Technologia i rozwiązania

Projektowanie gier w środowisku Unity 3.x

Twórz w pełni funkcjonalne i profesjonalne gry 3D!

Helion

Will Goldstone

PACKT PUBLISHING

Tytuł oryginału: Unity 3.x Game Development Essentials

Tłumaczenie: Jacek Janusz

ISBN: 978-83-246-3984-7

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/prgun3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 2

Podstawy tworzenia prototypów i skryptów

Jedną z najlepszych metod zdobywania wiedzy i doświadczenia podczas stawiania pierwszych kroków w dziedzinie projektowania gier jest tworzenie prototypów własnych pomysłów. Środowisko Unity przoduje w tym zagadnieniu, udostępniając wizualny edytor sceny oraz publiczne zmienne składowe, które widnieją na panelu *Inspector* (inspektor) w postaci odpowiednich ustawień. Zapoznanie się z działaniem edytora Unity rozpoczniemy od wygenerowania prototypu prostej mechaniki gry, wykorzystując w tym celu kształty podstawowe oraz tworzenie elementarnych skryptów.

W tym rozdziale zostaną przedstawione następujące zagadnienia:

- tworzenie nowego projektu w środowisku Unity,
- importowanie pakietów z zasobami,
- praca z obiektami gry w widokach *Scene* (scena) i *Hierarchy* (hierarchia),
- dodawanie materiałów,
- tworzenie skryptów w językach C Sharp (C#) oraz JavaScript,
- zmienne, funkcje i polecenia,
- użycie polecenia `Translate()` w celu przemieszczenia obiektów,
- użycie prefabrykatów do przechowywania obiektów,
- użycie polecenia `Instantiate()` w celu konkretyzacji obiektów.

Twój pierwszy projekt w środowisku Unity

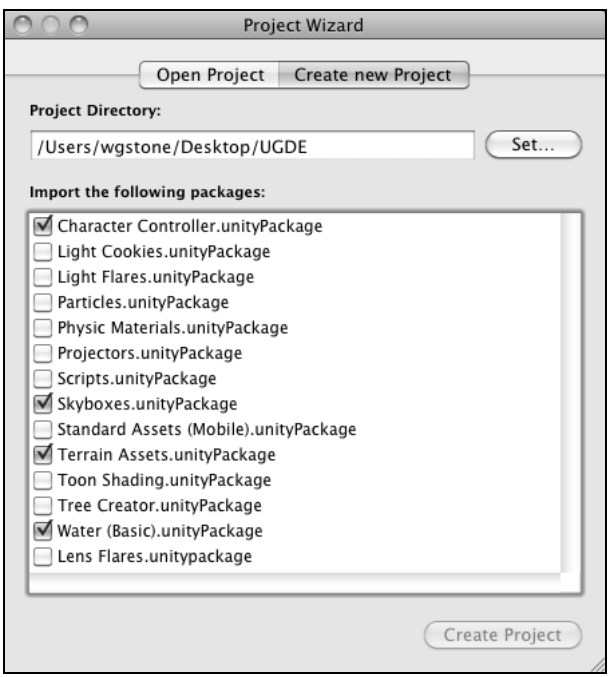
Środowisko Unity jest oferowane w dwóch wersjach: standardowej (darmowej) oraz płatnej (licencjonowanej) dla profesjonalnych projektantów. W niniejszej książce będziemy omawiać opcje, do których mają dostęp użytkownicy wersji darmowej.

Jeśli uruchomisz środowisko Unity po raz pierwszy, zostanie do niego wczytany przykładowy projekt. Zapoznawanie się z najlepszymi wzorami, stosowanymi w najwyższej jakości projektach, jest bardzo przydatne. Jednakże w przypadku początkujących użytkowników przeglądanie profesjonalnie utworzonych zasobów i skryptów mogłoby być deprymujące, dlatego też pominiemy projekt przykładowy i rozpoczniemy prace od samego początku!

W menu głównym środowiska Unity wybierz opcję *File/New Project* (plik/nowy projekt), co spowoduje wyświetlenie okna dialogowego *Project Wizard* (kreator projektu — na poniższym rysunku przedstawiono wersję dla komputera typu Mac). Kliknij zakładkę *Create new Project* (stwórz nowy projekt).

Uwaga

Jeśli chciałbyś, by za każdym razem po uruchomieniu środowiska Unity okno dialogowe *Project Wizard* było automatycznie wyświetlane, powinieneś wywoływać aplikację z naciśniętym klawiszem *Alt* (Mac i PC). Taki sposób działania można również skonfigurować w opcji menu *File/Preferences* (plik/ustawienia).



Kliknij przycisk *Set*[1] (umieść), aby zdefiniować lokalizację folderu dla nowego projektu Unity. W niniejszej książce został on nazwany *UGDE* i umieszczony na pulpicie, by mieć do niego łatwy dostęp.

Kreator projektu pozwala także na zaimportowanie wielu pakietów z zasobami, które są udostępnione za darmo przez firmę Unity Technologies. Zawierają one złożone skrypty, gotowe obiekty i inne elementy graficzne, które są przydatne podczas rozpoczynania pracy z różnymi rodzajami projektów. Te pakiety możesz także zaimportować w każdej chwili, wybierając opcję menu *Assets/Import Package* (zasoby/import pakietu), a następnie odpowiednią pozycję z dostępnej listy. Wybierając opcję *Assets/Import Package/Custom Package* (zasoby/import pakietu/pakiet użytkownika), mógłbyś również zaimportować pakiet znajdujący się w dowolnej lokalizacji na Twoim dysku twardym. Taka funkcjonalność pozwala współdzielić zasoby z innymi użytkownikami oraz umożliwia instalację pakietów, które uzyskałeś poprzez wybranie opcji *Window/Asset Store* (okno/sklep z zasobami).

Z listy pakietów, które można zaimportować, wybierz następujące (jak zaprezentowano na powyższym rysunku):

- *Character Controller* (kontroler postaci),
- *Skyboxes* (skybox — symulacja nieba i horyzontu),
- *Terrain Assets* (zasoby terenu),
- *Water (Basic)* (woda — wersja podstawowa).

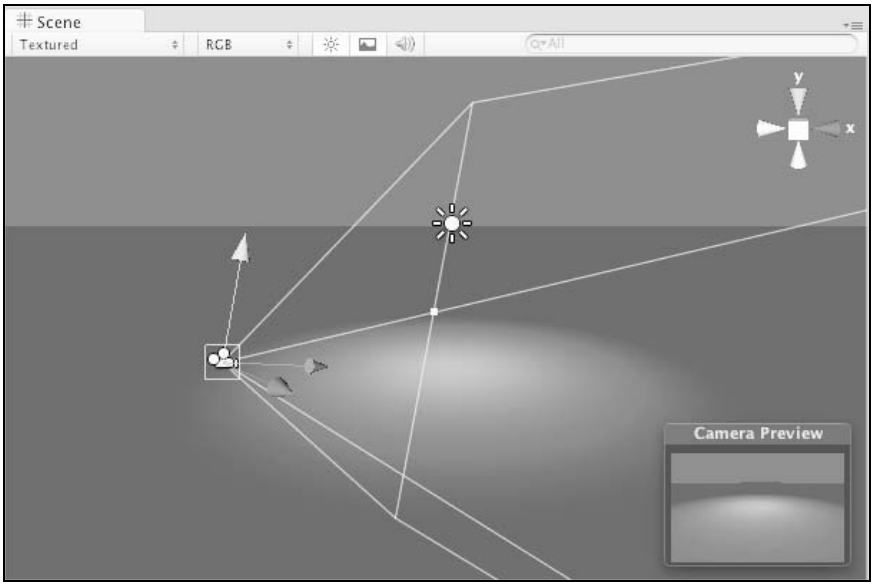
Po zaznaczeniu pakietów kliknij przycisk *Create Project* (stwórz projekt), znajdujący się w dolnej części okna dialogowego. Spowoduje to rozpoczęcie procesu tworzenia nowego projektu, podczas którego będziesz mógł obserwować paski postępu, reprezentujące import poszczególnych pakietów.

Podstawowe środowisko prototypowe

Aby stworzyć proste środowisko, w którym zostanie wygenerowany prototyp określonej mechaniki gry, działania powinniśmy rozpocząć od zdefiniowania podstawowej listy obiektów — zostanie im przypisana funkcjonalność pozwalająca graczowi celować i strzelać do ściany składającej się z prostych sześcianów.

Po zakończeniu pracy środowisko prototypowe będzie się składać z powierzchni zbudowanej z prostych sześcianów, kamery pozwalającej na obserwowanie świata gry w trzech wymiarach oraz źródła światła punktowego, umożliwiającego podświetlenie obszaru, w którym będzie się toczyć akcja gry. Wygenerowany świat powinien wyglądać tak jak na poniższym rysunku:

[1] W wersji dla PC przycisk nazywa się *Browse* (przeglądaj) — *przyp. tłum.*



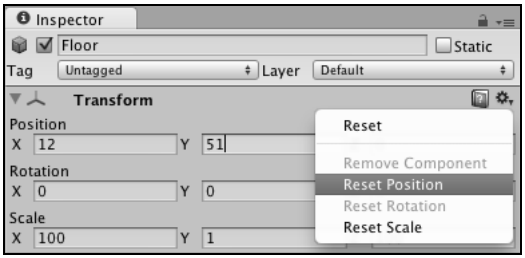
Definiowanie sceny

Ponieważ każda nowa scena zawiera domyślnie obiekt `Main Camera` (główna kamera), działania rozpoczniemy od utworzenia powierzchni dla środowiska prototypowego.

W panelu *Hierarchy* (hierarchia) kliknij przycisk *Create* (stwórz), a następnie z rozwijanego menu wybierz opcję *Cube* (sześcian). Elementy znajdujące się na liście są także dostępne w opcji *GameObject/Create Other* (obiekt gry/stwórz inny) menu głównego aplikacji. Obecnie możesz zauważyć obiekt o nazwie *Cube*, znajdujący się w panelu *Hierarchy*. Wybierz go, a później naciśnij klawisz *F2* (PC) lub *Return* (Mac) albo dwukrotnie z opóźnieniem kliknij nazwę obiektu, żeby ją zmienić. Wprowadź słowo *Floor* (powierzchnia), a następnie naciśnij klawisz *Enter* (PC) lub *Return* (Mac), by zatwierdzić wprowadzenie zmian.

