Przechowywanie połowu

W poprzedniej części zapewniliśmy graczowi sposób na złapanie potwora, rozwijając scenę Catch. Jednak, jeśli pamiętasz, gracz mógł złapać potwora, ale to wszystko. Z pewnością nie miał gdzie umieścić potwora po złapaniu go. Cóż, tu zbudujemy system przechowywania ekwipunku gracza. Umożliwi to graczom łapanie i przechowywanie potworów lub innych przedmiotów w razie potrzeby. Następnie oczywiście spędzimy czas na tworzeniu interfejsu użytkownika, aby uzyskać dostęp do potworów i ewentualnie innych przedmiotów w magazynie. Większość czasu spędzimy na rozwijaniu systemu ekwipunku, którego gracze będą używać do przechowywania złowionych ryb i innych przedmiotów. Zaczniemy od rdzenia naszego systemu Inventory, czyli bazy danych. Następnie przejdź do budowania elementów interfejsu potrzebnych graczowi, aby uzyskać dostęp do ekwipunku. Po drodze połączymy stworzone wcześniej sceny i dokończymy pierwszą działającą wersję naszej gry. Oto ogólny przegląd tego, co będziemy robić:

* System inwentaryzacji
* Zapisywanie stanu gry
* Konfigurowanie usług
* Przeglądanie kodu
* Operacje CRUD na potworach
* Aktualizacja sceny Catch
* Tworzenie sceny inwentaryzacji
* Dodawanie przycisków menu
* Połączenie gry
* Problemy z rozwojem mobilnym

System inwentaryzacji

Jeśli kiedykolwiek grałeś w jakąkolwiek przygodę lub grę fabularną, z pewnością znasz system ekwipunku gracza. System ekwipunku jest niezbędnym elementem tych gier i będzie również dla nas. Dlatego poświęcimy trochę czasu na przeglądanie funkcji, których potrzebujemy dla naszego systemu. Poniżej znajduje się lista funkcji, których potrzebujemy w naszym systemie ekwipunku:

Trwałe: gry mobilne są podatne na wyłączanie lub przerywanie. Dlatego inwentarz musi utrzymywać stan między sesjami gier w bazie danych lub innymi metodami przechowywania. Stan zapisu powinien być również solidny i szybko wykonywany. W tym celu moglibyśmy użyć płaskiego pliku lub bazy danych. Płaski plik będzie generalnie prostszy w użyciu, ale baza danych jest bardziej niezawodna i łatwo rozszerzalna.

Uwaga: plik płaski można uznać za bazę danych. Jednak w naszej dyskusji będziemy odnosić się do bazy danych jako zorganizowanego mechanizmu przechowywania, który obsługuje dobrze zdefiniowaną definicję danych i język zapytań.

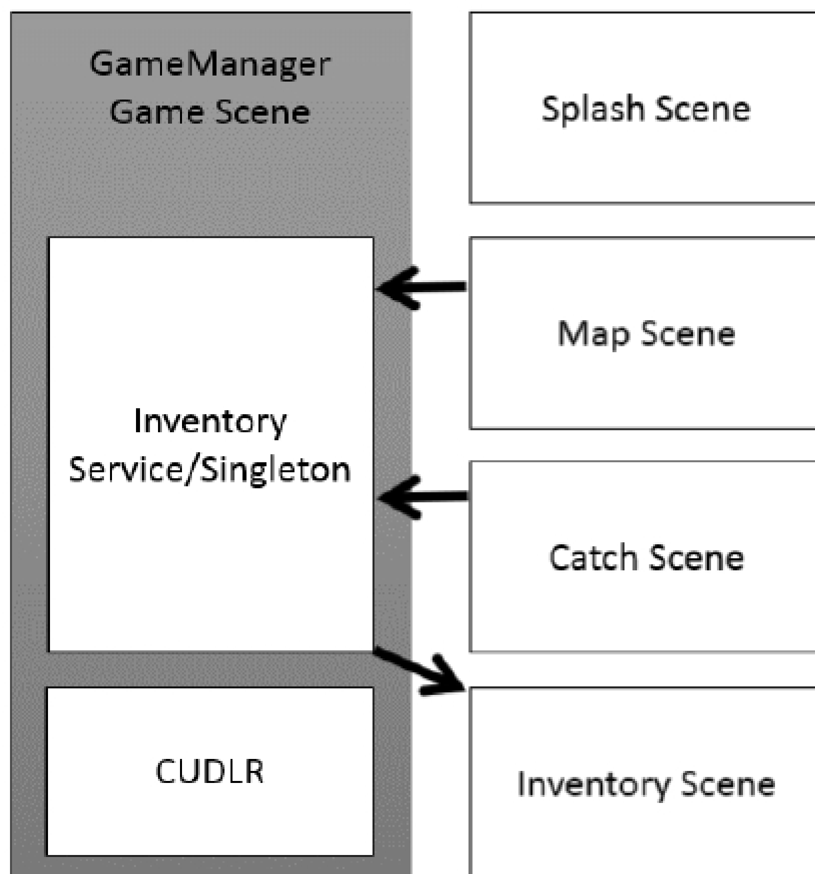
Wieloplatforma: Bazowa baza danych lub mechanizm przechowywania musi działać na dowolnej platformie, na której chcemy dystrybuować naszą grę, która w tej chwili jest ukierunkowana na Androida i iOS. Z tego powodu plik płaski może być oczywistym wyborem; jednak, jak zobaczymy, istnieją również inne dobre opcje międzyplatformowe.

Relacyjne: Nie chodzi tylko o relacyjną bazę danych, ale o wzajemne powiązanie obiektów. Na przykład, możemy chcieć dać naszym potworom własne przedmioty, takie jak nóż lub kapelusz szefa kuchni. Relacyjna baza danych sprawdziłaby się tutaj dobrze, ale istnieją inne opcje, takie jak baza obiektów lub wykresów. Oczywiście płaski plik XML może również reprezentować relacje, ale wygląda na to, że skłaniamy się ku rozwiązaniu bazodanowemu. Naszym idealnym rozwiązaniem byłaby relacyjna baza danych, która działa również jak obiektowa baza danych.

Rozszerzalny: Nasz system ekwipunku zacznie się od jednego typu przedmiotu, potwora. Jednak z pewnością będziemy chcieli później łatwo wspierać inne elementy. Ponownie, baza danych wygrywa tutaj.

Dostępność: Nasz system ekwipunku będzie musiał być dostępny dla wielu części lub scen w naszej grze. Dlatego prawdopodobnie chcemy, aby nasz asortyment był usługą i/lub typem singletona. Moglibyśmy umieścić system inwentaryzacji w usłudze, ale sensowne może być również uczynienie go singletonem.

Jeśli postępujesz zgodnie z funkcjami wymienionymi w poprzednich punktach, wygląda na to, że planujemy korzystać z bazy danych. Naszą preferencją będzie relacyjna baza danych dostępna również za pośrednictwem obiektów. Ponadto chcemy używać naszego ekwipunku jako usługi, jak usługa Monster, ale także jako singletona, jak klasa GameManager. Rzućmy okiem na poniższy diagram, który pokazuje, jak ten system powinien działać w ramach naszej gry:



Jak pokazuje poprzedni diagram, nowa usługa Inventory będzie częścią sceny gry i będzie współdziałać ze scenami mapy i połowu. W ten sposób gracz będzie mógł uzyskać dostęp do swojego ekwipunku z każdej z tych scen. Interfejs ekwipunku zostanie umieszczony w nowej scenie ekwipunku, którą oczywiście będzie zarządzał Menedżer gry. Mamy prawie wszystko, czego potrzebujemy do stworzenia naszej nowej usługi inwentaryzacji i sceny, z wyjątkiem typu i implementacji bazy danych, której będziemy używać do obsługi naszego systemu inwentaryzacji. W następnej sekcji zajmiemy się wyborem bazy danych i wykorzystaniem jej jako rdzenia nowej usługi.

Zapisywanie stanu gry

Nasza gra obecnie nie zapisuje żadnego stanu, ale do tej pory nie była nam potrzebna. Lokalizacja gracza była bezpośrednio dostępna z GPS urządzenia, a potwory wokół nich zostały stworzone przez naszą prowizoryczną usługę potworów. Jednak chcemy, aby nasi gracze polowali, łapali i zbierali potwory lub inne przedmioty w ramach gry. W tym celu musimy zapewnić trwałe przechowywanie w postaci bazy danych. W przeciwnym razie, gdy gracz wyłączy grę, wszystkie zebrane przez niego przedmioty znikną. Gry uruchomione na urządzeniu mobilnym są szczególnie podatne na przypadkowe wyłączenie lub awarie, co oznacza, że potrzebujemy solidnego rozwiązania pamięci masowej. Jeśli wyszukasz bazę danych w sklepie Unity Asset, zobaczysz wiele bezpłatnych i płatnych opcji. Jednak użyjemy alternatywy open source z GitHub o nazwie SQLite4Unity3d. Ten pakiet jest doskonałym opakowaniem dla SQLite, świetnej relacji międzyplatformowej bazy danych. W rzeczywistości w sklepie Asset dostępnych jest wiele różnych wersji opakowań baz danych SQLite. Co oznacza oprogramowanie nad innymi to kilka powodów wymienionych tutaj:

* Open source: to może być złe lub dobre. W tym przypadku jest to dobre, ponieważ jest bezpłatne i nadal obsługiwane. Nie wszystkie open source są darmowe lub dobrze obsługiwane, więc musisz być ostrożny.

