O'REILLY®

Unity
Tworzenie gier mobilnych

Jon Manning
Paris Buttfield-Addison

Helion

Tytuł oryginału: Mobile Game Development with Unity

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-4207-1

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/unitgm
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 9.

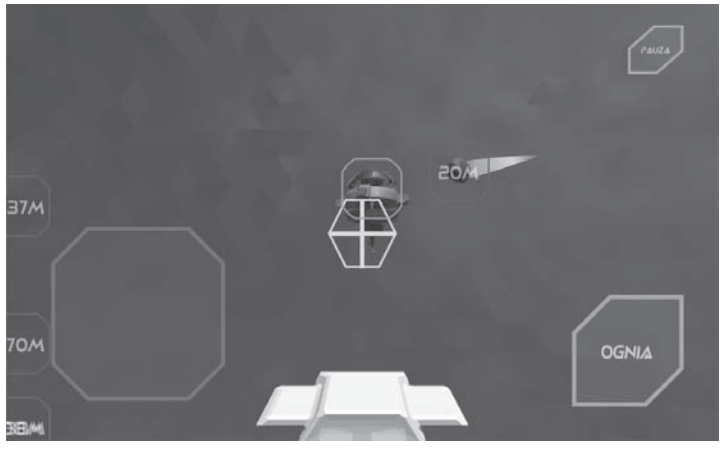
Tworzenie kosmicznej strzelanki

Unity nie jest jedynie platformą, która świetnie nadaje się do budowania gier 2D — z powodzeniem można jej także używać do tworzenia zawartości 3D. Unity zaprojektowano jako silnik 3D na długo przed dodaniem do niej narzędzi do tworzenia gier 2D, dlatego też mechanizmy do opracowywania gier w przestrzeni zostały udostępnione jako pierwsze.

W tym rozdziale dowiesz się, jak można wykorzystać środowisko Unity do napisania gry o nazwie *Rockfall* — kosmicznej strzelanki 3D. Gry tego typu zyskały największą popularność w połowie lat 90. ubiegłego wieku, kiedy to takie produkcje jak *Star Wars: X-Wing* (1993) oraz *Descent: Freespace* (1998) zapewniały graczom możliwości latania w przestrzeni kosmicznej, strzelania do przeciwników i ogólnie pojętego rozwalania wszystkiego, co się rusza. Gry tego typu są blisko powiązane z symulatorami lotów, jednak dzięki temu, że nie oczekuje się od nich zachowania ścisłych realiów fizyki lotu, ich twórcy mogą implementować więcej zabawnych mechanizmów.

Nie chcemy przez to powiedzieć, że nie istnieją symulatory lotów o charakterze gier arkadowych, jednak znacznie łatwiej znaleźć kosmiczną strzelankę tego typu niż realistyczny symulator lotu. Najlepszym wyjątkiem od tej reguły, przynajmniej w ostatnich latach, jest *Kerbal Space Program*. Jest on tak bardzo realistyczny pod względem fizycznej symulacji lotów kosmicznych, że stanowi niemal całkowite przeciwieństwo gry opisanej w tym rozdziale. Jeśli naprawdę zależy Ci na poznaniu mechaniki lotów orbitalnych oraz tego, co się stanie, kiedy w perycentrum zaczniemy przyspieszać w kierunku centrum grawitacji, zdecydowanie jest to gra, po którą musisz sięgnąć. Z tego względu całkiem uzasadnione będzie stwierdzenie, że termin „symulator kosmiczny", choć znacznie częściej używany w odniesieniu do gier, takich jak prezentowana w tym rozdziale, lepiej byłoby zastąpić określeniem „symulator walki kosmicznej". Ale nie czepiajmy się słówek... Lepiej zabierzmy się za strzelanie z laserów.

Pod koniec tej części książki powstanie gra wyglądająca tak, jak na rysunku 9.1.



Rysunek 9.1. Gotowa gra

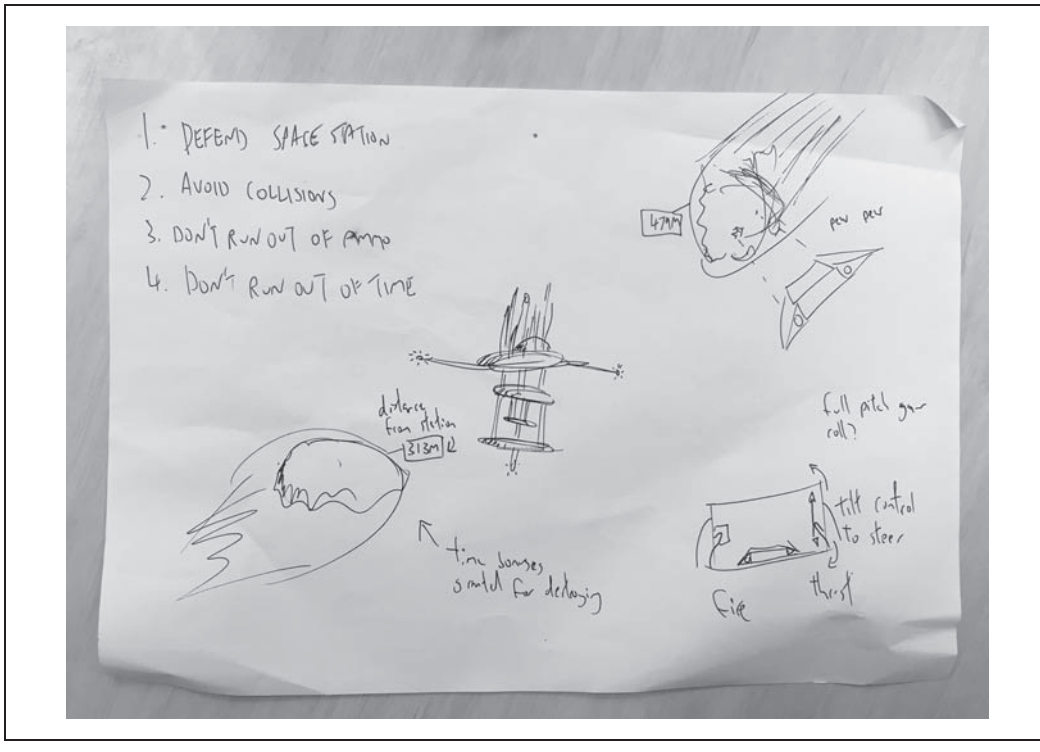
Projektowanie gry

Zaczynając projektowanie gry, musimy określić pewne kluczowe ograniczenia.

- Rozgrywka powinna trwać co najwyżej kilka minut.
- Elementy sterujące powinny być bardzo proste; najlepiej byłoby, gdyby ograniczały się do „leć" oraz „strzelaj".
- Gra powinna koncentrować się na wielu, stosunkowo krótkich wyzwaniach, a nie na jednym, dłuższym. Oznacza to wielu małych przeciwników, zamiast jednej walki z potężnym wrogiem. (To dokładnie na odwrót niż w przypadku gry *Krasnal w kopalni pełnej skarbów* — gry 2D, którą zajmowaliśmy się w poprzedniej części książki).
- Gra powinna koncentrować się niemal wyłącznie na strzelaniu laserami w przestrzeni kosmicznej. Gier tego typu jest zdecydowanie za mało. Takich gier *nigdy nie jest* zbyt wiele.

