Praca inżynierska – Adam Martenka

Strona tytułowa

Spis treści

[1. Wprowadzenie 3](#_Toc64557877)

[1.1. Zastosowanie 3](#_Toc64557878)

[1.2. Dlaczego Unity 4](#_Toc64557879)

[2. Przegląd istniejących rozwiązań 4](#_Toc64557880)

[3. Projekt 5](#_Toc64557881)

[3.1. Podstawowe elementy projektu w Unity 5](#_Toc64557882)

[3.2. Podstawowe elementy projektu Unity w kontekście mojej pracy 11](#_Toc64557883)

[4. Implementacja 20](#_Toc64557884)

[4.1. Etap pierwszy 21](#_Toc64557885)

[4.2. Etap drugi 27](#_Toc64557886)

# Wprowadzenie

Tematem mojej pracy dyplomowej jest stworzenie generatora wirtualnego terenu na potrzeby wizualizacji trójwymiarowej. Program stworzony został w technologii Unity w języku C#. Aplikacja umożliwia wygenerowanie dowolnego terenu oraz dodanie na nim elementów wody. Poza samą możliwością stworzenia terenu istnieje również możliwość dodania elementów flory i fauny do stworzonego świata. Dodane zwierzęta potrafią chodzić, szukają wody i pożywienia oraz rozglądają się, czy w pobliżu nie ma jakiegoś innego zwierzęcia, które mogli by upolować, lub przed którym powinni uciekać. Niektóre osobniki odżywiają się dodanymi przez nas roślinami a inne poszukują zwierząt, które mogliby zjeść. Poza roślinami, którymi odżywiają się niektóre istoty, istnieje możliwość dodania różnego gatunku drzew. Pełnią one jednak jedynie funkcje ozdobne.

Postanowiłem stworzyć taki generator, ponieważ chce pogłębiać swoją wiedzę w pisaniu gier w technologii Unity. Ponadto zawsze fascynowało mnie zachowanie dzikich zwierząt, a możliwość obserwacji ich zachowań w stworzonym przez nas terenie wydała mi się bardzo interesująca.

## Zastosowanie

Podstawową funkcjonalnością projektu jest generowanie terenu na potrzeby wizualizacji trójwymiarowej oraz obserwacja dodanych na niego zwierząt i roślin. Wygenerowany teren może posłużyć nie tylko do obserwacji ale również do paru innych rzeczy. Między innymi, stworzony świat może zostać użyty w innym projekcie. Napisany przeze mnie program może posłużyć jako teren do innej gry napisanej w Unity. Stworzony świat może również posłużyć jako gotowa scena do programów, które zajmują się tworzeniem trójwymiarowych grafik lub animacji takich jak 3ds Max lub Blender.

Stworzony przeze mnie program może być również użyty jako narzędzie do symulacji i wizualizacji. Czasami zdarza się, że zachodzi potrzeba zwizualizowania jakiegoś terenu na potrzeby budownictwa lub podczas projektowania ogrodów. Wtedy program może zostać użyty do odtworzenia istniejącego już terenu i sprawdzenia czy istnieje możliwość stworzenia zaplanowanych wybudowania jakiegoś budynku, osiedla czy stworzenia ogrodu zoologicznego.

Należy pamiętać, że projekt można jeszcze rozwijać. Po zaimplementowaniu dodatkowych elementów, aplikacja może posłużyć do wielu więcej rzeczy. W przypadku dodania elementów pasu startowego czy lądowiska oraz odpowiedniego modelu kokpitu i wzbogaceniu o skrypty umożliwiające lot, lądowanie i startowanie, projekt może posłużyć jako symulator lądowania samolotów czy helikoptera.

Tak naprawdę projekt może zostać użyty do wszelakich gier czy też symulacji, gdzie potrzebny jest otwarty świat.

## Dlaczego Unity

Projekt stworzony został przeze mnie w technologii Unity, czyli bardzo popularnym i rozbudowanym silniku gier. Umożliwia on tworzenie gier na platformę PC ale i na urządzenia mobilne. Najbardziej popularnym językiem w którym są one pisane jest C#. Niegdyś pisano również językach takich jak Boo czy UnityScript(czyli, czegoś stworzonego na wzór JavaScript). Obecnie jednak gry pisze się wyłącznie w języku C#. Najpopularniejszymi grami stworzonymi w Unity są: Angry Birds 2, Among Us czy Hearthstone.

Technologia ta może być kojarzona z niszowymi produkcjami ponieważ panel przedstawiający fakt, że gra stworzona jest na silniku Unity, który pokazywany jest przy starcie gry, wymagany jest w produkcjach, które generują przychód poniżej 100.000$ rocznie. W przypadku, gdy gra zarabia więcej producent silnika nie wymaga aby jego logo prezentowane było na starcie gry.

Do wykonania mojego projektu postanowiłem wykorzystać technologie Unity z wielu różnych przyczyn. Przede wszystkim chciałem aby narzędzie, z którego będę korzystał w żadnym stopniu nie ograniczało moich pomysłów. Silnik ten ma ogromnie możliwości i w realizacji aplikacji ograniczała mnie jedynie wyobraźnia, moje umiejętności programowania oraz stopień znajomości samego Unity.

Innym powodem, dla którego zdecydowałem się na użycie tej technologii jest fakt, że bardzo chciałbym pogłębić swoje umiejętności tworzenia gier w Unity ale i w programowaniu przy pomocy języka C#.

# Przegląd istniejących rozwiązań

# Projekt

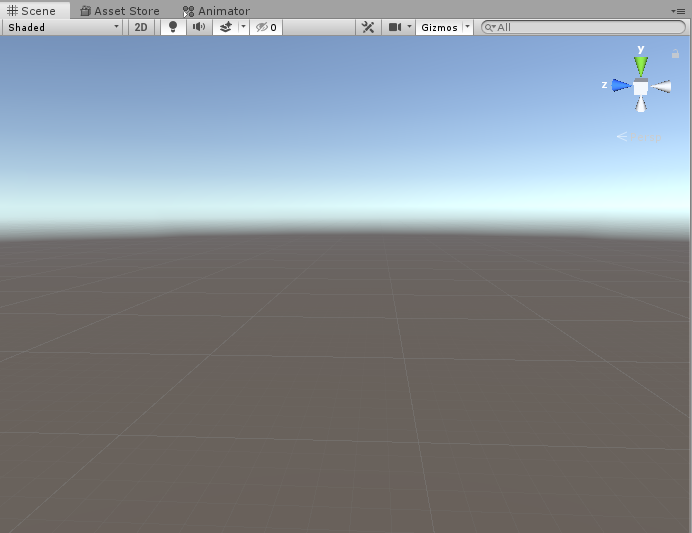
Każdy projekt stworzony w Unity ma kilka podstawowych składowych, bez których nie mógłby funkcjonować. Do stworzenia mojego projektu użyłem właśnie tych elementów, ale również dodanego przez Projektanta silnika sklepu w którym można, również za darmo, pobrać elementy świata przedstawionego, dzięki czemu nie musiałem wszystkich elementów tworzyć od zera. Pobrałem m.in.: wygląd zwierząt, drzew i krzewów.

W tym rozdziale postaram się omówić podstawowe elementy silnika a następnie, opiszę jak wyglądają one na przykładzie mojej aplikacji. Przedstawię powiązania między elementami w mojej aplikacji oraz w jaki sposób poszczególne skrypty działają na dane rzeczy.

## Podstawowe elementy projektu w Unity

Podczas tworzenia gry w Unity mamy wpływ na wiele rzeczy. Miedzy innymi możemy zdecydować, czy gra, którą chcemy tworzyć ma być w wymiarze 2D czy 3D, możemy wybrać edytor tekstu , w którym kod zamierzamy pisać lub zdecydować na jaką platformę ma zostać wydana nasza gra. Są jednak elementy, których użycie jest narzucone nam przed producenta silnika i aby móc stworzyć interesującą nas grę czy też symulacje jesteśmy zmuszeni ich użyć. Poniżej przedstawię te najważniejsze:

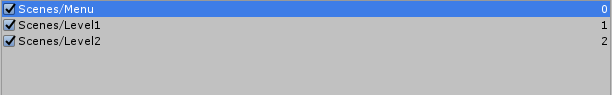
**Scena**



Rysunek Podstawowy widok sceny w Unity

Scena jest obszarem gry, na który można dodawać elementy, które chcemy dodać do naszej rozgrywki. Może być również wykorzystana jako niezależny od innych scen poziom gry lub można na niej stworzyć Menu, które wyświetlane będzie na początku gry. W projekcie może istnieć więcej niż jedna scena, a przechodzenie pomiędzy nimi może być zaprogramowane w skrypcie. Widoczna na Rys.1. scena, to widok podstawowy, który widzimy po uwożeniu gry.

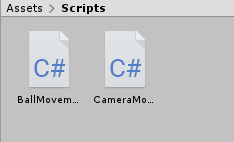
Stworzone sceny posiadają swój numer, co ułatwia zaprogramowanie płynnego przechodzenia między nimi.



Rysunek Sceny przedstawiające menu oraz dwa poziomy, które są kolejno ponumerowane.

