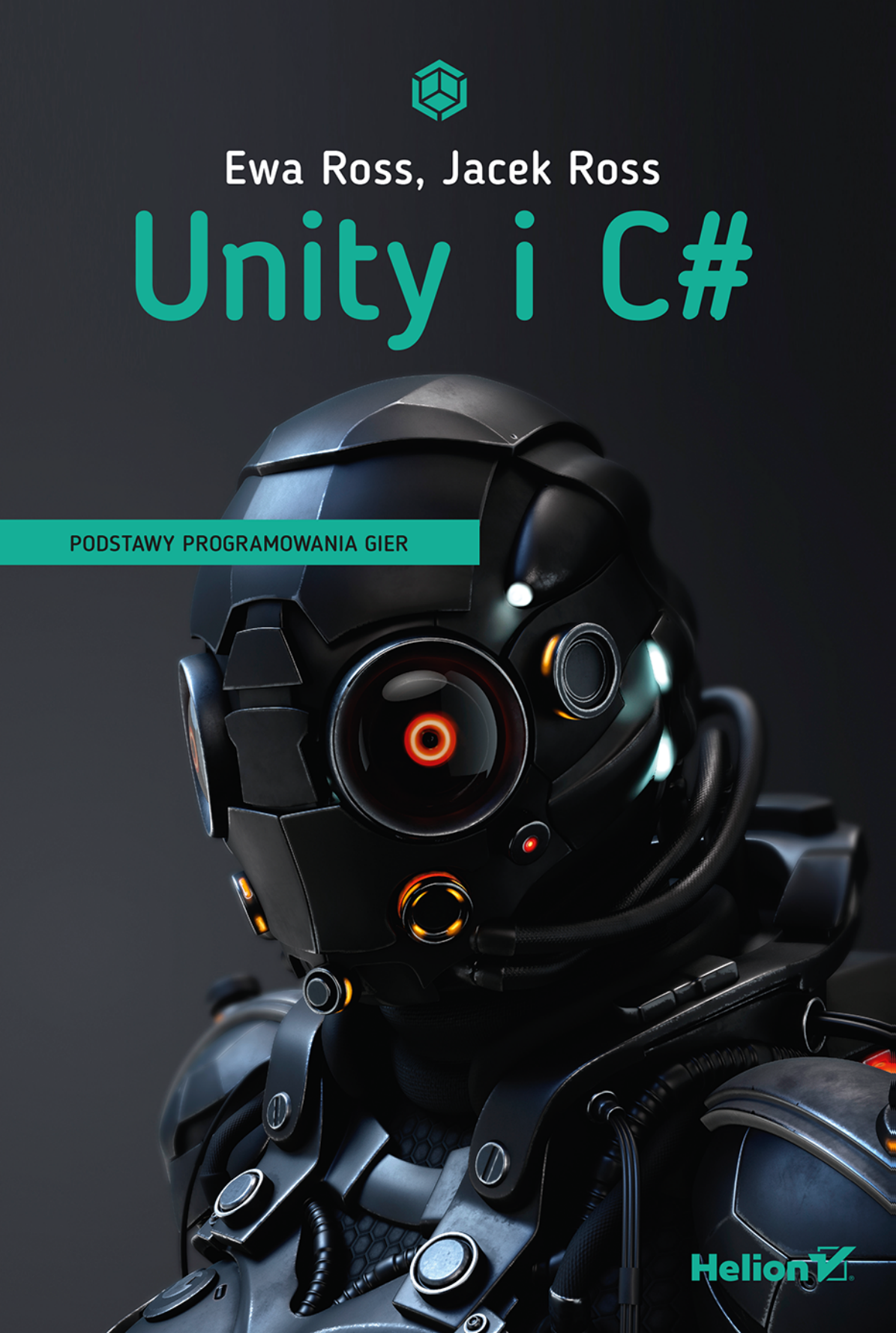
Ewa Ross, Jacek Ross
Unity i C#
PODSTAWY PROGRAMOWANIA GIER
Helion

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki
Grafika na okładce została wykorzystana za zgodą Shutterstock.com

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

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

Kody źródłowe wybranych przykładów dostępne są pod adresem:
https://ftp.helion.pl/przyklady/unityv.zip

ISBN: 978-83-283-9237-3

Printed in Poland.