Niemal zawsze warto zacząć od zapisania na kartce papieru założeń dotyczących głównych, ogólnych koncepcji i pomysłów. Takie „rozważania na papierze" zapewniają nieuporządkowane podejście do zagadnienia, które ułatwia odkrywanie nowych pomysłów, świetnie pasujących do ogólnego planu. My także postąpiliśmy w ten sposób i szybko naszkicowaliśmy ogólny pomysł na grę (patrz rysunek 9.2).

Ten szkic celowo jest bardzo ogólny i narysowany zdecydowanie za szybko; jednak można na nim wskazać kilka pomysłów: asteroidy lecące w kierunku stacji kosmicznej, użytkownika sterującego statkiem kosmicznym za pomocą specjalnego joysticka i strzelającego laserami po dotknięciu wyświetlonego przycisku. Na rysunku widać także kilka dodatkowych szczegółów, stanowiących wynik rozważań o sposobie prezentowania scen tego typu; chodzi o takie elementy jak etykiety pokazujące odległość asteroid od stacji kosmicznej czy też rozważania o tym, jak gracz będzie trzymał urządzenie.

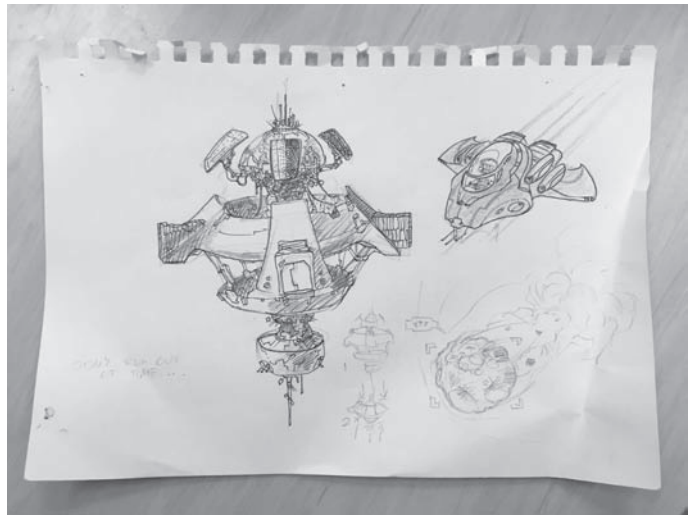


Rysunek 9.2. Początkowa idea gry naszkicowana na kartce papieru

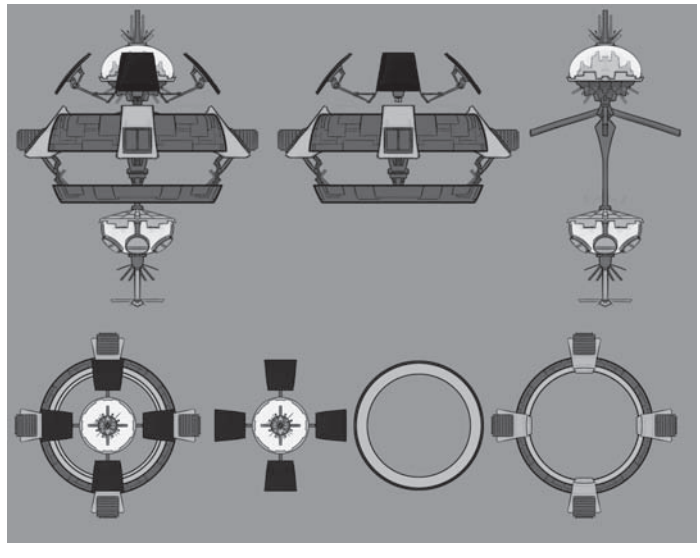
Po przygotowaniu tego ogólnego szkicu zwróciliśmy się do zaprzyjaźnionego artysty Reksa Smeala i poprosiliśmy o przekształcenie naszego rysunku w coś nieco bardziej konkretnego. Choć nie był to absolutnie konieczny etap prac nad projektem gry, jednak ułatwił nam uchwycenie jej ogólnego charakteru. Przede wszystkim zdaliśmy sobie sprawę, że dość dużo uwagi trzeba będzie poświęcić centralnej stacji kosmicznej, której użytkownik ma bronić, gdyż musi wyglądać jak coś, czego warto bronić. Kiedy opowiedzieliśmy znajomemu artyście, o co ma chodzić w grze, przygotował szkic przedstawiony na rysunku 9.3. Po ustaleniu szczegółów Rex zaprojektował coś, co lepiej nadawało się do zamodelowania w grze (patrz rysunek 9.4).

Bazując na tym projekcie, zamodelowaliśmy stację kosmiczną w programie Blender. Na etapie projektowania stacji zdecydowaliśmy o stylu grafik składających się z niewielkiej liczby wielokątów (podejście zainspirowane przez Heather Penn oraz Thimothego Reynoldsa); ze względu na swoją prostotę dobrze nada się do zastosowania w tej grze. (Nie oznacza to wcale, że tworzenie grafiki z wykorzystaniem tego podejścia jest proste i łatwe; chodzi o to, że jego zastosowanie jest łatwiejsze, podobnie jak rysowanie ołówkiem jest łatwiejsze od malowania farbami olejnymi).

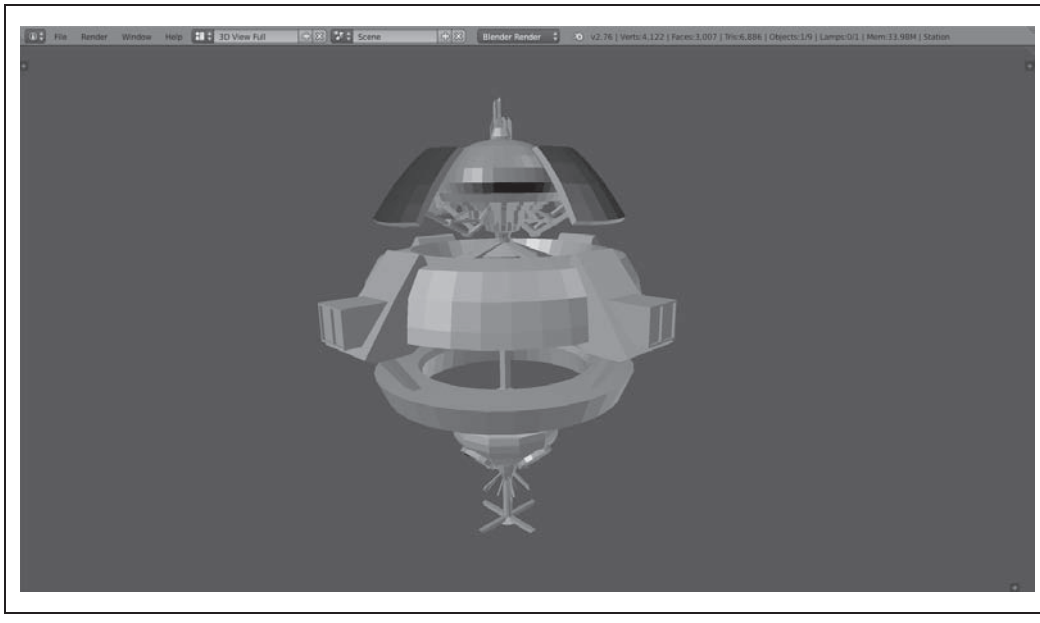
Zamodelowana stacja została przedstawiona na rysunku 9.5. Dodatkowo przygotowaliśmy także model statku kosmicznego oraz asteroidy. Także te dwa modele przygotowaliśmy w programie Blender, a ich efekty zostały przedstawione na rysunkach 9.6 oraz 9.7.