* Relacyjna baza danych: SQLite to lekka relacyjna baza danych, która również jest otwarta i oparta na społeczności. Relacyjna baza danych jest dla nas dobrym rozwiązaniem, ponieważ obsługuje relacje danych i zapewnia dobrze znany język definicji danych. Język używany do tworzenia zapytań i definiowania danych w tej bazie danych nazywa się SQL, stąd nazwa. Na szczęście nie będziemy musieli wchodzić do SQL, ponieważ wrapper SQLite4Unity3d zarządza tym za nas.

* Model danych obiektów/jednostek: Model danych obiektów lub encji umożliwia programiście zarządzanie danymi w bazie danych za pomocą obiektów, zamiast bezpośrednio pisać w języku dodatkowym, takim jak SQL. Opakowania SQLite4Unity3d zapewnia doskonałą implementację mapowania relacyjnego obiektu opartego na kodzie (najpierw klasy) lub modelu danych zdefiniowanego przez jednostkę. Podejście oparte na kodzie pozwoli nam najpierw zdefiniować nasze obiekty, a następnie, w czasie wykonywania, dynamicznie skonstruować naszą bazę danych, aby dopasować nasze definicje obiektów. Nie martw się, jeśli to wszystko brzmi obco; niedługo przejdziemy do szczegółów.

Uwaga : Table first jest przeciwieństwem kodu/klasy pierwszego podejścia używanego do definiowania encji. Tabele bazy danych są definiowane jako pierwsze, a następnie kod/klasy są wyprowadzane w procesie kompilacji. Tabela pierwsza jest preferowana dla tych, którzy chcą rygorystycznej definicji danych.

Teraz, gdy mamy już tło, ubrudźmy sobie ręce, ładując opakowanie bazy danych i inny potrzebny kod. Na szczęście opakowanie i kod bazy danych są spakowane w ramach importu pojedynczego zasobu. Wykonaj następujące czynności, aby zaimportować pakiet zasobów:

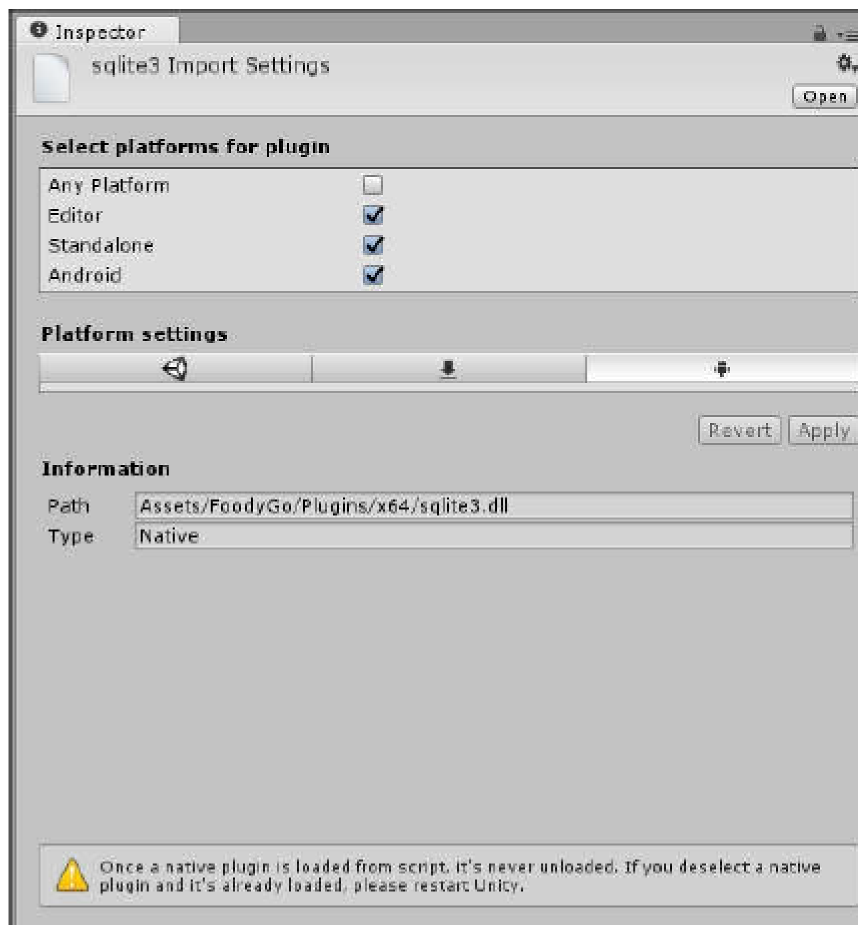
1. Otwórz edytor Unity i kontynuuj od miejsca, w którym zostawiliśmy projekt pod koniec rozdziału 5, Catching the Prey w AR, z załadowaną sceną Catch.

2. Z menu wybierz Zasoby | Importuj pakiet | Pakiet niestandardowy...

3. Po otwarciu okna dialogowego Importuj pakiet... przejdź do pobrany folder kodu źródłowego C5A i wybierz plik Chapter5_import1.unitypackage. Następnie kliknij Otwórz, aby zaimportować plik.

4. Po załadowaniu Import Unity Package wystarczy zweryfikować, co jest importowane, a następnie kliknąć przycisk Importuj. Spowoduje to zaimportowanie nowych i zaktualizowanych skryptów, a także niektórych wtyczek zarządzających integracją SQLite.

5. W oknie Projekt wybierz folder Assets/FoodyGo/Plugins/x64. W folderze wybierz wtyczkę sqlite3. Następnie w oknie Inspektora potwierdź, że wtyczka jest skonfigurowana do wdrożenia na Twoim urządzeniu. Poniższy zrzut ekranu pokazuje przykład dla Androida, ale byłby taki sam nawet dla iOS:



6. Jeśli chcesz wprowadzić zmiany w panelu, kliknij przycisk Zastosuj na dole. Spowoduje to zapisanie i zastosowanie zmian. Importowanie pakietu i konfigurowanie wtyczek było dość proste. W następnej sekcji sprawdzimy, czy wszystko jest poprawnie skonfigurowane.

Konfigurowanie usług

Teraz, gdy mamy nowe wtyczki opakowujące SQLite, skrypt SQLite i inne zaimportowane skrypty, skonfigurujemy niektóre usługi w naszej scenie Catch, aby je przetestować:

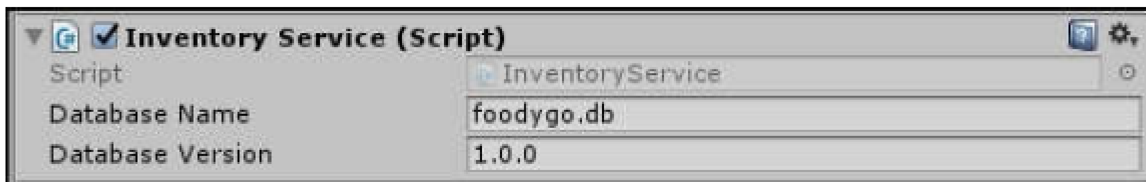
1. Z menu wybierz GameObject | Utwórz puste. Zmień nazwę tego nowego obiektu na Usługi i zresetuj transformację do zera w oknie Inspektora.

2. Wybierz nowy obiekt Usługi w oknie Hierarchia i kliknij prawym przyciskiem myszy (naciśnij Control i kliknij na komputerze Mac), aby otworzyć menu kontekstowe. Z menu kontekstowego wybierz Utwórz puste. Spowoduje to utworzenie nowego pustego obiektu podrzędnego dołączonego do obiektu Usługi. Zmień nazwę tego nowego obiektu Inventory w oknie Inspektora.

3. Powtórz krok 2, ale tym razem nazwij nowy obiekt CUDLR.

4. Wybierz obiekt Inventory w oknie Hierarchia. Z folderu Assets/FoodyGo/Scripts/Services przeciągnij skrypt Inventory na obiekt Inventory.

5. Po wybraniu obiektu Inventory przejrzyj ustawienia komponentu Inventory w oknie Inspektora w następujący sposób:



6. Usługa inwentaryzacji ma kilka krytycznych parametrów, które wyjaśniono tutaj:

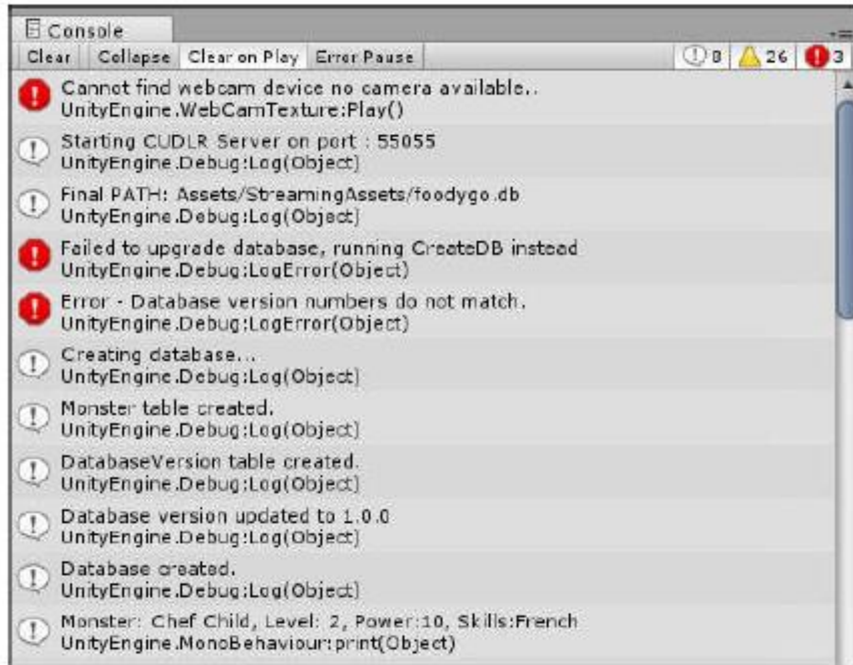
Nazwa bazy danych: To jest nazwa bazy danych. Zawsze należy używać rozszerzenia .db, które jest standardem dla baz danych SQLite.