Spis treści

Lekcja 4.

Instrukcje warunkowe

Instrukcje warunkowe — część I

Podczas omawiania algorytmów widzieliśmy, jak wykonanie kodu programu może rozgałęziać się w zależności od pewnego warunku. Przykładowo, jeśli zmienna `X` ma wartość większą niż 10, dodaj do niej 1, a jeśli mniejszą, dodaj 2. Zapisuje się to tak jak na listingu 4.1.

LISTING 4.1. Przykład instrukcji warunkowej

```
if (X > 10)
    X += 1;
else
    X += 2;
```

W nawiasach po słowie kluczowym `if` powinien znaleźć się **warunek** — zestaw obliczeń, których wynikiem jest `true` (tak) albo `false` (nie). Następnie pojawia się operacja (tutaj: dodania wartości 1 do `X`), która jest wykonywana, jeśli warunek ma wartość `true`. Po słowie kluczowym `else` możemy wpisać operację wykonywaną, jeśli warunek ma wartość `false`. Część po `else` nie jest wymagana.

Zamiast pojedynczej operacji można wpisać wiele operacji, umieszczając je w nawiasach klamrowych jak na listingu 4.2.

LISTING 4.2. Przykład rozbudowanej instrukcji warunkowej

```
if (X > 10)
{
    X += 1;
    Y = 0;
}
else
{
    X += 2;
    Y = -1;
}
```

Jako warunku można użyć również zmiennej typu `bool`.

Operatory, które służą do tworzenia warunków, to:

- **==**, czyli operator równości, np. `A == B` zwraca `true`, tylko jeśli `A` ma taką wartość jak `B`;
- **!=**, czyli operator różnicy, np. `A != B` zwraca `true`, tylko jeśli `A` ma inną wartość niż `B`;
- **>** lub **<**, czyli operatory większości i mniejszości, np. `A > B` zwraca `true`, tylko jeśli `A` jest większe niż `B`;
- **>=** lub **<=**, czyli operatory „większy lub równy" oraz „mniejszy lub równy", np. `A <= B` zwraca `true`, tylko jeśli `A` jest mniejsze lub równe `B`.

Operatory te można bez przeszkód stosować na typach prostych takich jak `int` czy `bool`, ale na typach obiektowych można stosować tylko operatory `==` i `!=`, które sprawdzają, czy dwie zmienne mają przypisany dokładnie ten sam obiekt.

Specjalną wartością dla zmiennych obiektowych jest **`null`**. Oznacza ona, że nie jest przypisany żaden obiekt. Uwaga: próba wywołania funkcji dla zmiennej obiektowej o wartości `null` zawsze skończy się błędem wykonania programu. Dobrą praktyką jest więc sprawdzanie, czy taka wartość jest przypisana do zmiennej, zawsze kiedy nie ma co do tego absolutnej pewności — jak np. na listingu 4.3.

LISTING 4.3. Przykład użycia słowa kluczowego null

```
void Funkcja(JakasKlasa1 JakasZmienna)
{
    if (JakasZmienna != null)
        JakasZmienna.CiekawaFunkcja(7);
}
```

Gdyby do funkcji `Funkcja` w parametrze `JakasZmienna` trafiła wartość `null`, wywołanie metody `CiekawaFunkcja` zwróciłoby błąd. Sprawdzamy jednak za pomocą warunku, czy w zmiennej znajduje się `null`. Jeśli nie, na pewno jest to jakiś obiekt, dla którego możemy wywołać metodę `CiekawaFunkcja`.

Zadania do samodzielnego wykonania

Projekt JakimJestemTypem — część I

Czas: około 30 – 40 minut.

Zadanie

Zadaniem kursanta jest utworzenie prostej gry (można wykorzystać jeden z projektów z lekcji 1.), w której będzie się znajdować kilka brył 3D mających jeden z trzech komponentów:

- komponent z polem typu całkowitego,
- komponent z polem typu zmiennoprzecinkowego,
- komponent z polem typu `bool`.