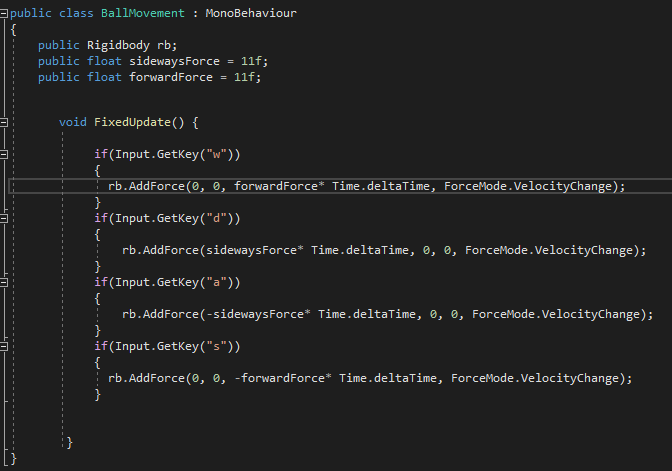
**Skrypty**

Skrypt, czyli kod napisany w języku C#, lub innym, który umożliwia nam pisanie w Unity. To dzięki nim mamy możliwość ingerowania w to, co dzieje się w naszej grze. To właśnie kod, który wpisujemy w tych plikach pozwala nam na zmienianie scen, ruch postaci lub ingerencje w dodanych przez nas elementów poziomu. Skrypty można tworzyć niezależnie od siebie ale najlepiej, aby każdy z nich odpowiadał za jedną rzecz, taką jak na przykład ruch, zmiana scen lub rozwój postaci.



Rysunek Przykładowe skrypty

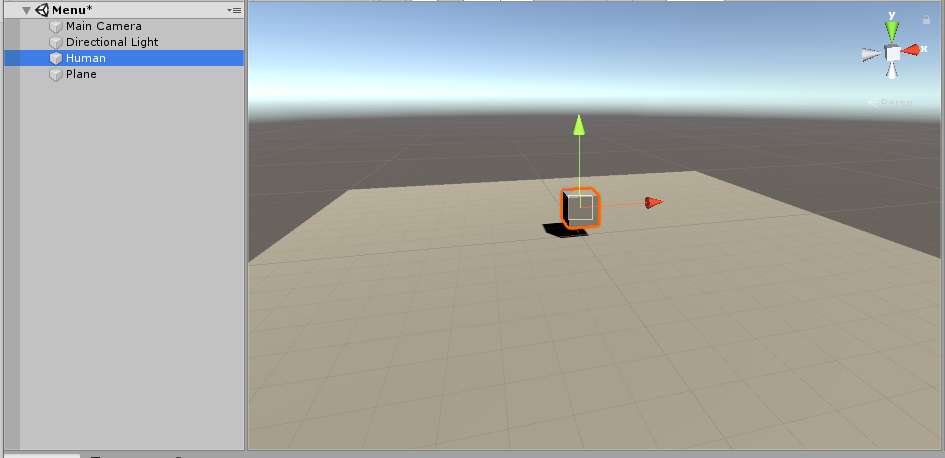
Na rysunku 4. widać zawartość zaprezentowanego powyżej skryptu BallMovement. Skrypt, napisany w języku C# odpowiedzialny jest za poruszanie się danego elementu za pomocą przycisków W, A, S i D. Skrypt ten został dodany do obiektu Ball.



Rysunek Fragment skryptu BallMovement

**GameObjects**

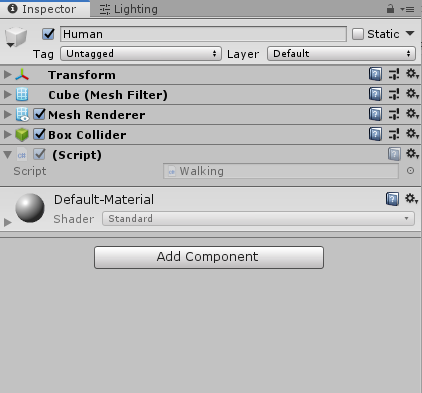
Są to elementy gry, które dodajemy do sceny. Funkcjonowanie ich zaprogramowane jest w skryptach. GameObject może mieć dowolną ilość skryptów, które mogą odpowiadać za każdą interakcje z światem stworzonym lub to co dany GameObject ma robić. Elementy te mogą być zwierzętami, drzewami, pojazdami, trawą czy skała a wszystko co mają robić można dzięki skryptom do nich przypisać.



Rysunek Dodany do sceny GameObject o nazwie Human

Dodane przeze mnie obiekty mają swoje wbudowane przez producenta silnika cechy, które można zobaczyć w Inspectorze. Po naciśnięciu na konkretny element na scenie, wyświetla się on w panelu bocznym. Właśnie tam można dodać, stworzone przez nas skrypty.

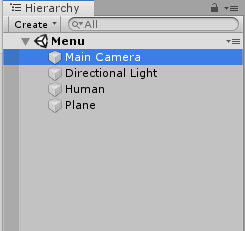
Do GameObjectu o nazwie Human został dodany skrypt o nazwie Walking, Rys.5.



Rysunek Podgląd cech i funkcjonalności GameObjectu o nazwie Human

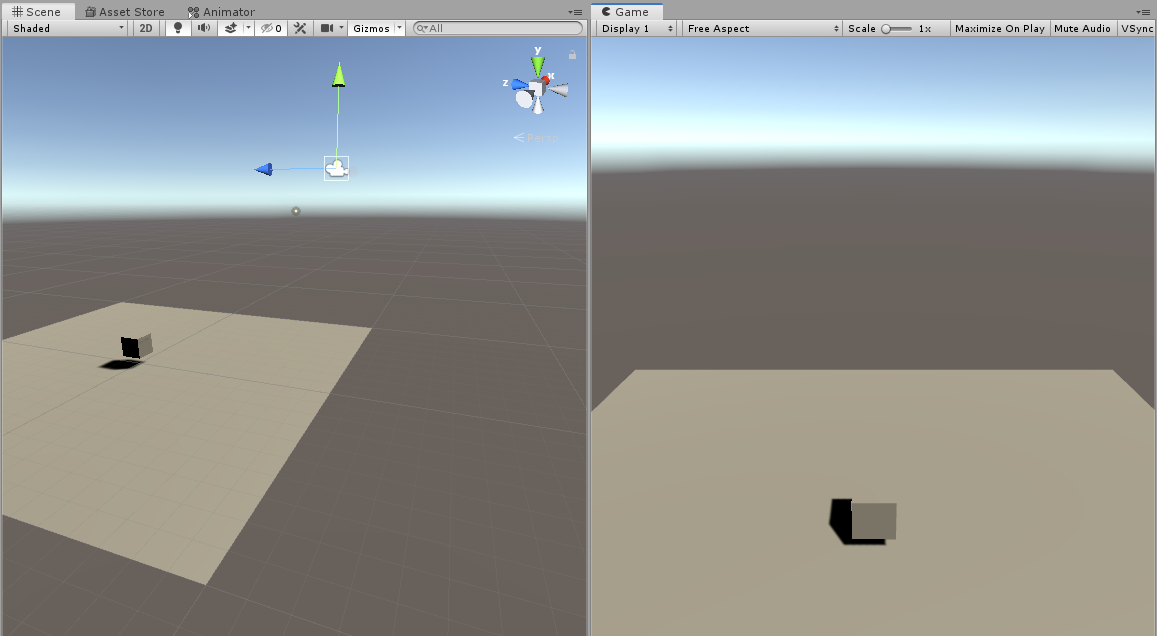
**Kamera**

W każdym projekcie Unity znajduje się kamera. Jest to jeden z elementów, jak scena, który jest dodany od momentu stworzenia projektu. Ten element, jest o tyle ważny, że gdyby nie on to nie widzielibyśmy w naszej grze niczego. W kadrze kamery znajdują się elementy, które widzi gracz po uruchomieniu gry.



Rysunek Kamera opisana jest pod nazwą "Main Camera" w głównej hierarchii projektu

Ustawienie kamery gracz może zmieniać dowolnie. W oknie Scene widzimy wszystkie elementy w stworzonym świecie. Można również w nim ustawić kamerę oraz kadr, który widoczny będzie w grze. W oknie „Game”, widzimy to, co znajduje się w oku kamery, a zarazem to co widzi gracz po uruchomieniu gry.



Rysunek W oknie "Scene" zaznaczona jest kamera a w oknie "Game" widać co jej kadrze.

Ruch kamery można modyfikować dowolnie w skryptach, co zaprezentuje w późniejszym etapie pracy.

## Podstawowe elementy projektu Unity w kontekście mojej pracy

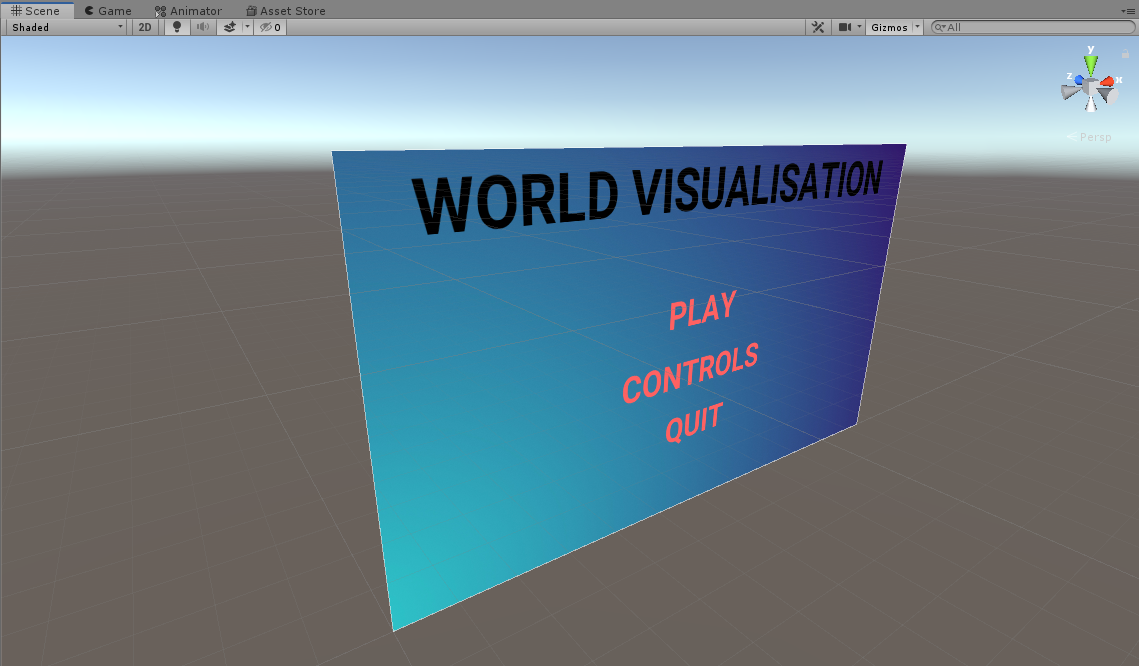
W tym podrozdziale omówię stworzony przeze mnie projekt, bazując na podstawowych elementach Unity, które opisałem w poprzednim podrozdziale. Dodatkowo opowiem o tym jak konkretne elementy funkcjonują i w jaki sposób na siebie działają. Poza tym, co opisałem wcześniej przedstawię dodatkowe składowe, które są kluczowe do funkcjonowania mojego projektu.

