Łapanie na haczyk

Teraz, gdy nasz gracz porusza się po świecie rzeczywistym i wirtualnym, nadszedł czas, aby przejść do innych aspektów naszej gry. Jeśli pamiętasz, w Foody GO gracze muszą złapać potwory z jedzeniem. Potwory z jedzeniem to genetycznie zmodyfikowane wypadki laboratoryjne. Te potwory wędrują teraz wszędzie, a także są niesamowitymi kucharzami i szefami kuchni. Po złapaniu potwora gracz może wyszkolić go, aby był lepszym kucharzem lub zabrać go do restauracji do pracy i zdobywać punkty. Po ustaleniu tego nieco tła, będziemy pracować nad spawnowanie i śledzeniem potworów wokół gracza. Ta część będzie mieszanką pracy z Unity oraz pisania lub edytowania nowych skryptów. Projekt naszej gry jest na tyle wyjątkowy, że nie możemy już polegać na standardowych zasobach. Ponadto wcześniej unikaliśmy zawiłości matematyki GIS i GPS, aby nas nie zniechęcić. Zamiast tego zdecydowaliśmy się tylko krótko wspomnieć o funkcjach biblioteki GIS. Na szczęście powinieneś mieć teraz wystarczającą wiedzę GIS, aby przenieść swoją naukę na wyższy poziom i dostać się do matematyki. Jeśli matematyka nie jest twoim przedmiotem. Obejmiemy następujące tematy:

* Tworzenie nowej usługi potwora

* Zrozumienie odległości w mapowaniu

* Dokładność GPS

* Sprawdzanie potworów

* Wyświetlanie współrzędnych w przestrzeni świata 3D

* Dodawanie potwora do mapy

* Budowanie prefabrykatu potwora

* Śledzenie potworów w interfejsie użytkownika

Zanim do tego przejdziemy, jeśli masz Unity z ostatniej otwartej części, z załadowanym projektem gry, przejdź do następnej sekcji. W przeciwnym razie otwórz Unity i załaduj projekt gry FoodyGO. Następnie upewnij się, że scena Mapa jest załadowana. Kiedy otworzysz jeden z zapisanych plików projektu, prawdopodobnie będziesz musiał również załadować scenę początkową. Unity często tworzy nową domyślną scenę, zamiast próbować zgadywać, która scena powinna zostać załadowana.

Tworzenie nowej usługi potwora

Skoro potrzebujemy sposobu na śledzenie potworów wokół gracza, czy jest lepszy sposób na zrobienie tego niż za pomocą nowej usługi? Serwis potworów będzie musiał wykonać kilka zadań, jak następuje:

* Śledź lokalizację graczy

* Zapytanie o potwory w pobliżu

* Śledź potwory w zasięgu gracza

* Utwórz instancję potwora, gdy jest wystarczająco blisko gracza

Na razie nasza usługa potworów będzie wyszukiwać i śledzić potwory tylko lokalnie na urządzeniu gracza. Nie tworzymy jeszcze serwisu internetowego potworów, w którym wielu graczy będzie konsumować i oglądać te same potwory. Jednak w Części 7, Tworzenie świata AR, przekonwertujemy naszą usługę tak, aby korzystała z usługi zewnętrznej, aby lepiej zaludniać nasze potwory. Otwórz Unity i postępuj zgodnie z tymi wskazówkami, aby rozpocząć pisanie nowego skryptu usługi:

1. W oknie Projekt otwórz folder Assets/FoodyGo/Scripts/Services. Kliknij prawym przyciskiem myszy (naciśnij Ctrl i kliknij prawym przyciskiem myszy na komputerze Mac), aby otworzyć menu kontekstowe i wybierz Utwórz | Skrypt C# do tworzenia nowego skryptu. Zmień nazwę skryptu MonsterService.

2. Kliknij dwukrotnie nowy plik MonsterService.cs, aby otworzyć go w MonoDevelop lub wybranym edytorze kodu.

3. Tuż po użyciach i przed definicją klasy dodaj następujący wiersz:

namespace packt.FoodyGO.Services {

4. Przewiń w dół do końca kodu i zakończ przestrzeń nazw przez dodanie zakończenia:

}

5. Jak zapewne pamiętasz, dodaliśmy do naszego kodu przestrzeń nazw w celu nazwania konfliktów i organizacji.

6. Tuż w definicji klasy dodaj nową linię:

public GPSLocationService

gpsLocationService;

7. Ta linia pozwala nam dodać w edytorze odnośnik do naszej usługi GPS. Pamiętaj, że chcemy również śledzić lokalizację gracza. Po zakończeniu edycji zapisz plik.

8. Potwierdź, że twój skrypt wygląda teraz tak:

using UnityEngine;

using System.Collections;

namespace packt.FoodyGO.Services

{

public class MonsterService :

MonoBehaviour

{

public GPSLocationService

gpsLocationService;

// Use this for initialization

void Start()

{

}

// Update is called once per frame

void Update()

{

}

}

}

Teraz dodamy nowy skrypt do nowej gry MonsterService , obiekt w oknie Hierarchia, wykonując następujące czynności:

1. Wróć do Unity i upewnij się, że scena Map jest załadowana.

2. Wybierz i rozwiń obiekt Usługi w oknie Hierarchia.

3. Kliknij prawym przyciskiem myszy (naciśnij Control i kliknij prawym przyciskiem myszy na Macu) obiekt Usługi, aby otworzyć menu kontekstowe i wybierz Utwórz pusty, aby utworzyć nowy pusty obiekt podrzędny.

4. Wybierz i zmień nazwę nowego obiektu Monster w Inspektorze okna.

5. Otwórz folder Assets/FoodyGo/Scripts/Services w oknie Project

6. Przeciągnij nowy skrypt MonsterService i upuść go na nowy obiekt Monster.

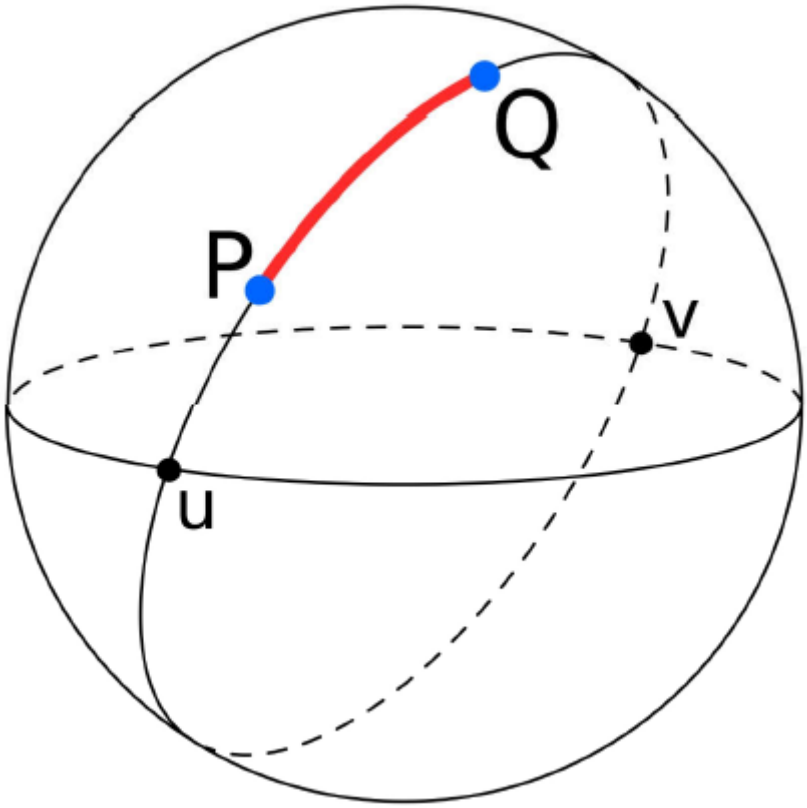
7. Wybierz obiekt Monster w oknie Hierarchia.

8. Przeciągnij obiekt GPS, również pod obiektem Usługi, do gniazda Gps Location Service dla komponentu skryptu Monster Service w obiekcie Monster w oknie Inspektora.

Oczywiście mamy jeszcze kilka rzeczy do zrobienia, ale to jest dobry pierwszy krok w dodaniu naszej nowej usługi Monster. W następnej sekcji zajmiemy się matematyką związaną z obliczaniem odległości i pozycji na świecie. Następnie wrócimy do dodawania kolejnych funkcjonalności do naszej nowej usługi.