Obiekt `RigidbodyFPSController` z gry powinien posiadać komponent o nazwie `Gracz`, którego zadaniem będzie zbadanie po zaistnieniu kolizji z jedną z brył, jaki komponent posiada bryła, a następnie pobranie z niego wartości i dodanie do wewnętrznej sumy:

- wartości pola całkowitego,
- wartości pola zmiennoprzecinkowego zrzutowanej na typ całkowity,
- 1, jeżeli wartość pola typu `bool` to `true`.

Po każdym dodaniu `Gracz` powinien wypisać na konsoli aktualną wartość wewnętrznej sumy.

Rozwiązanie

Plik *Gracz.cs*:

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

public class Gracz : MonoBehaviour {
    private float Suma;
    void Start () {
        Suma = 0f;
    }
    private void OnCollisionEnter(Collision collision)
    {
        KomponentZLiczbaCalkowita ObiektZLiczbaCalkowita =
        ↪collision.collider.GetComponent<KomponentZLiczbaCalkowita>();
        KomponentZLiczbaZmiennoprzecinkowa ObiektZLiczbaZmiennoprzec =
        ↪collision.collider.GetComponent<KomponentZLiczbaZmiennoprzecinkowa>();
        KomponentZPolemBool ObiektZPoleBool =
        ↪collision.collider.GetComponent<KomponentZPolemBool>();
```

```
        if(ObiektZLiczbaCalkowita != null)
        {
            int wartosc = ObiektZLiczbaCalkowita.Liczba;
            Debug.Log("Trafiony ma typ z liczbą całkowitą: " + wartosc.ToString());
            Suma = Suma + wartosc;
        } else if (ObiektZLiczbaZmiennoprzec != null)
        {
            int wartosc = (int)ObiektZLiczbaZmiennoprzec.Liczba;
            Debug.Log("Trafiony ma typ z liczbą zmiennoprzecinkową, zmieniamy ją na: " +
            ↪wartosc.ToString());
            Suma = Suma + wartosc;
        }
        else if (ObiektZPoleBool != null)
        {
            int wartosc = 0;
            if(ObiektZPoleBool.PoleBool == true )
            {
                wartosc = 1;
                Debug.Log("Trafiony ma typ z polem bool o wartości true, dodajemy 1 do sumy");
            } else Debug.Log("Trafiony ma typ z polem bool o wartości false");
            Suma = Suma + wartosc;
        }
        Debug.Log("Suma wynosi: " + Suma.ToString());
    }
}
```

Plik *KomponentZLiczbaCalkowita.cs*:

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

public class KomponentZLiczbaCalkowita : MonoBehaviour {
    public int Liczba;
}
```

Plik *KomponentZLiczbaZmiennoprzecinkowa.cs*:

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

public class KomponentZLiczbaZmiennoprzecinkowa : MonoBehaviour {
    public float Liczba;
    // Update is called once per frame
    void Update () {
            int JestemIntem;
            float JestemFloatem = 2.7f;
            JestemIntem = (int)JestemFloatem;
    }
}
```

Plik *KomponentZPolemBool.cs*:

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