**Scena**

W stworzonym przeze mnie projekcie są dwie główne sceny. Są to: Menu oraz Game.

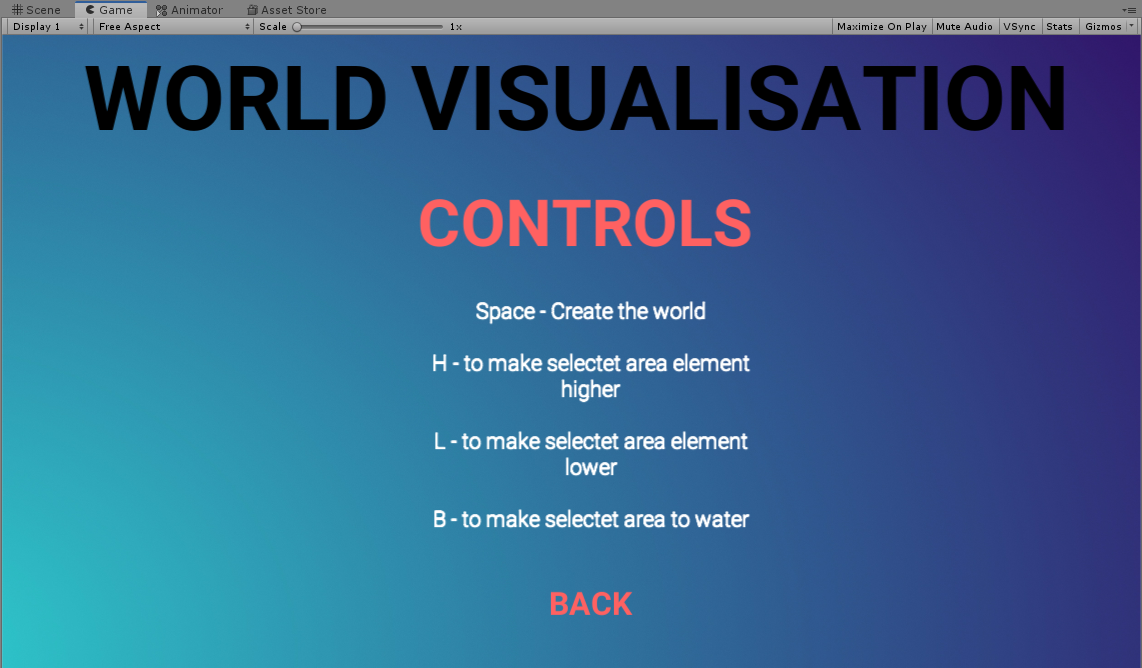


Rysunek Menu główne apliacji



Rysunek Menu główne z okna "Scene"

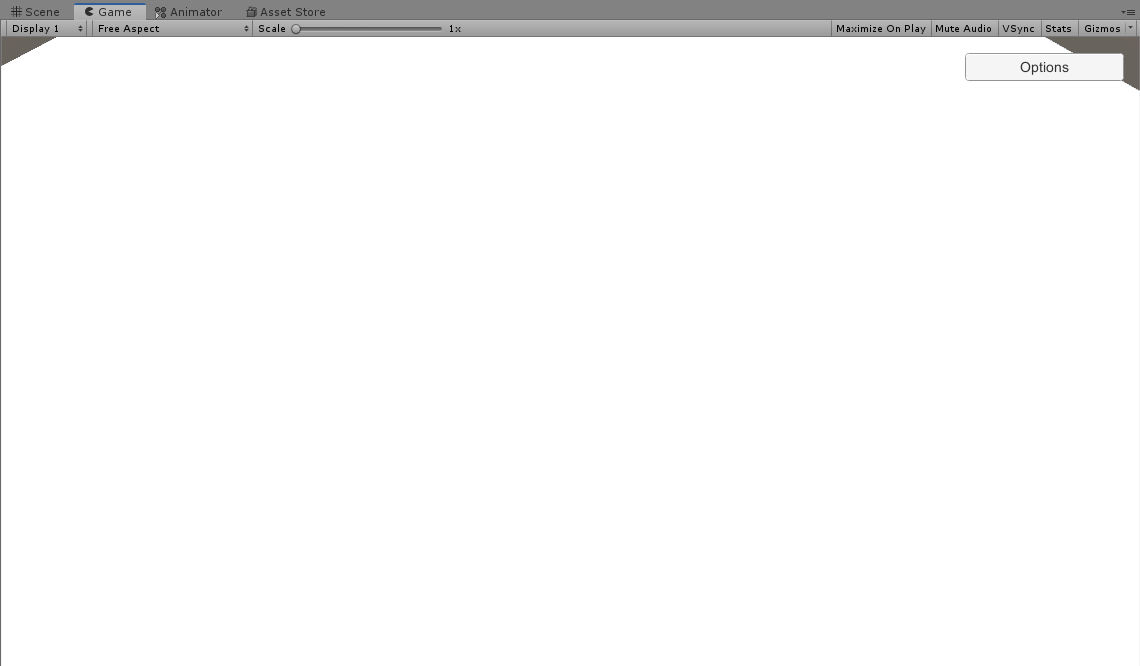
Pierwsza strona przedstawia menu główne, które widoczne jest na starcie aplikacji. Na samej górze menu głównego widzimy nazwę aplikacji a pod nią trzy przyciski. Po naciśnięciu pierwszego zostaniemy przekierowaniu do drugiej sceny i rozpoczniemy tworzenie elementów świata. Przycisk o nazwie „Controls” przenosi nas do drugiego panelu na tej scenie, na którym wytłumaczone jest sterowanie gry ( Rys. 10.). Ostatni przycisk odpowiada za wyłączenie aplikacji.



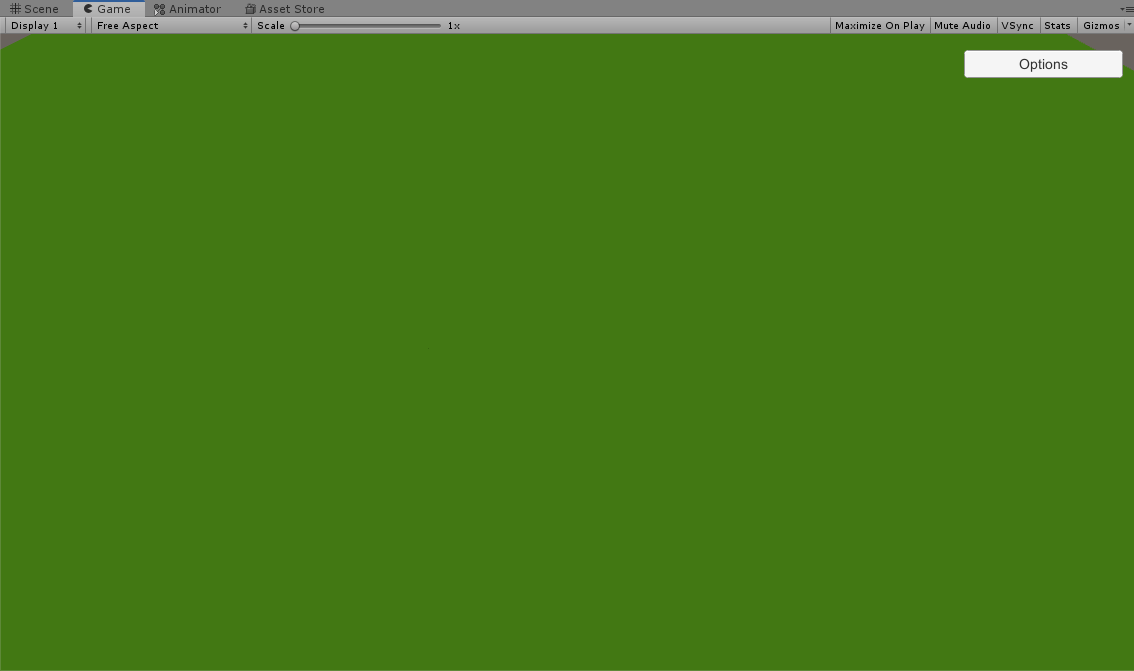
Rysunek Opcja "Controls" w menu głównym

Druga scena, która nazywa się „Game” odpalana jest po przyciśnięciu przycisku Play w menu głównym i to tam właśnie odbywa się cała funkcjonalność gry. Przejście pomiędzy scenami zostało zaprogramowane w skrypcie o nazwie „MainMenu”, którego dokładne funkcjonowanie opisze w dalszej części pracy.

Gdy przejdziemy do drugiej sceny naszym oczom ukazuje się sam generator. Na początku widzimy biały panel(Rys.12.), na którym jest możliwość tworzenia dowolnego terenu wirtualnego. Rozpoczęcie tworzenia następuje po naciśnięciu przycisku „Space” na klawiaturze. Wtedy to na białym panelu tworzy się zielona powłoka imitująca trawę(Rys.13.).