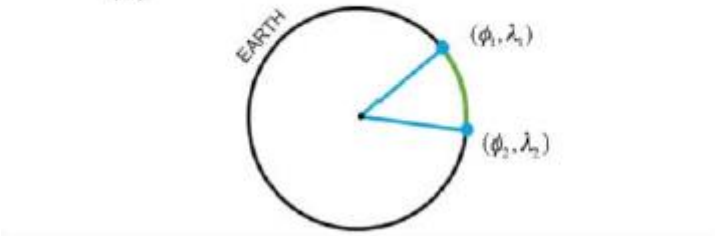
Zrozumienie odległości w mapowaniu

W poprzednich częściach nie musieliśmy martwić się o odległość, ponieważ pracowaliśmy tylko ze stałym punktem, czyli pozycją gracza. Teraz chcemy, aby gracz wyszukał i znalazł potwory. Chcemy, aby nasza usługa dotycząca potworów określała, czy potwór jest wystarczająco blisko, aby można go było zobaczyć lub usłyszeć. Aby to zrobić, nasza usługa Monster musi być w stanie obliczyć odległość między dwoma współrzędnymi mapowania: pozycją gracza i miejscem ukrycia się potwora. Możesz zapytać: „Dlaczego to takie trudne, czy Unity nie robi tego cały czas?” Odpowiedź brzmi tak i nie. Unity świetnie nadaje się do obliczania odległości liniowych między dwoma punktami w przestrzeni 2D i 3D. Pamiętaj jednak, że nasze współrzędne mapowania są w rzeczywistości w stopniach dziesiętnych wokół kuli, Ziemi. Aby poprawnie obliczyć odległość między dwoma punktami na kuli, musimy narysować linię na kuli, a następnie zmierzyć odległość. Oto diagram, który, miejmy nadzieję, powinien to lepiej wyjaśnić:



Jak pokazuje poprzedni diagram, mierzymy łuk, a nie odległość w linii prostej między punktami współrzędnych P i Q. W ramach ćwiczenia pomyśl o zmierzeniu odległości między u i v na diagramie. Wyobraź sobie, że latasz dookoła świata z miasta w punkcie u do miasta w punkcie v. Jaką masz nadzieję, że Twoja linia lotnicza wykorzystała do obliczenia paliwa? Aby poprawnie znaleźć odległość między dwoma lokalizacjami współrzędnych na mapie, używamy formuły zwanej hasrsine, jak pokazano na poniższym diagramie:

$$\text{haversine}\left(\frac{d}{r}\right) = \text{haversine}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\text{haversine}(\lambda_2 - \lambda_1)$$



Po niewielkiej manipulacji algebraicznej poprzedni wzór sprowadza się do postaci:

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

Jeśli nie jesteś matematykiem, w tym momencie twoje oczy mogą się zamykać lub być może starają się kontynuować, ze strachu, że twoja głowa eksploduje. Cóż, nie martw się. Choć te równania wyglądają onieśmielająco, wprowadzenie tej formuły do kodu jest dość proste. Sztuka polega na tym, aby nie dać się przytłoczyć i po prostu rozbijać to po kawałku. Jeśli chodzi o was, matematycy, przepraszam, że nie zagłębiacie się w szczegóły równania, ale reszta z nas jest przekonana, że sami możecie to rozwiązać. Zacznijmy więc i przenieś tę formułę do nowej biblioteki Math:

1. W oknie Projekt przejdź do folderu Assets/FoodyGo/Scripts/Mapping.

2. Kliknij prawym przyciskiem myszy (naciśnij klawisz Ctrl i kliknij prawym przyciskiem myszy na komputerze Mac) folder mapowania, aby otworzyć menu kontekstowe i wybierz Utwórz | Skrypt C#. Zmień nazwę skryptu MathG.

3. Kliknij dwukrotnie nowy skrypt MathG, aby otworzyć go w MonoDevelop lub wybranym przez siebie preferowanym edytorze.

4. Dodaj przestrzeń nazw, wstawiając następujący wiersz po instrukcji using:

namespace packt.FoodyGO.Mapping {

5. Pamiętaj o zamknięciu przestrzeni nazw przez dodanie końcowego nawiasu klamrowego } w ostatniej linii.

6. Zmień definicję klasy, aby odpowiadała następującym:

public static class MathG {

7. Ponieważ jest to biblioteka, oznaczyliśmy klasę jako static i usunęliśmy MonoBehaviour.

8. Całkowicie usuń metody Start i Update.

9. Dodaj nową instrukcję using dla systemu w następujący sposób:

using system;

10. W tym momencie twoja biblioteka podstawowa powinna wyglądać tak:

```
using UnityEngine;

using System.Collections;

using System;

namespace packt.FoodyGO.Mapping

{

public static class MathG

{

}

}
```

Jeśli nie masz doświadczenia w pisaniu skryptów lub pisaniu skryptów w Unity, zdecydowanie zalecamy wykonanie ćwiczenia z kodowania. Nie ma lepszego sposobu na naukę skryptów niż po prostu to zrobić. Jednakże, jeśli jesteś starym profesjonalistą lub po prostu wolisz przeczytać rozdziały i spojrzeć na kod

później, to również jest w porządku. Wystarczy otworzyć folder Chapter_4_Asset z pobranego kodu źródłowego, a wszystkie ukończone skrypty będą tam. Podstawowa nowa biblioteka MathG jest gotowa; dodajmy funkcję odległości haversine w następujący sposób:

1. Utwórz nową metodę w klasie MathG, wpisując:

public static float

Distance(MapLocation mp1,

MapLocation mp2){}

2. Wewnątrz metody Odległość wprowadź następujący pierwszy wiersz kodu:

double R = 6371; //avg radius of earth in km

3. Następnie wprowadzamy jeszcze trochę kodu inicjującego:

//convert to double in order to

increase

//precision and avoid rounding errors

double lat1 = mp1.Latitude;

double lat2 = mp2.Latitude;

double lon1 = mp1.Longitude;

double lon2 = mp2.Longitude;

4. Po przekonwertowaniu szerokości lub długości geograficznej z liczby zmiennoprzecinkowej na podwójną obliczamy różnicę i konwertujemy wartości na radiany. Większość funkcji matematycznych trygonometrycznych wymaga wprowadzenia w radianach, a nie w stopniach. Wpisz następujący kod:

// convert coordinates to radians

lat1 = deg2rad(lat1);

lon1 = deg2rad(lon1);

lat2 = deg2rad(lat2);

lon2 = deg2rad(lon2);

// find the differences between the coordinates

var dlat = (lat2 - lat1);

var dlon = (lon2 - lon1);

5. Oblicz odległość korzystając ze wzoru haversine wpisując kod:

// haversine formula

var a = Math.Pow(Math.Sin(dlat / 2),

2) + Math.Cos(lat1) *

```
Math.Cos(lat2) * Math.Pow(Math.Sin(dlon /

2), 2);

var c = 2 * Math.Atan2(Math.Sqrt(a),

Math.Sqrt(1 - a));

var d = c * R;
```

Uwaga : Zwróć uwagę, że używamy funkcji System.Math, które zapewniają podwójną precyzję, aby uniknąć możliwych błędów zaokrąglania. Unity dostarcza również bibliotekę Mathf, ale domyślnie jest to zmiennoprzecinkowa.

6. Jak widać, dla uproszczenia formuła jest podzielona na kilka linijek kodu. Nie ma nic szczególnie trudnego. Na koniec zwracamy wartość z funkcji przekonwertowaną z powrotem na zmiennoprzecinkową i w metrach. Wpisz ostatnią linię w metodzie:

```
//convert back to float and from km to m

return (float)d * 1000;
```

7. Na koniec musimy dodać nową metodę konwersji stopni na radiany. Jeśli zauważyłeś, użyliśmy go w poprzednim kodzie. Tuż pod metodą Odległość wprowadź następującą metodę:

```
public static double deg2rad(double

deg)

{

var rad = deg * Math.PI /

180;

// radians = degrees *

pi/180

return rad;

}
```

Uwaga: Teraz, kiedy możemy obliczyć odległość między dwoma współrzędnymi mapowania, potrzebujemy sposobu na przetestowanie naszego wzoru. Otwórz skrypt MonsterService w edytorze skryptów i wykonaj następujące czynności:

8. Po deklaracji gpsLocationService dodaj następujący wiersz:

```
private double lastTimestamp;
```

9. Następnie dodaj następujący kod w metodzie Update:

```
if (gpsLocationService != null &&

gpsLocationService.IsServiceStarted &&

gpsLocationService.PlayerTimestamp

> lastTimestamp)
```

```
{
lastTimestamp =
gpsLocationService.PlayerTimestamp;
var s =
MathG.Distance(gpsLocationService.Longitude,
gpsLocationService.Latitude,
gpsLocationService.mapCenter.Longitude,
gpsLocationService.mapCenter.Latitude);
print("Player distance from map
tile center = " + s);
}
```

10. Większość tego kodu powinna już wyglądać znajomo. Instrukcja if wykonuje ten sam zestaw testów, aby upewnić się, że usługa lokalizacji GPS działa i ma nowe punkty danych. Gdy ma nowe punkty danych, obliczamy odległość od aktualnej szerokości lub długości geograficznej oraz od aktualnego początku mapy. Następnie używamy print do wypisania wyników.

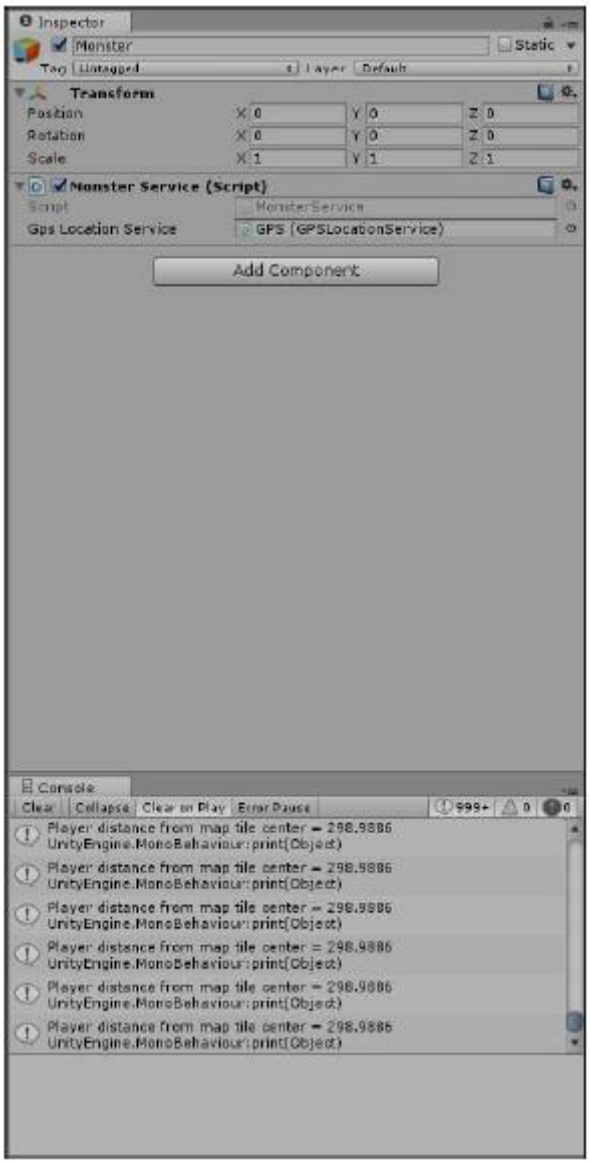
11. Ostatnią rzeczą, którą musimy zrobić, to dodać nowe użycie na górze pliku, aby uwzględnić nasze nowe funkcje MathG:

```
using packt.FoodyGO.Mapping;
```