Wersja bazy danych: Ustawia wersję bazy danych. Wersja powinna zawsze mieć postać major.minor.revision i powinna zawierać tylko wartości liczbowe. Omówimy szczegółowo, jak uaktualnianie bazy danych działa później.

7. Wybierz obiekt CUDLR w oknie Hierarchia. Z folderu Assets/CUDLR/Scripts przeciągnij skrypt serwera na obiekt CUDLR. Jeśli przegapiłeś Część 2, Mapowanie lokalizacji gracza, oraz sekcję dotyczącą konfiguracji konsoli CUDLR, wróć tam.

8. Z menu wybierz Okno | Konsola. Przeciągnij i przypnij okno konsoli tuż pod oknem inspektora.

9. Naciśnij Play w edytorze i uruchom scenę Catch. Nie musisz grać w tę grę; po prostu przejrzyj dane wyjściowe, jak pokazano w oknie konsoli:



10. Dane wyjściowe konsoli powinny być zgodne z danymi wyjściowymi pokazanymi w przykładzie (z wyjątkiem błędu kamery internetowej). Jak widać w wynikach, baza danych jest faktycznie tworzona podczas uruchamiania gry.

11. Zatrzymaj grę, klikając ponownie przycisk Graj. Następnie ponownie uruchom grę. Zauważ, że dane wyjściowe okna konsoli są teraz inne. Przy drugim uruchomieniu gry nie trzeba było tworzyć bazy danych, ponieważ została już utworzona.

12. Zbuduj i wdróż swoją grę na urządzenie mobilne. Jak zawsze, upewnij się, że skonfigurowałeś odpowiednie sceny w ustawieniach kompilacji. W takim przypadku dla kompilacji należy sprawdzić tylko scenę Catch.

13. Gdy gra jest uruchomiona, otwórz przeglądarkę i wprowadź adres CUDLR, którego wcześniej używałeś do łączenia się z konsolą. Jeśli nie masz pewności, jak to zrobić, wróć do Rozdziału 2, Mapowanie lokalizacji gracza, na temat konfigurowania CUDLR.

14. Dane wyjściowe CUDLR powinny wyglądać bardzo podobnie do danych wyjściowych widocznych w oknie konsoli:

```
Database not in Persistent path
Database written
Final PATH: /storage/emulated/0/Android/data/com.packt.FoodyGO/files/foodygo.db
Creating database...
Monster table created.
DatabaseVersion table created.
Database version updated to 1.0.0
Database created.
Monster: Chef Child, Level: 2, Power:10, Skills:French
```

15. Jeśli nie widzisz czegoś podobnego na wyjściu CUDLR, sprawdź ustawienia wtyczki z poprzedniej sekcji lub zapoznaj się z Częścią 10, Rozwiązywanie problemów.

16. Zamknij grę na swoim urządzeniu. Ponownie otwórz grę i ponownie przejrzyj dane wyjściowe na konsoli CURLR. Ponownie, nie powinieneś widzieć utworzonej nowej bazy danych, ponieważ jest ona już stworzona dla gry

Sprawdzam kod

Jak widać w poprzednim ćwiczeniu konfiguracyjnym i testowaniu, zaimportowana przez nas usługa Inventory ma już dołączone opakowanie bazy danych. Przyjrzyjmy się niektórym ze zmian w skrypcie, które zaimportowaliśmy, a także przejdźmy do szczegółów skryptu InventoryService:

1. Kliknij dwukrotnie CatchSceneController znajdujący się w folderze Assets/FoodyGo/Scripts/Controllers w oknie projektu, aby otworzyć skrypt w wybranym edytorze.

2. Jedyne, co do tej pory zmieniło się w CatchSceneController, to nowa metoda Start wywołująca naszą usługę Inventory. Wykonaj następującą metodę przeglądania:

```
void Start()
{
var monster =
InventoryService.Instance.CreateMonster();
print(monster);
}
```

3. Wewnątrz metody Start InventoryService jest wywoływany jako singleton przy użyciu właściwości Instance. Następnie wywoływana jest metoda CreateMonster w celu wygenerowania nowego obiektu potwora. Na koniec obiekt potwora jest drukowany w oknie konsoli za pomocą metody drukowania.

4. Kod w metodzie Start jest w zasadzie tylko tymczasowym kodem testowym, który przeniesiemy później. Doceń jednak łatwość dostępu, jaką zapewnia nam wzór singleton.

5. Mamy jeszcze jeden krok, zanim przyjrzymy się usłudze InventoryService. Otwórz skrypt Monster znajdujący się w folderze Assets/FoodyGo/Scripts/Database. Jeśli pamiętasz, wcześniej używaliśmy klasy Monster do śledzenia lokalizacji naszego spawnu w MonsterService. Zamiast tego zdecydowaliśmy się uprościć klasę Monster, aby była używana tylko do utrwalania ekwipunku/bazy danych i promować naszą starszą klasę do nowej lokalizacji MonsterSpawnLocation. Skrypt MonsterService został również zaktualizowany, aby korzystać z nowej lokalizacji MonsterSpawnLocation.

6. Przyjrzymy się bliżej nowemu obiektowi Monster, którego użyjemy do zachowania w ekwipunku/bazie danych:

```
public class Monster

{

[PrimaryKey, AutoIncrement]

public int Id { get; set; }

public string Name { get; set; }

public int Level { get; set; }

public int Power { get; set; }

public string Skills { get; set; }

public double CaughtTimestamp { get; set;

}

public override string ToString()

{

return string.Format("Monster: {0},

Level: {1}, Power:{2}, Skills:{3}",

Name, Level, Power, Skills);

}

}
```

7. Pierwszą rzeczą, która jest oczywista, jest użycie właściwości do definiowania atrybutów Monster, co jest odejściem od Unity i bardziej tradycyjnego C#. Następnie zwróć uwagę, że górna właściwość Id ma kilka atrybutów oraz dołączono PrimaryKey i AutoIncrement. Jeśli znasz relacyjne bazy danych, od razu zrozumiesz ten wzorzec. Dla mniej znanych, wszystkie rekordy/obiekty w naszej bazie danych wymagają unikalnego identyfikatora zwanego kluczem podstawowym. Ten identyfikator, w tym

przypadku zwany Id, pozwala nam później szybko zlokalizować obiekt. Atrybut AutoIncrement pozwala nam wiedzieć, że właściwość Id, czyli liczba całkowita, zostanie automatycznie zwiększona po utworzeniu nowego obiektu. Uwalnia nas to od samodzielnego zarządzania identyfikatorami obiektów i oznacza, że właściwość Id zostanie automatycznie ustawiona przez bazę danych.

8. Nie będziemy się teraz zbytnio przejmować innymi właściwościami, ale zamiast tego zwróćmy uwagę na nadpisaną metodę ToString. Zastąpienie ToString pozwala nam dostosować dane wyjściowe obiektu i jest przydatne do debugowania. Zamiast sprawdzać wszystkie właściwości i drukować je w konsoli, możemy uprościć to do print(monster), jak widzieliśmy wcześniej w metodzie CatchSceneController.Start.

9. Po usunięciu tła otwórz skrypt InventoryService znajdujący się w folderze Assets/FoodyGo/Scripts/Services. Jak widać, ta klasa ma wiele sekcji warunkowych w metodzie Start, aby uwzględnić różne platformy wdrożeniowe. Nie będziemy sprawdzać tego kodu, ale przyjrzymy się kilku ostatnim wierszom metody Start:

```
_connection = new SQLiteConnection(dbPath,
SQLiteOpenFlags.ReadWrite |
SQLiteOpenFlags.Create);
Debug.Log("Final PATH: " + dbPath);
if (newDatabase)
{
CreateDB();
}else
{
CheckForUpgrade();
}
```

10. Pierwsza linia tworzy nowe połączenie SQLiteConnection, które tworzy połączenie z bazą danych SQLite. Połączenie jest nawiązywane poprzez przekazanie ścieżki bazy danych (dbPath) i opcji. Opcje dostępne w połączeniu to żądanie uprawnień do odczytu/zapisu i, jeśli to konieczne, utworzenie bazy danych. W związku z tym, jeśli w dbPath nie zostanie znaleziona żadna istniejąca baza danych, zostanie utworzona nowa pusta baza danych. Następna linia po prostu zapisuje ścieżkę bazy danych do konsoli.

Uwaga: Debug.Log jest odpowiednikiem metody drukowania. Wcześniej używaliśmy druku dla uproszczenia i nadal będziemy to robić wszędzie tam, gdzie jest to właściwe.

11. Po nawiązaniu połączenia sprawdzamy, czy została utworzona nowa baza danych, sprawdzając zmienną boolean newDatabase. Zmienna newDatabase została wcześniej ustawiona nad sekcją kodu, określając, czy istniejąca już baza danych była już obecna. Jeśli newDatabase ma wartość true, wywołujemy CreateDB, w przeciwnym razie wywołujemy CheckForUpgrade.