```
public class KomponentZPolemBool : MonoBehaviour {
    public bool PoleBool;
}
```

ProjektRosliny — część I

Czas: około 45 – 60 minut.

Zadanie

W lekcji 2. jedno z zadań dodatkowych związane było z projektem o nazwie *ProjektRosliny*. Kursant miał za zadanie utworzenie dokumentacji projektowej do takiej właśnie gry. Nadszedł teraz czas na utworzenie rozwiązania dla pierwszej wersji tej gry.

Celem gry jest podniesienie sadzonek (po jednej) i zaniesienie ich do doniczek. Kliknięcie w doniczkę, gdy gracz niesie sadzonkę, powoduje zasadzenie drzewa. Gra kończy się, gdy gracz zasadzi poprawnie co najmniej 3 drzewa.

W projekcie o nazwie *ProjektRosliny_szablon* znajdziesz puste sceny zawierające zasoby graficzne niezbędne do realizacji gry, w tym m.in. sadzonki w grupie obiektów *Sadzonki*, rośliny dorosłe w grupie obiektów *Rosliny* oraz doniczki w grupie obiektów *Doniczki*.

Podpowiedź

Utwórz komponenty `Doniczka`, `Sadzonka` i `Gracz`. Niech komponent `Gracz` będzie przypisany do obiektu `FPSRigidbodyController` i niech zapamiętuje, czy gracz niesie sadzonkę, za pomocą publicznej zmiennej typu `Sadzonka`. Komponent `Doniczka` może reagować na dotknięcie swojego obiektu na scenie uruchomieniem funkcji `OnCollisionEnter` i wprowadzeniem odpowiednich zmian związanych z obiektami sadzonek i roślin. Powiąż sadzonkę z odpowiadającą jej rośliną, dodając w klasie `Sadzonka` publiczne pole typu `GameObject`. Do pola typu `GameObject` w edytorze Unity możesz przypisać dowolny obiekt, posiadający dowolną liczbę i typ komponentów.

Rozwiązanie

Plik *Gracz.cs*:

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

public class Gracz : MonoBehaviour {
    public Sadzonka NiesionaSadzonka;
    private int LiczbaPosadzonychRoslin;
    // Use this for initialization
    void Start () {
        LiczbaPosadzonychRoslin = 0;
    }
    public void PosadzonoRosline()
    {
```

```
        // przestajemy nieść sadzonkę, właśnie ją zasadziliśmy
        NiesionaSadzonka = null;

        // sprawdzamy, ile roślin zostało już posadzonych
        LiczbaPosadzonychRoslin++;
        if(LiczbaPosadzonychRoslin == 3)
        {   // zwycięstwo!
            Debug.Log("Zwycięstwo!!!");
            Collider ColliderGracza = GetComponent<Collider>();
            ColliderGracza.enabled = false;
        }
    }
}
```

Plik *Doniczka.cs*:

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

public class Doniczka : MonoBehaviour {
    private void OnCollisionEnter(Collision collision)
    {
        Gracz obiektGracz = collision.collider.GetComponent<Gracz>();
        // uwaga! sprawdzamy, czy gracz cokolwiek niesie, aby próba wejścia na doniczkę bez sadzonki nie
        // spowodowała błędu
        if (obiektGracz.NiesionaSadzonka != null)
        {
            Sadzonka doZasadzeniaSadzonka = obiektGracz.NiesionaSadzonka;
            GameObject doZasadzeniaRoslina = doZasadzeniaSadzonka.Roslina;
            // włączamy roślinę
            Transform transformRosliny = doZasadzeniaRoslina.GetComponent<Transform>();
            // GetComponentInChildren<Renderer>() szuka komponentu Renderer w podobiektach tego obiektu;
            // szukamy renderera obiektu o nazwie RoslinaX_Obiekt
            Renderer rendererRosliny = transformRosliny.GetComponentInChildren<Renderer>();
            rendererRosliny.enabled = true;
            // przemieszczamy roślinę, aby znalazła się w doniczce
            Transform transformDoniczki = GetComponent<Transform>();
            Vector3 pozycja = transformRosliny.position;
            pozycja.x = transformDoniczki.position.x;
            pozycja.y = transformDoniczki.position.y;
            pozycja.z = transformDoniczki.position.z;
            transformRosliny.position = pozycja;
            // powiadamiamy obiekt gracza, że została posadzona roślina
            obiektGracz.PosadzonoRosline();
        }
        else
            Debug.Log("Gracz nie ma sadzonki.");
    }
}
```

Plik *Sadzonka.cs*:

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

