Fot. 3.26. Zamrażanie postaci

Bardzo proste linie kodu, które mają na celu ograniczenie lub umożliwienie poruszania się naszej postaci. Linia 40 jest konieczna (zmiana wartości), bo nasza postać cały czas po zamrożeniu miałaby wyświetlaną animację biegania (zamiast stania w miejscu). Przejdźmy teraz trochę do teorii Unity. Mówiliśmy o metodzie Start oraz Update. Istnieje też metoda FixedUpdate. Polega ona na tym, że w przeciwieństwie do Update (wykonywany co klatkę), ta funkcja jest wykonywana zero, raz lub tyle ile chcemy razy na sekundę. Daje nam to możliwość płynniejszego poruszania się.

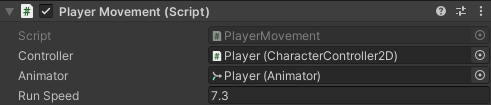


Fot. 3.27. Pozostałe linie kodu odnośnie wspomagacza

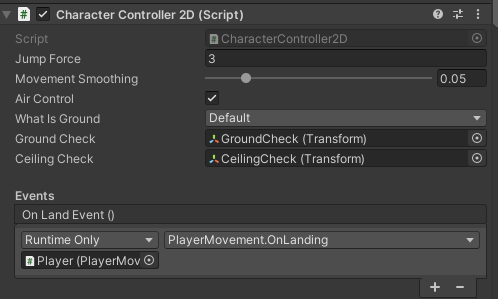
Przechodząc od razu do linii 32 widzimy tam odwołanie do funkcji odpowiedzialnej za ruch przekazujemy input horyzontalny oraz to czy jest kuca czy skacze (w naszym przypadku tylko może skakać, więc crouch równa się zawsze false). W ramach gotowej funkcji do chodzenia odbywa się ruch w danym kierunku.

Linia 17 to zebranie inputy horyzontalnego oraz ustawienie animatora na odpowiednią animację (po obliczeniach matematycznych->jeśli 0 to stoi, inaczej biega). Linia 21 odpowiada za wciśnięcie klawisza JUMP. Oczywiście na klawiaturze nie ma takiego przycisku, ale Unity oferuje nam napisanie opcji zamiast „W” to „JUMP”, ułatwia to programowanie. W przypadku wciśnięcia tego klawisza wykonuje się skok i animator pokazuje odpowiednią animację.

Podsumowując te skrypty. Nasze poruszanie się oraz cała postać to bardzo skomplikowana rzecz, która musiała być zaprogramowana na samym początku gry. Stąd też wzięło się uproszczenie w postaci gotowe skryptu. Widać wyraźnie, że w przypadku tego zagadnienia duża rolę odgrywa zintegrowana praca wszystkich rzeczy (animatora, Controllera oraz wspomagacza). Wszystko ukazuje nam piękny model poruszania się w naszej grze.



Fot. 3.28. Parametry wejściowe naszego wspomagacza



Fot. 3.28. Parametry wejściowe gotowe skryptu

Wracając na chwilę, by opisać zasadę działania GroundCheck i CeilingCheck. Opisywaliśmy w dziale odnośnie postaci, że składa się ona z kwadratowego oraz okrągłego Collidera2D. W naszym programie jak wspominaliśmy nie ma kucania. Jednak w przypadku zaimplementowania tego. CharacterController2D ukrywałby Collider2D górny i zostawiał tylko ten dolny (okrągły).

# 4. Problemy, plany rozbudowy, opinia oraz podsumowanie projektu

Zaczynając pracę w Unity każdy z nas miał problem, aby zainstalować środowisko i rozpocząć prace nad pierwszymi szczeblami projektu. Duży projekt wymaga dobrej organizacji pracy. Staraliśmy się robić w miarę możliwości wszyscy razem, aby każdy wszystko rozumiał co się dzieje w kodzie i jak to działa. Masa poradników odnośnie Unity była pomocna podczas pisania tego projektu.

Przechodząc do planów rozbudowy. Parę rzeczy było podanych w wyżej wymienionych liniach tekstu m.in.: ten ptak, który mógłby być drugim graczem.

Nie można ograniczać się do gotowej paczki tekstur, dlatego pewnie stworzyli byśmy własne tekstury oparte na modelu tych podstawowych.

Stworzyliśmy jeden poziom na potrzebę projektu, jednak nie widzimy przeciwieństw, aby zbudować ich więcej. Ważne, by mieć nowe pomysły na elementy gry i je wdrażać. Dodanie dźwięków i dialogów idzie wraz z nowymi poziomami.

Myślimy nad możliwością walki wręcz oraz nowymi przeciwnikami i bossami.

Nowe zadania i misje, aby urozmaicić rozgrywkę.

Cieszymy się, że skończyliśmy z dużą wiedza odnośnie tego środowiska i na pewno będziemy je wspominać bardzo przyjemnie.

Jeszcze raz serdecznie zapraszamy do pobrania i zagrania w naszą platformówkę:

<https://drive.google.com/file/d/1qNpeNMy0LSGONGufAyGxI4WGvVJ6Fx3f/view?usp=sharing>