12. Metoda CreateDB nie tworzy fizycznego pliku bazy danych na urządzeniu. Zamiast tego robi się to w kodzie połączenia, który omówiliśmy wcześniej. Metoda CreateDB tworzy instancje tabel obiektów lub schematu w bazie danych w następujący sposób:

```
private void CreateDB()
{
Debug.Log("Tworzenie bazy danych...");
zmienna min. =
_connection.GetTableInfo("Potwór");
if(minfo.Liczba>0)
_connection.DropTable<potwór>();
_connection.CreateTable<Monster>();
Debug.Log("Tablica potworów została utworzona.");
zmienna informacja =
_connection.GetTableInfo("Wersja bazy danych");
if(vinfo.Count>0)
_connection.DropTable<DatabaseVersion>();
_connection.CreateTable<DatabaseVersion>
();
Debug.Log("Tabela wersji bazy danych
Utworzony.");
_connection.Insert(nowa wersja bazy danych
{
Wersja = Wersja bazy danych
});
Debug.Log("Wersja bazy danych zaktualizowana do"
+ Wersja bazy danych);
Debug.Log("Baza danych została utworzona.");
}
```

13. Nie zniechęcaj się liczbą instrukcji Debug.Log w tej metodzie; najlepiej po prostu myśleć o nich jako o pomocnych komentarzach. Po zalogowaniu instalacji najpierw ustalamy, czy tabela Monster została już utworzona przy użyciu metody GetTableInfo na połączeniu. GetTableInfo zwraca kolumny/właściwości tabeli; jeśli nie ustawiono żadnych kolumn ani właściwości, minfo będzie miało wartość 0. Jeśli jednak tabela jest obecna, usuniemy ją lub usuniemy i utworzymy nową tabelę, korzystając z naszych obecnych właściwości Monster. Postępujemy zgodnie z tym samym wzorcem dla następnej tabeli DatabaseVersion. Jeśli GetTableInfo zwraca vinfo.Count > 0, usuń tabelę, w przeciwnym razie po prostu kontynuuj. Zobaczysz, że gdy dodamy więcej obiektów do InventoryService, będziemy musieli utworzyć nowe tabele w ten sam sposób. Opakowanie

SQLite4Unity3d zapewnia nam obiekt Framework Relational Mapping (ORM), który pozwala nam mapować obiekty do tabel relacyjnych baz danych. Dlatego czasami będziemy używać zamiennie terminów obiekt i tabela.

14. Po utworzeniu tabel obiektów tworzymy nowy obiekt DatabaseVersion i zapisujemy go w bazie danych za pomocą metody Insert na _connection. Obiekt DatabaseVersion jest bardzo prosty i ma tylko jedną właściwość o nazwie Version. Używamy tego obiektu/tabeli do śledzenia wersji bazy danych.

15. Pamiętaj, że jeśli nie musimy tworzyć nowej bazy danych, to sprawdzimy aktualizację metodą CheckForUpgrade w następujący sposób:

```
private void CheckForUpgrade()
{
try
{
var version = GetDatabaseVersion();
if (CheckDBVersion(version))
{
//newer version upgrade required
Debug.LogFormat("Database current
version {0} - upgrading to {1}", version,
DatabaseVersion);
UpgradeDB();
Debug.Log("Database upgraded");
}
}
catch (Exception ex)
{
Debug.LogError("Failed to upgrade
database, running CreateDB instead");
Debug.LogError("Error - " +
ex.Message);
CreateDB();
}
}
```

16. Metoda CheckForUpgrade najpierw pobiera bieżącą wersję pliku bazy danych, a następnie porównuje ją z wersją wymaganą przez kod w metodzie CheckDBVersion. Jeśli kod wymaga nowszej wersji bazy danych, ustawionej przez ustawienie DatabaseVersion w usłudze InventoryService, uaktualnia bazę danych. Jeśli baza danych nie wymaga aktualizacji, gra korzysta z aktualnej bazy danych. Jeśli jednak wystąpi błąd podczas sprawdzania wersji lub wystąpi jakiś inny błąd, kod założy, że coś jest nie tak z istniejącą bazą danych i po prostu utworzy nową wersję. Więcej czasu poświęcimy później na samą aktualizację bazy danych.

17. Na koniec przyjrzymy się metodzie CreateMonster wywoływanej przez CatchSceneController:

```
public Monster CreateMonster()

p ()

{

var m = new Monster

{

Name = "Chef Child",

Level = 2,

Power = 10,

Skills = "French"

};

_connection.Insert(m);

return m;

}
```

18. Metoda CreateMonster obecnie po prostu tworzy zakodowany na stałe obiekt Monster i wstawia go do bazy danych za pomocą metody _connection.Insert. Następnie zwraca nowy obiekt do kodu wywołującego. Jeśli masz doświadczenie w pracy z relacyjnymi bazami danych i pisaniu kodu SQL, miejmy nadzieję, że docenisz prostotę wkładki tutaj. W następnej sekcji zaktualizujemy CreateMonster i inne metody działania.

Operacje CRUD na potworach

Obecnie nasza usługa Inventory tworzy tylko tego samego potwora i potrzebujemy go do tworzenia nowych potworów i wykonywania innych operacji, takich jak odczytywanie, aktualizowanie i usuwanie potworów. Nawiasem mówiąc, standardowe operacje na bazie danych, takie jak tworzenie, odczyt,aktualizacja i usuwanie, są często określane jako CRUD. Więc podwiń rękawy; tak naprawdę zrobimy trochę kodowania i zbudujemy potwora CRUD. Ponownie otwórz skrypt InventoryService w wybranym edytorze i przewiń w dół do metody CreateMonster. Usuń metodę CreateMonster i wykonaj następujące instrukcje, aby ją zastąpić i dodać inne nowe metody:

* CREATE: Dodaj następującą metodę, aby zastąpić metodę CreateMonsters:

```
public Monster CreateMonster(Monster m)

{
```

```
var id = _connection.Insert(m);

m.Id = id;

return m;

}
```

Zamiast tworzyć zakodowanego potwora, teraz bierzemy obiekt potwora i wstawiamy go jako nowy obiekt/rekord w bazie danych. Zwraca to nowy automatycznie inkrementowany identyfikator, który ustawiliśmy w obiekcie Monster, i zwracamy go z powrotem do kodu wywołującego. Kolejna część naszego kodu pozwoli nam stworzyć szczegóły potwora.

* READ (single): Będziemy obsługiwać dwie wersje metody odczytu:

jeden do czytania lub znajdowania pojedynczego potwora, a drugi do czytania wszystkich potworów. Dodaj następujący kod, aby odczytać pojedynczego potwora:

```
public Monster ReadMonster(int id)

{

return _connection.Table<Monster>()

.where(m => m.Id == id).FirstOrDefault();

}
```

Ta metoda pobiera identyfikator i znajduje pasujący obiekt potwora w tabeli za pomocą metody Where, która jako parametr przyjmuje delegata funkcji. Wygląda na to, że kod używa Linq to SQL, ale tak nie jest. Where i FirstOrDefault są dodawane jako część implementacji SQLite, aby zachować wieloplatformowość.

Uwaga: iOS nie obsługuje obecnie Linq, co często powoduje zamieszanie dla programistów wywodzących się z tradycyjnego języka C# nawet na innych platformach, takich jak Linux czy Mac. Jeśli chcesz, aby Twoja aplikacja była zgodna z wieloma platformami, unikaj całkowicie korzystania z przestrzeni nazw System.Linq.

* READ (all): Obsługa czytania wszystkich potworów jest jeszcze prostsza:

```
public IEnumerable<Monster> ReadMonsters()

{

return _connection.Table<Monster>();

}
```

Wystarczy jeden wiersz kodu, aby wyciągnąć wszystkie potwory z bazy danych. Nie ma nic prostszego.

* UPDATE: Zaktualizuj kod potwora w następujący sposób:

```
public int UpdateMonster(Monster m)

{

return _connection.Update(m);

}
```

Metoda UpdateMonster pobiera obiekt potwora i aktualizuje go w bazie danych. Zostanie zwrócona wartość int zawierająca liczbę zaktualizowanych rekordów/potworów. Powinno to zawsze zwracać 1. Zwróć uwagę, że obiekt potwora przekazany do metody aktualizacji powinien mieć istniejący identyfikator. Jeśli właściwość Id obiektu potwora wynosi 0, należy zamiast tego użyć metody CreateMonster. Metoda CreateMonster utworzy nowego potwora w bazie danych i ustawi właściwość Id.

* DELETE: Wreszcie, gdy nie będziemy już potrzebować obiektu potwora, chcielibyśmy móc go usunąć za pomocą następującego kodu:

```
public int DeleteMonster(Monster m)

{

return _connection.Delete(m);

}
```

Metoda DeleteMonster jest podobna do metody UpdateMonster. Pobiera potwora, którego chcesz usunąć, a następnie usuwa go z bazy danych. Zwraca liczbę usuniętych obiektów, która powinna wynosić tylko 1. Ponownie, obiekt potwora musi mieć poprawną właściwość Id. Jeśli nie ma prawidłowego identyfikatora, to i tak nie istnieje w bazie danych.

Mamy nadzieję, że doceniasz łatwość, z jaką zakodowaliśmy te podstawowe operacje CRUD potworów. Posiadanie mapowania relacyjnego obiektu dostępnego jako część opakowania SQLite4Unity3d pozwoliło nam szybko zaimplementować trwałość bazy danych dla naszego inwentarza potworów. W żadnym momencie nie musimy nawet wypowiadać słów SQL, nie mówiąc już o napisaniu kodu SQL. Ponadto w przyszłości implementacja innych obiektów w usłudze Inventory powinna być równie łatwa.

Aktualizacja sceny Catch

Kiedy wdrożyliśmy nowe operacje CRUD do przechowywania naszych potworów w ekwipunku, złamaliśmy istniejący skrypt CatchSceneController. Jeśli pamiętasz, usunęliśmy starą przykładową metodę CreateMonster i napisaliśmy nową metodę, która po prostu tworzy wpis potwora w bazie danych. Oznacza to, że nie tylko musimy naprawić nasz zaktualizowany kod, ale także potrzebujemy sposobu na utworzenie instancji nowych losowych właściwości potworów. Jak zwykle, zanim naprawimy kontroler scen, zajmijmy się kwestią tworzenia nowych losowych właściwości potworów. Idealnym rozwiązaniem jest tutaj stworzenie prostej statycznej klasy zwanej MonsterFactory, która będzie losowo budować potwory. Postępuj zgodnie ze wskazówkami, aby zbudować nasz nowy skrypt MonsterFactory:

1. Kliknij prawym przyciskiem myszy (naciśnij klawisz Ctrl i kliknij na komputerze Mac) folder Assets/FoodyGo/Scripts/Services w oknie projektu. Z menu kontekstowego wybierz Utwórz | Skrypt C#. Zmień nazwę skryptu MonsterFactory.