12. Po zakończeniu edycji upewnij się, że wszystkie skrypty zostały zapisane, a następnie wróć do edytora Unity. Poczekaj kilka sekund, aż skompilują się aktualizacje skryptu. Upewnij się, że masz ustawioną usługę GPS do symulacji, a następnie naciśnij przycisk Odtwórz, aby rozpocząć testowanie. Jeśli przeskoczyłeś tutaj z innej części książki lub po prostu zapomniałeś, jak włączyć symulację GPS.

13. Gdy gra jest uruchomiona w edytorze, otwórz okno Unity Console, wybierając Okno | Konsola .

14. Przeciągnij zakładkę okna Konsola i dołącz ją do dolnej części Okna inspektora. Oto zrzut ekranu:



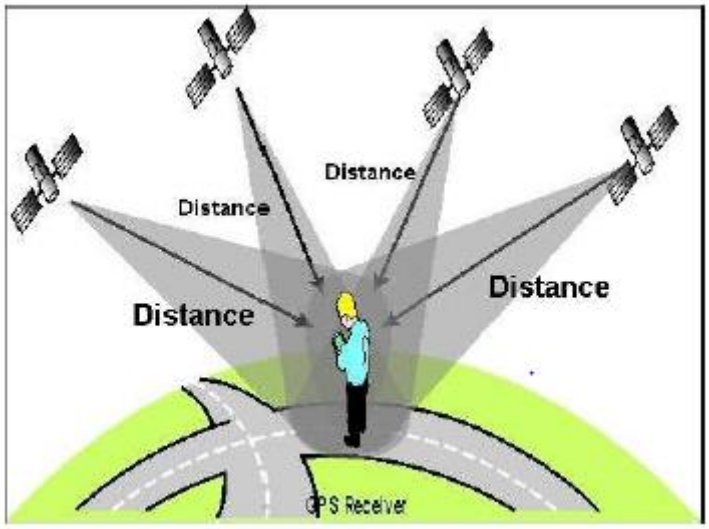
15. Powinieneś widzieć odległość gracza od środka kafelka mapy w metrach. W naszej obecnej konfiguracji pojedynczy kafelek mapy reprezentuje obszar o wymiarach około 600 x 600 metrów. Tak więc nasza postać nigdy nie powinna oddalić się na odległość większą niż 300 metrów od środka, w żadnym kierunku, zanim mapa się przerysuje. Jeśli twoje odległości znacznie się różnią, sprawdź funkcję MathG.Distance i upewnij się, że wpisałeś kod poprawnie.

16. Zbuduj i wdróż grę na swoje urządzenie mobilne. Uruchom grę z urządzenia mobilnego i podłącz konsolę CUDLR. Przejdź się po domu lub w zasięgu Wi-Fi i zobacz, jakie odległości otrzymujesz.

Kiedy spacerowałeś po swoim domu lub obszarze Wi-Fi, prawdopodobnie zauważyłeś, że odległości były niedokładne lub niespodziewanie zmieniły się. Ta niedokładność wynika z tego, że GPS w urządzeniu robi wszystko, co w jego mocy, aby obliczyć najlepszą możliwą pozycję za pomocą triangulacji satelitarnej. Jak zauważyłeś, Twoje urządzenie może czasami mieć trudności z podaniem dokładnej lokalizacji.

Dokładność GPS

W Części 2, kiedy przedstawiliśmy koncepcję śledzenia GPS, tylko pokrótce wspomnieliśmy, jak działa triangulacja satelitów i jaka jest dokładność GPS. W tamtym czasie dodanie dodatkowego poziomu szczegółowości byłoby po prostu przeciążeniem informacjami, a nie mieliśmy dobrego przykładu, aby zademonstrować dokładność GPS. Teraz, jak właśnie widzieliśmy w poprzedniej sekcji, dokładność GPS będzie miała wpływ na sposób interakcji gracza ze światem. W związku z tym poświęćmy chwilę, aby zrozumieć, w jaki sposób GPS oblicza lokalizację. Urządzenie GPS wykorzystuje sieć 24-32 satelitów krążących wokół Ziemi, zwaną siecią Globalnego Systemu Nawigacji Satelitarnej (GNSS). Satelity te okrążają Ziemię co 12 godzin i przekazują swoją zakodowaną czasowo lokalizację w sygnałach mikrofalowych. Urządzenie GPS odbiera następnie te sygnały z satelitów, na które ma dobrą widoczność. Oprogramowanie GPS w urządzeniu wykorzystuje te odczyty do obliczania odległości i wykorzystuje trilateralację do znalezienia własnej lokalizacji. Im więcej satelitów GPS może zobaczyć, tym dokładniejsze będą obliczenia. Poniższy diagram pokazuje, jak działa trilateracja:



Bystrzy czytelnicy mogli zauważyć, że przeszliśmy z triangulacji na trilateralację. To jest celowe; triangulacja wykorzystuje kąty do określenia lokalizacji. Jak widać na poprzednim schemacie, GPS faktycznie wykorzystuje odległości. Dlatego poprawnym i bardziej szczegółowym terminem jest trilateracja. Jeśli jednak próbujesz wyjaśnić znajomym dokładność GPS, nadal może być konieczne użycie terminu triangulacja. W tym momencie możesz pomyśleć, poza matematyką, jeśli widzimy wiele satelitów i możemy dokładnie obliczyć odległości, to nasza dokładność powinna być naprawdę wysoka. To prawda, ale istnieje wiele czynników, które mogą wypaczyć obliczenia. Oto lista rzeczy, które mogą zakłócać pomiar odległości w śledzeniu GPS:

* Przesunięcia zegara satelitarnego: Satelity używają zegarów atomowych gwarantowanych do pewnego poziomu precyzji dla publiczności. Przesunięcia są wprowadzane przez wojsko USA w celu celowego zmniejszenia celności.

* Warunki atmosferyczne: pogoda i zachmurzenie mogą wpływać na sygnały.

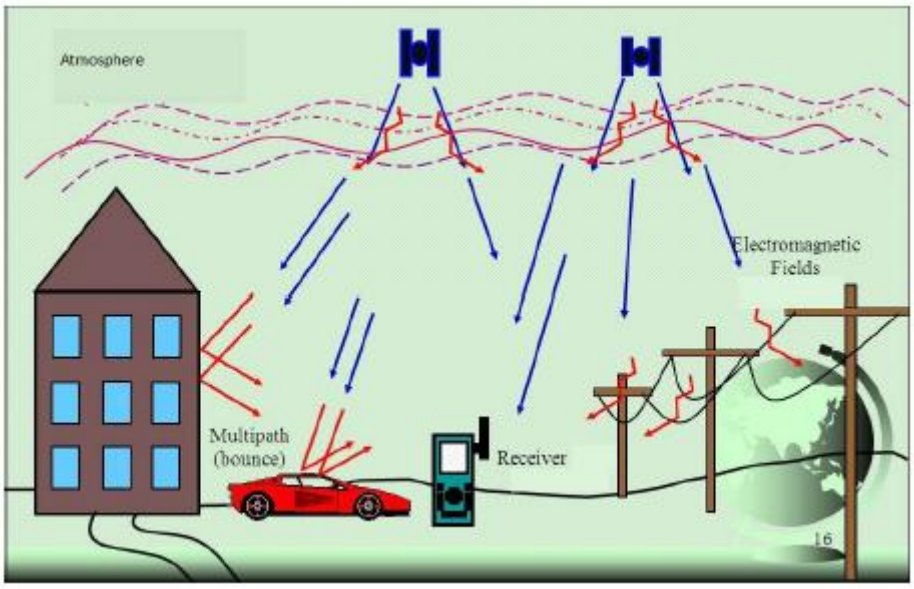
* Zegar odbiornika GPS: Dokładność zegara urządzenia będzie odgrywać główną rolę w dokładności. Na przykład telefon komórkowy nie ma zegara atomowego.

* Blokada sygnału: Sygnały satelitarne mogą być blokowane przez wysokie budynki, ściany, dachy, wiadukty, tunele i tak dalej.

* Pola elektromagnetyczne: linie energetyczne i mikrofale mogą mieć wpływ na ścieżkę sygnału.

* Odbicie sygnału: Najgorszym problemem dla śledzenia GPS jest odbicie sygnału. Sygnały mogą odbijać się od budynków, metalowych ścian i nie tylko.