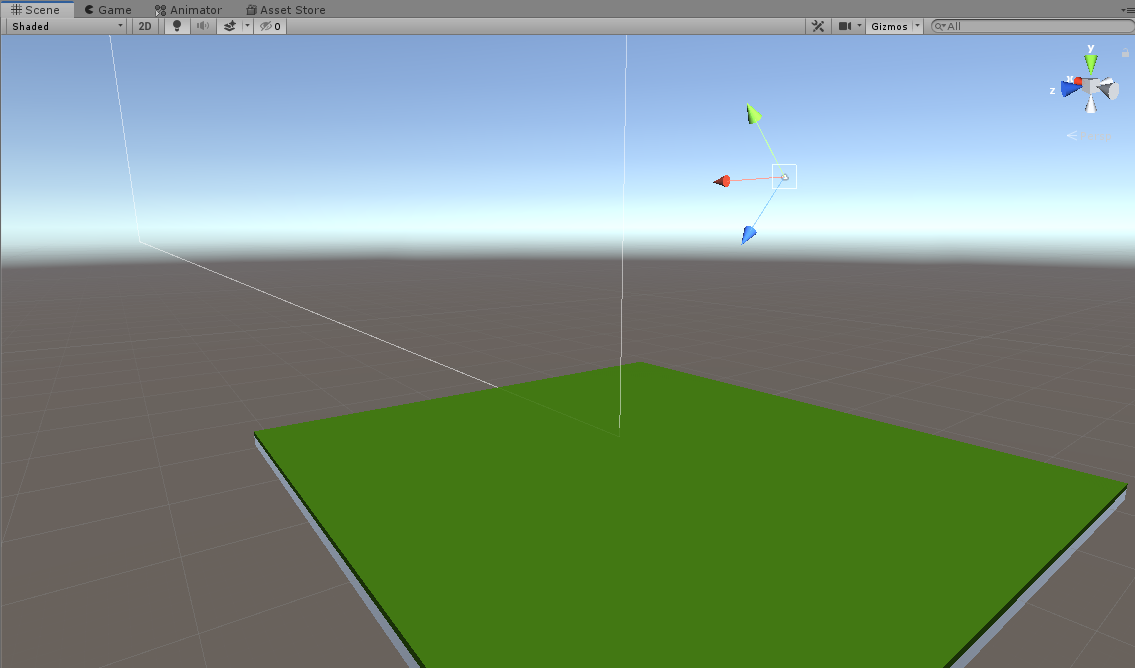


Rysunek Podstawa, na której tworzony będzie teren



Rysunek Na białym panelu powstaje trawa

To w jaki sposób zostało to zaprogramowane zostanie opisane w dalszej części pracy.

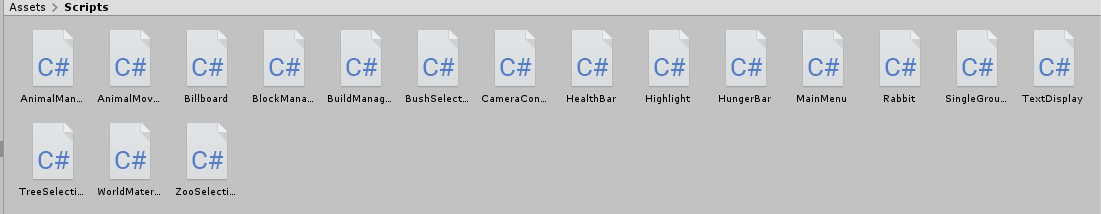


Rysunek . Widok na scenę drugą wraz z zaznaczoną kamerą

Istnieje możliwość zatrzymania, tworzenia terenu i wyjścia do menu głównego a co za tym idzie przejścia do sceny pierwszej.

**Skrypty**

W mojej aplikacji znajduje się wiele skryptów. Starałem się, aby każdy z nich odpowiadał za jedną konkretną rzecz albo chociaż jego działanie mieściło się w obrębie jednej funkcjonalności programu.



Rysunek Skrypty, znajdujące się w aplikacji

Działanie aplikacji można podzielić na dwa podstawowe etapy. Pierwszym z nich jest tworzenie terenu a drugim, jest dodawanie zwierząt i roślin czy drzew. Dlatego też skrypty w aplikacji można podzielić na te, które odpowiedzialne są za etap pierwszy oraz te, które usprawniają działanie etapu drugiego. Dla obu części tworzenia dodane są dodatkowe skrypty, które usprawniają elementy wizualne podczas jej tworzeni. Są też fragmenty kodu, które odpowiadają za elementy poboczne takie jak: ruch kamery, wyświetlanie teksty na ekranie, skrypty odpowiedzialne za interfejs użytkownika.

Najbardziej kluczowymi skryptami w mojej aplikacji są:

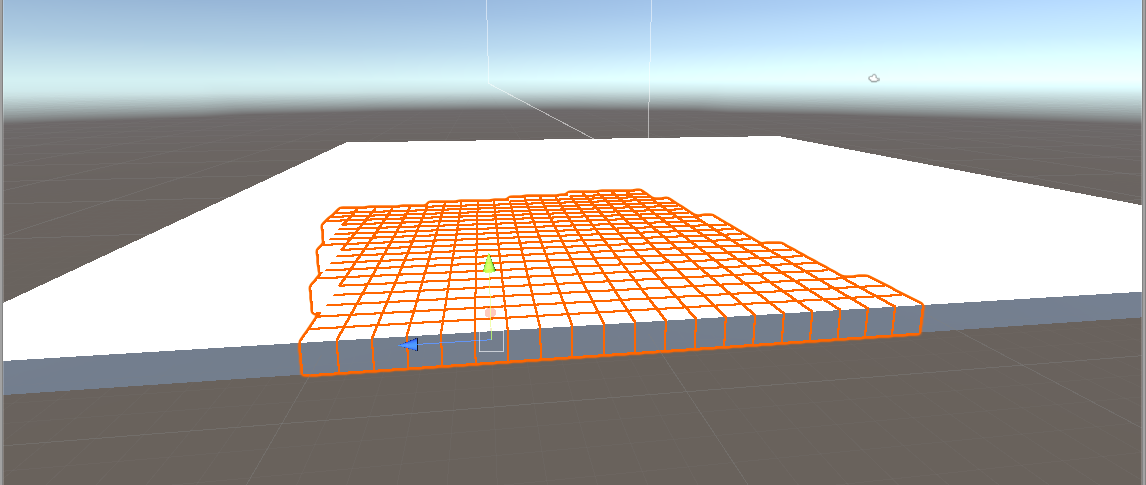
* BlockManager – bardzo ważny skrypt, w którym znajduje się większość funkcjim odpowiedzialnych za tworzenie terenu. Dzięki temu fragmentowi można modelować wysokość zaznaczonego terenu, dodawać wodę. W nim istnieje również fragment odpowiedzialny za tworzenie prostokąta umożliwiającego zaznaczenie znajdującego się w jego obrębie terenu.
* WorldMaterial- skrypt odpowiedzialny za stworzony teren. Dzięki niemu dodawane są konkretne elementy świata;
* BuildManager – skrypt odpowiada, za przekazanie informacji do skryptu „WorldMaterial”, jaki element zostanie na danym obszarze stworzony;
* AnimalMovement – w nim napisane jest cała mechanika ruchu zwierząt. Skrypt odpowiada za to gdzie udaje się zwierzę podczas swobodnego ruchu, gdy jest głodne lub gdy jest spragnione;
* CameraController – kod napisany w nim odpowiada za ruch kamery.

Dokładną funkcjonalność każdego skryptu opiszę w późniejszej części pracy.

**GameObjects**

Wszystko, co widzimy podczas odpalenia aplikacji jest obiektem gry. W tym podpunkcie postaram się opowiedzieć w jaki sposób zostały one wykorzystane do tego aby stworzyć ostateczną wersje projektu.

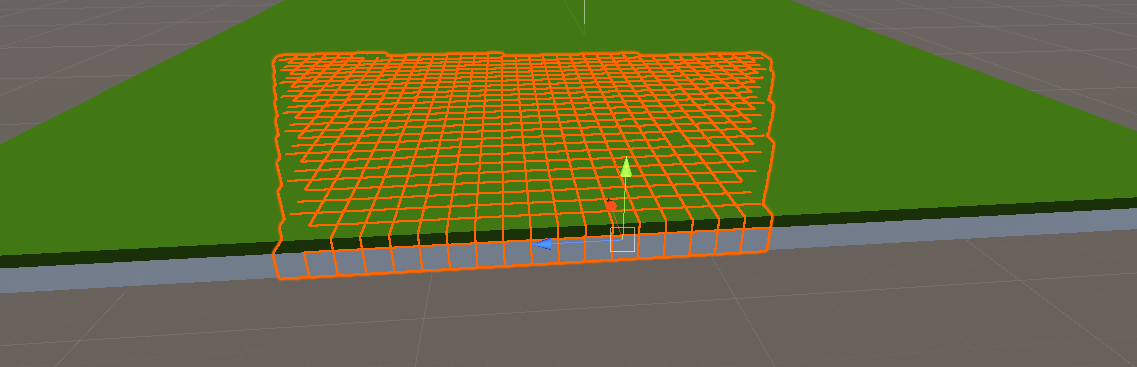
Z wielu obiektów gry składa się panel podstawowy, na którym to tworzony jest teren. Tak naprawdę panel, który widzimy na początku to wiele sześcianów ustawionych obok siebie tak by tworzyły spójną całość.



Rysunek Sześciany, które tworzą panel podstawowy

Zdecydowałem się na takie rozwiązanie ponieważ dzięki niemu możemy na każdym sześcianie stworzyć coś innego, a co za tym idzie istnieje większa swoboda podczas modelowania terenu.

Po wciśnięciu spacji na każdym sześcianie pojawia się „trawa” o wysokości połowy jednego elementu panelu podstawowego.