2. Kliknij dwukrotnie nowy skrypt, aby otworzyć go w wybranym edytorze.

3. Edytuj plik tak, aby skrypt odpowiadał następującym za pomocą packt.FoodyGO.Database;

```
using UnityEngine;

namespace packt.FoodyGO.Services

{
```

```
public static class MonsterFactory
{
}
}
```

4. Praktycznie usunęliśmy skrypt startowy, ponieważ potrzebujemy tylko prostej statycznej klasy.

5. Następnie dodamy listę losowych imion, które odmienimy w imię potwora. Dodaj następujące pole bezpośrednio w klasie:

```
public static class MonsterFactory
{
public static string[] names =
{
"Chef",
"Child",
"Sous",
"Poulet",
"Duck",
"Dish",
"Sauce",
"Bacon",
"Benedict",
"Beef",
"Sage"
};
```

6. Możesz dodać tyle innych nazw, ile chcesz do listy; pamiętaj tylko, że powinieneś po każdym wpisie poprzedzić przecinek, z wyjątkiem ostatniego.

7. Następnie stworzymy kilka innych pól do przechowywania umiejętności i innych maksymalnych właściwości:

```
public static string[] skills =
{
"French",
"Chinese",
"Sushi",
```

```
"Breakfast",
"Hamburger",
"Indian",
"BBQ",
"Mexican",
"Cajun",
"Thai",
"Italian",
"Fish",
"Beef",
"Bacon",
"Hog",
"Chicken"
};
public static int power = 10;
public static int level = 10;
```

8. Podobnie jak w przypadku listy nazwisk, możesz również dodawać własne wpisy do listy umiejętności. Pamiętaj, że umiejętność jest jak specjalność w kuchni lub produkcie spożywczym. Umiejętności te wykorzystamy w dalszej części gry, gdy wyślemy nasze potwory do restauracji w celu znalezienia pracy.

9. Po usunięciu właściwości, przejdziemy teraz do napisania metody i pomocników CreateRandomMonster:

```
public static Monster CreateRandomMonster()
{
var monster = new Monster
{
Name = BuildName(),
Skills = BuildSkills(),
Power = Random.Range(1, power),
Level = Random.Range(1, level)
};
return monster;
```

```
}
private static string BuildSkills()
{
var max = skills.Length - 1;
return skills[Random.Range(0, max)] + ","
+ skills[Random.Range(0, max)];
}
private static string BuildName()
{
var max = names.Length - 1;
return names[Random.Range(0, max)] + " " +
names[Random.Range(0, max)];
}
```

10. Kod jest dość prosty, ale omówimy kilka rzeczy. Używamy Random.Range w głównej metodzie i pomocnikach (BuildName, BuildSkills), aby zapewnić losowy zakres wartości. W przypadku metod pomocnika nazw i umiejętności te losowe wartości są używane jako indeks w tablicach nazw lub umiejętności w celu zwrócenia wartości ciągu. Losowe wartości są następnie łączone w zestaw umiejętności oddzielonych nazwami lub przecinkami.

```
public static int power = 10;
public static int level = 10;
```

11. Właściwości mocy i poziomu można łatwo ustawić ponownie za pomocą metody Random.Range. Użyj wartości 1, początku, do maksymalnej wartości właściwości, którą ustawiliśmy powyżej.

12. Jak zawsze, kiedy skończysz edycję skryptu, zapisz go.

13. Otwórz skrypt CatchSceneController w edytorze Unity z folderu Assets/FoodyGo/Scripts/Controllers.

Uwaga : W zależności od edytora kodu znalezienie pliku w edytorze skryptów może być trudne. Dlatego podczas otwierania nowych skryptów zawsze odwołujemy się do edytora Unity.

14. Wybierz i usuń metodę Start w górnej części pliku. Przepiszemy kod w następujący sposób:

```
void Start()
{
var m =
MonsterFactory.CreateRandomMonster();
print(m);
}
```

15. Te kilka linijek kodu oznacza, że po zainicjowaniu CatchScene zostanie wygenerowany nowy, losowy potwór.

16. Po zakończeniu edycji zapisz plik i wróć do edytora Unity. Poczekaj, aż skrypty się skompilują, a następnie naciśnij Play, aby rozpocząć scenę. Sprawdź w oknie konsoli dane wyjściowe nowych właściwości potworów, a zobaczysz więcej losowych wartości.

Przykład losowo generowanych właściwości potworów w konsoli

Tak więc teraz, gdy rozpocznie się CatchScene, zostaną utworzone losowe potwory, które gracz będzie mógł złapać. Chcemy jednak, aby atrybuty potworów określały, jak łatwo lub trudno jest graczowi je złapać. To, co musimy zrobić, to dodać kod, który sprawi, że potwór będzie trudniejszy do złapania i sposób na ucieczkę. Wykonaj następujące czynności, aby dodać trochę trudności do CatchScene:

1. Pierwszą rzeczą, którą zrobimy, jest dodanie nowych pól do skryptu CatchSceneController. Otwórz skrypt CatchSceneController z folderu Assets/FoodyGo/Scripts/Controllers.

2. Po istniejących polach i tuż nad metodą Awake dodaj nowe pola:

```
public Transform escapeParticlePrefab;

public float monsterChanceEscape;

public float monsterWarmRate;

public bool catching;

public Monster monsterProps;;
```

3. EscapeParticlePrefab będzie naszym efektem cząsteczkowym, gdy potwór ucieknie.monsterChanceEscape określa szansę ucieczki. monsterWarmRate określa, jak szybko potwór nagrzewa się po trafieniu. łapanie jest po prostu zmienną bool, której użyjemy do wyjścia z pętli. Wreszcie monsterProps przechowuje losowo generowane właściwości potworów.

4. Zmień kod w metodzie Awake na następujący kod:

```
monsterProps =

MonsterFactory.CreateRandomMonster();

print(monsterProps);

monsterChanceEscape = monsterProps.Power *

monsterProps.Level;

monsterWarmRate = .0001f *

monsterProps.Power;

catching = true;

StartCoroutine(CheckEscape());
```

5. Najpierw przechowujemy nowe losowe właściwości potworów w zmiennej o nazwie monsterProps. Następnie, jak widać po kodzie, szansę na ucieczkę uzyskujemy, mnożąc razem moc i poziom potwora. Po zmodyfikowaniu tego wskaźnika ciepła przez pomnożenie wartości bazowej przez moc. (Na razie

nie przejmuj się zbytnio zakodowanym na stałe .0001f.) Następnie ustawiamy nasz stan łapania na true i na koniec uruchamiamy współprogram o nazwie CheckEscape.

6. Teraz dodaj współprogram CheckEscape w następujący sposób:

```
IEnumerator CheckEscape()
{
while (catching)
{
yield return new WaitForSeconds(30);
if (Random.Range(0, 100) <
monsterChanceEscape && monster!= null)
{
catching = false;
print("Monster ESCAPED");
monster.gameObject.SetActive(false);
Instantiate(escapeParticlePrefab);
foreach (var g in
frozenDisableList)
{
g.SetActive(false);
}
}
}
}
```

7. Wewnątrz współprogramu CheckEscape znajduje się pętla while, która będzie działać tak długo, jak łapanie jest prawdziwe. Pierwsza instrukcja wewnątrz pętli daje lub skutecznie wstrzymuje pętlę na 30 sekund, co oznacza, że zawartość pętli while będzie uruchamiana tylko co 30 sekund. Po tej przerwie następuje test, aby sprawdzić, czy losowa wartość z zakresu 0-100 jest mniejsza niż nasza monsterChanceEscape. Jeśli tak, a potwór (MonsterController) nie jest pusty, potwór ucieka.

8. Kiedy potwór ucieka, dzieje się kilka rzeczy. Najpierw ustawiamy stan łapania na false, co zatrzymuje pętlę. Następnie wypisujemy komunikat do konsoli, zawsze dobra praktyka. Następnie wyłączamy monster.gameobject, tworzymy instancję tej cząsteczki ucieczki i na koniec wyłączamy elementy sceny. Aby wyłączyć elementy sceny, wykonujemy iterację po zamrożeniuDisableList.

9. Tuż w instrukcji if metody OnMonsterHit wprowadź podświetlony nowy wiersz kodu:

```
monster = go.GetComponent< MonsterController >
```

```
();
if (monster != null)
{
monster.monsterWarmRate = monsterWarmRate;
```

10. Ten wiersz po prostu aktualizuje monsterWarmRate potwora (MonsterController) do tej samej wartości, którą obliczyliśmy powyżej w metodzie Awake.

11. Pozostając w metodzie OnMonsterHit, dodaj jeszcze kilka linii zaraz po print("Monster FROZEN"); oświadczenie, jak podkreślono:

```
print("Monster FROZEN");
//save the monster in the player inventory
InventoryService.Instance.CreateMonster(monste
rProps);
```

12. Ten wiersz kodu po prostu zapisuje właściwości potwora lub obiekt potwora w ekwipunku bazy danych po ich złapaniu. Pamiętaj, że podczas dodawania nowego potwora do ekwipunku musimy użyć metody CreateMonster.