Poniższy diagram przedstawia przykłady problemów z sygnałem:



Zrozumienie, dlaczego urządzenie może nie być tak dokładne, jak powinno, może pomóc w zrozumieniu problemów podczas testowania gry. Spróbuj ponownie uruchomić grę i wędrować po swoim domu lub obszarze Wi-Fi z CUDLR połączonym z komputerem programistycznym. Sprawdź, czy rozumiesz, dlaczego urządzenie może podawać pewne niedokładności w lokalizacji. Więc teraz wiemy, że nasze urządzenie jest dokładne tylko do pewnego momentu. Na szczęście istnieje miara dokładności zwracana z każdej nowej akwizycji lokalizacji. Ta wartość dokładności zasadniczo da nam promień błędu w metrach dla raportowanego obliczenia. Przetestujmy to, wykonując następujące czynności:

1. Otwórz folder Assets/FoodyGo/Scripts/Services w oknie projektu i kliknij dwukrotnie skrypt MonsterService, aby otworzyć go w wybranym edytorze skryptów.

2. Zmień linię drukowania w metodzie Update, aby odpowiadała następującym:

print("Player distance from map tile

center = " + s + "

- accuracy " +

gpsLocationService.Accuracy);

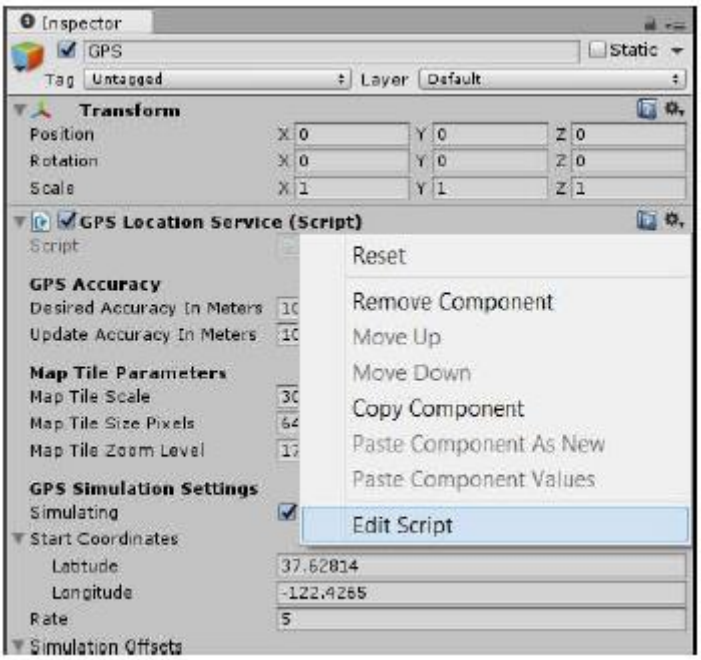
3. Zapisz skrypt po zakończeniu edycji i wróć do edytora Unity. Poczekaj kilka sekund na ponowną kompilację skryptów.

4. Zbuduj i wdróż grę na swoje urządzenie. Dołącz CUDLR i ponownie przetestuj grę, wędrując po swoim domu lub obszarze Wi-Fi.

Najprawdopodobniej zauważyłeś, że za dokładność została zwrócona wartość około 10. Mogły istnieć inne dziwne wartości, prawdopodobnie wartości z zakresu 500 lub 1000. Powodem, dla którego dokładność wynosi domyślnie około 10, jest to, że jest to ustawienie domyślne podczas uruchamiania usługi GPS. Jest to również typowa dokładność dla szerokiej gamy urządzeń mobilnych wyposażonych w GPS. Wraz z rozwojem technologii typowa dokładność GPS na nowszych urządzeniach mobilnych zbliża się do około trzech metrów. Ponieważ gracze w naszej grze będą szukać pieszo, chcemy, aby aktualizacje GPS i dokładność były obliczane w najlepszej możliwej rozdzielczości. Gracze szybko byliby sfrustrowani, gdyby musieli iść kilka metrów w jednym kierunku tylko po to, by odkryć, że jechali w złą stronę. Zmieńmy domyślną dokładność usługi lokalizacji GPS, wykonując następujące czynności:

1. W edytorze Unity wybierz i rozwiń obiekt Usługi w oknie Hierarchia.

2. Wybierz obiekt GPS, a następnie w oknie Inspektora kliknij ikonę koła zębatego obok usługi lokalizacji GPS (skryptu), aby otworzyć menu kontekstowe. Wybierz z menu opcję Edytuj skrypt. Poniższy zrzut ekranu przedstawia ikonę koła zębatego i menu:



3. Zostaniesz przekierowany do wybranego edytora skryptów z otwartym skryptem GPSLocationService.

4. Dodaj następujące deklaracje zmiennych i atrybut nagłówka tuż pod deklaracją zdarzenia OnMapRedraw:

```
public event OnRedrawEvent

OnMapRedraw; //add code after

this line

[Header("GPS Accuracy")]

public float DesiredAccuracyInMeters =

10f;
```

public float UpdateAccuracyInMeters =

10f;

5. Przewiń w dół do metody StartService i edytuj wywołanie Input.Location.Start() w następujący sposób:

// Start service before querying

location

Input.location.Start(DesiredAccuracyInMeters

,

UpdateAccuracyInMeters);

6. Po zakończeniu edycji zapisz swój skrypt. Wróć do edytora Unity i poczekaj na ponowną kompilację.

7. Wybierz obiekt GPS w oknie Hierarchia. W oknie Inspektora powinieneś teraz zobaczyć nowy nagłówek i właściwości, które właśnie dodaliśmy. Zmień oba ustawienia dokładności na 1 z 10.

8. Zbuduj i wdróż grę na swoje urządzenie mobilne.

9. Po uruchomieniu gry na urządzeniu mobilnym podłącz konsolę CUDLR. Poruszaj się po domu lub obszarze Wi-Fi i sprawdzaj konsolę CUDLR i notuj wszelkie zmiany w wynikach.

Mamy nadzieję, że Twoje urządzenie obsługuje dokładność lepszą niż 10 metrów i być może zauważyłeś zalew nowych aktualizacji GPS. Jeśli widzisz te aktualizacje, zauważysz również, że raportowana dokładność prawdopodobnie spadła do liczby mniejszej niż 10. Dla tych czytelników, którzy nie mogą zobaczyć zmiany, być może uda ci się przekonać znajomego z nowszym urządzeniem do przetestowania gry. Niestety, chociaż nasze urządzenie zgłasza aktualizacje co kilka metrów, częstsze aktualizacje mają swoją cenę, moc baterii. GPS zużywa dużo energii, aby stale odbierać sygnały satelitarne i wykonywać obliczenia odległości i trilateracji. Prosząc o częstsze aktualizacje, prawdopodobnie bateria urządzenia wyczerpie się szybciej. Jako twórca gry musisz zdecydować, jaka dokładność najlepiej sprawdza się w Twojej grze.

Szukam potworów

Świetnie, teraz, kiedy rozumiemy, jak określać odległość i jak dokładność GPS może zmienić trilaterację lokalizacji, nadszedł czas, aby zacząć śledzić potwory wokół postaci. Na razie użyjemy prostej metody losowego umieszczania potworów wokół gracza. W następnej części będziemy lokalizować potwory za pomocą serwisu internetowego. W tym momencie omówiliśmy już sporo skryptów i do końca rozdziału mamy więcej do zrobienia. Ponadto zmiany w skryptach, które musimy wprowadzić, są bardziej skomplikowane i podatne na błędy. Tak więc, aby nie narażać Cię na to zamieszanie, zaimportujemy następną sekcję zmian. W pozostałej części tego rozdziału będziemy przełączać się między ręcznymi edycją a importowaniem skryptów tam, gdzie będzie to właściwe. Wykonaj następujące instrukcje, aby wykonać pierwszy import zasobów skryptu:

1. Z menu edytora Unity wybierz Zasoby | Importuj pakiet | Pakiet niestandardowy…

2. Po otwarciu okna dialogowego Importuj pakiet... przejdź do miejsca, w którym umieściłeś pobrany kod źródłowy książki i otwórz folder Chapter_4_Asset.

3. Wybierz plik Chapter4_import1.unitypackage do zaimportowania i kliknij przycisk Otwórz.

4. Poczekaj na wyświetlenie okna dialogowego Import Unity Package. Upewnij się, że wszystkie skrypty są zaznaczone i kliknij przycisk Importuj.

5. Otwórz folder FoodyGo w oknie projektu i przejrzyj nowe skrypty.

Otwórzmy nowy skrypt MonsterService w edytorze kodu i zobaczmy, co się zmieniło:

1. Znajdź skrypt MonsterService w oknie projektu i kliknij go dwukrotnie, aby otworzyć go w edytorze kodu.

2. Pierwszą rzeczą, jaką możesz zauważyć na górze pliku, jest dodanie nowych instrukcji using i kilku nowych pól. Oto fragment nowych pól:

```
[Header("Monster Spawn Parameters")]

public float monsterSpawnRate = .75f;

public float latitudeSpawnOffset =

.001f;

public float longitudeSpawnOffset =

.001f;

[Header("Monster Visibility")]

public float monsterHearDistance =

200f;

public float monsterSeeDistance =

100f;

public float monsterLifetimeSeconds =

30;

public List<Monster> monsters;
```

3. Jak widać, dodano nowe pola, aby kontrolować pojawianie się potworów oraz z jakiej odległości potwory mogą być widziane lub słyszane. Na koniec mamy zmienną listy, która przechowuje nowy typ potwora. Nie będziemy spędzać czasu na oglądaniu klasy Monster, ponieważ w tej chwili jest to tylko kontener danych.

4. Następnie przewiń w dół do metody Update i zauważ, że kod testu odległości został usunięty i zastąpiony przez CheckMonsters(). CheckMonsters to nowa metoda, którą dodaliśmy do odradzania się i sprawdzania aktualnego stanu potworów.