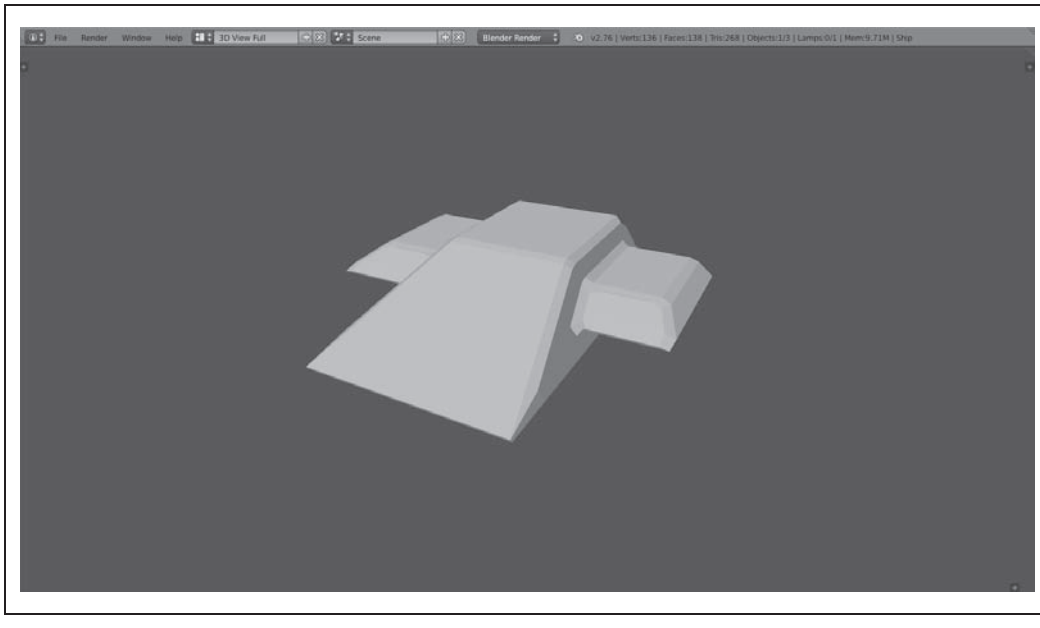
Rysunek 9.3. Początkowa koncepcja wyglądu gry przygotowana przez Reksa



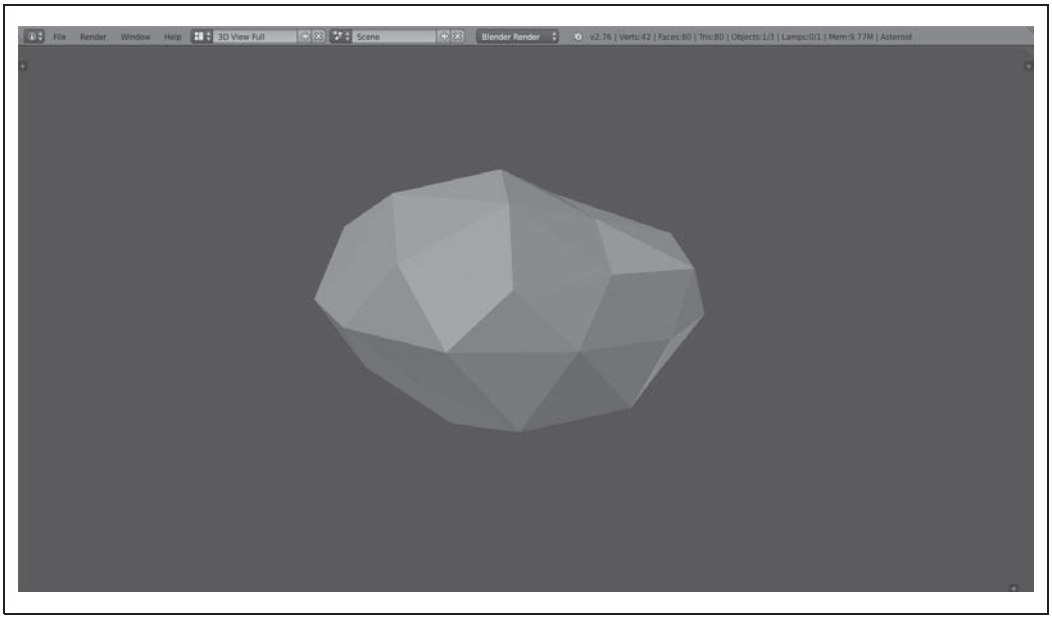
Rysunek 9.4. Poprawiona grafika koncepcyjna stacji kosmicznej, przygotowana do zamodelowania



Rysunek 9.5. Model stacji kosmicznej



Rysunek 9.6. Model statku kosmicznego



Rysunek 9.7. Model asteroidy

Pobieranie zasobów

Podczas tworzenia gry będziesz pracował nad kilkoma zasobami, między innymi efektami dźwiękowymi, modelami i teksturami. Zostały one przygotowane i są dostępne w przykładach dołączonych do książki. Pliki są umieszczone w katalogach, których struktura ułatwia ich przeglądanie i odnalezienie.

Wszystkie przykłady do książki można pobrać z serwera FTP wydawnictwa Helion: *ftp://ftp.helion.pl/przyklady/unitgm.zip*.

Architektura

Podstawowa architektura gry jest w zasadzie bardzo podobna do tej, której użyliśmy w grze *Krasnal w kopalni pełnej skarbów*. Główny obiekt menedżera gry odpowiada za utworzenie jej kluczowych obiektów, takich jak statek kosmiczny pilotowany przez gracza oraz stacja kosmiczna; jest on także informowany o zakończeniu gry, co następuje, kiedy gracz zginie.

Interfejs użytkownika tej gry będzie nieco bardziej złożony niż interfejs poprzedniej. Gra z krasnalem była sterowana dwoma przyciskami oraz wychyleniami urządzenia; jednak w grach 3D, w których gracz może się poruszać w dowolnym kierunku, sterowanie wychyleniami urządzenia nie sprawdza się najlepiej. Dlatego zamiast takiego rozwiązania ta gra zostanie wyposażona w „joystick ekranowy” — fragment ekranu, który będzie wykrywać dotknięcia i pozwalać na sterowanie statkiem poprzez przesuwanie palca w celu wskazania kierunku. Informacje z tego joysticka będą przekazywane do współużytkowanego menedżera wejścia, który z kolei będzie używany przez obiekt statku do aktualizowania swojego położenia.