13. Po zakończeniu edycji zapisz plik i wróć do edytora Unity. Upewnij się, że kod kompiluje się bez problemów.

14. Zanim będziemy mogli przetestować zmiany, musimy utworzyć i dodać nowy escapeParticlePrefab do CatchSceneController.

15. Z folderu Assets/Elementals/Prebabs(Mobile)/Fire w oknie Project przeciągnij prefabrykat Explosion do okna Hierarchy i upuść go. Zobaczysz na scenie efekt cząsteczkowy eksplozji.

16. Wybierz obiekt Eksplozja w oknie Hierarchia. Zmień nazwę obiektu na EscapePrefab i ustaw pozycję Transform komponentu Z na -3.

17. Teraz przeciągnij obiekt EscapePrefab do folderu Assets/FoodyGo/Prefabs, aby utworzyć nowy prefabrykat. Powodem, dla którego stworzyliśmy nowy prefabrykat, jest to, że zmieniliśmy domyślną pozycję obiektu względem naszej sceny.

18. Usuń EscapePrefab ze sceny, wybierając go w oknie Hierarchii i naciskając klawisz Delete.

19. Wybierz obiekt CatchScene w oknie Hierarchia. Przeciągnij EscapePrefab z folderu Assets/FoodyGo/Prefabs na puste miejsce Escape Particle Prefab w komponencie Catch Scene Controller w oknie Inspektora.

20. Zapisz scenę i projekt. Naciśnij Play w oknie edytora, aby uruchomić i przetestować grę. Spróbuj odtworzyć tę scenę kilka razy i zauważ różnicę w tym, jak łatwo lub trudno jest teraz złapać potwora. Zauważ, że potwory na wyższych poziomach mocy są teraz niemożliwe do złapania. Nie martw się, zrekompensujemy to później, gdy dodamy różne kulki poziomu zamrożenia do nowej sekcji ekwipunku w przyszłej części.

21. Oczywiście zbuduj i wdróż grę na swoje urządzenie mobilne i przetestuj. Tak jak w tej chwili, wydaje się, że scena po prostu się zawiesza, gdy potwór ucieka lub zostaje złapany. Naprawimy to później w GameManagerze, kiedy wszystko połączymy.

Do tej pory mamy teraz MonsterFactory, który generuje losowe potwory. Atrybuty potworów określają poziom trudności sceny Złap. Następnie, gdy potwór zostanie złapany, przechowujemy jego atrybuty w nowej usłudze InventoryService w bazie danych SQLite. Wygląda na to, że naszym następnym zadaniem jest zbudowanie interfejsu użytkownika dla ekwipunku, do którego przejdziemy w następnej sekcji.

Tworzenie sceny ekwipunku

Wspaniałą rzeczą w dzieleniu zawartości gry na sceny jest nasza zdolność do rozwijania i testowania każdego funkcjonalnego elementu oddzielnie. Nie musimy martwić się uruchamianiem innych gier, wydarzeniami czy funkcjonalnościami spowalniającymi nasz rozwój. Jednak w pewnym momencie musimy zebrać wszystkie te elementy razem i ważne jest, aby często testować całą grę. Zanim zaczniemy pracę nad sceną Inventory, dokonamy kolejnego pełnego zresetowania wszystkich skryptów, tak jak to zrobiliśmy w poprzednich częściach. Oznacza to, że będziemy importować kilka nowych i zaktualizowanych skryptów i nie będziemy w stanie szczegółowo opisywać rozległych zmian. W pozostałym rozdziale nie będziemy mieli czasu na podkreślenie żadnego interesującego kodu, ale zaleca się, abyś wędrował po kodzie w wolnym czasie. Tym, którzy dokonali własnych zmian w skryptach, zaleca się samodzielne utworzenie kopii zapasowych tych elementów. Postępuj zgodnie ze wskazówkami tutaj, aby wykonać import zasobów:

1. Zapisz scenę i projekt i wykonaj kopię zapasową projektu w innej lokalizacji, jeśli dokonałeś jakichkolwiek zmian, które chcesz zachować.

2. Z menu wybierz Zasoby | Importuj pakiet | Pakiet niestandardowy..., aby otworzyć okno dialogowe Importuj pakiet...

3. W oknie dialogowym przejdź do folderu z pobranym kodem źródłowym Chapter_6_Assets i wybierz plik Chapter6_import2.unitypackage. Następnie kliknij Otwórz, aby rozpocząć importowanie zasobów.

4. Gdy otworzy się okno dialogowe Import Unity Package, po prostu upewnij się, że wszystkie elementy są zaznaczone, a następnie kliknij przycisk Importuj.

Po załadowaniu zaktualizowanych i nowych skryptów zbudujmy nową scenę Inventory, postępując zgodnie z następującymi wskazówkami:

1. Z menu wybierz Plik | Nowa scena. Stworzy to nową pustą scenę z kamerą i kierunkowym światłem.

2. Z menu wybierz Plik | Zapisz scenę jako... Po wyświetleniu monitu nazwij ekwipunek sceny i zapisz go.

3. Z menu wybierz GameObject | Utwórz puste. Zmień nazwę nowego obiektu InventoryScene i zresetuj transformację do zera w oknie Inspektora.

4. Przeciągnij obiekty Main Camera i Directional Light na nowy obiekt InventoryScene w oknie Hierarchy. To zrewiduje obiekty, tak jak zrobiliśmy to w innych scenach.

5. Z menu wybierz GameObject | interfejs użytkownika | Płyta. Spowoduje to dodanie Panelu z elementem nadrzędnym Canvas i obiektem EventSystem.

6. Wybierz EventSystem w oknie Hierarchia i naciśnij Usuń, aby go usunąć. Pamiętaj, Unity doda jeden dla nas później.

7. Wybierz obiekt nadrzędny Canvas i zmień jego nazwę na InventoryBag w oknie Inspektora.

8. W oknie Hierarchia przeciągnij nowy obiekt InventoryBag do InventoryScene, aby uczynić go dzieckiem.

9. W oknie Hierarchia wybierz obiekt Panel. Następnie w oknie Inspektora zmień kolor komponentu obrazu, klikając gniazdo. Spowoduje to otwarcie okna dialogowego Kolor. Zmień wartość Hex na #FFFFFF na dole okna dialogowego, a następnie zamknij je. Spowoduje to ustawienie całego okna gry na białym tle.

Stanowi to podstawę naszego InventoryScene. Zanim zagłębimy się głębiej w niektóre komponenty, najpierw utworzymy prefabrykat przedmiotów z ekwipunku:

1. Wybierz obiekt InventoryBag w oknie Hierarchia. Następnie z menu wybierz GameObject | interfejs użytkownika | Przycisk. Spowoduje to dodanie nowego obiektu Button jako elementu podrzędnego InventoryBag.

2. Zmień nazwę przycisku na MonsterInventoryItem i ustaw transformację Rect | Anchor Presets, aby rozciągnąć do góry, przytrzymując klawisze obrotu i położenia w oknie inspektora.

3. Usuń składnik obrazu, klikając ikonę koła zębatego obok niego i wybierając Usuń składnik z menu rozwijanego.

4. Zobaczysz teraz komunikat ostrzegawczy w komponencie Button. Zmień właściwość Przejście komponentu przycisku na brak. Spowoduje to usunięcie ostrzeżenia.

5. Z folderu Assets/FoodyGo/Scripts/UI w oknie projektu przeciągnij skrypt MonsterInventoryItem na obiekt przycisku MonsterInventoryItem w oknie Hierarchy. Spowoduje to dodanie skryptu ekwipunku do obiektu.

6. Kliknij prawym przyciskiem myszy (naciśnij klawisz Ctrl i kliknij na komputerze Mac) element MonsterInventoryItem w oknie Hierarchia. Z menu kontekstowego wybierz UI | Surowy obraz.

7. Po zaznaczeniu obiektu Raw Image w oknie Inspektora zmień Raw Image | Właściwość tekstury, klikając ikonę tarczy obok pola. W oknie dialogowym Wybierz teksturę wybierz teksturę potwora. Zmień także Rect Transform | właściwości width i height do wartości 80.

8. W oknie Hierarchia wybierz podrzędny obiekt tekstowy MonsterInventoryItem i naciśnij Ctrl + D (polecenie + D na komputerze Mac), aby zduplikować obiekt.

9. Wybierz pierwszy obiekt Tekst i zmień jego nazwę na TopText w oknie Inspektora. Zmień także składnik Tekst | Akapit | Wyrównanie do środka-góra

10. Powtórz ten proces dla drugiego obiektu Text(1), ale zmień nazwę obiektu BottomText i ustaw wyrównanie akapitu na centerbottom.

11. W oknie Hierarchii przeciągnij obiekt Raw Image tuż pod MonsterInventoryItem, tak aby był pierwszym dzieckiem.

12. Z okna Hierarchii przeciągnij element MonsterInventoryItem do folderu Assets/FoodyGo/Prefabs w oknie projektu. To sprawi, że MonsterInventoryItem stanie się prefabrykatem. Zachowaj oryginalny obiekt na scenie.

Po utworzeniu naszego elementu ekwipunku wrócimy i dokończymy torbę ekwipunku:

1. Wybierz obiekt InventoryBag w oknie Hierarchia, a następnie z menu wybierz GameObject | interfejs użytkownika | Przewiń widok. Spowoduje to dodanie widoku przewijania obok panelu. Przeciągnij widok przewijania na panel, aby uczynić go dzieckiem.

2. Wybierz obiekt Scroll View w oknie Hierarchia. Następnie w oknie Inspektora zmień Rect Transform | Ustawienia kotwicy do rozciągania — rozciągaj, przytrzymując klawisze obrotu i pozycji

3. Przy wciąż zaznaczonym widoku przewijania usuń zaznaczenie opcji przewijania w poziomie w komponencie Scroll Rect. Chcemy tylko, aby ekwipunek można było przewijać w pionie.