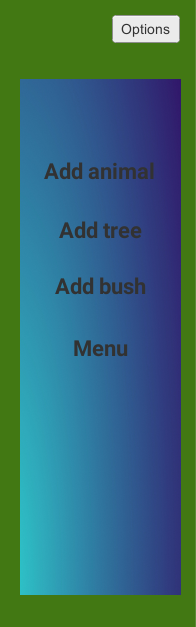


Rysunek Zielony element imitujący trawę na panelu podstawowym

Gdy już pierwszy, a zarazem kluczowy element, został dodany do panelu może rozpocząć się modelowanie terenu. W tym momencie można również dodać inne elementu gry, czyli zwierzęta i rośliny. Nie ma możliwości dodania elementu flory czy fauny do panelu podstawowego.

Bardzo ważnymi obiektami gry są również elementu interfejsu użytkownika. To z nich składa się cała pierwsza scena, oraz to dzięki nim istnieje możliwość dodania zwierząt i roślin to wygenerowanego terenu.

W prawym górnym rogu widzianego prze użytkownika ekranu jest przycisk. Po naciśnięciu na niego wyświetlany jest panel a na nim są przyciski, dzięki którym można dodać elementy flory i fauny na teren oraz można wyjść do menu głównego.



Rysunek Panel boczny

Po naciśnięciu któregoś przycisku zostanie wyświetlony kolejny GameObject. Znajdują się na nim obiekty gry, które możemy dodać do sceny

Na powyższych zrzutach ekranu widzimy kolejno: rośliny, którymi żywią się zwierzęta roślinożerne, drzewa oraz panel z wyborem zwierzęcia.

Wszystko widoczne na tych panelach można dodać do sceny i są to odrębne elementy gry.

Na poniższym screenie widzimy po jednym dodanym elemencie z każdej kategorii

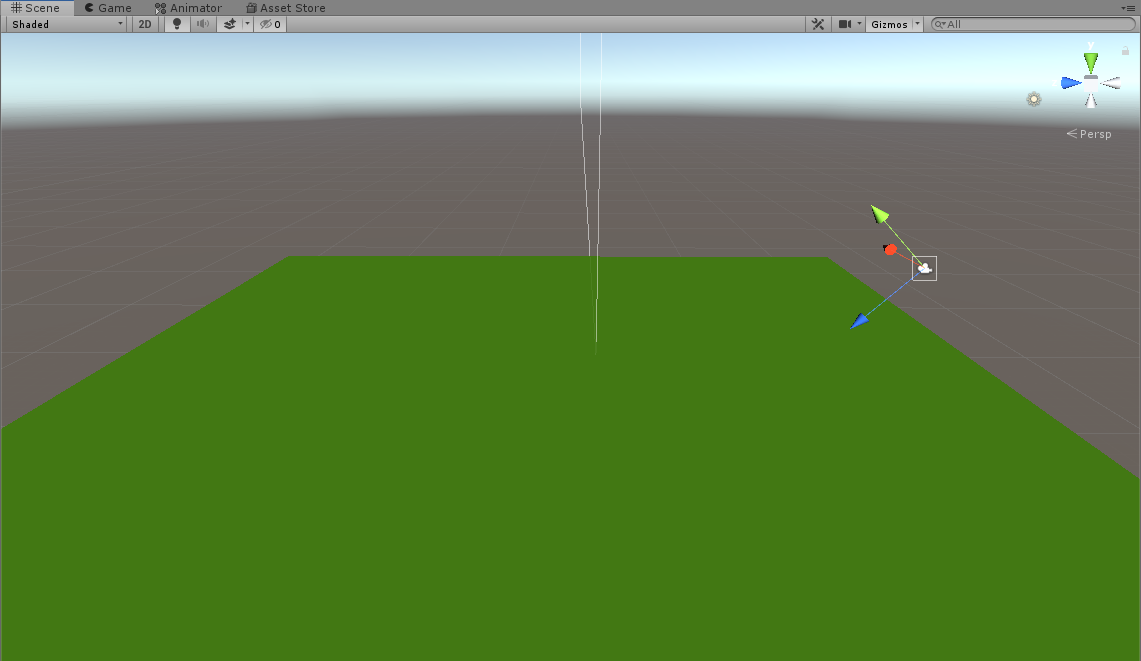


Rysunek Elementy świata

**Kamera**

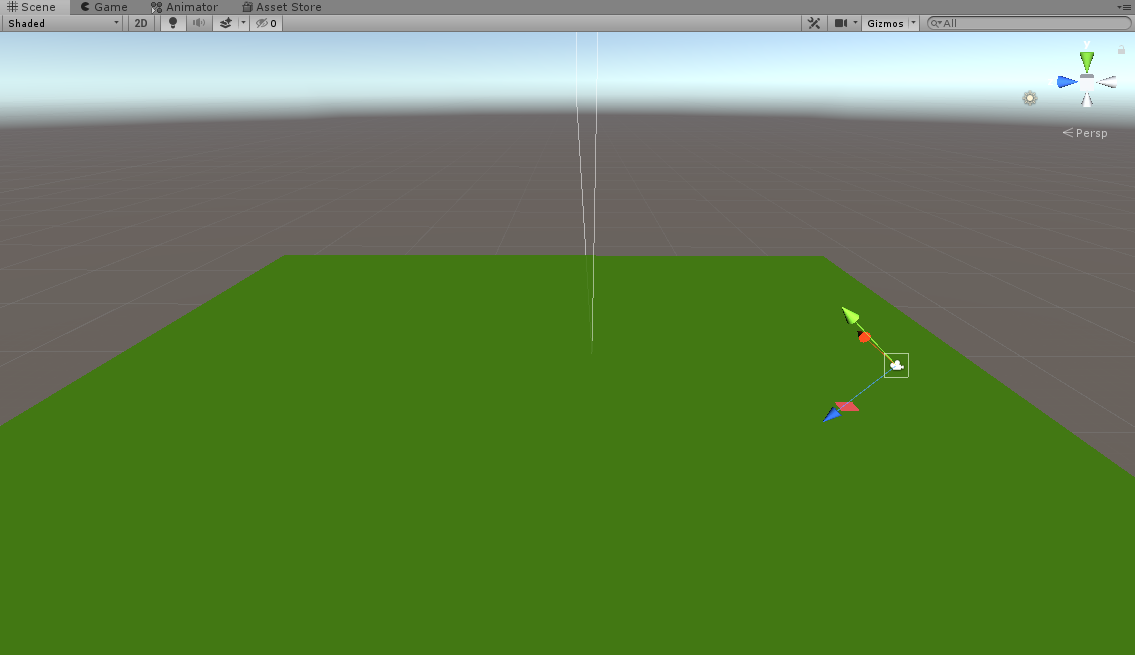
Tak jak napisałem we wcześniejszym etapie pracy, kamera ma za zadanie pokazanie fragmentu sceny, który będzie widoczny po uruchomieniu gry. W przypadku mojej aplikacji, jest ona umiejscowiona nad panelem podstawowym. Nie jest ona jednak umieszczona tam statycznie, ponieważ istnieje możliwość zmiany położenia kamery. Ruch ten jest ograniczony, aby nie było możliwości eksponowania tego, co jest poniżej panelu głównego.

Na poniższym screenie wiać jak ustawiona jest kamera na samym początku.



Rysunek Kamera na począku aplikacji

Natomiast ten screen prezentują jak blisko może przybliżyć się kamera podczas gry. Ruch jej został ograniczony w skrypcie, ponieważ w normalnych warunkach mogłaby ona przeniknąć przez podłoże.



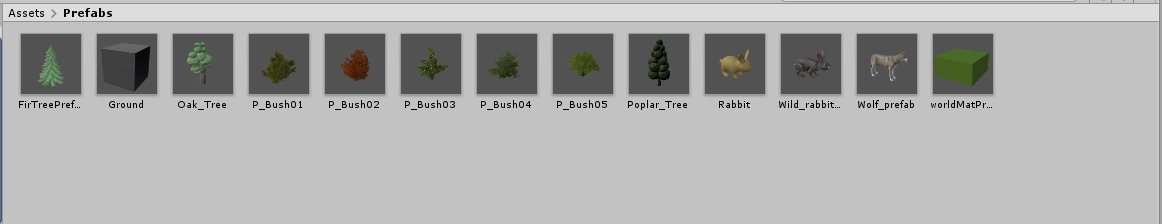
Rysunek Maksymalne przybliżenie kamery

Ujęcie z tak bliska jest bardzo przydatne w przypadku gdy chcemy dokładnie prześledzić ruch zwierząt lub zobaczyć wygląd stworzonych roślin czy drzew.

**Prefaby**

Prefaby to stworzone elementy, których kopie znajdują się w plikach aplikacji i które możemy użyć wiele razy w projekcie. Na przykład, gdy chcemy jakiś element dodawać wielokrotnie do gry to nie trzeba tworzyć go za każdym razem na nowo, lecz można zapisać go jako Prefab i dodawać do gry jako kopie. Nazwa tego elementu gry z reguły kończy się słowem ”Prefab”.

Prefaby, które znajdują się w moim projekcie widoczne są na rysunku 22.



Rysunek Prefaby w projekcie World Visualisation

# Implementacja

W tym rozdziale opisze kolejne etapy tworzenia programu. Przedstawię w jaki sposób zostały stworzone kluczowe dla funkcjonowania aplikacji elementy oraz przedstawię największe problemy jakie mnie spotkały podczas procesu tworzenia i w jaki sposób je rozwiązałem. Omówię również najważniejsze fragmenty kodu i przedstawię elementy silnika Unity, które ułatwiły mi tworzenie generatora.

Proces tworzenia podzielę na dwa etapy. W pierwszym opiszę to w jaki sposób i przy użyciu czego, stworzyłem możliwość generowania terenu, a w drugim przedstawię w jaki sposób zaprogramowałem możliwość dodawania flory i fauny oraz omówię skrypty, które odpowiadają za poruszanie się zwierząt.