Ze względu na konieczność zapewnienia zgodności rozpoczniemy tworzenie obiektów w punkcie zerowym świata, czyli centrum trójwymiarowego środowiska, w którym pracujemy. Aby upewnić się, że sześcian będący powierzchnią został umieszczony dokładnie w tym miejscu, sprawdź, czy w dalszym ciągu jest wybrany w panelu *Hierarchy*. Następnie zaznacz komponent *Transform* na panelu *Inspector* (inspektor) i zwróć uwagę na to, czy wszystkie wartości współrzędnych X, Y i Z są równe zeru. Jeśli tak nie jest, wprowadź zera w odpowiednich polach edycji lub kliknij ikonę w kształcie kółka zębatego, a potem wybierz z menu podręcznego opcję *Reset Position* (wyzeruj położenie).

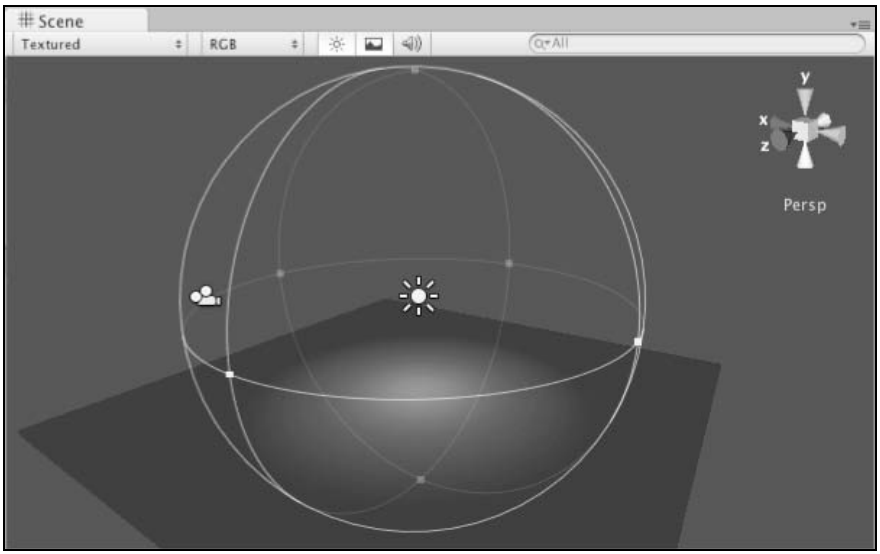
W dalszej kolejności powinniśmy przekształcić sześcian na powierzchnię, odpowiednio powiększając go w osi X i Y. W komponencie *Transform* przejdź do grupy *Scale* (skala), a następnie w polach edycji *X* i Z wprowadź wartości `100`, pozostawiając pole *Y* równe `1`.



Dodawanie prostego oświetlenia

Postaramy się teraz oświetlić obszar powierzchni prototypowej poprzez dodanie źródła światła punktowego. Kliknij przycisk *Create* w panelu *Hierarchy* (lub wybierz opcję menu głównego *GameObject/Create Other*), a następnie wybierz pozycję *Point Light* (światło punktowe). Umieść źródło światła w położeniu `(0, 20, 0)`, używając w tym celu odpowiednich wartości *Position* (pozycja) zawartych w komponencie *Transform*, dzięki czemu będzie się ono znajdować 20 jednostek nad powierzchnią.

Jak zapewne zauważyłeś, oznacza to, że światło nie dociera do całej powierzchni. Przeciągnij więc żółty punkt, który znajduje się na przecięciu linii tworzących zarys światła punktowego w widoku *Scene*, by uzyskać wartość parametru *Range* (zasięg) równą `40`. Parametr ten jest elementem komponentu *Light* (światło) dostępnego w widoku *Inspector*. Wykonane przez Ciebie działanie spowoduje, że na obiekcie tworzącym powierzchnię pojawi się lekka poświata.



Wskazówka

Pamiętaj, że większość komponentów jest powiązana z odpowiednimi narzędziami, służącymi do wizualnej edycji i znajdującymi się w widoku *Scene*. Modyfikacja wartości takich jak *Range* w komponencie *Light*, dostępnym w panelu *Inspector*, spowoduje uaktualnienie wyglądu obiektu w widoku *Scene* w trakcie wprowadzania zmian. Zostaną one zatwierdzone po naciśnięciu klawisza *Return* (*Enter*).

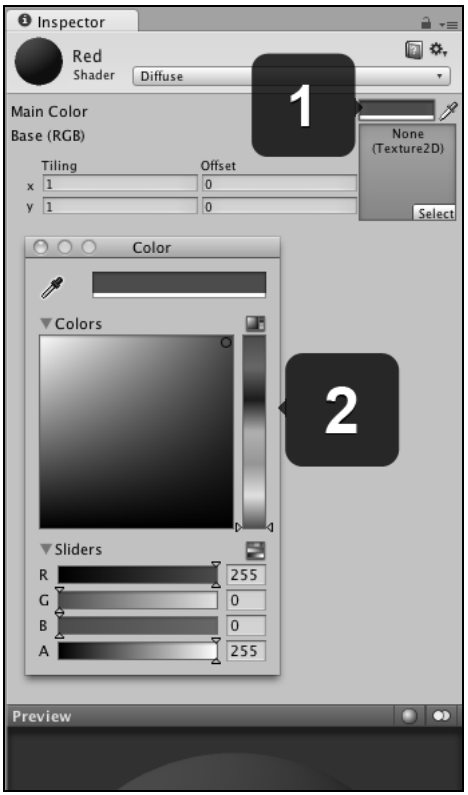
Kolejna cegła w ścianie

Utworzymy ścianę złożoną z sześcianów, która będzie celem dla pocisków. Najpierw utworzymy pierwszą cegłę, uzupełnioną w razie konieczności o komponenty, a następnie powielimy ją w tylu egzemplarzach, aż będziemy dysponowali gotową ścianą.

Tworzenie wzorcowej cegły

Aby uzyskać szablon, pozwalający na wygenerowanie wszystkich cegieł, musimy utworzyć wzorcowy obiekt, który będzie można później klonować. Procedura jest następująca:

1. Kliknij przycisk *Create*, znajdujący się w górnym obszarze panelu *Hierarchy*, a następnie wybierz opcję *Cube*. Używając wartości *Position* w komponencie *Transform*, umieść obiekt w położeniu `(0, 1, 0)`. Upewnij się, czy wciąż jest wybrany w panelu *Hierarchy*, a następnie przybliż jego widok przez przemieszczenie kursora myszy nad okno *Scene* i naciśnięcie klawisza *F*.
2. Uzupełnij obiekt *Cube* o parametry fizyczne, wybierając opcję menu głównego *Component/Physics/Rigidbody* (komponent/parametry fizyczne/bryła sztywna). Oznacza to, że Twój obiekt będzie od tej pory bryłą sztywną, charakteryzującą się takimi parametrami, jak masa czy ciężar. Może także oddziaływać na inne obiekty dzięki wykorzystaniu silnika fizycznego w celu uzyskania realistycznych wyników w świecie 3D.
3. Wreszcie stwórzmy kolor dla obiektu poprzez wygenerowanie odpowiedniego materiału. Materiały służą do uzupełniania obiektów trójwymiarowych o kolor i teksturę. Aby utworzyć nowy materiał, kliknij przycisk *Create*, znajdujący się w panelu *Project*, a następnie z menu rozwijanego wybierz opcję *Material* (materiał). Naciśnij klawisz *F2* (PC) lub *Return* (Mac), aby zmienić jego domyślną nazwę *New Material* (nowy materiał) na *Red* (czerwony).
4. Gdy materiał zostanie wybrany, jego właściwości są wyświetlane w panelu *Inspector*. Kliknij pole koloru (1), znajdujące się po prawej stronie etykiety *Main Color* (główny kolor), aby otworzyć okno dialogowe *Color Picker* (wybór koloru) (2). Będzie ono różnić się wyglądem w zależności od tego, czy używasz komputera typu Mac lub PC. Po prostu wybierz jakiś odcień koloru czerwonego, a następnie zamknij okno. Pole koloru *Main Color* powinno zostać odpowiednio zaktualizowane.



5. Aby użyć stworzonego materiału, przeciągnij go z panelu *Project* i upuść na sześcian znajdujący się w widoku *Scene* lub nazwę obiektu w oknie *Hierarchy*. Materiał zostanie użyty w komponencie *Mesh Renderer* (renderer siatki) dla danego obiektu oraz natychmiast zaprezentowany innym jego komponentom w panelu *Inspector*. Najważniejsze jest jednak to, że Twój sześcian powinien być teraz czerwony! Modyfikacja parametrów materiału przy użyciu opcji *Preview* (podgląd), zastosowanej do dowolnego obiektu, spowoduje wprowadzenie rzeczywistych zmian tylko w oryginalnym elemencie, ponieważ poprzez tę operację odwołujemy się do właściwego zasobu, a nie do nowo utworzonej instancji.

6. Ponieważ sześcian ma już kolor i właściwości fizyczne, wynikające z użycia komponentu *Rigidbody* (bryła sztywna), może zostać sklonowany i utworzyć ścianę złożoną z cegieł. Zanim jednak tego dokonamy, przyjrzyjmy się przez chwilę prawom fizycznym działającym w przestrzeni 3D. Upewnij się, że sześcian został wybrany, a następnie przypisz parametrowi *Y Position* (położenie Y) wartość `15`, zaś *X Rotation* (rotacja X) wartość `40`. Obie te właściwości należą do komponentu *Transform* w widoku *Inspector*. Naciśnij przycisk *Play* (lub skrót klawiszowy *Ctrl+P* dla PC albo *Command+P* dla komputerów Mac), znajdujący się w górnej części interfejsu Unity. Powinieneś zobaczyć spadający sześcian, który pod kątem uderza o powierzchnię i po wykonaniu półobrotu zatrzymuje się.