5. Przewiń w dół do metody CheckMonsters. Poniżej znajduje się pierwsza sekcja tej metody:

```
if (Random.value > monsterSpawnRate)

{

var mlat =

gpsLocationService.Latitude +
```

```
Random.Range(-latitudeSpawnOffset,

latitudeSpawnOffset);

var mlon =

gpsLocationService.Longitude +

Random.Range(-longitudeSpawnOffset,

longitudeSpawnOffset);

var monster = new Monster

{

location = new MapLocation(mlon,

mlat),

spawnTimestamp =

gpsLocationService.PlayerTimestamp

};

monsters.Add(monster);

}
```

6. Pierwsza linia tej metody sprawdza, czy nowy potwór powinien zostać spawnowany. Robi to za pomocą Unity Random.value, który zwraca losową wartość od 0,0 do 1,0 i porównuje ją z monsterSpawnRate. Jeśli potwór się odradza, nowe współrzędne szerokości lub długości geograficznej są obliczane na podstawie bieżącej lokalizacji GPS i losowego zakresu +/- przesunięcia odrodzenia. Następnie tworzony jest nowy obiekt danych potwora i dodawany do listy potworów.

7. Przewiń trochę w dół, a zobaczysz, że bieżąca lokalizacja gracza jest konwertowana na typ MapLocation. Robimy to, aby przyspieszyć obliczenia. W programowaniu gier przechowuj wszystko, czego możesz później potrzebować i unikaj tworzenia nowych obiektów.

8. W następnej linii znajduje się wywołanie nowego typu Epoch i zapisanie wyniku do chwili obecnej. Epoch to statyczna klasa narzędziowa, która zwraca aktualny czas Epoki w sekundach. Jest to ta sama skala czasu, której Unity używa do zwracania znaczników czasu z urządzenia GPS. Epoka lub czas uniksowy to standard pomiaru czasu zdefiniowany jako liczba sekund, które upłynęły od godziny 00:00:00, czyli 1, 1, 1970.

1. Następna w skrypcie jest pętla foreach, która sprawdza, czy odległość od potwora lub gracza jest poniżej progu widzenia lub słyszenia. Jeśli potwór jest widziany lub słyszany, instrukcja print wyświetla stan i odległość do gracza. Cała pozostała część kodu wygląda następująco:

```
//store players location for easy

access in distance

calculations

var playerLocation = new
```

```
MapLocation(gpsLocationService.Longitude,
gpsLocationService Latitude);
gpsLocationService.Latitude);
//get the current Epoch time in
seconds
var now = Epoch.Now;
foreach (Monster m in monsters)
{
var d = MathG.Distance(m.location,
playerLocation);
if (MathG.Distance(m.location,
playerLocation)
< monsterSeeDistance)
{
m.lastSeenTimestamp = now;
print("Monster seen, distance " +
d + " started at " +
m.spawnTimestamp);
continue;
}
if (MathG.Distance(m.location,
playerLocation) <
monsterHearDistance)
{
m.lastHeardTimestamp = now;
print("Monster heard, distance "
+ d + " started at "
+ m.spawnTimestamp);
continue;
}
```

2. Po zakończeniu przeglądania skryptu wróć do Unity i wybierz obiekt Monster Service w oknie Hierarchia. Spójrz w okno Inspektora i przejrzyj dodane ustawienia. Jeszcze niczego nie zmieniaj.

3. Po zakończeniu przeglądania zmian skompiluj i wdróż grę na swoim urządzeniu mobilnym. Dołącz CUDLR i wykonaj kolejny spacer po obszarze Wi-Fi. Podczas spaceru sprawdzaj, czy pojawiają się nowe potwory i odległość.

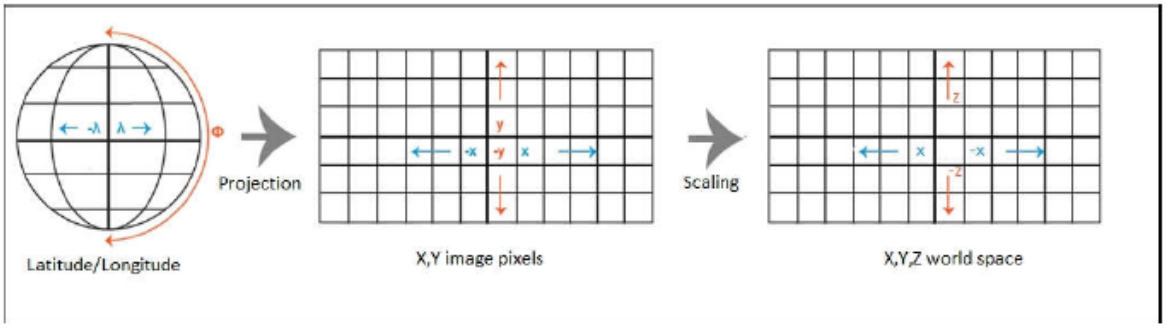
Świetnie, teraz mamy sposób na odradzanie się i śledzenie potworów wokół gracza, gdy się poruszają. Oczywistym następnym krokiem jest rozpoczęcie pokazywania naszych potworów na mapie. Jednak zanim to zrobimy, musimy dodać kod, aby przekonwertować współrzędne mapy na współrzędne świata gry dla naszego serwisu Monster.

Rzutowanie współrzędnych do przestrzeni świata 3D

Jeśli pamiętasz, w klasie CharacterGPSCompassController zastosowana przez nas metoda Update dokonała już konwersji ze współrzędnych mapy do przestrzeni świata 3D. Niestety, ten kod wymaga zależności od usługi lokalizacji GPS, aby określić skalę kafelków mapy świata. Tak więc, o ile chcielibyśmy stworzyć funkcję biblioteczną do konwersji, po prostu łatwiej będzie dodać ją jako metodę pomocniczą do usługi Monster. Na szczęście ta metoda pomocnicza została już dodana w ramach ostatniego importu zasobów skryptu. Po prostu wróć do edytora kodu i zakładając, że nadal masz otwartą usługę Monster z ostatniej sekcji, przewiń w dół do dołu pliku. Zauważysz, że do konwersji została dodana prywatna metoda, która jest pokazana w następujący sposób:

```
private Vector3 ConvertToWorldSpace(float

longitude, float latitude)

{

//convert GPS lat/long to world x/y

var x = ((GoogleMapUtils.LonToX(longitude)

- gpsLocationService.mapWorldCenter.x) *

gpsLocationService.mapScale.x);

var y = (GoogleMapUtils.LatToY(latitude)

- gpsLocationService.mapWorldCenter.y) *

gpsLocationService.mapScale.y;

return new Vector3(-x, 0, y);

}
```

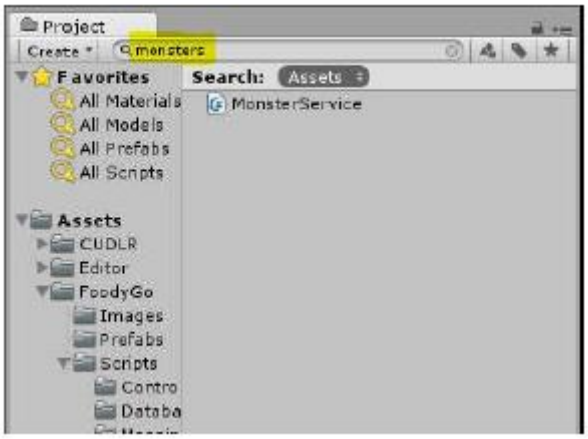
Jest to ten sam kod, którego używamy do konwersji współrzędnych gracza na przestrzeń świata. Zasadniczo rzutujemy współrzędne mapy na przestrzeń obrazu kafelków mapy x,y, a następnie przekształcamy je w przestrzeń świata. Miejmy nadzieję, że poniższy rysunek lepiej zilustruje tę koncepcję:



Dodawanie potworów do mapy

Tak więc teraz, gdy mamy już wszystko w porządku, nadszedł czas, aby zacząć umieszczać potwory na mapie. Na początek przynajmniej obiekty potworów. Otwórz Unity i skorzystaj z poniższych instrukcji, aby dodać kod instancji potwora do naszego serwisu Monster:

1. W polu wyszukiwania u góry okna Projektu wpisz potwory. Zasoby powinny automatycznie filtrować przedmioty z potworami w ich imieniu. Kliknij dwukrotnie skrypt MonsterService na przefiltrowanej liście, aby otworzyć skrypt w wybranym edytorze, jak pokazano na poniższym zrzucie ekranu:



2. Zaraz po deklaracji zmiennej GPSLocationService dodaj następujący wiersz kodu:

public GameObject monsterPrefab;

3. Przewiń w dół pliku i utwórz nową metodę SpawnMonster z następującym kodem:

private void SpawnMonster(Monster

monster)

{

var lon =

monster.location.Longitude;

var lat =

monster.location.Latitude;

var position =

ConvertToWorldSpace(lon, lat);

monster.gameObject =

(GameObject)Instantiate(monsterPrefab,

position, Quaternion.identity)

}

4. SpawnMonster to kolejna metoda pomocnicza, której użyjemy do odrodzenia naszego gotowego potwora. Metoda Instantiate dynamicznie tworzy i zwraca obiekt, biorąc pod uwagę prefabrykowany obiekt gry i pozycję lub obrót. Zwrócony obiekt gry jest następnie dodawany jako odniesienie do obiektu danych Monster, który zapewni później bezpośredni dostęp do obiektu gry.