Sterowanie poprzez przechylanie urządzenia w grach 3D jest znacznie trudniejszym rozwiązaniem, co nie oznacza wcale, że nie można go dobrze zaimplementować. *NOVA 3* to strzelanka z widokiem pierwszej osoby, korzystająca z wychylania urządzenia do celowania oraz obracania postaci i zapewniająca dużą dokładność tych czynności. Warto wypróbować tę grę, by przekonać się, w jaki sposób jej twórcy zaimplementowali sterowanie.

Model walki zastosowany w naszej grze celowo będzie raczej nierealistyczny. Najprostszym i jednocześnie najbardziej realistycznym rozwiązaniem byłoby zamodelowanie obiektów fizycznych, do których byłoby przykładane przyspieszenie oraz zastosowanie sił fizycznych do obracania statku kosmicznego. Jednak takim statkiem trudno byłoby latać, a gracz zbyt szybko mógłby się pogubić. Dlatego zdecydowaliśmy się na zastosowanie nieco oszukanej fizyki: statek zawsze będzie poruszał się do przodu ze stałą szybkością i nie będzie miał żadnego pędu. Co więcej, gracz nie będzie mógł obracać statku, a jakakolwiek próba obrotu zostanie skorygowana (czyli, w odróżnieniu od faktycznej przestrzeni kosmicznej, w naszej grze będzie istnieć pojęcie „góry").

Projekt i decyzje

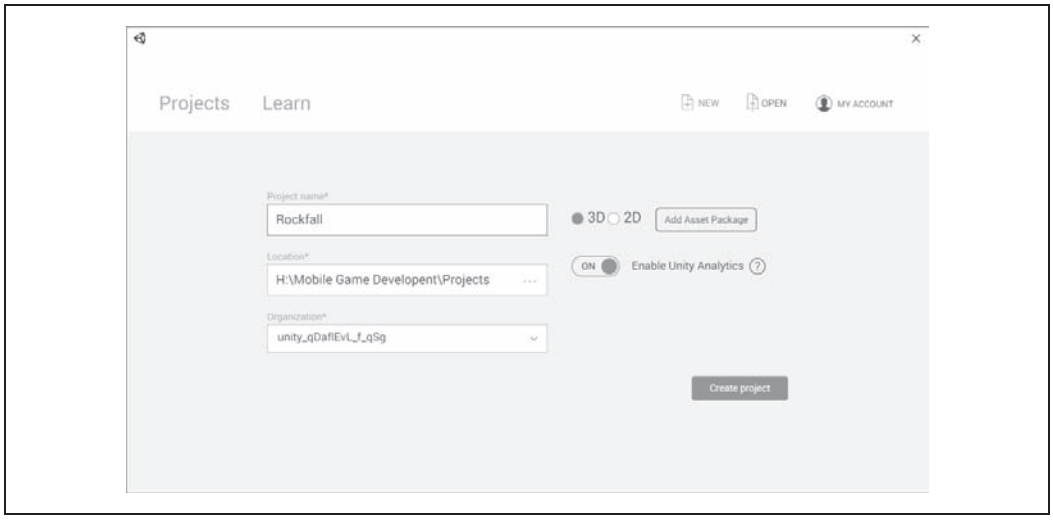
Wszystkie decyzje projektowe dotyczące gier prezentowanych w tej książce są całkowicie uznaniowe. Choć uznaliśmy, że nie będziemy stosować fizyki lotu, nie oznacza to wcale, że wykorzystanie takiej fizyki w symulatorach lotu o charakterze arkadowym jest rozwiązaniem, którego należy unikać. Warto wypróbować własne pomysły i sprawdzić, co z nich wyjdzie. Nie należy z góry uznawać, że gry mogą działać tylko w jeden, konkretny sposób dlatego, że autorzy jakiejś książki tak stwierdzili. W końcu mogli to wymyślić tylko na potrzeby tej książki.

Asteroidy będą prefabrykatami tworzonymi przez specjalny obiekt „generatora asteroid". Będzie on generował asteroidy co jakiś (ustalony) czas i wystrzeliwał je w kierunku stacji kosmicznej. Kiedy asteroida uderzy w stację, zmniejszy jej wartość punktów trafienia; kiedy wartość ta spadnie do zera, stacja ulegnie zniszczeniu, a gra się zakończy.

Tworzenie sceny

Tworzenie gry zaczniemy od przygotowania sceny. W tym celu utworzymy nowy projekt Unity, po czym dodamy do niego statek kosmiczny, który będzie latał po scenie. Oto czynności, od których zaczniemy.

1. *Utworzenie projektu*. Przede wszystkim należy otworzyć nowy projekt Unity o nazwie `Rockfall`, wybierając przy tym tryb *3D* (patrz rysunek 9.8).
2. *Zapisanie nowej sceny*. Kiedy Unity utworzy projekt i wyświetli pustą scenę, należy zapisać projekt, wybierając z menu opcje *File/Save*. Scenę trzeba zapisać w pliku *Main.scene*, w katalogu *Assets*.
3. *Zaimportowanie pobranych zasobów*. W tym celu należy dwukrotnie kliknąć plik *.unitypackage* skopiowany z przykładów dołączonych do książki (patrz punkt „Pobieranie zasobów" na stronie 166). Do projektu trzeba zaimportować wszystkie zasoby.



Rysunek 9.8. Tworzenie nowego projektu

Teraz możemy się już zabrać za opracowywanie statku kosmicznego.

Statek

Prace nad grą zaczniemy do przygotowania statku kosmicznego przy użyciu modelu pobranego z materiałów dołączonych do książki (patrz punkt „Pobieranie zasobów" na stronie 166).

Obiekt *Ship* będzie niewidocznym obiektem zawierającym wyłącznie skrypty; do niego dołączymy wiele obiektów podrzędnych, obsługujących konkretne aspekty prezentowania statku na ekranie.

1. *Utworzenie obiektu Ship*. Aby utworzyć obiekt *Ship*, należy wybrać z menu głównego opcje *GameObject/Create Empty*. Na ekranie zostanie wyświetlony nowy obiekt gry, którego nazwę zmieniamy na `Ship`.

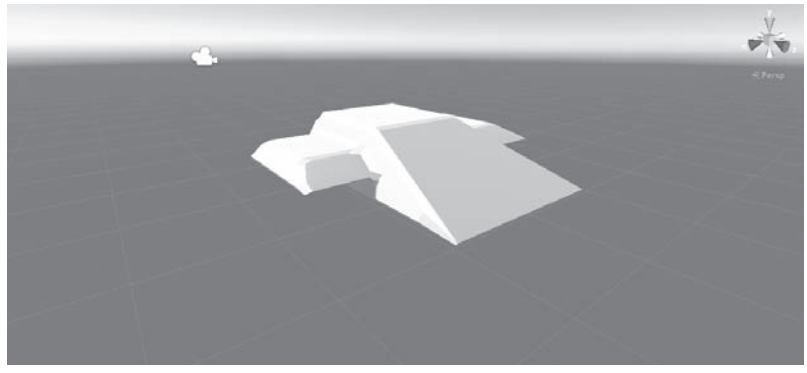
   Teraz zajmiemy się dodaniem modelu.