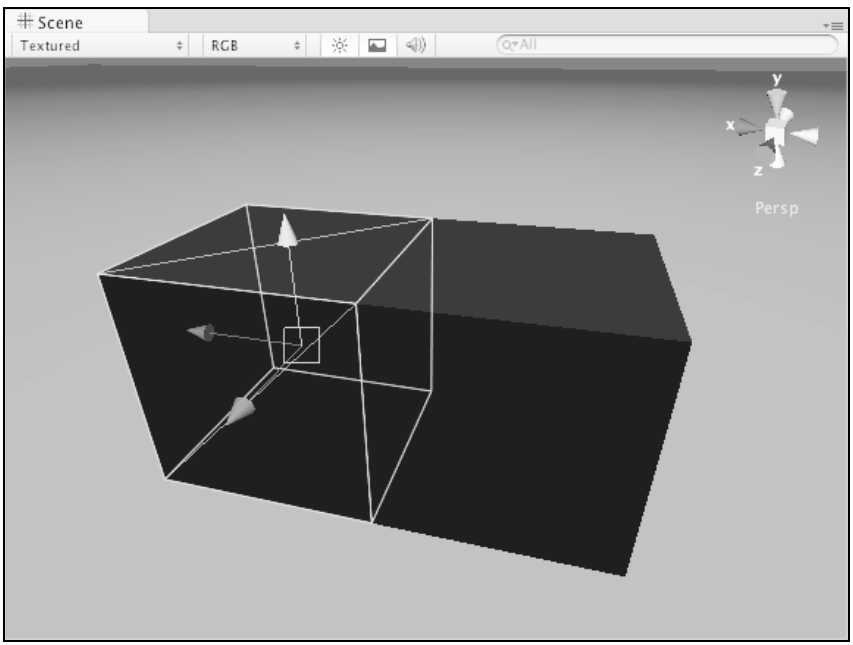
7. Naciśnij przycisk *Play* ponownie, aby zatrzymać symulację. Nie naciskaj przycisku *Pause*, ponieważ spowodowałoby to tylko tymczasowe zatrzymanie animacji, natomiast późniejsze zmiany w scenie nie zostałyby zapamiętane.
8. Przywróć poprzednie ustawienia dla sześcianu: parametrowi *Y Position* przypisz wartość `1`, a *X Rotation* — wartość `0`.

Sprawdziłeś, że cegła została poprawnie zdefiniowana i działa zgodnie z oczekiwaniami. Możesz więc rozpocząć tworzenie rzędu cegieł, z których będzie się składać nasza ściana.

I pstryk! Mamy rząd

Aby ułatwić rozmieszczanie obiektów, środowisko Unity pozwala na użycie opcji przyciągania. Parametry opcji przyciągania mogą być modyfikowane w opcji menu *Edit/Snap Settings* (edycja/ustawienia opcji przyciągania).

Aby zastosować opcję przyciągania, przytrzymaj naciśnięty klawisz *Control* (PC) lub *Command* (Mac) podczas używania narzędzia *Translate* (*W*) w celu rozmieszczenia obiektów w widoku *Scene*. By rozpocząć budowę ściany, przy użyciu skrótu klawiszowego *Ctrl+D* (PC) lub *Command+D* (Mac) skopiuj obiekt cegły, którym już dysponujemy. Następnie przeciągnij go, chwytając za uchwyt czerwonej osi i jednocześnie trzymając wciśnięty klawisz *Control* (PC) lub *Command* (Mac). Dzięki temu obiekt będzie się przemieszczał skokowo po siatce przyciągania. Przesuwaj cegłę w osi X, aż znajdzie się w bezpośrednim sąsiedztwie innej, tak jak przedstawiono na poniższym rysunku:



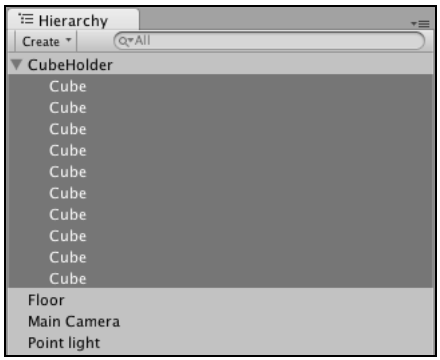
Powtarzaj procedurę kopiowania i rozmieszczania przy użyciu siatki przyciągania aż do momentu, gdy będziesz dysponować rzędem dziesięciu cegieł. Aby przyspieszyć tworzenie konstrukcji, umieścimy go w pustym obiekcie nadrzędnym, który następnie sklonujemy.

> **Przyciąganie do wierzchołków**
>
> Podstawowa procedura przyciągania, której użyliśmy wcześniej, działała poprawnie, ponieważ nasze sześciany charakteryzują się ogólną skalą o wielkości 1. Jednakże podczas skalowania obiektów o bardziej złożonych kształtach powinieneś zastosować opcję przyciągania do wierzchołków. Aby jej użyć, upewnij się, że wybrałeś narzędzie *Translate*, a następnie wciśnij i przytrzymaj klawisz *V*. Gdy umieścisz kursor myszy nad punktem wybranego wierzchołka, przeciągnij cały element do wierzchołka należącego do innego obiektu.

Grupowanie i duplikacja przy użyciu pustych obiektów

Stwórz pusty obiekt, wybierając opcję menu głównego *GameObject/Create Empty* (obiekt gry/stwórz pusty). Następnie umieść go w położeniu `(4.5, 0.5, -1)` przy użyciu komponentu *Transform* z panelu *Inspector*. Zmień jego nazwę na *CubeHolder*.

Zaznacz wszystkie sześciany w panelu *Hierarchy*: wybierz sześcian znajdujący się najwyżej, przytrzymaj klawisz *Shift*, a następnie wybierz sześcian najniższy. W dalszej kolejności przeciągnij zaznaczony zbiór sześcianów do pustego obiektu *CubeHolder*, znajdującego się w panelu *Hierarchy*, aby uczynić go obiektem nadrzędnym. Po wykonaniu tej operacji widok *Hierarchy* powinien się przedstawiać następująco:

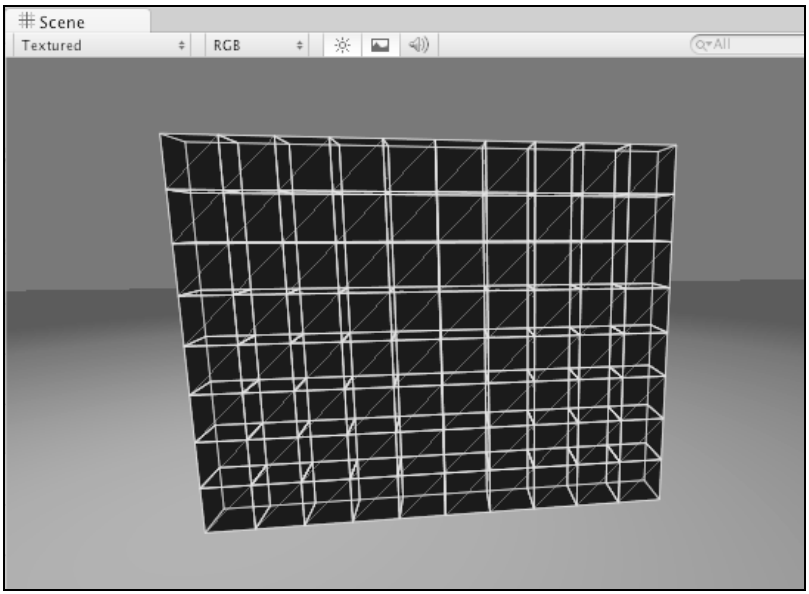


Możesz zauważyć, że obok nazwy obiektu nadrzędnego pojawiła się ikona w postaci strzałki. Oznacza to, że możesz ją kliknąć, co spowoduje rozwinięcie i zwinięcie listy z obiektami podrzędnymi. Aby zaoszczędzić miejsce w panelu *Hierarchy*, kliknij strzałkę, by ukryć wszystkie obiekty podrzędne, a następnie ponownie wybierz *CubeHolder*.

Gdy cały rząd cegieł został utworzony i przyporządkowany do obiektu nadrzędnego, możemy go w prosty sposób zduplikować poprzez przeciągnięcie z użyciem opcji przyciągania w osi Y. Tak jak poprzednio użyj skrótu klawiszowego służącego do klonowania (PC: *Ctrl+D*, Mac:

Command+*D*), a następnie wybierz narzędzie *Translate* (*W*) i zastosuj technikę przeciągania z przyciąganiem (PC: *Ctrl*, Mac: *Command*), by przemieścić rząd cegieł w górę o jedną jednostkę za pomocą zielonego uchwytu osi Y.

Powtarzaj powyższą procedurę, aż utworzysz osiem rzędów cegieł umieszczonych jeden nad drugim. Ściana powinna wyglądać jak na poniższym rysunku. Zwróć uwagę na to, że zostały na nim zaznaczone wszystkie obiekty *CubeHolder* znajdujące się w panelu *Hierarchy*.



Zbuduj i zniszcz!

Gdy zbudowaliśmy ścianę z cegieł, nadszedł czas na stworzenie prostej mechaniki gry, w której gracz może przemieszczać kamerę i wystrzeliwać pociski w kierunku ściany, by ją zburzyć.

Definiowanie widoku

Ustaw kamerę w taki sposób, by była skierowana na ścianę. W tym celu zaznacz obiekt *Main Camera*, znajdujący się w panelu *Hierarchy*, a następnie w komponencie *Transform* zdefiniuj jego pozycję równą `(4, 3, -15)`. Upewnij się także, że wszystkie parametry związane z obrotem są równe zeru.

Wprowadzenie do tworzenia skryptów

Aby rozpocząć naukę programowania, przyjrzymy się prostemu przykładowi, definiującemu tę samą funkcjonalność w językach C Sharp (C#) oraz JavaScript, podstawowych dla projektantów wykorzystujących środowisko Unity. Można również tworzyć skrypty oparte na języku Boo, lecz są one rzadko używane, pomijając osoby, które go dobrze znają.

> **Uwaga**
>
> Kolejne ćwiczenia możesz wykonywać, używając zarówno języka JavaScript, jak i C#. Jednakże w dalszej części książki powinieneś zdecydować się na ten język, który bardziej Ci odpowiada.

Rozpocznij od kliknięcia przycisku *Create* (stwórz), znajdującego się w panelu *Project* (projekt), a następnie wybrania opcji *Javascript* lub *C# Script*.

Twój skrypt zostanie umieszczony w panelu *Project* pod nazwą *NewBehaviourScript*, obok której widnieje niewielka ikona w kształcie strony z napisem *JS* lub *C#*. W trakcie dokonywania wyboru skryptu masz dostęp do podglądu jego kodu w oknie *Inspector* (inspektor). Znajduje się tam także przycisk *Open* (otwórz), którego kliknięcie powoduje otwarcie skryptu w domyślnym edytorze o nazwie Monodevelop. Możesz go także otworzyć, klikając dwukrotnie jego ikonę w panelu *Project*.