```
public class Sadzonka : MonoBehaviour {
    public GameObject Roslina;
    // Use this for initialization
    void Start () {
        // sprawdzamy, czy w edytorze na pewno zostało ustawione pole Roslina
        if (Roslina == null)
            Debug.LogError("Brak ustawienia pola Roslina! ");
        // na starcie dorosła roślina ma zniknąć
        Transform transformRosliny = Roslina.GetComponent<Transform>();
        // GetComponentInChildren<Renderer>() szuka komponentu Renderer w podobiektach tego obiektu;
        // szukamy renderera obiektu o nazwie RoslinaX_Obiekt
        Renderer rendererRosliny = transformRosliny.GetComponentInChildren<Renderer>();
        rendererRosliny.enabled = false;
    }
    private void OnCollisionEnter(Collision collision)
    {
        Gracz obiektGracz = collision.collider.GetComponent<Gracz>();
        obiektGracz.NiesionaSadzonka = this;
        // wyłączamy sadzonkę
        Transform transformSadzonki = GetComponent<Transform>();
        Renderer rendererSadzonki = transformSadzonki.GetComponentInChildren<Renderer>();
        Collider colliderSadzonki = transformSadzonki.GetComponentInChildren<Collider>();
        rendererSadzonki.enabled = false;
        colliderSadzonki.enabled = false;
    }
}
```

ProjektRosliny — część II

Czas: około 90 minut.

Zadanie

Grę utworzoną w poprzednim zadaniu, *ProjektRosliny*, rozbuduj do drugiej wersji. W kilku miejscach na ziemi powinny leżeć konewki. Gracz może podnieść konewkę i nieść ją równocześnie z sadzonką. Dotknięcie doniczki, gdy gracz niesie sadzonkę, powinno spowodować posadzenie sadzonki (w doniczce powinna pojawić się mała roślina), a dotknięcie doniczki, gdy jest w niej sadzonka i gracz niesie konewkę, powinno powodować, że wyrośnie drzewo. Warunki zwycięstwa nie zmieniają się, ale gracz może też przegrać — zablokuje się, jeżeli weźmie więcej niż jedną konewkę naraz.

W materiałach dołączonych do książki znajdziesz katalog *WateringCan*, a w nim zasoby graficzne dla konewki (w tym gotowy do użycia na scenie obiekt-prefabrykat o nazwie *WateringCanPrefab*).

Podpowiedź

Rozmiar kodu źródłowego rośnie i niektóre jego fragmenty są coraz bardziej rozbudowane albo wykonywane kilka razy. Przykładowo, z pewnością trzeba będzie kilka razy zmieniać ustawienie, czy sadzonka jest widoczna, czy nie. W takich przypadkach do dobrych praktyk należy przeniesienie fragmentu kodu, który za dane działanie odpowiada,

do osobnej funkcji. Dobrym miejscem jest klasa `Sadzonka`. Takie modyfikacje kodu źródłowego pod wpływem nowych wymagań są typowe dla programowania zwinnego i nazywają się **refaktoryzacją**.

Gra zawiera już sporo logiki działań i nie jest całkowicie liniowa. Może się zdarzyć, że źle zakodujesz któryś z warunków i w trakcie testowania gry pojawi się błąd spowodowany odwołaniem do pola w obiekcie `null`. Unity wyświetla wówczas w konsoli szczegółową informację o tym, w którym pliku źródłowym i w której jego linii wystąpił błąd, np.:

```
NullReferenceException: Object reference not set to an instance of an object
↪Doniczka.OnCollisionEnter(UnityEngine.Collision collision) (at Assets/Doniczka.cs:22)
```

W tym przypadku warto zajrzeć do pliku *Doniczka.cs* (komponent `Doniczka`), do linii o numerze 22 (numery linii wypisane są po lewej stronie okna z kodem źródłowym). W dalszej części kursu dowiesz się jeszcze, jak zatrzymywać wykonanie programu w konkretnej linii i jak badać stan zmiennych w tym momencie.

Rozwiązanie

Plik *Gracz.cs*:

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

public class Gracz : MonoBehaviour {
    public Sadzonka NiesionaSadzonka;
    public bool NiesionaKonewka;
    private int LiczbaPosadzonychRoslin;
    // Use this for initialization
    void Start () {
        LiczbaPosadzonychRoslin = 0;
    }
    public void PosadzonoSadzonke()
    {
        // przestajemy nieść sadzonkę, właśnie ją zasadziliśmy
        NiesionaSadzonka = null;
    }
    public void WyrosloDrzewo()
    {
        // przestajemy nieść sadzonkę, właśnie ją zasadziliśmy, nie niesiemy też już konewki
        NiesionaSadzonka = null;
        NiesionaKonewka = false;
        // sprawdzamy, ile roślin zostało już posadzonych
        LiczbaPosadzonychRoslin++;
        if(LiczbaPosadzonychRoslin == 3)
        {   // zwycięstwo!
            Debug.Log("Zwycięstwo!!!");
            Collider ColliderGracza = GetComponent<Collider>();
            ColliderGracza.enabled = false;
        }
    }
}
```