2. *Dodanie modelu*. Aby dodać model statku kosmicznego, należy otworzyć katalog *Models*, przeciągnąć model *Ship* i upuścić go na obiekcie *Ship*. W efekcie na scenie zostanie wyświetlony trójwymiarowy model statku, co pokazano na rysunku 9.9. Ze względu na to, że model został przeciągnięty na obiekt gry *Ship*, stał się jego elementem podrzędnym; to oznacza, że model będzie przesuwany po scenie wraz z nadrzędnym obiektem *Ship*.

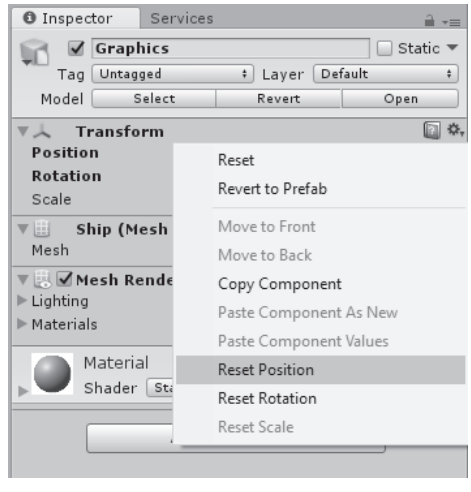
3. *Zmiana nazwy modelu*. Teraz można zmienić nazwę modelu na `Graphics`.

   Musimy zagwarantować, że obiekt *Graphics* będzie wyświetlany w tym samym miejscu, w którym znajduje się jego obiekt nadrzędny — *Ship*.

4. *Zaznaczenie obiektu Graphics*. Kiedy obiekt *Graphics* zostanie zaznaczony, należy kliknąć ikonę koła zębatego, umieszczoną w górnym, lewym rogu komponentu *Transform*, a następnie wybrać opcję *Reset Position* (patrz rysunek 9.10).



Rysunek 9.9. Model statku kosmicznego na scenie



Rysunek 9.10. Zerowanie położenia obiektu Graphics

W polach obrotu (*Rotation*) należy wpisać wartości (-90, 0, 0). To konieczne, gdyż statek został zaprojektowany w programie Blender, który używa innego systemu współrzędnych niż środowisko Unity; konkretnie rzecz biorąc, w Blenderze kierunki „góra" i „dół" określa oś Z, natomiast w Unity — oś Y. Aby rozwiązać ten problem, Unity automatycznie obraca modele przygotowane w Blenderze, by skompensować tę różnicę.

Chcemy, by statek zderzał się z innymi obiektami gry; w tym celu dodamy do niego zderzacz.

5. *Dodanie do obiektu statku zderzacza BoxCollider*. Aby dodać zderzacz, należy zaznaczyć obiekt *Ship* (czyli obiekt nadrzędny obiektu *Graphics*) i kliknąć przycisk *Add Component* umieszczony u dołu panelu *Inspector*. Z listy komponentów trzeba wybrać opcje *Physics/Box Collider*.

Po dodaniu zderzacza należy zaznaczyć pole wyboru *Is Trigger*, a w polach *Size* wpisać — odpowiednio — wartości (2, 1.2, 3). W ten sposób zostanie utworzony prostopadłościan otaczający obiekt gracza.

Nasz statek musi poruszać się do przodu ze stałą szybkością. W tym celu dodamy do niego skrypt przesuwający dowolny obiekt, do którego zostanie dołączony.

6. *Dodanie skryptu ShipThrust*. Gdy obiekt *Ship* wciąż będzie zaznaczony, należy kliknąć przycisk *Add Component* wyświetlony u dołu panelu *Inspector* i utworzyć nowy skrypt C# o nazwie *ShipTrust.cs*.

   Skrypt po utworzeniu należy otworzyć i dodać do niego poniższy fragment kodu.

```
public class ShipThrust : MonoBehaviour {

  public float speed = 5.0f;

  // Przesuwamy statek do przodu ze stałą prędkością
  void Update () {
    var offset = Vector3.forward * Time.deltaTime * speed;
    this.transform.Translate(offset);
  }
}
```

   Skrypt *ShipThrust* udostępnia jeden parametr — `speed`, który jest używany przez funkcję `Update` do przesuwania obiektu do przodu. Sam ruch jest generowany poprzez pomnożenie wektora przesunięcia przez parametr szybkości (`speed`) oraz przez właściwość `Time.deltaTime`, co zapewnia, że obiekt będzie się przesuwać do przodu z tą samą szybkością, niezależnie od tego, ile razy w ciągu sekundy zostanie wywołana funkcja `Update`.

   Koniecznie należy zwrócić uwagę, by skrypt *ShipTrust* został dołączony do obiektu *Ship*, a nie *Graphics*.

7. *Testowanie gry*. Po naciśnięciu przycisku *Play* można będzie zobaczyć, jak statek zaczyna się przesuwać do przodu.

Przesuwanie kamery za statkiem

Kolejnym etapem prac będzie przesuwanie kamery za poruszającym się statkiem kosmicznym. Można to zrobić na kilka różnych sposobów: najprostszym jest umieszczenie kamery w obiekcie *Ship*, co sprawi, że będzie się ona poruszać wraz z nim. Jednak takie rozwiązanie da raczej kiepskie efekty wizualne, gdyż sprawi, że statek nigdy nie będzie się obracał względem kamery.

Znacznie lepszym rozwiązaniem jest zachowanie kamery jako odrębnego obiektu i dodanie do niej skryptu, który zapewni, że wraz z upływem czasu będzie się ona powoli poruszać w odpowiednim kierunku. Oznacza to, że kiedy statek wykona gwałtowny skręt, skompensowanie tego manewru przez kamerę zajmie trochę czasu — czyli da dokładnie taki efekt, jaki powstałby w rzeczywistości, gdy operator starałby się utrzymać statek w kadrze.

1. *Dodanie skryptu SmoothFollow do głównej kamery*. W celu zaimplementowania ruchu kamery należy zaznaczyć obiekt *Main Camera*, kliknąć przycisk *Add Component* i dodać nowy skrypt C# o nazwie *SmoothFollow.cs*.