5. Następnie musimy dodać wywołanie SpawnMonster wewnątrz metody CheckMonsters. Znajdź następujący wiersz kodu w metodzie CheckMonsters:

m.lastSeenTimestamp = now;

6. Po tym wierszu wprowadź następujące wiersze kodu:

if (m.gameObject == null)

SpawnMonster(m);

7. To, co robimy tutaj, to testowanie, czy potwór ma już dołączony obiekt spawnowany. Jeśli tak się nie stanie – a powinno być widoczne – wywołujemy SpawnMonster, aby utworzyć instancję nowego potwora.

8. Zapisz skrypt w swoim edytorze kodu i wróć do Unity. Poczekaj, aż Unity skompiluje zaktualizowane skrypty.

9. Utwórz nowy obiekt gry kostki, wybierając GameObject | Obiekt 3D | Kostka z menu. Zmień nazwę obiektu monsterCube w oknie Inspektora.

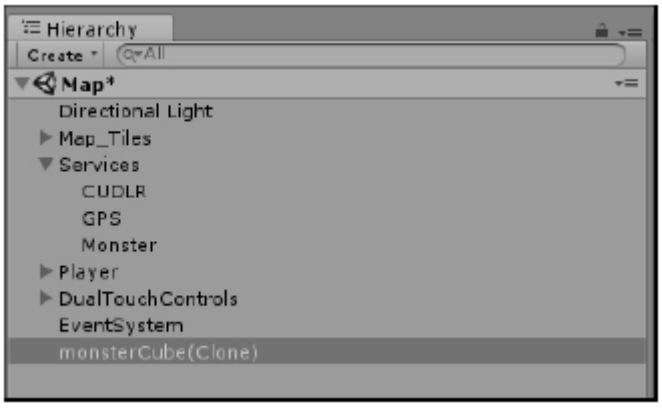
10. Otwórz folder Assets/FoodyGo/Prefabs w oknie projektu. Następnie przeciągnij nowy obiekt gry monsterCube do folderu Prefabs, aby utworzyć nowy element prefabrykowany.

11. Usuń obiekt gry monsterCube z okna Hierarchia.

12. Wybierz i rozwiń obiekt Usługi w oknie Hierarchia, a następnie wybierz obiekt Potwór.

13. Z folderu Assets/FoodyGo/Prefabs przeciągnij prefabrykat monsterCube na puste miejsce Monster Prefab w komponencie Monster Service w oknie Inspektora.

14. Naciśnij przycisk Graj, aby uruchomić grę w edytorze. Upewnij się, że usługa GPS przeprowadza symulację. Podczas trwania symulacji, wokół gracza powinien pojawić się obiekt monsterCube. Jeśli po pewnym czasie nie zobaczysz żadnych potworów, zmniejsz częstotliwość pojawiania się potworów do wartości około 0,25 w usłudze potworów. Poniżej znajduje się przykładowy zrzut ekranu, zauważ, jak klony monsterCube są dodawane do hierarchii:



Oczywiście nasze klocki nie wyglądają na bardzo przekonujące potwory, więc zróbmy coś z tym. Użyjemy innej postaci Reallusion jako bazy dla naszego potwora. Jeśli sobie przypominasz, Reallusion to firma, która tworzy te wspaniałe postacie iClone, których używamy do postaci naszego gracza. Wykonaj poniższe instrukcje, aby przygotować nową postać potwora:

1. Otwórz okno magazynu zasobów, wybierając Okno | Magazyn zasobów .

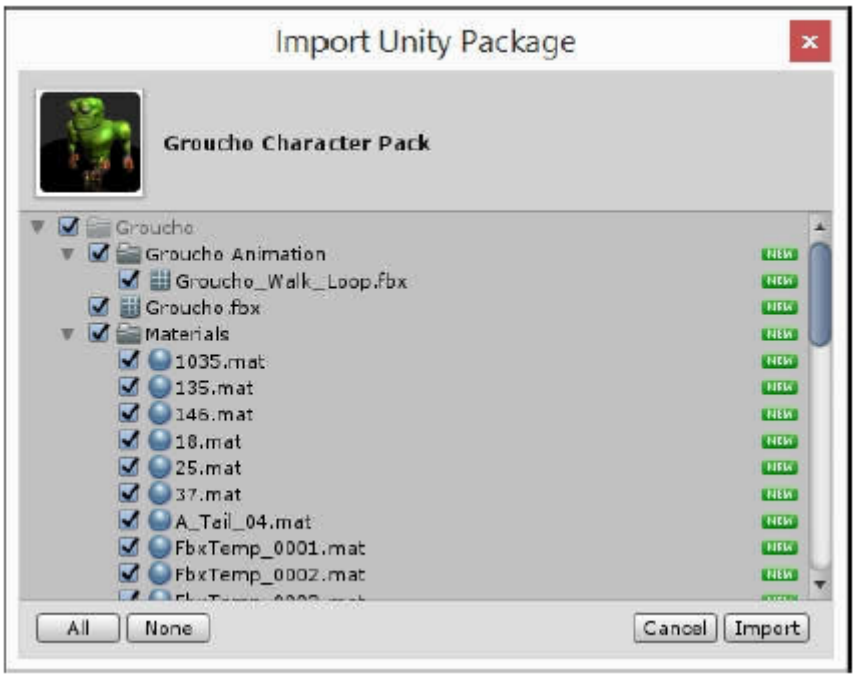
2. Po załadowaniu strony Asset Store wpisz w polu wyszukiwania groucho i naciśnij Enter lub kliknij szukaj.

3. Będzie płatna i darmowa wersja postaci Groucho

katalogowany; wybierz darmową wersję z listy.

4. Po załadowaniu strony zasobu kliknij przycisk Pobierz, aby pobrać i zaimportować zasób. Ponownie, może to chwilę potrwać; więc weź napój lub po prostu zrelaksuj się.

5. Po zakończeniu pobierania otworzy się okno dialogowe Importuj pakiet Unity. Po prostu upewnij się, że wszystko jest zaznaczone i kliknij Importuj w następujący sposób:



6. Po zaimportowaniu postaci otwórz folder Assets/Groucho/Prefab w oknie projektu. Następnie przeciągnij prefabrykat groucho do okna Hierarchia.

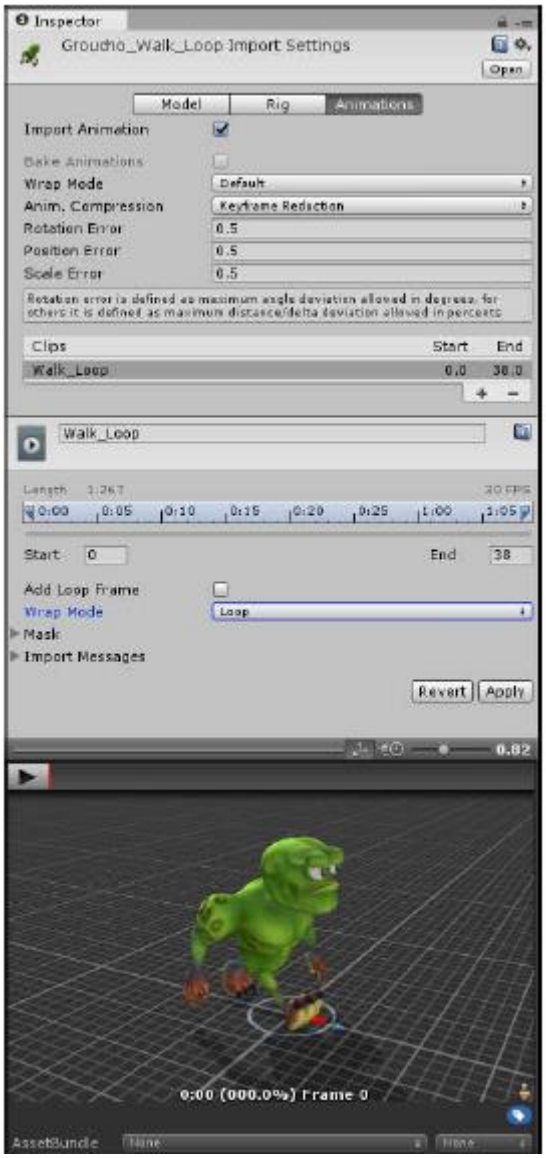
7. Wybierz obiekt Groucho w oknie Hierarchia. W oknie Inspektora zresetuj transformację obiektów, wybierając ikonę koła zębatego obok komponentu Przekształć, aby otworzyć menu kontekstowe. Następnie wybierz Resetuj z menu kontekstowego. Postać Groucho powinna teraz nachodzić na twoją postać iClone.

8. Zmień nazwę obiektu Groucho na Monster w oknie Inspektora.

9. Kliknij ikonę celu obok pola Animacja komponentu Animacja w oknie Inspektora. Wybierz Walk_Loop z okna dialogowego, a następnie zamknij.

10. Zaimportowana animacja pętli spaceru dla postaci Groucho nie jest domyślnie importowana do pętli. Musimy rozwiązać problem pętli animacji, wybierając animację Walk_Loop, którą właśnie ustawiliśmy w oknie Inspektora. To podświetli animację w oknie projektu.