Plik *Doniczka.cs*:

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

public class Doniczka : MonoBehaviour {
    public Sadzonka posadzonaSadzonka;
    // Use this for initialization
    void Start () {
        posadzonaSadzonka = null;

    }

    // Update is called once per frame
    void Update () {
    }
    private void OnCollisionEnter(Collision collision)
    {
        Gracz obiektGracz = collision.collider.GetComponent<Gracz>();
        // sprawdzamy, czy gracz próbuje podlać sadzonki w doniczce
        if (posadzonaSadzonka != null && obiektGracz.NiesionaKonewka)
        {
            NiechWyrosnieDrzewo(obiektGracz);
        }
        // sprawdzamy, czy gracz niesie sadzonkę do posadzenia;
        // uwaga: gracz może posadzić sadzonkę na miejscu innej sadzonki — przegra wówczas grę
        else if (obiektGracz.NiesionaSadzonka != null)
        {
            posadzonaSadzonka = obiektGracz.NiesionaSadzonka;
            // przenosimy sadzonkę w nowe miejsce
            Transform transformSadzonki = posadzonaSadzonka.GetComponent<Transform>();
            Transform transformDoniczki = GetComponent<Transform>();
            Vector3 pozycja = transformSadzonki.position;
            pozycja.x = transformDoniczki.position.x;
            pozycja.y = transformDoniczki.position.y;
            pozycja.z = transformDoniczki.position.z;
            transformSadzonki.position = pozycja;
            posadzonaSadzonka.WlaczRenderer(true);
            obiektGracz.PosadzonoSadzonke();
        }
        else
            Debug.Log("Gracz nie ma sadzonki.");
    }
    // jeśli wszystkie warunki zostaną spełnione, wywołamy tę funkcję, aby wyrosło drzewo;
    // w 1. wersji projektu kod ten znajdował się w funkcji OnCollisionEnter,
    // ale gdy rozmiar kodu pojedynczej funkcji rośnie, dla czytelności warto go podzielić
    private void NiechWyrosnieDrzewo(Gracz obiektGracz)
    {
        GameObject doZasadzeniaRoslina = posadzonaSadzonka.Roslina;
        // włączamy roślinę
        Transform transformRosliny = doZasadzeniaRoslina.GetComponent<Transform>();
        // GetComponentInChildren<Renderer>() szuka komponentu Renderer w podobiektach tego obiektu;
        // szukamy renderera obiektu o nazwie RoslinaX_Obiekt
        Renderer rendererRosliny = transformRosliny.GetComponentInChildren<Renderer>();
        rendererRosliny.enabled = true;
        // przemieszczamy roślinę, tak aby znalazła się w doniczce
```

```
        Transform transformDoniczki = GetComponent<Transform>();
        Vector3 pozycja = transformRosliny.position;
        pozycja.x = transformDoniczki.position.x;
        pozycja.y = transformDoniczki.position.y;
        pozycja.z = transformDoniczki.position.z;
        transformRosliny.position = pozycja;
        // wyłączamy sadzonkę w doniczce
        posadzonaSadzonka.WlaczRenderer(false);
        // powiadamiamy obiekt gracza, że została posadzona roślina
        obiektGracz.WyrosloDrzewo();
        // rośnie już drzewo, a nie sadzonka
        posadzonaSadzonka = null;
    }
}
```

Plik *Sadzonka.cs*:

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

public class Sadzonka : MonoBehaviour {
    public GameObject Roslina;
    // Use this for initialization
    void Start () {
        // sprawdzamy, czy w edytorze na pewno zostało ustawione pole Roslina
        if (Roslina == null)
            Debug.LogError("Brak ustawienia pola Roslina! ");
        // na starcie dorosła roślina ma zniknąć
        Transform transformRosliny = Roslina.GetComponent<Transform>();
        // GetComponentInChildren<Renderer>() szuka komponentu Renderer w podobiektach tego obiektu;
        // szukamy renderera obiektu o nazwie RoslinaX_Obiekt
        Renderer rendererRosliny = transformRosliny.GetComponentInChildren<Renderer>();
        rendererRosliny.enabled = false;
    }
    private void OnCollisionEnter(Collision collision)
    {
        Gracz obiektGracz = collision.collider.GetComponent<Gracz>();
        obiektGracz.NiesionaSadzonka = this;
        // wyłączamy sadzonkę
        WlaczRenderer(false);
        WlaczCollider(false);
    }
    public void WlaczCollider(bool CzyWlaczyc)
    {
        Collider ColliderSadzonki = GetComponentInChildren<Collider>();
        ColliderSadzonki.enabled = CzyWlaczyc;
    }
    public void WlaczRenderer(bool CzyWlaczyc)
    {
        Renderer RendererSadzonki = GetComponentInChildren<Renderer>();
        RendererSadzonki.enabled = CzyWlaczyc;
    }
}
```