   Po otworzeniu pliku trzeba zapisać w nim poniższy fragment kodu.

```
public class SmoothFollow : MonoBehaviour
{

  // Obiekt docelowy, za którym podąża kamera
  public Transform target;

  // Wysokość kamery nad obiektem docelowym
  public float height = 5.0f;

  // Odległość kamery od obiektu docelowego, bez uwzględniania wysokości
  public float distance = 10.0f;

  // O ile wolniejsze będą zmiany obrotu i wysokości
  public float rotationDamping;
  public float heightDamping;

  // Funkcja aktualizująca jest wywoływana podczas generowania każdej klatki
  void LateUpdate()
  {
    // Przerywamy, jeśli nie ma obiektu docelowego
    if (!target)
      return;

    // Wyznaczamy aktualne kąty obrotu
    var wantedRotationAngle = target.eulerAngles.y;
    var wantedHeight = target.position.y + height;

    // Zapisujemy aktualne położenie i kierunek kamery
    var currentRotationAngle = transform.eulerAngles.y;
    var currentHeight = transform.position.y;

    // Tłumimy kąt obrotu względem osi Y
    currentRotationAngle = Mathf.LerpAngle(currentRotationAngle,
      wantedRotationAngle, rotationDamping * Time.deltaTime);

    // Tłumimy wysokość
    currentHeight = Mathf.Lerp(currentHeight,
      wantedHeight, heightDamping * Time.deltaTime);

    // Przekształcamy kąt na obrót
    var currentRotation = Quaternion.Euler(0, currentRotationAngle, 0);

    // Ustawiamy położenie kamery na płaszczyźnie X-Z na
    // liczbę metrów określoną zmienną "distance"
    transform.position = target.position;
    transform.position -= currentRotation * Vector3.forward * distance;

    // Ustawiamy położenie kamery, używając nowej wysokości
    transform.position = new Vector3(transform.position.x,
      currentHeight, transform.position.z);

    // I w końcu, ustawiamy kamerę w tym samym kierunku, w którym patrzy
    // obiekt docelowy
```

```
        transform.rotation = Quaternion.Lerp(transform.rotation,
          target.rotation, rotationDamping * Time.deltaTime);

    }
}
```

Skrypt *SmoothFollow.cs* przedstawiony w książce bazuje na kodzie udostępnionym przez Unity. Zmodyfikowaliśmy go nieco, by lepiej dostosować do potrzeb symulatora lotów. Jeśli chcesz przeanalizować pierwotny kod, znajdziesz go w pakiecie *Utility*. Aby go zaimportować, wystarczy wybrać z menu głównego opcje *Assets/Import Package/* ↪*Utility*. Po zakończeniu importowania pierwotny skrypt można będzie wyświetlić, wybierając z menu opcje *Standard Assets/Unity*.

Zasada działania skryptu *SmoothFollow* opiera się na wyliczeniu punktu w przestrzeni, w którym powinna znaleźć się kamera, a następnie punktu *pomiędzy* tą lokalizacją oraz bieżącym położeniem kamery. Stosowanie tego skryptu w kolejnych klatkach da efekt, jakby kamera stopniowo zbliżała się do punktu docelowego, lecz jednocześnie zwalniała. Co więcej, ponieważ punkt docelowy, w którym kamera powinna się znaleźć, zmienia się w każdej klatce, zatem cały czas będzie ona pozostawać gdzieś za nim; a to jest dokładnie ten efekt, o który nam chodziło.

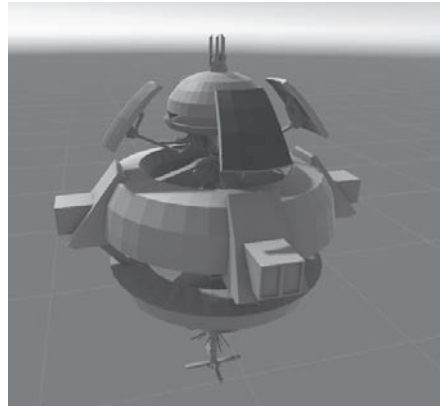
2. *Konfiguracja komponentu SmoothFollow*. W tym celu wystarczy przeciągnąć obiekt *Ship* i upuścić go w polu *Target*.
3. *Sprawdzenie gry*. Teraz należy uruchomić grę, klikając przycisk *Play*. Kiedy to nastąpi, panel *Game* nie będzie już prezentował poruszającego się statku, zamiast tego kamera będzie podążać za statkiem. Można się o tym łatwo przekonać, spoglądając na panel *Scene*.

Stacja kosmiczna

Stację kosmiczną, której bezustannie zagrażają nadlatujące asteroidy, można opracować dokładnie w ten sam sposób jak statek kosmiczny: najpierw utworzymy pusty obiekt gry, a następnie dołączymy do niego model. Co więcej, stacja kosmiczna jest prostsza od statku, gdyż jest całkowicie bierna — wisi jedynie w przestrzeni i jest pod ostrzałem asteroid. Oto czynności, które należy wykonać, by utworzyć obiekt stacji kosmicznej w grze.

1. *Utworzenie obiektu-kontenera dla stacji kosmicznej*. Najpierw należy zbudować nowy, pusty obiekt gry i nadać mu nazwę `Space Station`.
2. *Dodanie modelu jako obiektu podrzędnego*. Aby dodać model stacji, trzeba otworzyć katalog *Models*, przeciągnąć model *Station* i upuścić go na obiekcie gry *Space Satation*.
3. *Wyzerowanie położenia obiektu modelu stacji*. Następnie należy zaznaczyć obiekt *Station* dodany w poprzednim punkcie i kliknąć prawym przyciskiem myszy komponent *Transform*. Z wyświetlonego menu trzeba wybrać opcję *Reset Position*, podobnie jak zrobiliśmy podczas dodawania modelu statku kosmicznego.

Po zakończeniu tych czynności stacja kosmiczna powinna wyglądać tak, jak na rysunku 9.11.



Rysunek 9.11. Stacja kosmiczna

Po dodaniu modelu stacji kosmicznej warto poświęcić nieco uwagi jego strukturze i uzupełnić go o odpowiedni zderzacz. Ten zderzacz jest bardzo istotnym elementem, gdyż asteroidy (które w końcu dodamy do gry) muszą w coś uderzać.

Dodawanie zderzacza należy zacząć od zaznaczenia obiektu modelu i rozwinięciu wszystkich jego elementów podrzędnych. Model ten składa się z wielu siatek podrzędnych, z których główna nosi nazwę *Station*. Należy ją zaznaczyć.

Jeśli teraz spojrzymy na panel *Inspector*, zobaczymy w nim, oprócz komponentów *Mesh Filter* oraz *Mesh Renderer*, komponent *Mesh Collider* (patrz rysunek 9.12). Jeśli komponent ten *nie będzie widoczny*, przeczytaj uwagę „Modele i zderzacze".

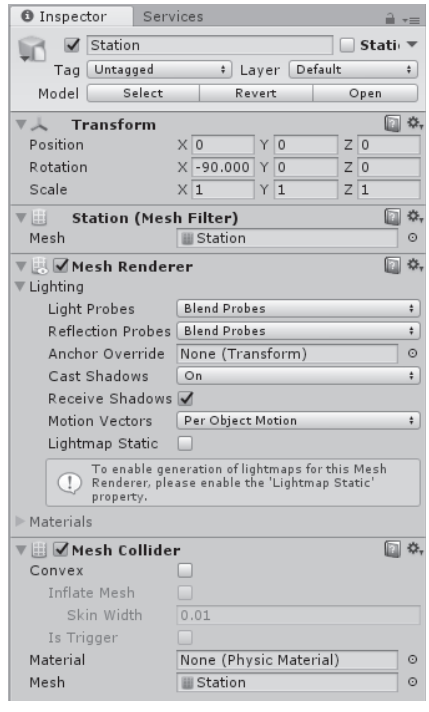
Modele i zderzacze

Podczas importowania modelu Unity może od razu utworzyć dla niego zderzacz. Kiedy zaimportowałeś model z pakietu *Assets*, zostały także zaimportowane ustawienia utworzone dla tego modelu, które zawierały informację, by dodać zderzacz stacji. (Dokładnie to samo zrobiliśmy dla modeli statku kosmicznego oraz asteroid).