Nowy skrypt definiujący zachowanie lub klasę

Bez względu na to, czy zdecydujesz się na język C#, czy JavaScript, przydatne będzie przeczytanie obu dalszych fragmentów, ponieważ zawierają on ogólne informacje o tworzeniu skryptów i mogą również pomóc w podjęciu decyzji o wyborze określonego języka.

W terminologii środowiska Unity każdy nowo utworzony skrypt odpowiada zdefiniowaniu nowej klasy. Jeśli jesteś początkującym programistą, potraktuj klasę jako zbiór działań, właściwości i innych zapamiętanych informacji, do których można mieć dostęp poprzez jej nazwę.

Na przykład klasa o nazwie `Pies` mogłaby zawierać takie właściwości, jak `kolor`, `rasa`, `rozmiar` i `płeć`. Oprócz nich istniałyby takie działania, jak `biegnij` czy `przynieś patyk`. Właściwości mogą być opisane jako **zmienne**, natomiast działania mogą zostać zdefiniowane w postaci **funkcji**, zwanych także metodami.

Aby odwołać się do zmiennej `rasa`, która jest właściwością klasy `Pies`, moglibyśmy podać nazwę klasy, następnie użyć znaku kropki, a wreszcie wprowadzić nazwę zmiennej:

```
Pies.rasa;
```

Do wywołania funkcji klasy `Pies` moglibyśmy zastosować poniższy zapis:

```
Pies.przyniesPatyk();
```

Funkcje możemy również wywoływać z argumentami — nie mają one jednak nic wspólnego z tymi, których używamy codziennie podczas dyskusji z innymi osobami! Przyjmij założenie, że argumenty pozwalają na modyfikowanie działania funkcji. Na przykład w przypadku funkcji `przyniesPatyk` moglibyśmy użyć argumentu, który definiuje, jak szybko pies powinien przynieść patyk. Jej wywołanie mogłoby wyglądać tak:

```
Pies.przyniesPatyk(25);
```

Mimo że zaprezentowane wyżej przykłady są abstrakcyjne, często pozwalają odwzorować kod programu na sytuacje wzięte z życia, co sprawia, że nabierają sensu. Podczas czytania tej książki postaraj się do nich wracać. Możesz także tworzyć własne odwzorowania, co pozwoli Ci lepiej zrozumieć działanie klas informacji oraz ich właściwości.

W czasie pisania skryptu w języku C# lub JavaScript tworzysz nową klasę (lub klasy) zawierającą właściwości (zmienne) oraz polecenia (funkcje), których możesz użyć w określonym miejscu w swojej grze.

Jak wygląda od środka działanie skryptu w języku C#?

Gdy w środowisku Unity utworzysz nowy skrypt w języku C#, od razu otrzymasz pewien kod, którego możesz użyć:

```
using UnityEngine;
using System.Collections;
public class NewBehaviourScript : MonoBehaviour {
    // Use this for initialization
    void Start () {
    }
    // Update is called once per frame
    void Update () {
    }
}
```

Kod rozpoczyna się od dwóch niezbędnych wierszy, zawierających odwołania do samego silnika Unity:

```
using UnityEngine;
using System.Collections;
```

Następnie jest tworzona nazwa klasy, zgodna z nazwą skryptu. W języku C# musisz zdefiniować taką nazwę skryptu, która będzie odpowiadać nazwie zadeklarowanej w nim klasy. Na początku dokumentu widzimy zapis `public class NewBehaviourScript : MonoBehaviour {`, ponieważ `NewBehaviourScript` jest domyślną nazwą, nadawaną przez środowisko Unity każdemu nowo utworzonemu skryptowi. Jeśli w panelu *Project* zmienisz nazwę skryptu, system Unity automatycznie zmieni także nazwę zdefiniowanej w nim klasy.

Kod składający się z klas

Podczas pisania kodu większość funkcji, zmiennych i innych składników skryptu będzie umieszczana wewnątrz klasy języka C#. *Wewnątrz* w tym kontekście oznacza, że musi wystąpić po deklaracji klasy i kończyć się zamykającym nawiasem klamrowym }, znajdującym się na końcu skryptu. Podczas analizowania poleceń zawartych w tej książce możesz założyć, że Twój kod powinien zostać umieszczony wewnątrz klasy zdefiniowanej w skrypcie. Każdy wyjątek od tej reguły zostanie jawnie wskazany. Ta zasada nie jest tak rygorystycznie przestrzegana w języku JavaScript, ponieważ cały skrypt jest w nim definiowany jako oddzielna klasa. Aby zapoznać się ze szczegółami, przeczytaj podrozdział zatytułowany „Jak wygląda od środka działanie skryptu w języku JavaScript?".

Funkcje podstawowe

Środowisko Unity zawiera wiele własnych funkcji, które mogą być używane w celu uruchomienia różnych opcji silnika gry. Dwie z nich są szczególnie ważne podczas tworzenia nowego skryptu w języku C#.

Uwaga

Funkcje języka C# (zwane również metodami) najczęściej rozpoczynają się od słowa `void`. Użyta nazwa określa typ wartości, która jest zwracana przez funkcję, czyli danej otrzymanej po jej wywołaniu. Ponieważ większość funkcji jest tworzona wyłącznie w celu wykonania określonych instrukcji, a nie zwracania jakichś informacji, na początku ich deklaracji będziesz często spotykał słowo `void`. Oznacza ono po prostu, że wywołanie funkcji nie zwróci żadnych wartości.

Funkcje te są następujące:

- **`Start()`**: Zostaje ona wywołana przy pierwszym uruchomieniu sceny, dlatego jest często używana (jak wynika z sugestii zawartej w kodzie) w celach inicjalizacyjnych. Na przykład mógłbyś dysponować zmienną przechowującą wynik, której musi zostać przypisana wartość `0` podczas rozpoczynania gry. W funkcji `Start()` mogłaby także zostać wywołana procedura, która umieszcza postać gracza we właściwym miejscu gry na początku danego poziomu.
- **`Update()`**: Ponieważ stany różnych elementów gry mogą się zmieniać w trakcie jej trwania, funkcja ta jest wywoływana w każdej iteracji podczas działania Twojego programu i jest odpowiedzialna za ich sprawdzanie.

Zmienne w języku C#

Aby zapamiętać informację w zmiennej języka C#, należy użyć następującej składni:

typDanych nazwaZmiennej = wartość;

Na przykład:

```
int currentScore = 5;
```

lub:

```
float currentVelocity = 5.86f;
```

Zwróć uwagę na to, że w powyższych przykładach użyto danych numerycznych: `int` oznacza **liczbę całkowitą**, natomiast `float` **liczbę zmiennoprzecinkową**. Ten drugi przypadek wymaga dodatkowo w języku C# umieszczenia litery `f` po wartości samej liczby. Różni się to w pewien sposób od składni używanej w języku JavaScript. Więcej szczegółów zostanie zaprezentowanych w kolejnym punkcie „Zmienne w języku JavaScript".

Jak wygląda od środka działanie skryptu w języku JavaScript?

Nowo utworzony plik w języku JavaScript zawiera mniej danych niż odpowiadający mu plik języka C#, ponieważ cały skrypt jest uważany za definicję klasy. Zakłada się, że znajdujące się w nim treści zawierają się między niewidocznymi dla użytkownika znacznikami otwierającymi i zamykającymi klasę, gdyż sama jej deklaracja jest ukryta.

Możesz także zauważyć, że wiersze `using UnityEngine;` i `using System.Collections;` są również niewidoczne w pliku języka JavaScript, dlatego nowo utworzony skrypt zawiera tylko funkcję `Update()`:

```
function Update() {
}
```

Deklaracja funkcji w języku JavaScript wygląda inaczej i wymaga użycia słowa kluczowego `function` przed jej nazwą. Deklaracje zmiennych i innych elementów skryptu są także zapisywane w odmienny sposób — w kolejnych punktach zapoznamy się z odpowiednimi przykładami.

Zmienne w języku JavaScript

Składnia służąca do deklarowania zmiennych w języku JavaScript wygląda następująco:

```
var nazwaZmiennej : TypDanych = wartość;
```

Jak widać, deklaracja jest zawsze poprzedzona słowem kluczowym `var`. Na przykład:

```
var currentScore : int = 0;
```

lub:

```
var currentVelocity : float = 5.86;
```

Jak zapewne zauważyłeś, typ `float`, inaczej niż w języku C#, nie wymaga podania litery `f` występującej za wartością. Podczas analizy kolejnych skryptów napisanych w dwóch różnych językach będziesz także mógł stwierdzić, że język C# ma często bardziej rygorystyczne reguły, określające sposób tworzenia programów, szczególnie w przypadku niejawnie deklarowanych typów danych, które mają być używane.

Komentarze

Zarówno w języku C#, jak i w JavaScript możesz tworzyć komentarze za pomocą następujących konstrukcji:

```
// dwa ukośniki oznaczają pojedynczy wiersz komentarza
```

lub:

```
/* ukośnik i gwiazdka rozpoczynają komentarz zawierający się w wielu wierszach.
Jego zakończeniem są gwiazdka i ukośnik */
```

Podczas czytania tej książki komentarze mogą pomóc Ci przypomnieć sobie zasadę działania tworzonych przez Ciebie fragmentów kodu. Pamiętaj, że komentarze nie wykonują się, dlatego możesz w nich umieszczać dowolne teksty, włączając w to również wiersze kodu. Dopóki będą się one zawierać w komentarzu, nie zostaną potraktowane jako działający kod.

Atakowanie ściany

A teraz zastosujmy zdobytą wiedzę w praktyce i zamieńmy istniejącą scenę w prototyp interakcyjnej fabuły. W panelu *Project* zmień nazwę nowo utworzonego skryptu poprzez jego wybranie, naciśnięcie klawisza *F2* (PC) lub *Return* (Mac), a następnie wprowadzenie tekstu skryptu *Shooter*.