11. Następnie wybierz obiekt nadrzędny Groucho_Walk_Loop. Właściwości importu animacji zostaną następnie wyświetlone w oknie Inspektora, jak pokazano na kolejnym zrzucie ekranu:



12. Zmień tryb owijania na Loop, a następnie kliknij przycisk Zastosuj.

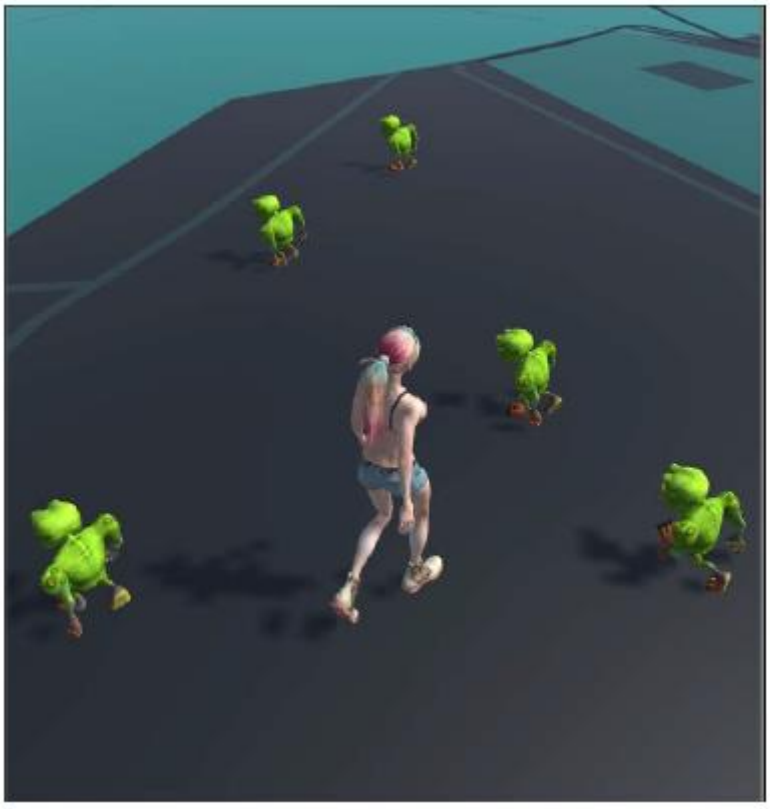
13. Wybierz obiekt Potwór w oknie Hierarchii i przeskaluj potwora, zmieniając wartości skali w komponencie Transform z 1 na .5 w x,y,z. Potwór musi być mniejszy i mniej onieśmielający.

14. Przeciągnij i upuść obiekt gry Monster do folderu Assets/FoodyGo/Prefabs w oknie projektu, aby utworzyć nowy szablon Monster.

15. Usuń obiekt gry Monster z okna Hierarchii, wybierając i naciskając klawisz Delete.

16. Wybierz i rozwiń obiekt Usługi w oknie Hierarchia. Następnie wybierz usługę Monster, aby podświetlić ją w oknie Inspektora. Przeciągnij i upuść Monster prefab z folderu Assets/FoodyGo/Prefabs w oknie projektu do pola Monster Prefab w komponencie Monster Service w Oknie inspektora.

17. Uruchom grę w edytorze, naciskając Play. Upewnij się, że symulacja GPS jest ustawiona na symulację. Poniższy przykładowy zrzut ekranu pokazuje wysoką częstotliwość pojawiania się potworów:



Świetnie, teraz na mapie pojawiają się nasze potwory. Podczas uruchamiania gry prawdopodobnie zauważyłeś, że pojawiło się kilka nowych problemów. Oto lista problemów, które musimy rozwiązać:

* Gdy mapa się wycentruje, potwory się nie wycentrują

* Potwory pozostają widoczne na mapie nawet po tym, jak gracz wyjdzie poza zasięg widoczności

* Wszystkie potwory skierowane są w tym samym kierunku

* Nadal nie mamy możliwości śledzenia słyszalnych potworów

Aby naprawić pierwsze trzy problemy, wykonamy kolejny import zasobów skryptu, a następnie przejrzymy zmiany. Następnie naprawimy ostatni problem, dodając element interfejsu użytkownika do sceny. Wykonaj poniższe instrukcje, aby zaimportować nowe skrypty i inne potrzebne nam zasoby:

1. Zaimportuj pakiet zasobów, wybierając Zasoby | Importuj pakiet | Pakiet niestandardowy...

2. Po otwarciu okna dialogowego Importuj pakiet... przejdź do folderu z pobranym kodem źródłowym Chapter_4_Assets, wybierz Chapter4_import2.unitypackage, a następnie kliknij Otwórz .

3. Kiedy otworzy się okno dialogowe Import Unity Package, po prostu potwierdź, że wszystkie zasoby są wybrane, a następnie kliknij Importuj .

4. Potwierdź, czy nowe zasoby zostały zaimportowane, otwierając folder Assets/FoodyGo w oknie Projekt. Powinieneś zobaczyć kilka nowych folderów, takich jak Obrazy i Skrypty/UI. Wszystkie pierwsze trzy błędy zostały naprawione za pomocą kilku dodatków do skryptu MonsterService. Otwórz skrypt MonsterService w wybranym przez siebie edytorze i przejrzyj poprawki i odpowiednie zmiany podane na poniższej liście:

1. Potwory muszą się wycentrować po przerysowaniu mapy:

* Pierwszy problem został rozwiązany przez dołączenie do zdarzenia OnMapRedraw usługi GPSLocationService. Jeśli sobie przypominasz, to wydarzenie zostanie wywołane, gdy środkowy kafelek mapy zostanie przerysowany. Tutaj możesz zobaczyć zmiany w kodzie:

```
//event hookup inside Start()

gpsLocationService.OnMapRedraw +=

GpsLocationService_OnMapRedraw;

//event method

private void

GpsLocationService_OnMapRedraw(GameObject

g)

{

//map is recentered,

recenter all monsters

foreach(Monster m in

monsters)

{

if(m.gameObject !=

null)

{

var newPosition =

ConvertToWorldSpace(m.location.Longitude,
```

```
m.location.Latitude);

m.gameObject.transform.position =

newPosition;

}

}

}
```

* Ta metoda w pętli przechodzi przez potwory w serwisie MonsterService, sprawdza, czy mają one instancję obiektu gry. Jeśli tak, obiekt gry zostanie przeniesiony na mapę.

2. Potwory pozostają widoczne po zobaczeniu:

* Następna poprawka jest również stosunkowo prosta i zawiera tylko kilka dodatków do metody CheckMonsters. Pierwsza poprawka dotyczy sytuacji, gdy potwór nie jest widziany ani słyszany; chcemy mieć pewność, że nie są widoczne. Robimy to sprawdzając, czy pole gameObject potwora nie jest puste, a następnie ustawiamy właściwość Active na false za pomocą SetActive(false), co jest równoznaczne z uczynieniem obiektu niewidocznym. Oto sekcja tego kodu:

```
//hide monsters that

can't be seen

if(m.gameObject != null)

{

m.gameObject.SetActive(false);

}
```

* Poprzednio, jeśli potwór był widziany, po prostu odradzaliśmy nowego potwora, jeśli pole gameObject było puste. Teraz musimy również upewnić się, że jeśli potwór ma gameObject, obiekt jest aktywny i widoczny. Robimy to prawie dokładnie tak, jak powyżej, ale teraz upewniamy się, że obiekt gry jest aktywny i widoczny za pomocą Ustaw Aktywny (prawda). Poniżej znajduje się sekcja kodu do przeglądu:

```
if (m.gameObject == null)

{

print("Monster seen,

distance " + d + " started at " +

m.spawnTimestamp);

SpawnMonster(m);

}

else

{
```

```
m.gameObject.SetActive(true);

//make sure the monster is visible

}
```

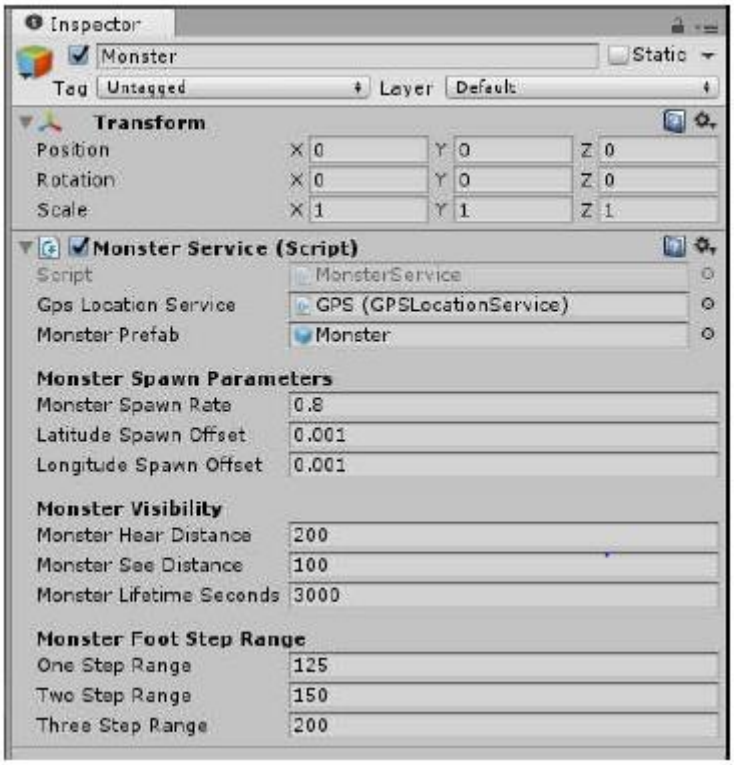
3. Wszystkie potwory zwrócone są w tym samym kierunku:

* Naprawimy ten ostatni problem, ustawiając losowy obrót potworów wokół osi y wektora Up. Oto sekcja kodu, zaktualizowana w metodzie SpawnMonster:

```
private void SpawnMonster(Monster

monster)

{

var lon =

monster.location.Longitude;

var lat =

monster.location.Latitude;

var position =

ConvertToWorldSpace(lon, lat);

var rotation =

Quaternion.AngleAxis(Random.Range(0,

360),

Vector3.up);

monster.gameObject =

(GameObject)Instantiate(monsterPrefab,

position, rotation);

}
```