4. W oknie Hierarchia rozwiń obiekt Scroll View, a następnie rozwiń również podrzędny obiekt Viewport. Spowoduje to ujawnienie obiektu dolnego poziomu o nazwie Content. Wybierz obiekt Zawartość.

5. W oknie Inspektora ustaw Rect Transform | Zakotwicz Presets do góry - rozciągnij ponownie, przytrzymując klawisze obrotu i pozycji.

6. Po wybraniu obiektu Treść w oknie Hierarchia z menu wybierz Komponent | Układ | Grupa układu siatki, aby dodać komponent do obiektu Content. Powtórz proces, wybierając Komponent | Układ | Monter rozmiaru zawartości.

7. Z folderu Assets/FoodyGo/Scripts/UI w oknie Project przeciągnij skrypt InventoryContent na obiekt Content w oknach Hierarchy lub Inspector.

8. Po zaznaczeniu obiektu Content w oknie Inspektora przeciągnij obiekt Scroll View na pustą zawartość Inventory | Przewiń właściwość Rect.

9. Następnie z folderu Assets/FoodyGo/Prefabs w oknie projektu przeciągnij prefabrykat MonsterInventoryItem na pustą zawartość ekwipunku | Miejsce na prefabrykaty w ekwipunku.

10. Potwierdź lub ustaw wartości dla Rect Transform, Grid Layout Group, Content Size Fitter i Inventory Content na wartości.

11. Na koniec w oknie Hierarchii przeciągnij element MonsterInventoryItem na obiekt Content, aby stał się obiektem potomnym. Następnie dezaktywuj obiekt, odznaczając pole wyboru obok nazwy obiektu w oknie Inspektora. Po prostu użyjemy tego obiektu jako odniesienia.

Do tej pory zbudowaliśmy większość sceny Inventory, a teraz wszystko, co musimy zrobić, to połączyć różne rzeczy. Wykonaj następujące instrukcje, aby zakończyć scenę:

1. Przeciągnij skrypt InventorySceneController z Assets/FoodyGo/Scripts/Controllers w oknie Project i upuść go na obiekt InventoryScene w oknie Hierarchy.

2. Wybierz obiekt InventoryScene. Przeciągnij obiekt Content z okna Hierarchy na pusty kontroler sceny Inventory | Miejsce na zawartość ekwipunku w oknie Inspektora.

3. Z folderu Assets w oknie Project przeciągnij scenę Catch do okna Hierarchy. Pozwala nam to zobaczyć, jak obie sceny nakładają się na siebie.

4. Wybierz i przeciągnij obiekt Usługi ze sceny Catch i upuść go na scenę Inventory. Spowoduje to dodanie Usług do sceny Inventory. Jeśli pamiętasz, używaliśmy tych usług tylko do testowania i planowaliśmy usunąć je ze sceny Catch później.

5. Kliknij prawym przyciskiem myszy (naciśnij Ctrl i kliknij, na Macu) na scenie Catch aby otworzyć menu kontekstowe. Wybierz opcję Usuń scenę i po wyświetleniu monitu o zapisanie kliknij przycisk Zapisz.

6. Naciśnij Play, aby uruchomić scenę i zobaczyć wyniki. Poniżej znajduje się przykładowy zrzut ekranu kilku schwytanych potworów treningowych złapanych podczas testowania gry w scenę Catch

Miejmy nadzieję, że spędziliście trochę czasu na łapaniu potworów w testowej scenie połowu po połączeniu usługi ekwipunku i teraz widzicie te złapane potwory. Jeśli nie masz żadnych potworów, nie martw się, połączymy grę przed końcem części. Być może zauważyłeś, że nasze przedmioty z ekwipunku potworów są w rzeczywistości przyciskami, ale nic nie robią. To również jest w porządku, później dodamy szczegółową sekcję ekwipunku. Na razie jednak chcemy zakończyć rozdział, łącząc wszystkie sceny w odpowiednią grę.

Dodawanie przycisków menu

Aby połączyć ze sobą nasze sceny, potrzebujemy danych wejściowych gracza, aby wywołać zdarzenie. Mamy ustawienie wydarzenia, gdy gracz kliknie potwora w scenie mapy. Jednak chcemy również, aby gracz przeniósł się ze sceny Map/Catch do sceny Ekwipunku iz powrotem. W tym celu dodamy kilka przycisków interfejsu użytkownika do każdej ze scen. Ponieważ mamy już otwartą scenę Inventory, zacznijmy od dodania nowego przycisku do tej sceny:

1. Kliknij prawym przyciskiem myszy (naciśnij klawisz Ctrl i kliknij, na komputerze Mac) obiekt InventoryBag w oknie Hierarchia i wybierz UI | Przycisk z menu kontekstowego. Spowoduje to dodanie przycisku tuż pod panelem.

Rozwiń obiekt Przycisk, a następnie wybierz podrzędny obiekt Tekst i naciśnij klawisz Delete, aby go usunąć.

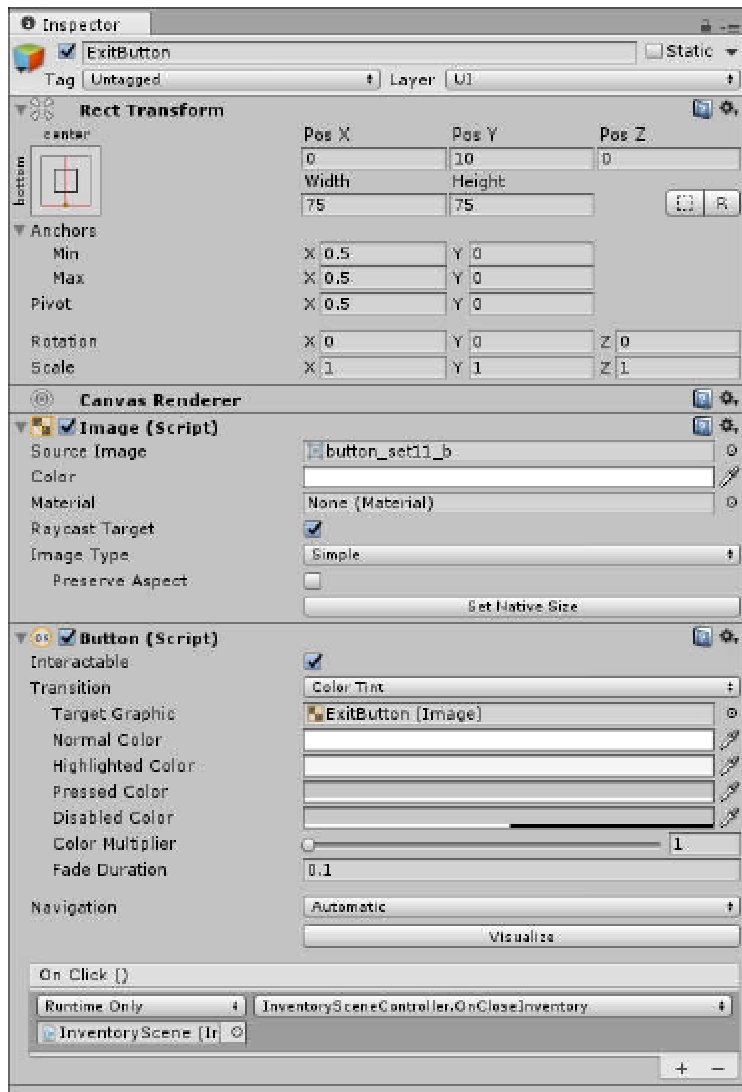
2. Wybierz nowy obiekt Button i zmień jego nazwę na ExitButton w oknie Inspektora.

3. W oknie Inspektora ustaw Rect Transform | Zakotwicz ustawienia wstępne do środka dolnej części, przytrzymując klawisze obrotu i pozycji. Następnie zmień szerokość i wysokość właściwości Rect Transform na 75, a Pos Y na 10.

4. Zmień obraz | Obraz źródłowy, klikając ikonę dziesiątki obok slotu, aby otworzyć okno dialogowe Wybierz duszka. Wybierz duszka o nazwie button_set11_b z okna dialogowego.

5. Na koniec podłączymy przycisk, klikając + pod przyciskiem | Właściwość zdarzenia On Click. Spowoduje to utworzenie nowego miejsca na wydarzenie. Następnie przeciągnij obiekt InventoryScene z okna Hierarchia na puste miejsce Brak(obiekt). Z menu rozwijanego Brak funkcji wybierz InventorySceneController | OnCloseInventory.

6. Pełna konfiguracja przycisku ExitButton jest pokazana na poniższym zrzucie ekranu:



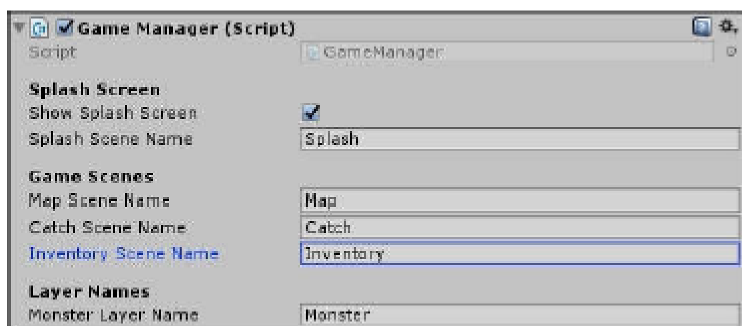
Dodany właśnie przycisk ExitButton służy do zamykania sceny Inventory i powrotu do sceny, która ją otworzyła. Zanim otrzymamy sceny Map and Catch, najpierw musimy przenieść nasze usługi na scenę Game i zająć się kilkoma drobnymi aktualizacjami. Wykonaj instrukcje, aby zaktualizować scenę gry:

1. Z folderu Assets w oknie Project przeciągnij scenę Game do okna Hierarchy.

2. Przeciągnij obiekt Usługi ze sceny Inwentarz do sceny Gra. Będzie to nowa stała siedziba obiektu Usługi. Zmień nazwę usług na _Services w oknie inspektora. Ponownie używamy _ do oznaczenia obiektów, które nie mają zostać zniszczone.