Jeśli wykorzystujesz język C#, pamiętaj o upewnieniu się, że Twoja deklaracja klasy znajdująca się wewnątrz skryptu odpowiada jego nazwie:

```
public class Shooter : MonoBehaviour {
```

Jak wcześniej wspomniano, użytkownicy wykorzystujący język JavaScript nie będą musieli stosować się do powyższej reguły. By zastosować w praktyce wiedzę o korzystaniu ze skryptów w środowisku Unity, napiszemy skrypt sterujący kamerą i pozwalający na strzelanie pociskami w kierunku postawionej przez nas ściany.

Na początku utworzymy trzy zmienne:

- **`bullet`**: ta zmienna jest typu `Rigidbody`, ponieważ przechowuje odwołanie do obiektu o właściwościach fizycznych, który zostanie przez nas utworzony;
- **`power`**: jest to zmienna o typie zmiennoprzecinkowym, którą wykorzystamy w celu ustalenia siły strzału;
- **`moveSpeed`**: kolejna zmienna o typie zmiennoprzecinkowym, której użyjemy, by zdefiniować prędkość poruszania kamerą za pomocą klawiszy strzałek.

Powyższe zmienne muszą być **składowymi z dostępem publicznym**, co pozwoli je wyświetlić w panelu *Inspector* w postaci modyfikowanych ustawień. Za chwilę zobaczysz, jak to działa!

Deklarowanie zmiennych publicznych

Ważne jest, by zrozumieć działanie zmiennych o dostępie publicznym, ponieważ pozwala on na odwoływanie się do nich z innych skryptów. Ma to znaczenie podczas projektowania gier, gdyż umożliwia definiowanie prostszej komunikacji międzyobiektowej. Zmienne publiczne są także przydatne z tego względu, że występują w postaci ustawień, które można wizualnie modyfikować w panelu *Inspector*, gdy tylko skrypt zostanie przypisany do obiektu. Zmienne prywatne zachowują się przeciwnie — zostały zaprojektowane, by być dostępne tylko z poziomu skryptu, klasy lub funkcji, w których są zdefiniowane. Nie pojawiają się również w postaci ustawień w panelu *Inspector*.

Język C#:

Zanim rozpoczniemy działania, usuniemy funkcję `Start()` ze skryptu, ponieważ nie będziemy jej używać. W tym celu po prostu skasuj tekst `void Start { }`. Aby stworzyć wymagane zmienne, umieść poniższy fragment kodu w swoim skrypcie poniżej wiersza otwierającego klasę:

```
using UnityEngine;
using System.Collections;
public class Shooter : MonoBehaviour {
   public Rigidbody bullet;
   public float power = 1500f;
   public float moveSpeed = 2f;
   void Update () {
   }
}
```

Zwróć uwagę na to, że w powyższym przykładzie usunęliśmy domyślne komentarze, by zaoszczędzić miejsce.

Język JavaScript:

Aby stworzyć zmienne o dostępie publicznym w języku JavaScript, musisz się upewnić, że zostaną one zadeklarowane poza istniejącymi funkcjami. Należy je po prostu umieścić na samej górze skryptu. Zadeklaruj więc trzy wymagane zmienne, wstawiając poniższy kod na początku skryptu *Shooter*:

```
var bullet : Rigidbody;
var power : float = 1500;
var moveSpeed : float = 5;
   function Update () {
   }
```

Przypisywanie skryptów do obiektów

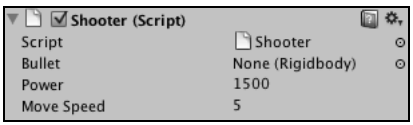
Aby skrypt mógł zostać użyty w grze, musi być przypisany jako komponent do jednego z obiektów gry znajdujących się w istniejącej scenie.

Zapisz skrypt, wybierając z głównego menu edytora opcję *File/Save* (plik/zapisz), a następnie wróć do środowiska Unity. Oferuje ono kilka sposobów przypisywania skryptu do obiektu:

1. Przeciągnij go z panelu *Project* i upuść na nazwie obiektu w panelu *Hierarchy*.
2. Przeciągnij go z panelu *Project* i upuść na wizualną reprezentację obiektu w panelu *Scene*.
3. Wybierz obiekt, dla którego chcesz przypisać skrypt. Następnie przeciągnij i upuść skrypt w niezajętym obszarze w widoku *Inspector* tego obiektu.
4. Wybierz obiekt, do którego chcesz przypisać skrypt. Wybierz opcję menu głównego *Component/Scripts* (komponent/skrypty), a następnie nazwę swojego skryptu.

Najczęściej jest stosowana pierwsza metoda. W naszym przypadku będzie to rzeczywiście najlepszy wybór, ponieważ przeciąganie skryptu do kamery w widoku *Scene* mogłoby być uciążliwe z tego powodu, że nie zawiera ona wyraźnego obszaru, na którym można by go upuścić.

Przeciągnij więc nowy skrypt *Shooter* z panelu *Project* i upuść go na nazwie *Main Camera*, znajdującej się w panelu *Hierarchy*, co spowoduje jego przypisanie. Będziesz mógł zauważyć, że skrypt zostanie wyświetlony jako nowy składnik poniżej istniejącego komponentu `Audio Listener`. W panelu *Inspector* stwierdzisz także istnienie trzech zmiennych publicznych: `Bullet`, `Power` i `Move Speed`:



Jak możesz zauważyć, środowisko Unity zmodyfikowało nazwy zmiennych, rozpoczynając je od wielkiej litery. Dodatkowo, w przypadku zmiennej `moveSpeed`, wielka litera wewnątrz jej nazwy została potraktowana jako początek nowego słowa. Dzięki temu uzyskano dwa wyrazy oddzielone spacją, które są widoczne w panelu *Inspector* jako jedna zmienna publiczna.

Można również stwierdzić, że zmienna `Bullet` nie została jeszcze zainicjalizowana, lecz oczekuje się, iż obiekt, który zostanie do niej przypisany, będzie używał komponentu bryły sztywnej (`Rigidbody`). Często zwany jest on po prostu obiektem typu `Rigidbody`. Pomimo tego, że w środowisku Unity wszystkie elementy w scenie mogą być traktowane jako obiekty gry, podczas definiowania w skrypcie obiektu o typie `Rigidbody` będziemy mogli się odwoływać jedynie do właściwości i funkcji klasy o tej właśnie nazwie. Nie stanowi to jednak problemu — po prostu sprawia, że skrypt staje się bardziej efektywny niż w przypadku, gdybyśmy odwoływali się do całej klasy `GameObject`. Aby dowiedzieć się więcej na ten temat, zapoznaj się z dokumentacją związaną z oprogramowaniem skryptowym dotyczącym wspomnianych klas:

- `GameObject`: *http://unity3d.com/support/documentation/ScriptReference/GameObject.html*,
- `RigidBody`: *http://unity3d.com/support/documentation/ScriptReference/Rigidbody.html*.

Uwaga

Pamiętaj, że podczas modyfikowania zmiennych publicznych, dostępnych w panelu *Inspector*, dowolna ich zmiana zamiast zastąpić wartości, które zostały zapisane w skrypcie, po prostu je przesłoni.

Kontynuujmy prace związane z tworzeniem skryptu i uzupełnijmy go o pewną interaktywność. Wróć więc z powrotem do edytora, do którego został on wczytany.

Poruszanie kamerą

Teraz wykorzystamy zmienną `moveSpeed` oraz odczytamy stan klawiatury, aby poruszać kamerą i faktycznie stworzyć prosty mechanizm celowania dla strzału. Będzie on polegać na ustawianiu kamery w kierunku miejsca, w które będziemy strzelali.

Ponieważ chcielibyśmy używać klawiszy strzałek na klawiaturze, musimy się najpierw dowiedzieć, w jaki sposób w kodzie można się do nich odwoływać. Środowisko Unity wykorzystuje wiele sygnałów wejściowych, które można obserwować i modyfikować przy użyciu **menedżera wejść** (*Input Manager*) — patrz menu *Edit/Project Settings/Input* (edycja/ustawienia projektu/wejście).

Jak widać na poniższym rysunku, istnieją dwa domyślne parametry typu *Input* (wejście): *Horizontal* (poziomy) i *Vertical* (pionowy). Wykorzystują one zasadę działania osi współrzędnych, dzięki czemu naciskanie przycisku *Positive Button* (przycisk dodatni) generuje wartość równą `1`, natomiast naciskanie przycisku *Negative Button* (przycisk ujemny) tworzy wartość `-1`. Zwolnienie dowolnego przycisku oznacza, że wartość typu *Input* ponownie staje się równa `0`, podobnie jak miałoby to miejsce podczas używania sprężystego joysticka analogowego, znajdującego się w urządzeniu typu gamepad.

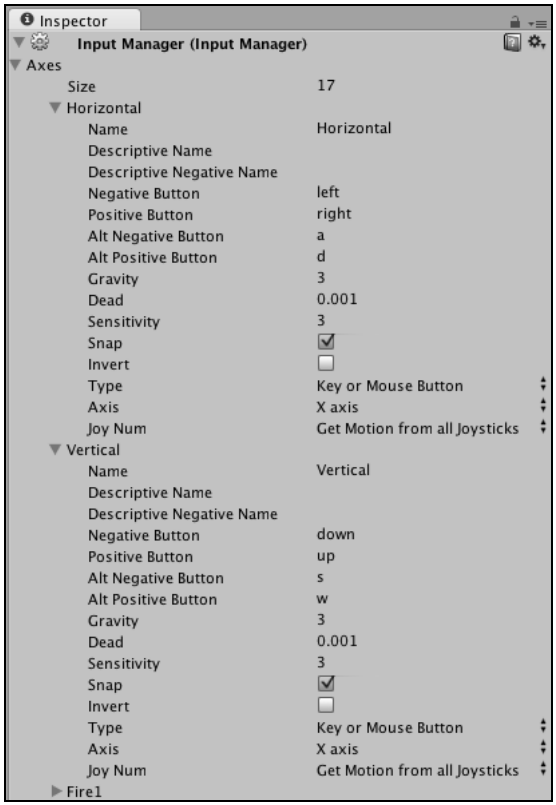
Ponieważ typ `Input` jest również nazwą klasy, a wszystkie nazwane elementy w menedżerze wejść są osiami lub przyciskami, w skrypcie możemy użyć następującego zapisu:

```
Input.GetAxis("Horizontal");
```

Powyższy wiersz pozwala na odczytanie bieżącej wartości przycisków poziomych, zawierającej się w zakresie od `-1` do `1` w zależności od tego, co jest naciskane przez użytkownika. Zastosujmy go w naszym skrypcie, używając zmiennych lokalnych do reprezentowania osi.