## Etap pierwszy

Tworzenie pierwszego etapu rozpocząłem od dodania pojedynczego elementu panelu podstawowego. Następnie dodany przeze mnie sześcian wielokrotnie zduplikowałem, aż do uzyskania sporych rozmiarów panelu. Wszystkie stworzone elementy dodałem do jednego pustego GameObjectu i nazwałem go WholeGround.



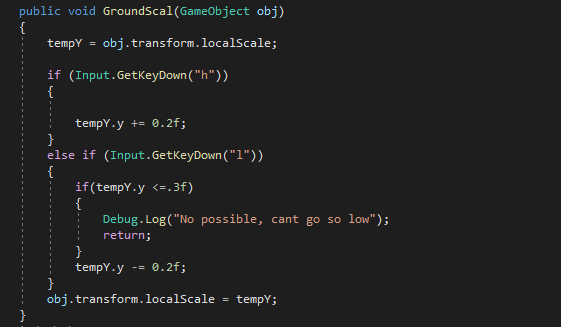
Rysunek GameObject o nazwie WholeGround

Gdy miałem już panel podstawowy postanowiłem utworzyć element, który umożliwiać będzie modelowanie terenu. Stworzony on został bardzo podobnie do WholeGround, czyli dodałem zwykłą kostkę oraz dodałem ją do pustego GameObjectu o nazwie worldMatPrefab. Element ten dodałem do folderu z Prefabami.

Kiedy miałem już potrzebny mi element postanowiłem dodać skrypt, który nazwałem BlockManager. Następnie napisałem mechanizm, który umożliwia zwiększanie i zmniejszanie danego elementu.

Tę funkcjonalność zawarłem w funkcji GraoundScal, która przyjmuje GameObject(w tym przypadku worldMatPrefab) a następnie przy użyciu przycisku „h” zwiększa go o 0.2 jednostki, oraz zmniejsza o 0.2 jednostki przy naciśnięciu przycisku „l”. Dodatkowo funkcja zawiera blokadę, która uniemożliwia zmniejszanie elementu gdy jego rozmiar jest mniejszy bądź równy 0.3 jednostki.

Na poniższym zrzucie ekrany widać całą funkcję GroundScal.



Rysunek Funkcja GroundScal w skrypcie BlockManager

Obiekt ten ma specjalny materiał, który zmienia swój kolor w zależności od tego jaka jest jego wysokość. Gdy jest on utworzony ma kolor zielony, który imituje trawę. Następnie im poziom jest wyższy kolor ten robi się coraz ciemniejszy, do momentu, gry przypomina skały górskie.

**Dodawanie do list**

Następnie postanowiłem stworzyć funkcjonalność, która umożliwiłaby dodawanie worldMatPrefab na WholeGround, aby wyglądało to jak element ziemi, który można dowolnie zwiększać i zmniejszać. W tym celu, w skrypcie BlockManager postanowiłem stworzyć dwie listy GameObjectów, o nazwach: „created” oraz „marked”, w których odpowiednio znajdowałby się elementy stworzone oraz te, które zostały zaznaczone i których wysokość można modyfikować.

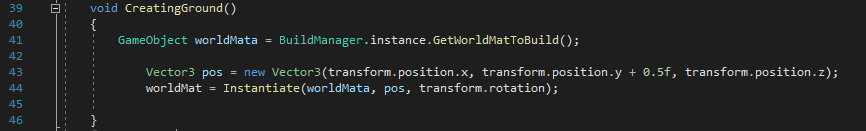


Rysunek Listy marked oraz created

**Lista „created”**

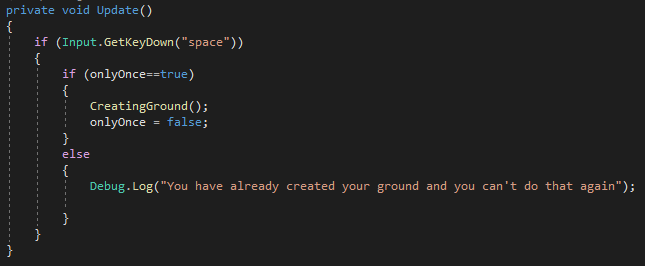
Tworzenie worldMatPrefab odbywa się w jeszcze innym skrypcie zwanym SingleGround, w którym znajduje się tylko właśnie ta funkcjonalność. Działa ona następująco:

Na każdy element panelu podstawowego zostaje dodany worldMatPrefab. Funkcjonalność ta znajduje się w funkcji o nazwie CreatingGround.



Rysunek Funkcja CreatungGround w skrypcie SingleGround

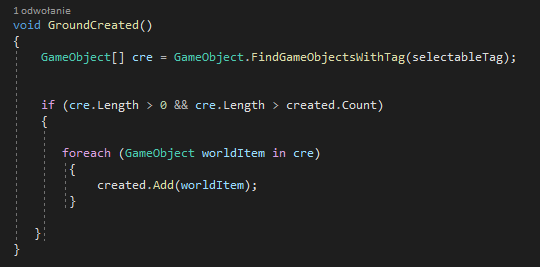
Funkcja CreatingGround zostaje wywołana w momencie, gdy użytkownik naciśnie przycisk „Space” na klawiaturze. Może to zrobić tylko raz, ponieważ gdy spróbuje zrobić to ponownie zostanie wyświetlony komunikat, że materiał już został stworzony i nie można go dodać drugi raz. Wszystko to znajduje się w funkcji Update.



Rysunek Funkcja Update w skrypcie SingleGround

Zmienna onlyOnce, którą widać na screenie powyżej pełni tutaj bardzo kluczową rolę, ponieważ to dzięki niej tworzenie ziemi odbywa się tylko raz. Jest to zmienna typu bool i gdy spacja zostanie naciśnięta zmienia się ona w „false”. CreatingGround jest wywoływanie tylko w momencie gdy onlyOnce ma wartość „true”. Wartość ta nie jest edytowana w żadnym innym fragmencie tego skryptu.

Kiedy ziemia została już stworzona, dodawanie jej do listy created odbywa się z powrotem w skrypcie BlockManager. Konkretnie ma to miejsce w funkcji GroundCreating.



Rysunek Funkcja GroundCreated w skrypcie GroundCreating

Funkcjonowanie skryptu:

Do tabeli GameObjectów o nazwie cre, dodawane są wszystkie elementy z tagiem selectableTag. Tag ten mają wszystkie elementy worldMatPrefab. Następnie sprawdzane jest czy długość „cre” jest większa od 0 oraz czy jest ona większa od liczebności listy created. Ma to na celu zatrzymanie dodawania elementów, które są już w liście. Kolejnym etapem jest dodanie każdego elementu z cre do created za pomocą pętli foreach, przy użyciu zmiennej pomocniczej worldItem.

Kiedy wszystkie stworzone elementy ziemi są już w liście created, należało zabrać się za stworzenie funkcjonalności, która miała na celu zaznaczać elementy.

**Lista „marked”**

Elementy, które znajdują się w liście „marked” są tymi, których poziom można zwiększać i zmniejszać przy użyciu przycisków „h” oraz „l”. Funkcjonalność ta umożliwia modelowanie dokładnie takiego terenu jaki chcemy.

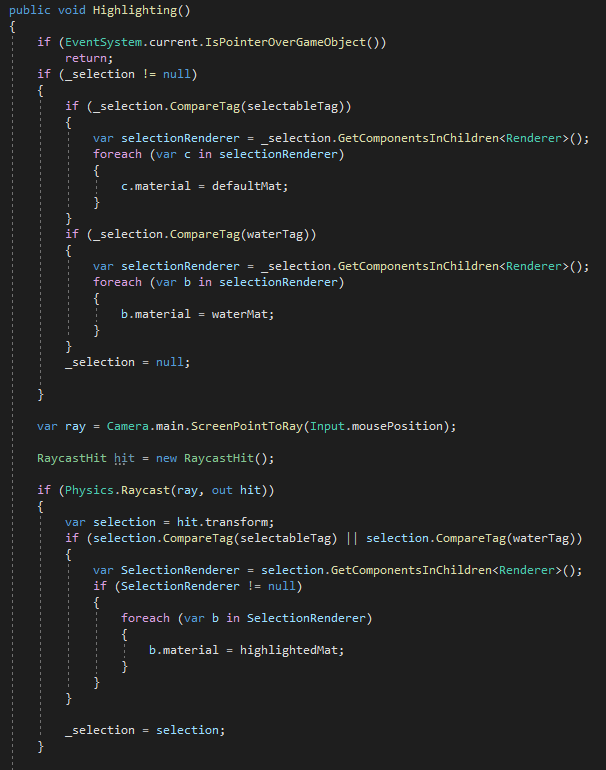
Zaznaczanie odbywa się na dwa sposoby: Po naciśnięciu interesującego nas konkretnego elementu oraz zaznaczeniu większej ilości elementów przy użyciu pola zaznaczania. Elementy znajdujące się w jego obrębie są dodawane do listy „marked”

Fragmenty ziemi, które są zaznaczone, tymczasowo zmieniają swój kolor na czerwony. To samo dzieje się z elementami, nad którymi obecnie znajduje się kursor. Funkcjonalność ta stworzona została w funkcji Highlighting.

Zanim będzie wykonana funkcja Highlighting, program sprawdza, czy lista created nie jest pusta.



Rysunek



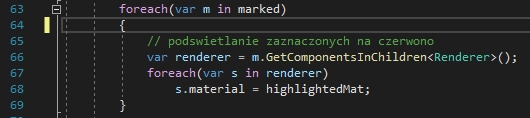
Rysunek Funkcja Highlighting

Działanie funkcji:

Podstawowa funkcjonalność odbywa się dzięki tak zwanym Raycastom, które pobierają informacje na temat obiektu, który znajduje się na jego drodze. W Przypadku tej funkcji działanie raycastu zostało przypisane do pozycji myszki. Dzięki temu można pobrać informacje na temat obiektu na który wskazuje obecnie kursor.