Plik *Konewka.cs*:

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

public class Konewka : MonoBehaviour {
    private void OnCollisionEnter(Collision collision)
    {
        Gracz obiektGracz = collision.collider.GetComponent<Gracz>();
        obiektGracz.NiesionaKonewka = true;
        // wyłączamy tę konewkę
        Collider colliderKonewki = GetComponent<Collider>();
        gameObject.SetActive(false);
        colliderKonewki.enabled = false;
    }
}
```

Projekt JakimJestemTypem — część II

Czas: około 60 minut.

Zadanie

Projekt *JakimJestemTypem* należy rozbudować o nowy komponent, który zamiast pola z typem prostym będzie zawierał trzy pola z typami obiektowymi. Tymi typami będą klasy, które utworzyliśmy w trakcie lekcji (a więc komponenty z typami prostymi).

Inaczej mówiąc, w rozwiązaniu powinien pojawić się następujący schemat:

- `KomponentzPolemObiektowym`,
- pole `ObiektZIntem` typu `KomponentZLiczbaCalkowita`,
- pole `ObiektZFloatem` typu `KomponentZLiczbaZmiennoprzecinkowa`,
- pole `ObiektZBoolem` typu `KomponentZPolemBool`.

`KomponentzPolemObiektowym` powinien mieć funkcję, która sama wyznaczy sumę dla posiadanych przez niego obiektów.

Gdy skończysz, poeksperymentuj według własnego uznania, dodając więcej obiektów na scenę, dodatkowe złożone komponenty itp.

Podpowiedź

Pola obiektowe uzupełnij w edytorze (w przyszłości dowiesz się, jak tworzyć nowe obiekty w kodzie). Aby to zrobić, najpierw musi istnieć obiekt na scenie z odpowiednim komponentem. Utwórz kilka ukrytych obiektów bez reprezentacji graficznej, które będą posiadały wyłącznie komponent `Transform` i ten z komponentów z lekcji, który chcesz zastosować i przypisać do pola obiektowego.

Rozwiązanie

Plik *Gracz.cs*:

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

public class Gracz : MonoBehaviour {
    private float Suma;
    // Use this for initialization
    void Start () {
        Suma = 0f;
    }
    private void OnCollisionEnter(Collision collision)
    {
        KomponentZLiczbaCalkowita ObiektZLiczbaCalkowita =
        ↪collision.collider.GetComponent<KomponentZLiczbaCalkowita>();
        KomponentZLiczbaZmiennoprzecinkowa ObiektZLiczbaZmiennoprzec =
        ↪collision.collider.GetComponent<KomponentZLiczbaZmiennoprzecinkowa>();
        KomponentZPolemBool ObiektZPoleBool =
        ↪collision.collider.GetComponent<KomponentZPolemBool>();
        KomponentZPolemObiektowym ObiektZObiektami =
        ↪collision.collider.GetComponent<KomponentZPolemObiektowym>();
        if (ObiektZLiczbaCalkowita != null)
        {
            int wartosc = ObiektZLiczbaCalkowita.Liczba;
            Debug.Log("Trafiony ma typ z liczbą całkowitą: " + wartosc.ToString());
            Suma = Suma + wartosc;
        }
        if (ObiektZLiczbaZmiennoprzec != null)
        {
            int wartosc = (int)ObiektZLiczbaZmiennoprzec.Liczba;
            Debug.Log("Trafiony ma typ z liczbą zmiennoprzecinkową, zmieniamy ją na: " +
            ↪wartosc.ToString());
            Suma = Suma + wartosc;
        }
        if (ObiektZPoleBool != null)
        {
            int wartosc = 0;
            if(ObiektZPoleBool.PoleBool == true )
            {
                wartosc = 1;
                Debug.Log("Trafiony ma typ z polem bool o wartości true, dodajemy 1 do sumy");
            } else Debug.Log("Trafiony ma typ z polem bool o wartości false");
            Suma = Suma + wartosc;
        }
        if (ObiektZObiektami != null)
        {
            int wartosc = ObiektZObiektami.PodajSwojaSume();
            Debug.Log("Suma dla komponentu z polami obiektowymi wynosi: " +
            ↪wartosc.ToString());
            Suma = Suma + wartosc;
        }
        Debug.Log("Suma wynosi: " + Suma.ToString());
    }
}
```