Dzięki temu możemy później zmodyfikować wartość zmiennej, używając operacji mnożenia, dzięki czemu uzyskamy maksymalną wartość większą od `1`. Pozwoli nam to na przemieszczanie kamery z prędkością większą niż jedna jednostka na iterację.

Dostępu do tej zmiennej nie musimy mieć w panelu *Inspector*, ponieważ samo środowisko Unity będzie przypisywać jej odpowiednie wartości w zależności od stanu sygnałów wejściowych. Wynika stąd, że powinna ona zostać zdefiniowana jako **zmienna lokalna**.



Zmienne lokalne, prywatne i publiczne

Zanim będziemy kontynuować prace związane z tworzeniem skryptu, zapoznajmy się ze zmiennymi lokalnymi, prywatnymi i publicznymi:

- **Zmienne lokalne**. Są to zmienne tworzone wewnątrz funkcji. Nie będą prezentowane w panelu *Inspector* i są dostępne wyłącznie dla funkcji, w której się znajdują.
- **Zmienne prywatne**. Są tworzone na zewnątrz funkcji i dlatego dostęp do nich mają wszystkie funkcje danej klasy. Nie są one jednak widoczne w panelu *Inspector*.
- **Zmienne publiczne**. Są tworzone na zewnątrz funkcji, dostęp do nich mają wszystkie funkcje danej klasy, a także inne skrypty. Zmienne publiczne są widoczne w panelu *Inspector* i mogą być tam modyfikowane.

Zmienne lokalne i odczytywanie wejść

Tworzenie zmiennych lokalnych w językach C# i JavaScript zostało przedstawione w poniższych fragmentach kodu:

Język C#:

```
void Update () {
   float h = Input.GetAxis("Horizontal") * Time.deltaTime * moveSpeed;
   float v = Input.GetAxis("Vertical") * Time.deltaTime * moveSpeed;
```

Język JavaScript:

```
function Update () {
   var h : float = Input.GetAxis("Horizontal") * Time.deltaTime * moveSpeed;
   var v : float = Input.GetAxis("Vertical") * Time.deltaTime * moveSpeed;
```

Zadeklarowane zmienne: `h` dla osi poziomej i `v` dla osi pionowej mogłyby zostać dowolnie nazwane, jednakże używanie pojedynczych liter jest po prostu szybsze. Ogólnie mówiąc, zazwyczaj powinno się tworzyć standardowe nazwy, ponieważ niektóre pojedyncze litery nie mogą być używane. Na przykład nazwy zmiennych `x`, `y` i `z` są używane do przechowywania wartości współrzędnych i dlatego też zarezerwowano je wyłącznie do tych zastosowań.

Ponieważ wartości osi mogą zawierać się w przedziale od `-1` do `1`, zmienne są liczbami dziesiętnymi i musimy je deklarować z użyciem typu zmiennoprzecinkowego. Mnoży się je następnie za pomocą symbolu gwiazdki `*` przez parametr `Time.deltaTime`. Oznacza to, że wyznaczana wartość jest dzielona przez liczbę klatek na sekundę (parametr `deltaTime` jest czasem tworzenia kolejnych klatek lub czasem liczonym od ostatniego wywołania funkcji `Update()`). Wynika stąd, iż wartość osiąga pewną spójną wielkość na sekundę, niezależnie od szybkości klatek.

W kolejnym kroku zwiększamy wartość wynikową, mnożąc ją przez zmienną publiczną, którą utworzyliśmy wcześniej, czyli `moveSpeed`. Oznacza to, że mimo iż wartości `h` i `v` są zmiennymi lokalnymi, możemy wciąż na nie wpływać poprzez modyfikowanie zmiennej publicznej `moveSpeed`, dostępnej w panelu *Inspector*, ponieważ jest ona składnikiem używanego przez nas równania. Jest to często spotykane rozwiązanie podczas tworzenia skryptów, gdyż wykorzystuje ono zaletę użycia publicznie dostępnych ustawień połączonych z określonymi wartościami generowanymi przez funkcję.

Zrozumienie zasady działania polecenia Translate

Aby rzeczywiście użyć stworzonych zmiennych w celu przemieszczenia obiektu, musimy zastosować polecenie `Translate`. Podczas implementowania dowolnego fragmentu kodu musisz być pewien, że rozumiesz jego działanie.

`Translate` jest poleceniem pochodzącym z klasy `Transform`: *http://unity3d.com/support/documentation/ScriptReference/Transform.html*.

Jest to klasa informacji przechowująca właściwości położenia, obrotu i skali danego obiektu. Zawiera ona również funkcje, które mogą być używane w celu jego przemieszczania lub obracania.

Oczekiwane użycie funkcji `Translate` wygląda następująco:

```
Transform.Translate(Vector3);
```

Istnienie parametru `Vector3` oznacza, że funkcja `Translate` wymaga jego użycia w formie głównego argumentu. Dane reprezentowane przez `Vector3` są po prostu informacjami zawierającymi wartości współrzędnych X, Y i Z. W naszym konkretnym przypadku oznaczają koordynaty przesunięcia obiektu.

Implementacja funkcji Translate

Zaimplementujmy polecenie `Translate`, używając utworzonych przez nas wartości wejściowych `h` i `v` i umieszczając je w parametrze `Vector3`.

Języki C# i JavaScript:

Umieść poniższy wiersz kodu wewnątrz funkcji `Update()` w Twoim skrypcie, a dokładniej pomiędzy jej otwierającym { i zamykającym } nawiasem klamrowym. Zwróć uwagę na to, że zapis wygląda identycznie w obu językach:

```
transform.Translate(h, v, 0);
```

Możemy tu skorzystać ze słowa `transform`, ponieważ wiemy, że dowolny obiekt, któremu udostępnimy skrypt, będzie wykorzystywać komponent `Transform`. Przyłączone komponenty danego obiektu mogą być adresowane poprzez użycie ich nazw zapisanych małymi literami, natomiast dostęp do komponentów innych obiektów wymaga zastosowania polecenia `GetComponent` i odpowiedniej nazwy zaczynającej się od wielkiej litery, na przykład:

```
GameObject.Find("NazwaInnegoObiektu").GetComponent<Transform>.Translate(h,v,0);
```

W naszym przypadku nie musimy wykorzystywać powyższej formy zapisu. Dostęp do komponentów przyłączonych do innych obiektów został dokładniej opisany w podrozdziale rozdziału 4., zatytułowanym „Komunikacja międzyskryptowa oraz składnia z kropką”.

W skrypcie użyjemy bieżącej wartości zmiennej `h` dla osi X oraz `v` dla osi Y, natomiast osi Z przekażemy po prostu wartość `0`, ponieważ nie chcemy się przesuwać do przodu ani do tyłu.

Zapisz teraz swój skrypt, używając opcji menu głównego *File/Save* (plik/zapisz), a następnie wróć do środowiska Unity. Zapamiętaj tworzoną scenę za pomocą opcji *File/Save Scene As* (plik/zapisz scenę jako) i nazwij ją *Prototype*.

Środowisko Unity zaproponuje zapisanie jej w domyślnym folderze *Assets* (zasoby). Powinieneś zawsze się upewniać, że zapamiętujesz sceny w tym folderze, ponieważ w przeciwnym razie

nie będziesz miał do nich dostępu z panelu *Project* (projekt). By zapewnić idealny porządek, możesz także utworzyć podkatalog wewnątrz folderu *Assets*, w którym będziesz przechowywać swoje sceny. Takie działanie nie jest jednak wymagane, lecz ogólnie uważa się je za dobry wzorzec postępowania.

Testujemy bieżącą wersję gry

W środowisku Unity możesz testować grę w każdym momencie, przy założeniu, że skrypty nie zawierają żadnych błędów. Jeśli w skryptach wykryto jakieś problemy, będziesz musiał je rozwiązać przed użyciem trybu grania.

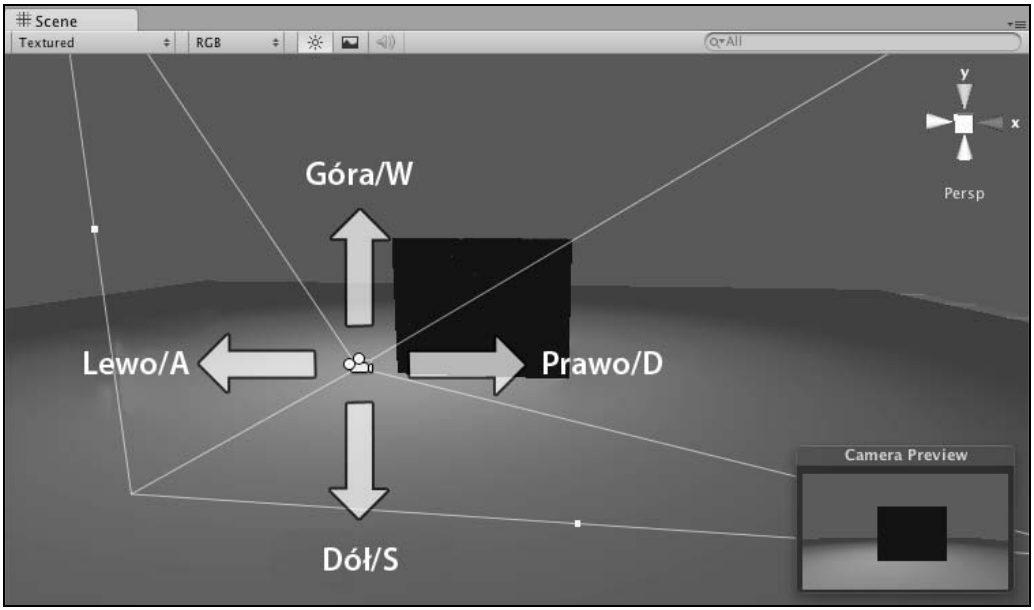
Gdy błędy zostaną naprawione, pasek *Console* (konsola), znajdujący się na dolnej krawędzi interfejsu Unity, nie będzie zawierać żadnych informacji. Pasek *Console* wyświetla najnowsze wpisy pojawiające się w konsoli Unity. Możesz ją wyświetlić, wybierając opcję menu *Window/Console* (okno/konsola) (skrót klawiszowy to *Ctrl+Shift+C* dla PC lub *Command+Shift+C* dla komputerów Mac). Każdy błąd zostanie wyświetlony na czerwono. Dwukrotne jego kliknięcie umożliwi wyświetlenie odpowiedniego fragmentu skryptu, który spowodował pojawienie się problemu. Większość błędów polega na pominięciu jakiegoś znaku lub błędnym zapisie, dlatego też zawsze dokładnie sprawdzaj, co napisałeś.