Kolor zmieniany jest na czerwony dla obiektów o tagach „SelectableTag” oraz „waterTag” co widać w linijce … . Jeśli myszka zejdzie z danego obiektu kolor wraca na ten, który miał przed najechaniem. Jeśli była to woda to obiekt zmienia kolor na waterMat(linijka …), a jeśli był to standardowy klocek kolor jest zmieniany na defaultMat(linijka ...).

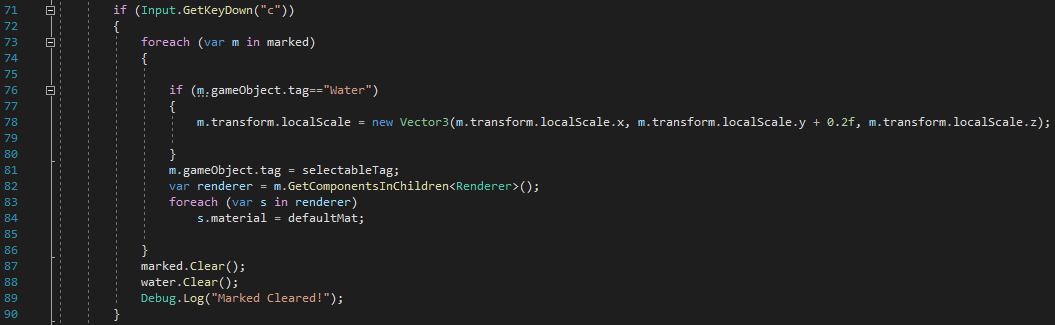
Zmiana koloru wszystkich zaznaczonych elementów na czerwony odbywa się w funkcji Update.



Rysunek Zmiana koloru zaznaczonych elementów

Za pomocą pętli foreach, wszystkie elementy, które znajdują się w liście marked zostają zmienione na czerwono.

Jeśli zostanie naciśnnięty przycisk „c” wszystkie elementy z marked zostaną usunięte a kolor z czerwonego zmieni się na taki jaki miały poprzednio. Ta funkcjonalność została stworzona za pomocą poniższego kodu



Rysunek Usuwanie elementów zaznaczonych

**Pole zaznaczania**

Zasada działania tego pola jest taka sama, jak w przypadku pola zaznaczania na pulpicie w systemach operacyjnych z rodziny Windows. Pole tworzy się za pomocą lewego przycisku myszy i przeciągnięcia w dowolną stronę kursora. Wszystko co znajduje się w obrębie pola zostaje zaznaczone(rys.32.).



Rysunek Pole zaznaczania

Widoczne pole jest elementem UI, które aktywowane jest gdy, użytkownik naciśnie lewy przycisk myszy w jednym miejscu, a w innym zostaje on puszczony. Na podstawie tych punktów obliczane jest pole, i wszystkie elementy znajdujące się w nim zostają zaznaczone. **Sprawdzić!!!**

Możliwość zrobienia wody z zaznaczonych elementów i zmienienia boola na isWater.

## Etap drugi

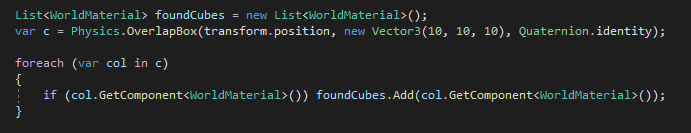
Projektowanie drugiego etapu zdominowała praca nad zachowaniem pierwszego dodanego do projektu zwierzęcia, czyli królika. Najtrudniejszym zadaniem było zaprogramowanie ruchu, czyli, swobodnego chodzenia, unikania przeszkód, szukania i podchodzenia do pokarmu i wody. Chciałem aby ruch odbywał się płynnie, a samo szukanie celu przez zwierzę było odpowiednio zoptymalizowane.

Ten etap rozpocząłem od znalezienia wyglądu zwierzęcia w sklepie z Assetami. Pozwala on na kupno bądź pobranie za darmo elementów, które można wykorzystać w projekcie. Ja postanowiłem kupić paczkę z wieloma różnymi zwierzętami wraz z animacjami poruszania się.

Gdy miałem już wygląd oraz animacje wziąłem się za programowanie skryptu, który umożliwiałby poruszanie się zwierzęcia. Chciałem aby skrypt był uniwersalny dla każdego zwierzęcia dlatego nie dałem do niego żadnych cech typowych dla królika.

Skrypt ten nazywa się AnimalMovement i jego kluczową funkcjonalnością jest analizowanie tego co znajduje się dookoła zwierzęcia i stwierdzenie, czy jest to miejsce do którego można iść lub czy dookoła znajduje się woda, gdy istota jest spragniona lub czy gdzieś w okolicy znajduje się pożywienie.

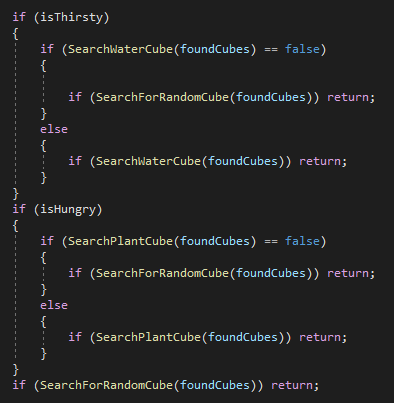
Skanowanie pobliskich elementów świata odbywa się w funkcji SearhCubes().



Rysunek Elementy umożliwiające poszukiwanie zwierzęcia

Na powyższym screenie z wspomnianej funkcji widać fragment kodu, który pobiera wszystkie elementy znajdujące się o 10 jednostek w każdym kierunku świata od zwierzęcia i dodaje się je do Listy foundCubes. Odbywa się to za pomocą metody Physic.OverlapBox. Tworzy ona trójwymiarowe pudełko i pobiera informacje o elementach znajdujących się w jego obrębie. W powyższym przypadku pobiera ona trzy zmienne. Pierwsza, to punkt centralny pudełka, w moim przypadku jest to dokładna pozycja zwierzęcia. Druga określa jego wielkość a trzecia odpowiada za rotacje.

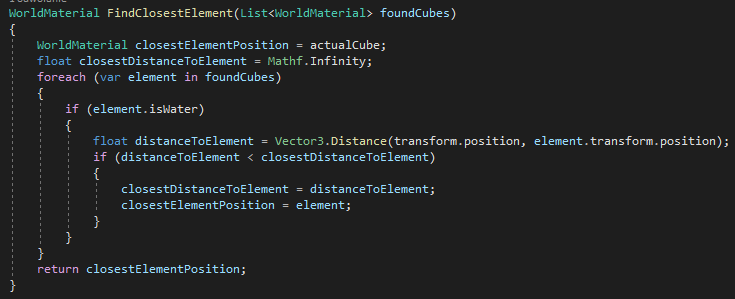
Następnie w tej samej funkcji sprawdzane jest, czy dane zwierzę jest spragnione lub głodne. Jeśli ani jedno ani drugie zwierzę wyszukuje wolnego miejsca, gdzie nie ma wody, rośliny, drzewa oraz sprawdza, czy nie jest ono za wysoko lub za nisko. Ta ostatnia czynność sprawdzana jest w innym miejscu i czynność ta wykonywana jest za każdym razem, gdy zwierzę chce się gdzieś przemieścić. Jeśli jednak istota jest głodna lub spragniona wyszukuje w swoim pobliżu wody lub pożywienia. Gdy nic w okolicy nie znajdzie to szuka innego, wolnego miejsca z którego będzie mógł się „rozejrzeć” za tym czego potrzebuje.



Rysunek Fragment funkcji SearhCubes

Funkcje SearchWaterCube oraz SearchPlanCube są to funkcje typu bool, które zwracają fałsz, gdy nie uda im się znaleźć wody ani jedzenia.

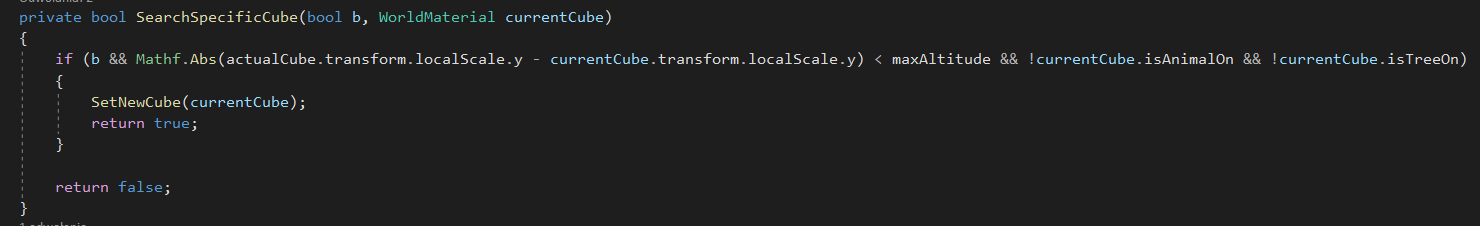
W przypadku poszukiwania wody, ważne jest aby zwierzę szło, do tej, która jest najbliżej niego, ponieważ jeśli wybierze element, który jest wodą ale znajduje się dalej od brzegu, to i tak do niego by poszedł i przez to chodziłby po wodzie, która jest na drodze do jego celu. Aby temu zapobiec napisałem funkcję FindClosestElement, która zwraca najbliższy element który jest wodą i znajduje się w zasięgu zwierzęcia.



Rysunek Funkcja FindClosestElement

Funkcja dla każdego elementu z listy, który jest wodą, sprawdza dystans między nią a aktualną pozycją w której znajduje się zwierzę. Przy pomocy zmiennych pomocniczych wyszukuje najbliższą kostkę i ustala ją jako najbliższy element.

Gdy funkcja znalazła już najbliższą wodę, element ten jest wysyłany do funkcji SearchSpecificCube, która sprawdza, czy znaleziony element nie jest za wysoko czy za nisko, lub czy nie znajduje się na nim drzewo lub krzew.