Plik *KomponentZPolemObiektowym.cs*:

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

public class KomponentZPolemObiektowym : MonoBehaviour {
    public KomponentZLiczbaCalkowita ObiektZIntem;
    public KomponentZLiczbaZmiennoprzecinkowa ObiektZFloatem;
    public KomponentZPolemBool ObiektZBoolem;
    public int PodajSwojaSume()
    {
        int Suma = 0;
        if (ObiektZIntem != null)
            Suma += ObiektZIntem.Liczba;
        if (ObiektZFloatem != null)
            Suma += (int)ObiektZFloatem.Liczba;
        if (ObiektZBoolem != null)
            if (ObiektZBoolem.PoleBool == true)
                Suma++;
        return Suma;
    }
}
```

Pliki *KomponentZLiczbaCalkowita.cs*, *KomponentZLiczbaZmiennoprzecinkowa.cs* i *Komponent ↪ZPolemBool.cs* są takie same jak w zadaniu „Projekt JakimJestemTypem — część I”.

Skorowidz

K

M

O

P

T

U

V

W

Z

Zdobądź pracę marzeń – zostań programistą Unity!

- **Poznaj środowisko pracy**
- **Naucz się tworzyć sceny**
- **Buduj ciekawe projekty**

Unity to zintegrowane środowisko umożliwiające tworzenie trójwymiarowych i dwuwymiarowych gier oraz różnego rodzaju interaktywnych treści, takich jak wizualizacje i animacje komputerowe. Wieloplatformowość rozwiązania, prostota używania, bogate funkcje, możliwość generowania materiałów prezentowanych na różnych urządzeniach oraz obsługa różnych języków skryptowych zadecydowały o niezwykłej popularności Unity i spowodowały, że to jeden z najczęściej wykorzystywanych silników gier komputerowych.

Jeśli marzysz o karierze twórcy gier, lecz obawiasz się, że nie uda Ci się opanować sztuki programowania, przekonaj się, że to wcale nie jest takie trudne! Dzięki tej książce sprawnie i pewnie wkroczysz w świat tworzenia gier komputerowych za pomocą języka C# w środowisku Unity. Dowiesz się, co trzeba zrobić, aby opanować niezbędne podstawy. Za sprawą dobrze przemyślanych lekcji osiągniesz kolejne stopnie wtajemniczenia, a zadania i podpowiedzi umożliwią Ci utrwalenie wiedzy oraz sprawdzenie jej w praktyce. Bez nadmiaru teorii i prosto do celu – właśnie tak nauczysz się Unity!

- **Instalacja i przegląd środowiska pracy**
- **Podstawowe operacje i komponenty Unity**
- **Tworzenie skryptów oraz kompilacja projektu**
- **Inżynieria oprogramowania, procesy i dokumentacja**
- **Podstawy programowania w języku C#**
- **Typy danych i konstrukcje programistyczne**
- **Debugowanie programu i używanie zewnętrznego kodu**

Tworzenie gier z Unity? To nic trudnego!

Helion
helion.pl
HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-9237-3

Cena: 49,90 zł