Jeśli Twoja gra jest wolna od błędów, kliknij przycisk *Play* (granie), znajdujący się w górnej części okna Unity, aby przejść do trybu grania. Będziesz mógł poruszać obiektem `Main Camera` za pomocą klawiszy strzałek góra, dół, lewo i prawo lub ich odpowiedników: *W*, *A*, *S* i *D*, tak jak zaprezentowano na poniższym rysunku.

Gdy przetestowałeś już grę i upewniłeś się, że działa poprawnie, ponownie naciśnij przycisk *Play*, aby opuścić tryb grania.

> **Uwaga**
>
> Opuszczenie trybu grania przed kontynuowaniem pracy jest ważne, ponieważ podczas jego wyłączania wszystkie zmodyfikowane ustawienia komponentów i obiektów używanych w bieżącej scenie nie zostaną zapamiętane. Pozostawienie środowiska Unity w trybie grania i wprowadzanie dalszych modyfikacji będzie oznaczało, że po prostu utracisz wyniki swojej pracy.

Ukończmy prototyp mechaniki gry poprzez uzupełnienie go o możliwość wystrzeliwania pocisków w kierunku ściany, aby ją zburzyć.



Tworzenie pocisku

Aby wystrzelić pocisk w kierunku ściany, musimy w ramach bieżącej sceny najpierw go utworzyć, a następnie zapamiętać w postaci **prefabrykatu**.

> **Uwaga**
>
> Prefabrykat jest obiektem gry, przechowywanym w projekcie w postaci zasobu. Zasób ten może być konkretyzowany, czyli tworzony w trakcie działania gry, a następnie przetwarzany poprzez użycie kodu.

Tworzenie prefabrykatu pocisku

Rozpocznij od kliknięcia przycisku *Create* (stwórz), znajdującego się na górnej krawędzi panelu *Hierarchy* (hierarchia). Następnie z podręcznego menu wybierz opcję *Sphere* (kula). Jak wcześniej wspomniano, dostęp do tworzenia kształtów podstawowych jest również możliwy za pomocą opcji menu głównego *GameObject/Create/Other* (obiekt gry/utwórz/inny).

Teraz upewnij się, że utworzona przez Ciebie kula została zaznaczona w panelu *Hierarchy*, a później przemieść kursor myszy na widok *Scene* (scena) i naciśnij klawisz *F*, by skoncentrować się na wybranym obiekcie.

Uwaga

Jeśli Twoja kula ma takie samo położenie, jak któryś z innych obiektów, możesz w prosty sposób przejść do narzędzia translacji (*W*), a następnie przeciągnąć uchwyt odpowiedniej osi, aby odsunąć ją od zasłaniającego elementu. Po tej operacji ponownie skoncentruj widok na kuli poprzez naciśnięcie klawisza *F*.

Spoglądając na panel *Inspector* (inspektor), możesz zauważyć, że podczas dodawania nowych obiektów o kształtach podstawowych środowisko Unity oprócz istniejącego komponentu `Transform` automatycznie przypisuje im trzy nowe komponenty. Są to:

1. **`Mesh Filter`** (filtr siatki). Służy do modyfikowania kształtu.
2. **`Renderer`** (renderer). Służy do modyfikowania wyglądu.
3. **`Collider`** (zderzacz). Służy do zarządzania interakcjami (zwanymi kolizjami) z innymi obiektami.

Tworzenie i przypisywanie materiału

Wygląd pocisku da się zmodyfikować poprzez stworzenie materiału, który możemy przekazać do renderera. Gdy chcesz zmienić wygląd obiektu, powinieneś najczęściej odszukać odpowiednie ustawienia związane z jakimś komponentem typu `Renderer`. W przypadku obiektów 3D będzie to `Mesh Renderer` (renderer siatki), a dla systemów cząstek będzie to `Particle Renderer` (renderer cząstek) itd.

Aby zachować porządek, utworzymy nowy podkatalog wewnątrz folderu *Assets*. Będziemy w nim przechowywać wszystkie materiały używane w bieżącym projekcie. W panelu *Project* kliknij przycisk *Create*, a następnie z podręcznego menu wybierz opcję *Folder* (katalog). Zmień jego nazwę na *Materials* poprzez naciśnięcie klawisza *F2* (PC) lub *Return* (Mac). Teraz umieść w nim materiał czerwonej cegły, który został wcześniej przez Ciebie stworzony.

Uwaga

Aby w panelu *Project* utworzyć nowy zasób w istniejącym folderze, po prostu go wybierz, a następnie rozpocznij operację tworzenia poprzez kliknięcie przycisku *Create*.

Teraz utworzymy potrzebny materiał i zastosujemy go w naszym obiekcie:

1. Upewnij się, że folder *Materials* jest zaznaczony, a następnie kliknij przycisk *Create*, znajdujący się w panelu *Project*. Z wyświetlonego menu podręcznego wybierz opcję *Material* (materiał). Spowoduje to utworzenie zasobu *New Material*, którego nazwę powinieneś zmienić na *bulletColor* lub podobną, przypominającą Ci, że powinien on zostać przypisany do pocisku.
2. Mając zaznaczony nowo utworzony materiał, kliknij obszar koloru, aby otworzyć okno *Color Picker*. Później wybierz jakiś odcień koloru niebieskiego i zamknij to okno.

3. Po wybraniu koloru przeciągnij materiał *bulletColor* z panelu *Project* i upuść go na nazwie kuli w panelu *Hierarchy*. Spowoduje to przypisanie materiału do interesującego Cię obiektu.

> **Uwaga**
>
> Jeśli chcesz sprawdzić, jak będzie wyglądać materiał przypisany do trójwymiarowego obiektu w środowisku Unity, możesz go przeciągnąć do widoku *Scene*, a następnie umieścić kursor myszy nad określoną siatką. System Unity zaprezentuje podgląd obiektu w wybranym kolorze. Aby unieważnić wybór, możesz przesunąć kursor poza obszar widoku lub nacisnąć klawisz *Esc*. By zastosować materiał, zwalniasz po prostu przycisk myszy.

Dodawanie parametrów fizycznych za pomocą komponentu Rigidbody

Poprzez dodanie komponentu `Rigidbody` musimy zapewnić to, że silnik fizyczny będzie mógł sterować pociskiem *Sphere*. Wybierz obiekt *Sphere* z panelu *Hierarchy*, a następnie opcję *Component/Physics/Rigidbody* (komponent/parametry fizyczne/bryła sztywna) z menu głównego.

Komponent bryły sztywnej został dodany, a jego parametry mogą być modyfikowane w panelu *Inspector*. Na potrzeby obecnego prototypu nie musimy jednak niczego zmieniać.

Przechowywanie obiektów jako prefabrykatów

Ponieważ powinniśmy wystrzeliwać pocisk w chwili, gdy gracz naciśnie jakiś klawisz, nie chcemy, aby znajdował się on cały czas w scenie. Zamiast tego powinien być przechowywany i konkretyzować się dopiero podczas naciskania klawisza. Z tego powodu będziemy zapisywać nasz obiekt w postaci prefabrykatu, a następnie używać skryptu w celu jego konkretyzacji (to znaczy tworzenia jego instancji) w momencie naciskania klawisza.

> **Uwaga**
>
> Prefabrykaty w środowisku Unity pozwalają na przechowywanie obiektów gry, które zostały zdefiniowane w określony sposób. Na przykład możesz odpowiednio skonfigurować obiekt wrogiego żołnierza, zawierający pewne skrypty i właściwości, które definiują jego zachowanie. Następnie możesz zapisać ten obiekt w postaci prefabrykatu i w razie konieczności konkretyzować go. Podobnie mógłbyś używać innego żołnierza, zachowującego się odmiennie, który byłby kolejnym prefabrykatem. Mógłbyś także utworzyć instancję pierwszego obiektu, a później zmodyfikować ustawienia jego komponentów, aby po konkretyzacji poruszał się wolniej lub szybciej. System prefabrykatów udostępnia szeroki zakres swobody w tym względzie.

Kliknij przycisk *Create* (stwórz), znajdujący się na górze panelu *Project* (projekt), a następnie wybierz opcję *Folder* (folder). Zmień nazwę nowo utworzonego katalogu na *Prefabs*. W kolejnym kroku przeciągnij obiekt *Sphere* z panelu *Hierarchy* (hierarchia) i upuść go na folder *Prefabs* w panelu *Project*. Przeciągnięcie obiektu gry w dowolny obszar panelu *Project* spowoduje, że zostanie on zapamiętany jako prefabrykat. Folder *Prefabs* został przez nas utworzony jedynie w celu zapewnienia porządku i stosowania się do dobrych wzorców postępowania. Zmień nazwę nowo utworzonego prefabrykatu na *Projectile*.

Możesz już usunąć oryginalny obiekt *Sphere* z panelu *Hierarchy*: wybierz go, a następnie naciśnij klawisz *Delete* (PC) lub *Command+Backspace* (Mac). Alternatywą dla tego rozwiązania jest kliknięcie prawym klawiszem myszy obiektu w panelu *Hierarchy* i wybranie opcji *Delete* (usuń) z menu podręcznego.

Wystrzelenie pocisku

Wróćmy do skryptu *Shooter*, który właśnie tworzymy. W tym celu kliknij dwukrotnie jego ikonę, znajdującą się w panelu *Project*, lub wybierz go, a następnie kliknij przycisk *Open* (otwórz), wyświetlany na górze okna *Inspector* (inspektor).

Użyjemy wcześniej zadeklarowanej zmiennej `bullet`, będącej odwołaniem do określonego obiektu, który zamierzamy skonkretyzować. Gdy tylko obiekt zostanie utworzony z prefabrykatu, zastosujemy odpowiednie działanie, aby wystrzelić go w kierunku ściany znajdującej się w scenie.