3. Usuń scenę Inventory, klikając ją prawym przyciskiem myszy (naciśnij klawisz Ctrl i kliknij, na komputerze Mac) i wybierając opcję Usuń scenę z menu kontekstowego. Po wyświetleniu monitu zapisz scenę, klikając Zapisz.

4. Wybierz obiekt _GameManager w oknie Hierarchii i zaktualizuj nazwy scen skryptów Game Manager, jak pokazano na zrzucie ekranu:



Następnie dodamy nowy przycisk, aby uzyskać dostęp do sceny Inwentarz na scenie Mapa. Postępuj zgodnie z instrukcjami, aby dodać nowy przycisk:

1. Kliknij dwukrotnie scenę Mapa w folderze Zasoby w oknie Projekt. Spowoduje to zamknięcie sceny Gry, a po wyświetleniu monitu upewnij się, że zapisałeś zmiany.

2. Rozwiń obiekt MapScene w oknie Hierarchia i kliknij prawym przyciskiem myszy (naciśnij Ctrl i kliknij, na Macu), aby otworzyć menu kontekstowe. Z menu wybierz UI | Przycisk, aby dodać nowy przycisk.

3. Wybierz przycisk utworzony w poprzednim kroku iw oknie Inspektora zmień jego nazwę na HomeButton. Ustaw przekształcenie prostokąta | Anchor Presets do dołu do środka, przytrzymując klawisze pozycji/obrotu. Zmień przekształcenie prostokąta | Szerokość/Wysokość na 80 i Pos Y na 10. Następnie zmień Obraz | Obraz źródłowy do button_set06_b, klikając ikonę tarczy i wybierając duszka w oknie dialogowym Wybierz duszka.

4. Przeciągnij skrypt HomeButton z folderu Assets/FoodyGo/Scripts/UI w oknie Projekt i upuść go na obiekt HomeButton w oknie Hierarchia.

5. Wybierz i przeciągnij obiekt HomeButton z okna Hierarchii do folderu Assets/FoodyGo/Prefabs w oknie Projekt, aby uczynić go nowym prefabrykatem.

Wreszcie ostatnia scena, którą musimy zrobić, to scena Catch, która zakończy naszą podróż w obie strony. Wykonaj następujące czynności, aby dodać przycisk HomeButton do sceny:

1. Kliknij dwukrotnie scenę Catch w folderze Assets okna Project. Spowoduje to zamknięcie sceny mapy, a po wyświetleniu monitu upewnij się, że zapisałeś zmiany.

2. Rozwiń obiekt CatchScene w oknie Hierarchia. Z folderu Assets/FoodyGo/Prefabs w oknie projektu przeciągnij prefabrykat HomeButton i upuść go na obiekt Catch_UI. Zobaczysz teraz HomeButton narysowany na górze CatchBall w scenie.

3. Wybierz przycisk HomeButton iw oknie Inspektora zmień Rect Transform | Zakotwicz ustawienia wstępne w prawym górnym rogu, oczywiście przytrzymując klawisze obrotu/pozycji. Następnie zmień Pos X i Pos Y na -10.

4. Zapisz scenę i projekt.

Teraz, po dodaniu przycisków przejścia scen, zaktualizowaniu wszystkich skryptów i ustawieniu wszystkiego innego, nadszedł czas, aby zebrać wszystko razem i uruchomić grę.

Połączenie gry

Wow, teraz, gdy nasza gra składa się z pięciu scen, nadszedł czas, aby połączyć wszystko w pełną grę. Jest tylko kilka rzeczy, które musimy zrobić, aby wszystko było połączone. Wykonaj poniższe instrukcje, aby skonfigurować ustawienia kompilacji i przetestować grę:

1. Otwórz scenę gry w edytorze; to będzie teraz nasza scena startowa.

2. Z menu wybierz Plik | Ustawienia budowania, aby otworzyć okno dialogowe Ustawienia budowania. Przeciągnij sceny z folderu Zasoby w oknie Projekt do Sceny w obszarze Budowania. Zmień kolejność scen, przeciągając je, aby pasowały do tego zrzutu ekranu:

3. Zbuduj i wdróż grę na swoje urządzenie mobilne i przetestuj ją. Zagraj w grę i spróbuj łapać potwory, sprawdzać ekwipunek i tak dalej.

Gra działa dobrze i jest kilka błędów, ale jedną rzeczą, którą możesz od razu zauważyć, jest to, że przyciski i elementy ekwipunku nie są tego samego rozmiaru, do którego zaprojektowaliśmy. Nie martw się, problemem skalowania interfejsu użytkownika zajmiemy się w następnej sekcji. Naprawimy niektóre inne błędy, które mogłeś zauważyć, przechodząc przez inne części.

Problemy z rozwojem mobilnym

Jeśli zwracałeś uwagę, być może zauważyłeś na początku, że tekst naszej sceny Splash nie miał odpowiedniego rozmiaru po wdrożeniu na urządzeniu mobilnym. Powodem, dla którego zostawiliśmy ten problem do tej pory, jest chęć zaprojektowania interfejsu użytkownika tak, aby był niezależny od rozmiaru ekranu. Zmorą każdego programisty mobilnego jest konsekwentne wspieranie niemal nieskończonej różnorodności rozmiarów ekranów. Na niektórych platformach wymaga to opracowania

obrazów/obrazów o wielu rozdzielczościach dla elementów interfejsu użytkownika. Na szczęście system Unity UI ma kilka fajnych opcji skalowania rozmiaru ekranu, które powinny działać na większości naszych platform. Należy jednak pamiętać, że nie wszystkie rozwiązania są w 100% idealne, a na niektórych platformach istnieje możliwość skalowania artefaktów. Aby naprawić nasze obecne problemy z renderowaniem ekranu interfejsu użytkownika, skonfigurujemy komponent Canvas Scaler na wszystkich naszych kanwach interfejsu użytkownika. Wykonaj następujące czynności, aby skonfigurować Canvas Scaler:

1. Otwórz dowolną ze scen gry, które mają elementy interfejsu użytkownika (Mapa, Catch, Splash i Ekwipunek).

2. Znajdź i wybierz element Canvas. Poniżej znajduje się lista elementów Canvas w każdej scenie:

* Scena powitalna: Canvas

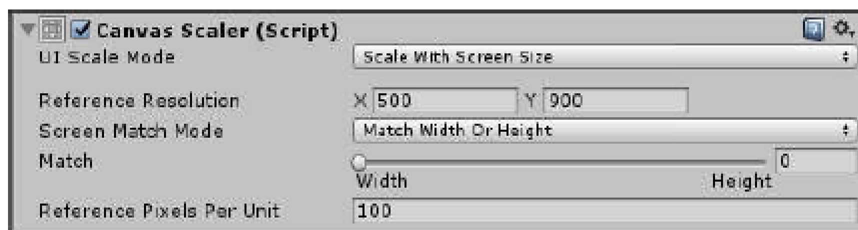
* Scena mapy: UI_Input

* Scena złapania: Catch_UI, Caught_UI

* Scena ekwipunku: InventoryBag

3. Jeśli kanwa nie zawiera komponentu Canvas Scaler, dodaj go za pomocą przycisku Dodaj komponent.

4. Ustaw wszystkie właściwości Canvas Scaler na następujące:



5. Powtórz ten proces dla wszystkich scen i obiektów Canvas. Pamiętaj o zapisaniu scen po wprowadzeniu zmian.

6. Zbuduj i wdróż grę na swój telefon po wprowadzeniu wszystkich zmian. Przetestuj grę i zauważ, jak elementy interfejsu użytkownika skalują się teraz zgodnie z ich przeznaczeniem.

Podsumowanie

Zaczęliśmy powoli od zrozumienia, jak przechowywać złapane potwory gracza. Następnie przyjrzeliśmy się kilku opcjom bazy danych, zanim zdecydowaliśmy się na wieloplatformowe narzędzie do relacyjnego mapowania obiektów dla SQLite, o nazwie SQLite4Unity3d. Opakowaliśmy bazę danych w naszej nowej usłudze Inventory, a następnie napisaliśmy operacje CRUD dla naszych potwornych przedmiotów z ekwipunku. Na tej podstawie zdecydowaliśmy, że potrzebujemy lepszego sposobu na losowe generowanie potworów, więc opracowaliśmy fabrykę potworów. To pozwoliło nam zapętlić się i dokończyć scenę Catch, włączając fabrykę potworów do generowania potworów i usługę ekwipunku do przechowywania złapanych potworów. Ponieważ potwory są przechowywane w bazie danych, opracowaliśmy nową scenę ekwipunku, aby zobaczyć złapane potwory. Na koniec powiązaliśmy wszystko razem za pomocą przycisków menu interfejsu użytkownika i połączyliśmy wszystkie sceny razem jako pełną grę. Oczywiście część zakończyła się rozwiązaniem niektórych problemów z wdrażaniem platformy. W następnym rozdziale wrócimy do odkrywania świata AR wokół gracza i

mapy. Rozszerzymy obiekty i miejsca, z którymi gracz może wchodzić w interakcje na mapie, korzystając z kilku sprytnych usług. Będzie to również wymagało od nas przejścia na wyższy poziom naszej wiedzy GIS/GPS, aby zbadać zapytania przestrzenne i inne zaawansowane koncepcje.