Śledzenie potworów w interfejsie użytkownika

W ostatnim wydaniu chcemy, aby gracz mógł śledzić potwory, które są w pobliżu, ale nie są widziane. Zrobimy to, dostarczając graczowi wizualną wskazówkę w postaci ikony lub obrazu śladów. Jeden krok lub odcisk łapy/pazura jest bardzo blisko, dwa odciski nie są tak blisko, a trzy odciski są w zasięgu słuchu. Ponieważ mamy tylko jeden typ potwora, przynajmniej na razie, pokażemy graczowi tylko jedną ikonę przedstawiającą najbliższego potwora. Zanim przejdziemy do kodu, przyjrzyjmy się nowym właściwościom, które zostały dodane do usługi MonsterService w celu obsługi angage kroków. Rozwiń obiekt Usługi, a następnie wybierz obiekt Potwór. Poniżej znajduje się widok okna Inspektora usługi Monster:



Jak widać, w oknie Inspektora dodano nową sekcję do komponentu Monster Service. Nowa sekcja określa, w jakich zakresach aktywowane są poszczególne kroki, przy czym wartość jest maksymalnym zakresem. Na przykład, jeśli najbliższy potwór znajduje się w odległości 130 metrów, gracz zobaczy dwa kroki, ponieważ 130 jest większe niż 125 ustawione dla zasięgu w jednym kroku , ale mniejsze niż 150 dla zasięgu w dwóch krokach . Otwórz skrypt MonsterService z powrotem w swoim ulubionym edytorze kodu. Poniżej przedstawiono zmiany w skrypcie, w których określany i ustawiany jest zakres kroków:

* Pierwsza zmiana dotyczy metody CheckMonsters wewnątrz instrukcji if, która sprawdza, czy potwór jest słyszalny:

```
var footsteps =

CalculateFootsetpRange(d);

m.footstepRange = footsteps;
```

* Drugą zmianą jest dodanie nowej funkcji CalculateFootstepRange. Ta metoda po prostu określa zasięg na podstawie parametrów zasięgu kroków i wygląda następująco:

```
private int

CalculateFootstepRange(float distance)

{

if (distance < oneStepRange) return

1;

if (distance < twoStepRange) return
```

2;

if (distance < threeStepRange)

return 3;

return 4;

}

Aby pokazać graczowi zakres kroków, dodamy widok ikon do interfejsu użytkownika w następujący sposób:

1. Wróć do Unity i wybierz scenę Map w oknie Hierarchia. Wybierz GameObject | interfejs użytkownika | Surowy obraz . Spowoduje to rozwinięcie obiektu DualTouchControls i dodanie nowego obiektu RawImage jako elementu podrzędnego.

2. Zmień nazwę obiektu RawImage na Footsteps w oknie Inspektora.

3. Po wybraniu obiektu Footsteps otwórz folder Assets/FoodyGo/Scripts/UI. Przeciągnij skrypt FootstepTracker na obiekt Footsteps w oknie Inspektora. Spowoduje to dodanie komponentu Footstep Tracker (Script) do okna Inspektora w następujący sposób:

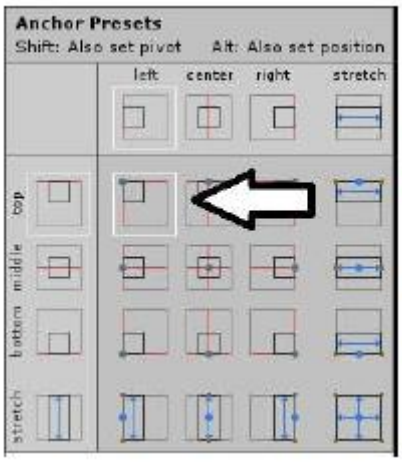
4. Rozwiń obiekt Usługi w oknie Hierarchia. Przeciągnij i upuść obiekt Monster Service w otwartym polu Monster Service w komponencie skryptu Footstep Tracker w oknie Inspektora.

5. Kliknij ikonę tarczy obok pola One Footstep. Otworzy się okno dialogowe Wybierz teksturę. Przewiń w dół w oknie dialogowym i wybierz łapy1, a następnie zamknij okno. Spowoduje to dodanie tekstury paws1 do pola One Footstep.

6. Zrób to samo dla pól Dwa kroki i Trzy kroki w następujący sposób:

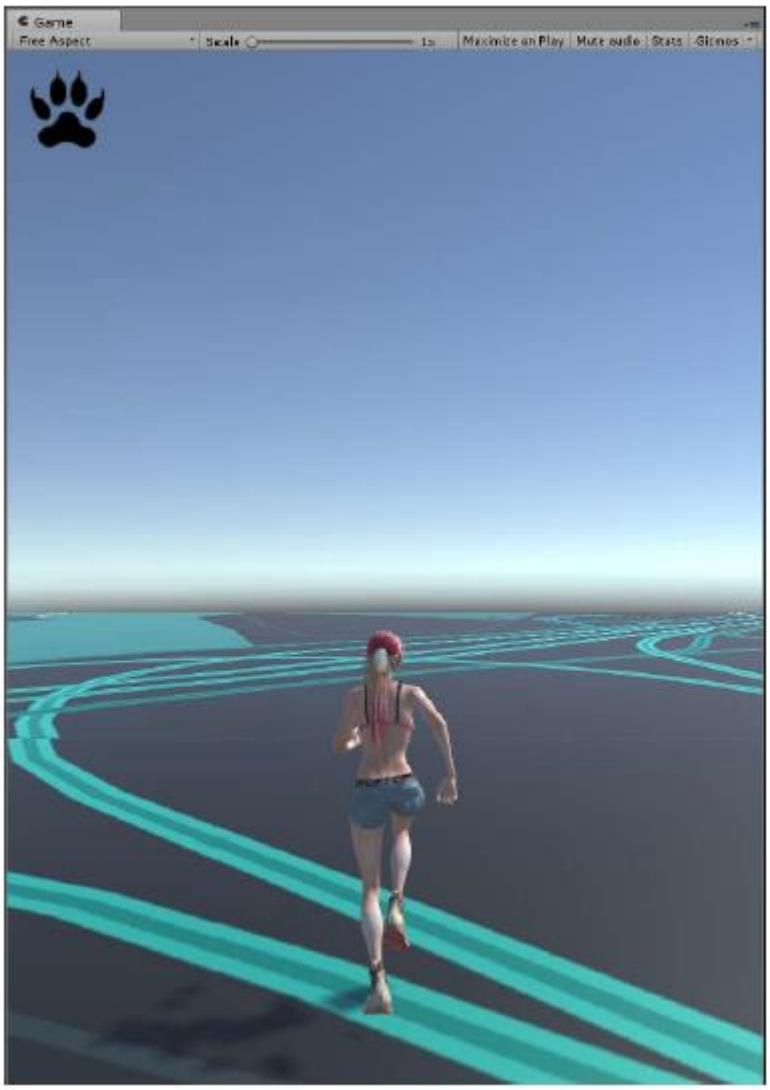
7. Otwórz menu Zakotwiczenia predefiniowane, klikając ikonę Rect Transform w oknie Inspektora.

8. Przy otwartym menu Anchor Presets przytrzymaj i naciśnij klawisze Shift i Alt, a następnie kliknij w lewym górnym rogu, jak pokazano w następnym oknie dialogowym:



9. Powinieneś teraz zobaczyć pusty biały kwadrat w lewym górnym rogu okna gry. Tutaj pojawi się ikona śladów.

10. Uruchom grę w edytorze Unity, naciskając Play. Upewnij się, że usługa GPS działa w trybie symulacji. Gdy twoja postać się teraz porusza, powinieneś zobaczyć, jak włącza się ikona odgłosu kroków z liczbą łap reprezentującą odległość do najbliższego potwora, jak pokazano na poniższym zrzucie ekranu:



Po zakończeniu testowania gry w edytorze skompiluj ją i wdróż na Twoje urządzenie mobilne. Następnie przejdź się po swoim domu lub okolicy i spróbuj wytropić potwory. Sprawdź, jak blisko potworów możesz się zbliżyć. Podczas testów na żywo pamiętaj o różnych odległościach, które ustalamy w serwisie Monster. Zastanów się, czy którakolwiek z tych wartości odległości wymaga zmiany.

Podsumowanie

Przez większość tej części napisaliśmy i zaktualizowaliśmy różne podstawowe skrypty potrzebne do śledzenia potworów na mapie. Zaczęliśmy od zrozumienia podstawowej matematyki do obliczania odległości przestrzennych. Następnie przeszliśmy do zrozumienia dokładności GPS i czynników wpływających na dokładność. Następnie przeszliśmy do skryptowania skryptu obsługi potwora i innych skryptów zależnych. Następnie dodaliśmy kod tworzenia prefabrykatów do skryptu potwora, aby pokazać prosty prefabrykat na mapie. Następnie zaimportowaliśmy do gry nową postać potwora i skonfigurowaliśmy nowy prefabrykat potwora. Dzięki testom ustaliliśmy, że pozostało jeszcze kilka problemów do rozwiązania. Następnie rozwiązaliśmy je poprzez aktualizację i przegląd skryptu. W końcu wdrożyliśmy sposób śledzenia słyszalnych potworów za pomocą prostej ikony odgłosów kroków, którą dodaliśmy jako nowy element interfejsu użytkownika. W kolejnej części pozwalamy graczowi spróbować złapać potwory w alternatywnym widoku rzeczywistości. To będzie pierwsza część, w

którym będziemy badać AR w naszej grze, wraz z wieloma innymi koncepcjami gry, takimi jak fizyka ciała sztywnego, animacja i efekty cząsteczkowe.