W funkcji `Update()` po wierszu `transform.Translate(h, v, 0)` dodaj następujący kod, niezależnie od tego, jakiego języka programowania używasz:

```
if(Input.GetButtonUp("Fire1")){
}
```

Polecenie `if` sprawdza, czy klawisz przypisany do wejściowego przycisku `Fire1` został zwolniony. Domyślnie jest on odwzorowany na lewy klawisz *Ctrl* lub lewy przycisk myszy. Możesz go jednak przypisać do innego klawisza, wprowadzając odpowiednią modyfikację w menedżerze wejść, co można zrobić przez wybranie opcji menu głównego *Edit/Project Settings/Input* (edycja/ustawienia projektu/wejścia).

Użycie funkcji Instantiate() do konkretyzowania obiektów

Wewnątrz polecenia `if` (to znaczy pomiędzy otwierającym i zamykającym nawiasem klamrowym) umieść następujący wiersz kodu:

Język C#:

```
Rigidbody instance = Instantiate(bullet, transform.position,
↪transform.rotation) as Rigidbody;
```

Język JavaScript:

```
var instance: Rigidbody = Instantiate(bullet, transform.position,
↪transform.rotation);
```

Jak widać, stworzyliśmy nową zmienną zwaną `instance`. Przechowujemy w niej referencję do funkcji tworzącej nowy obiekt o typie `Rigidbody`.

Polecenie `Instantiate` wymaga podania trzech parametrów:

```
Instantiate(co należy stworzyć, gdzie należy stworzyć, wielkość obrotu);
```

W naszym przypadku chcemy, aby została utworzona instancja obiektu lub prefabrykatu, który został przypisany do zmiennej publicznej `bullet`. Położenie i obrót instancji pobierzemy z komponentu `transform`, używanego w obiekcie, do którego został przydzielony skrypt, czyli `Main Camera`. Dlatego też w skryptach będziesz mógł często zauważyć zapis `transform.position`, odnoszący się do ustawień związanych z komponentem `transform` i dotyczący położenia obiektu, do którego został przyłączony dany skrypt.

Zauważ, że w języku C# musisz umieścić słowo `Rigidbody` za wywołaniem funkcji `Instantiate`, aby jawnie wskazać typ danych.

Przyłożenie wektora siły do bryły sztywnej

Po utworzeniu obiektu musimy go natychmiast wystrzelić, używając w tym celu polecenia `AddForce()`. Działa ono tak:

```
Rigidbody.AddForce(kierunek i wielkość wektora siły podane jako typ Vector3);
```

Zanim więc przyłożymy wektor siły, musimy stworzyć referencję do kierunku, w którym chcemy strzelać. Kamera jest zwrócona w stronę ściany z cegieł, więc sensowne jest, by oddawać strzały w tym właśnie kierunku. Poniżej wiersza zawierającego funkcję `Instantiate()`, a jednocześnie wciąż w obrębie polecenia `if`, umieść następujący kod:

Język C#:

```
Vector3 fwd = transform.TransformDirection(Vector3.forward);
```

Język JavaScript:

```
var fwd: Vector3 = transform.TransformDirection(Vector3.forward);
```

Utworzyliśmy zmienną `fwd` o typie `Vector3` i przypisaliśmy jej kierunek do przodu dla komponentu `transform` w obiekcie, do którego jest przyłączony nasz skrypt.

Polecenie `TransformDirection` może zostać użyte w celu przekształcenia lokalnego kierunku — w naszym przypadku kierunku wskazywania kamery — na kierunek świata. Obiekty oraz świat mają własne systemy współrzędnych, więc może się zdarzyć, że kierunek wskazywania do przodu dla jakiegoś obiektu nie musi koniecznie odpowiadać właściwemu kierunkowi świata. Wynika stąd, iż w takich przypadkach konwersja jest niezbędna. Konstrukcja `Vector3.forward`, użyta w powyższym kontekście, jest zwykłym uproszczeniem zapisu `Vector3(0, 0, 1)`. Występuje w nim wartość jednostkowa długości dla osi Z.

Wreszcie będziemy mogli wystrzelić pocisk przez odwołanie się do naszej zmiennej, która reprezentuje nowo utworzony obiekt `instance`, a następnie użycie polecenia `AddForce()` w celu przyłożenia siły w kierunku zdefiniowanym w zmiennej `fwd`, mnożąc ją przez wcześniej utworzoną zmienną publiczną `power`. Dodaj poniższy wiersz kodu bezpośrednio pod tym, który poprzednio umieściłeś w skrypcie:

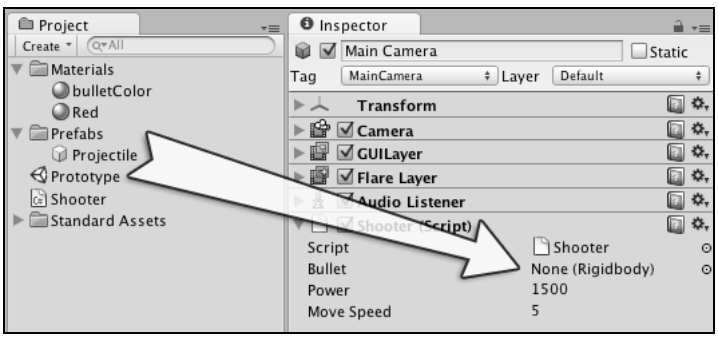
Języki C# i JavaScript:

```
instance.AddForce(fwd * power);
```

Zapamiętaj skrypt i wróć do środowiska Unity.

Zanim przetestujemy działanie ukończonej mechaniki gry, musimy przypisać prefabrykat *Projectile* do zmiennej publicznej `Bullet`. W tym celu wybierz obiekt `Main Camera` w panelu *Hierarchy*, aby wyświetlić skrypt *Shooter* w postaci komponentu w panelu *Inspector*.

Następnie przeciągnij prefabrykat *Projectile* z panelu *Project* i upuść go na zmienną `Bullet` w panelu *Inspector* w miejscu, w którym znajduje się napis *None (Rigidbody)*, jak przedstawiono na poniższym rysunku:



Po wykonaniu tej czynności będziesz mógł zobaczyć nazwę *Projectile* w obszarze zmiennej `Bullet`. Zapisz scenę za pomocą opcji *File/Save Scene* (plik/zapisz scenę), a później przetestuj grę, naciskając przycisk *Play* znajdujący się w górnej części interfejsu.

Teraz będziesz mógł przemieszczać kamerę i strzelać pociskami za pomocą lewego klawisza *Ctrl* na klawiaturze. Jeśli chcesz zmienić siłę wystrzału pocisku, po prostu zmodyfikuj zmienną

publiczną *Power* poprzez wybranie obiektu `Main Camera`, a następnie wprowadzenie innej wartości w miejsce liczby `1500` przypisanej do komponentu `Shooter (Script)`. Pamiętaj, aby zawsze nacisnąć przycisk *Play* w celu zakończenia testowania.

Podsumowanie

Gratulacje! Właśnie stworzyłeś swój pierwszy prototyp w środowisku Unity.

W tym rozdziale poznałeś zasady używania interfejsu Unity, obiektów gry i komponentów, a także podstawy tworzenia skryptów. Miejmy nadzieję, że wiedza ta będzie solidnym fundamentem, na którym zostanie zbudowane dalsze doświadczenie w tworzeniu gier komputerowych w środowisku Unity.

A teraz chwila odpoczynku. Zagraj w stworzony przez siebie prototyp gry, a może stwórz kolejny w oparciu o informacje zaprezentowane w tym rozdziale. Być może jesteś jednak żądny wiedzy — jeśli tak, kontynuuj czytanie książki!

Skoro poznałeś już podstawowe operacje dostępne w środowisku projektowym Unity, zajmijmy się właściwą grą, która będzie omawiana w dalszych rozdziałach tej książki.

W następnym rozdziale rozpoczniemy tworzenie gry zwanej *Survival Island* (wyspa przetrwania). Poznasz, w jaki sposób można będzie wykorzystać narzędzia tworzenia terenu, aby ukształtować tropikalną wyspę, która ma nawet swój wulkan! W dalszej kolejności umieścisz na niej prefabrykat postaci gracza i zwiedzisz nowo utworzony raj tropikalny!

Projektowanie gier w środowisku Unity 3.x

Silniki gier, takie jak Unity, są znanymi, cenionymi, a przede wszystkim potężnymi narzędziami ułatwiającymi tworzenie gier. Środowisko Unity jest jednym z najczęściej używanych oraz najbardziej cenionych pakietów, pozwalających na projektowanie gier komputerowych. Może ono być wykorzystywane przez bardzo różnych użytkowników, poczynając od hobbystów, a kończąc na dużych firmach. Pozwala tworzyć gry oraz interaktywne aplikacje dla przeglądarek internetowych, komputerów stacjonarnych, urządzeń przenośnych czy konsol. Dzięki intuicyjnemu i prostemu w obsłudze zestawowi narzędzi Unity oraz niniejszej książce także i Ty możesz stać się twórcą gier komputerowych.

Jeżeli jesteś projektantem gier lub masz dobry pomysł na grę, który chciałbyś wcielić w życie, z pomocą przyjdzie Ci ta książka. Dzięki niej błyskawicznie przygotujesz prototyp lub nawet całościowe rozwiązanie! W trakcie lektury nauczysz się projektować gry z wykorzystaniem silnika Unity 3, skryptów w języku C# oraz JavaScriptu. Niezwykle istotne jest to, że książka da Ci solidne podstawy w zakresie rozumienia kluczowych koncepcji związanych z projektowaniem gier — od fizyki świata gry aż do systemu cząstek. Ponadto dowiesz się, jak zapewnić najwyższą wydajność Twoim rozwiązaniom, budować interakcje oraz dzielić się z graczami gotowym rozwiązaniem. Żeby zacząć przygodę z tworzeniem gier, nie musisz posiadać żadnej specjalistycznej wiedzy, wystarczy Ci tylko ta książka! Brzmi kusząco? Spróbuj sam!

Tylko krok dzieli Cię od własnej gry komputerowej!

Sięgnij po tę książkę, a następnie:

- zaprojektuj własną grę komputerową
- zrealizuj projekt i podziel się efektami ze znajomymi
- spełnij swoje marzenia oraz zwizualizuj najlepsze pomysły wirtualnych światów
- zapewnij najwyższą wydajność Twojej grze

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-246-3984-7

Cena: 77,00 zł

Informatyka w najlepszym wydaniu