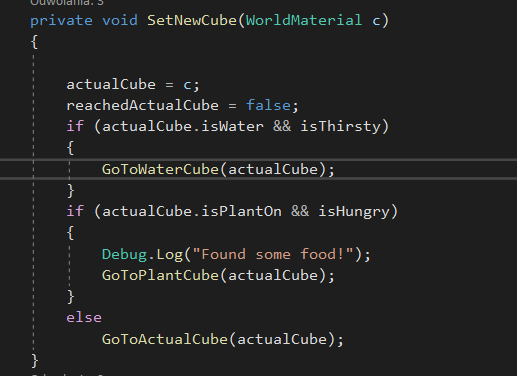
Rysunek Search specific cube

Aby dokonać sprawdzenia, czy klocek do którego zmierza postać jest na odpowiedniej wysokości należy odjąć od siebie wysokość kostki do której chce podążyć zwierzę od wysokości kostki na której obecnie znajduje się istota. Wynik tego działania zmieniany jest na wartość bezwzględną i sprawdzany, czy jest mniejszy od wartości ustalonej w zmiennej maxAltitude, której wartość to 0.5.

Działanie to oznacza, że jeśli miejsce do którego zmierza zwierze jest mniejsze lub większe od 0.5 jednostki to nie uda się do niego. Aby istota poszła na wybrane miejsce pozostałe elementy warunku if muszą zostać spełnione.

Pozostałe elementy ifa sprawdzają, czy docelowe miejsce nie jest drzewem, lub czy inne zwierzę nie zmierza już w tamtym kierunku

Gdy spełnione zostaną wszystkie wymogi z funkcji SearchSpecificCube, dana kostka zostanie wysłana do funkcji SetNewCube, która sprawdza, czy kostka jest wodą i zwierzę jest spragnione lub czy kostka jest rośliną i zwierzę jest głodne. Gdy zostaną spełnione któreś z tego zestawu wymagań to istota powędruję do tego miejsca w celu napicia się lub najedzenia.



Rysunek Funkcja SetNewCube

Jak widać na rys.37, w przypadku gdy zwierzę jest głodne i kostka jest jedzeniem zostanie ona wysłana do funkcji GoToPlantCube a w przypadku gdy dane miejsce jest wodą i istota jest spragniona to funkcja, która zostaje uruchomiona to GoToWaterCube. Jeśli żadne z tych wymagań nie zostało spełnione to zwierzę udaje się do wolnej kostki za pomocą funkcji GoToActualCube.

Powyższe funkcję umożliwiały znalezienie odpowiednich kostek ale teraz przedstawię kluczowe funkcje umożliwiające podchodzenie do rośliny oraz wody.

**Funkcja GoToWaterCube**

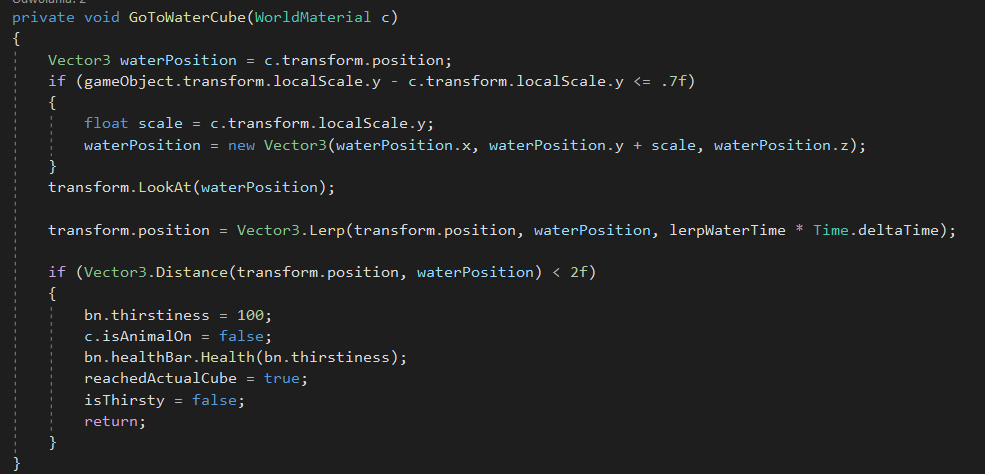
Gdy do funkcji GoToWaterCube zostanie przekazana kostka oznacza to, że jest ona wodą oraz znajduje się na odpowiedniej wysokości dla zwierzęcia. Jednak jest jeszcze parę rzeczy, które muszą zostać zrobione zanim zwierzę uda się do upragnionej wody.

Podstawową rzeczą jest dostosowanie wysokości kostki. W tej funkcji dokonywane jest podobne sprawdzenie jak w przypadku funkcji SearchSpecificCube, jednak tym razem wartość, której nie może przekroczyć wynik odejmowania wynosi 0.7 jednostki. Gdy wymaganie zostanie spełnione wartości y pozycji, w której znajduje się punkt docelowy i wysokości klocka są do siebie dodawane.

Takie coś jest konieczne, ponieważ zwierzę udaje się do konkretnej pozycji a ta jest zawsze taka sama. W przypadku zwiększenia wysokości klocka podczas modelowania terenu, jego pozycja nie zmienia się dlatego należy pobrać jego wysokość i dodać ją do wartości y aby zwierzę nie próbowało pójść w głąb ziemi.

Wartość ta przypisywana jest to zmiennej waterPosition, następnie zwierzę spogląda w kierunku wody i za pomocą wbudowanej w Unity funkcji Vector.lerp udaje się ono do wyznaczonej wody.

Jeśli istota znajduje się 2 jednostki od pozycji wody to w tym momencie rozpoczyna się picie. Zwierzę zostaje od razu napojone do maksymalnej wartości. Po napiciu się dane dane miejscie już nie jest celem zwierzęcia, dlatego zmienna typu bool o nazwie isAnimalOn zostaje zmieniona na false. Pasek napojenia zwierzęcia zostaje zaktualizowany oraz inne zmienne pomocnicze zostają zmienione co oznacza to, że zwierzę już jest napite i może szukać innego miejsca docelowego.



Rysunek GoToWaterCube

**Funkcja GoToPlantCube**

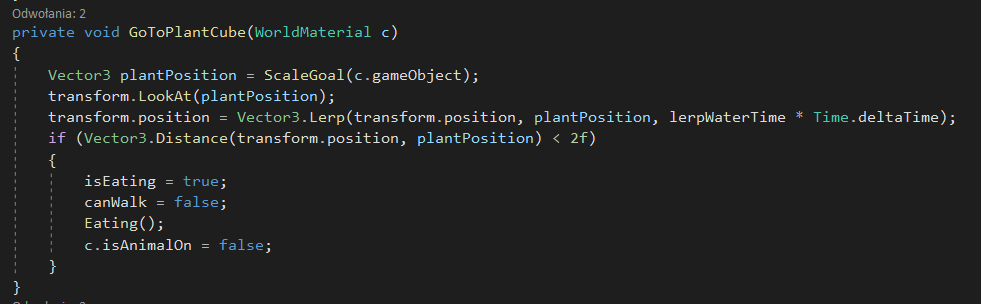
Funkcja ta jest bardzo podobna do tej, którą opisałem powyżej, jednak różnią się tym, że do wyskalowania pozycji użyłem dodatkowej funkcji, której nie można było użyć w przypadku skalowania wody oraz samo jedzenie odbywa się w innej, przeznaczonej do tego funkcji.

Zacznę od opisania funkcji ScaleGoal, która odpowiada za ustalenie odpowiedniej wysokości elementu do którego zmierza zwierzę. W tym przypadku można użyć funkcji bardziej uniwersalniej, ponieważ podczas dodawania rośliny do świata, wysokość elementu ziemi nie jest zmniejszana jak w przypadku tworzenia wody.

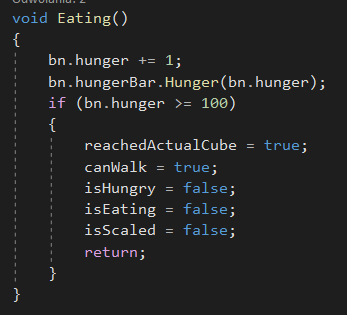
Zadaniem tego fragmentu kodu jest pobranie wartości y z podanego do niej obiektu gry. Następnie wysokość ta jest po prostu dodawana do no pozycji na osi y. Aby funkcja zwiększała pozycje tylko raz na jej końcu odbywa się sprawdzenie, czy pozycja jest taka sama jak suma skali i pozycji. Jeśli jest, oznacza to, że skalowanie już się odbyło i zwracana jest ta sama wartość jaka została przyjęta jeśli nie to zwracana jest wartość zeskalowana.

Inną funkcją pomocniczą w tym fragmencie kodu jest Eating. Odpowiada ona za samo jedzenie. Odbywa się to jednak stopniowo a nie od razu ja w przypadku picia.

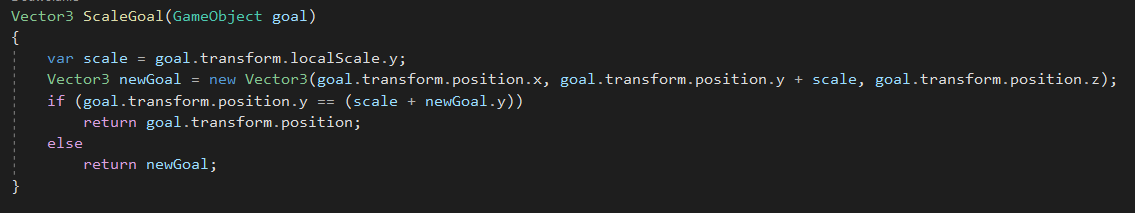
Poniżej zaprezentuje wszystkie trzy funkcje.



Rysunek GoToPlantCube



Rysunek Eating



Rysunek ScaleGoal