Jeśli zderzacz nie jest widoczny lub zaimportowałeś własny model i chciałbyś się przekonać, jak został on skonfigurowany, zaznacz sam model (czyli plik w katalogu *Models*) i przejrzyj jego ustawienia, aby wyświetlić i zmienić jego ustawienia (patrz rysunek 9.13). Przede wszystkim zwróć uwagę na zaznaczone pole wyboru *Generate Collider*.

Sześcian skybox

Obecnie w naszej grze używany jest domyślny sześcian *skybox* tworzony przez Unity, a mający spełniać wymagania gier, których akcja toczy się na powierzchni ziemi. Zmiana tego sześcianu na rozwiązanie zaprojektowane tak, by odpowiadało wyglądowi przestrzeni kosmicznej, będzie krokiem milowym na drodze do zapewnienia odpowiedniego wyglądu budowanej gry.



Rysunek 9.12. Zderzacz stacji kosmicznej

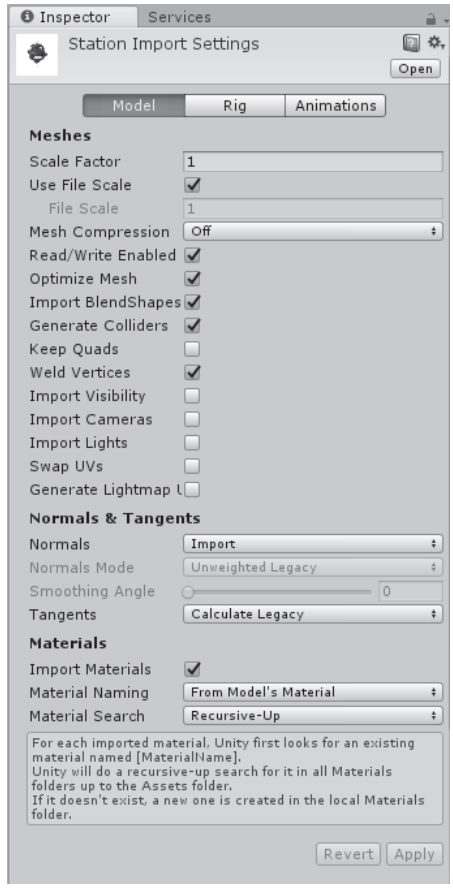
Zgodnie z tym, co sugeruje nazwa, sześciany *skybox* to wirtualne sześciany, które zawsze są rysowane poniżej wszystkich innych elementów sceny i nigdy nie poruszają się względem kamery. Sprawia to wrażenie, jakby tekstury umieszczone na tym sześcianie były nieskończenie daleko — stąd też sprawiają one wrażenie nieba i horyzontu.

Aby zapewnić odczucie, że gracz znajduje się wewnątrz kuli, a nie sześcianu, tekstury wyświetlane na sześcianie *skybox* w tworzonej grze muszą zostać zniekształcone w taki sposób, by krawędzie ich styku były niewidoczne. Można to zrobić na wiele sposobów, na przykład przy użyciu kilku wtyczek do programu Photoshop, jednak większość tych narzędzi służy do przetwarzania zdjęć. Bardzo trudno zdobyć zdjęcia przestrzeni kosmicznej zrobione w celu wykorzystania w grach wideo — znacznie łatwiejszym rozwiązaniem będzie zastosowanie narzędzia tak, by takie zdjęcia przygotować.

Tworzenie sześcianu skybox

Po pobraniu obrazów przygotowanych dla sześcianu *skybox* można dodać je do gry. W tym celu utworzymy nowy materiał, którego następnie użyjemy w ustawieniach oświetlenia sceny. Oto czynności, które należy wykonać.

1. *Utworzenie materiału Skybox*. Aby opracować nowy materiał, należy wybrać z menu opcje *Assets/Create/Material*. Nowemu materiałowi nadajemy nazwę `Skybox` i przenosimy do katalogu o tej samej nazwie.



Rysunek 9.13. Ustawienia importu modelu stacji kosmicznej

2. *Konfiguracja materiału*. Następnie należy zaznaczyć nowy materiał i zmienić wybrany mechanizm cieniowania (ang. *shader*) ze *Standard* na *6 Sided*. Spowoduje to zmianę zawartości panelu *Inspector* — pojawią się w nim pola pozwalające na określenie wszystkich sześciu tekstur (patrz rysunek 9.14).

   Następnie należy odszukać tekstury umieszczone w katalogu *Skybox* i przeciągnąć każdą z nich do odpowiedniego pola — teksturę *Up* należy upuścić na polu *Up*, teksturę *Front* na polu *Front* i tak dalej.

   Postać panelu *Inspector* po określeniu wszystkich sześciu tekstur przedstawia rysunek 9.15.

3. *Połączenie materiału z ustawieniami oświetlenia*. W celu zmiany ustawień oświetlenia należy wybrać z menu *Window* opcje *Lightning/Settings*. Spowoduje to wyświetlenie panelu *Lighting*; w jego górnej części będzie dostępne pole *Skybox Material*, należy na nie przeciągnąć i upuścić utworzony wcześniej materiał *Skybox* (patrz rysunek 9.16).

Rysunek 9.14. Ustawienia sześcianu skybox bez określonych tekstur

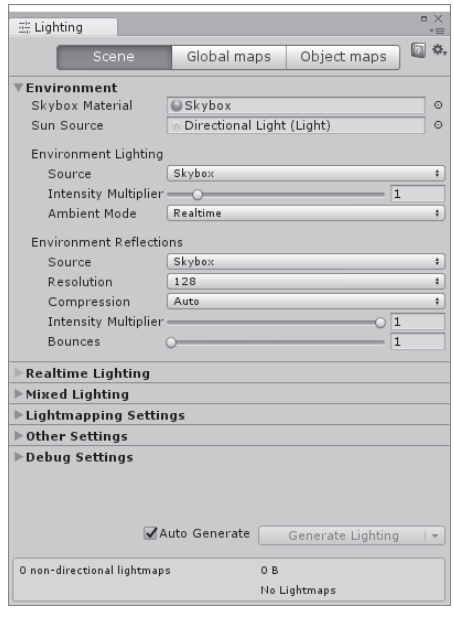
Po wprowadzeniu tych zmian niebo zostanie zastąpione obrazami przestrzeni kosmicznej, widocznymi na rysunku 9.17. Co więcej, system oświetlenia Unity będzie korzystać z informacji udostępnianych przez sześcian *skybox*, aby modyfikować sposób oświetlania obiektów; jeśli przyjrzymy się uważnie, zauważymy, że zarówno statek kosmiczny, jak i stacja mają delikatny zielonkawy odcień; zmiana jest spowodowana zielonymi obrazami przestrzeni kosmicznej, zastosowanymi w sześcianie *skybox*.

Obiekt Canvas

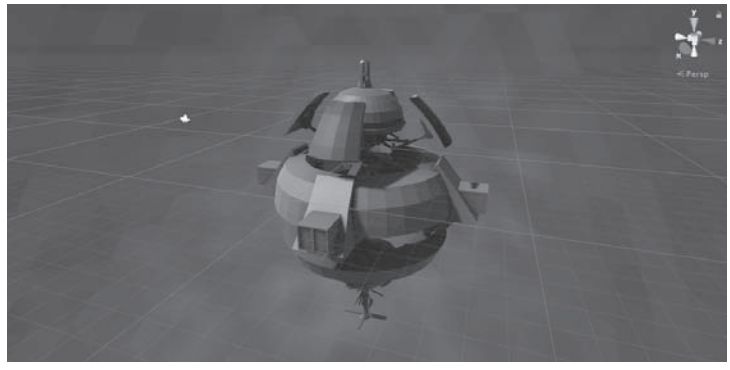
Na razie nasz statek kosmiczny zawsze będzie się poruszał jedynie do przodu, gdyż gracz nie może jeszcze nim sterować. Już niebawem zabierzemy się za dodawanie interfejsu użytkownika pozwalającego na sterowanie statkiem, jednak najpierw musimy dodać do gry i skonfigurować obiekt *Canvas*, na którym będzie wyświetlany ten interfejs użytkownika. Oto czynności, które należy wykonać.

1. *Utworzenie obiektu Canvas*. W tym celu należy wybrać z menu głównego opcje *GameObject/*
   ↪*UI/Canvas*. Spowoduje to utworzenie obiektów *Canvas* oraz *EventSystem*.
2. *Skonfigurowanie obiektu*. Aby rozpocząć konfigurowanie obiektu *Canvas*, przede wszystkim trzeba go zaznaczyć. Następnie w panelu *Inspector* trzeba znaleźć pole *Render Mode* dołączonego komponentu *Canvas*. Jego wartość należy zmienić na *Screen Space — Camera*. Spowoduje to wyświetlenie nowych opcji, pozwalających na określenie ustawień charakterystycznych dla tego trybu renderowania.

Rysunek 9.15. Ustawienia sześcianu skybox po określeniu tekstur

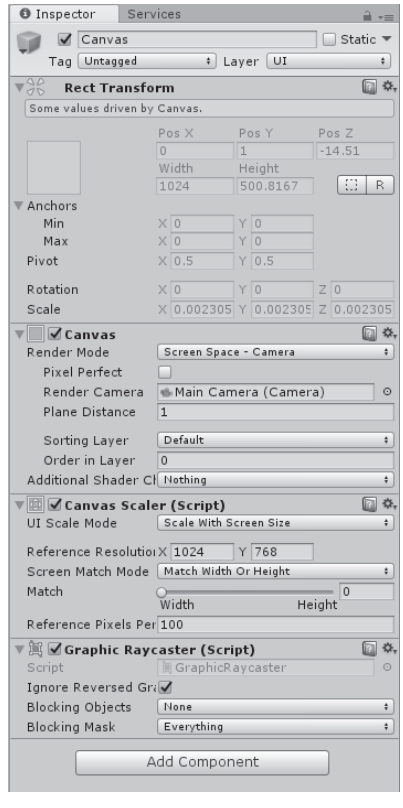


Rysunek 9.16. Konfiguracja ustawień oświetlenia



Rysunek 9.17. Nowe ustawienia sześcianu skybox zastosowane w grze

Na polu *Render Camera* należy upuścić obiekt *Main Camera*, a wartość w polu *Plane Distance* zmienić na 1 (co pokazano na rysunku 9.18). W ten sposób obiekt *Canvas* zostanie umieszczony dokładnie o jedną jednostkę od kamery.



Rysunek 9.18. Panel Inspector dla obiektu Canvas

Następnie ustawienie w polu *UI Scale Mode* należy zmienić z *Canvas Scaler* na *Scale with Screen Size*, a rozdzielczość wzorcową (podawaną w polach *X* i *Y* poniżej nagłówka *Reference Resolution*) trzeba ustawić — odpowiednio — na 1024 i 768, wtedy jej proporcje będą odpowiadać iPadowi.

Po przygotowaniu obiektu *Canvas* możemy zacząć dodawać do niego komponenty.

Podsumowanie

Scena naszej nowej gry jest już gotowa i możemy zacząć implementować systemy niezbędne do prowadzenia rozgrywki. W następnym rozdziale rzucimy się w czerń kosmosu i zaimplementujemy system sterowania statkiem kosmicznym.

Skorowidz

F

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

W

Z

O'REILLY®

Zaprojektuj własny świat z Unity!

Unity jest zintegrowanym środowiskiem do tworzenia trójwymiarowych i dwuwymiarowych gier komputerowych oraz innych materiałów interaktywnych. Szczególnie dobrze nadaje się do pisania gier, które mają działać na wielu różnych urządzeniach i w różnych systemach operacyjnych, szczególnie w środowisku mobilnym. Co istotne, Unity pozwala na prowadzenie prac w szybkim tempie, a sam sposób tworzenia jest nieskomplikowany i bardzo intuicyjny. Wszystko to sprawia, że za pomocą Unity nawet osoba bez praktycznego doświadczenia szybko zacznie tworzyć efektowne gry na urządzenia z systemami iOS i Android.

W tej książce wyczerpująco opisano proces tworzenia gier. Wyjaśniono tu zarówno podstawowe, jak i bardziej zaawansowane pojęcia i techniki związane ze stosowaniem środowiska Unity. Najpierw przedstawiono podstawowe informacje na jego temat. Zaprezentowano struktury gier, grafiki, skryptów, dźwięków, fizyki oraz systemów cząsteczkowych. Następnie opisano pełny proces tworzenia gry 2D oraz 3D. W książce zostały omówione również bardziej zaawansowane zagadnienia związane ze stosowaniem środowiska Unity, takie jak programowanie oświetlenia sceny, graficzny interfejs użytkownika, rozszerzanie edytora Unity, Unity asset store, a także wdrażanie gier oraz cechy zależne od platform systemowych.

Najważniejsze zagadnienia:

- podstawy Unity i pisanie skryptów
- stosowanie zasad fizyki oraz budowa systemów cząsteczkowych
- tworzenie arkadowej gry akcji z przewijanym tłem
- projektowanie symulatora walki w kosmosie
- korzystanie z niestandardowych narzędzi

Dr Jon Manning oraz **dr Paris Buttfield-Addison** są współzałożycielami firmy Secret Lab, zajmującej się pisaniem gier oraz narzędzi do ich tworzenia. W swoim dorobku mają takie gry jak *ABC Play School*, *Night in the Woods* oraz *Qantas Joey Playbox*. Poza grami tworzą YarnSpinner – framework do projektowania gier narracyjnych. Wcześniej pracowali jako programiści gier mobilnych oraz menedżerowie produktu w firmie Meebo (wykupionej przez Google).

ISBN 978-83-283-4207-1
9 788328 342071

Cena: 67,00 zł