



Ewa Ross, Jacek Ross

Unity i C#

PODSTAWY PROGRAMOWANIA GIER



Helion 

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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

Copyright © Ewa Ross, Jacek Ross 2018, 2022

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Wstęp	9
O czym jest książka?	9
Ale czy na pewno przeciętna osoba bez wrodzonych cech programisty może się nauczyć programować?	11
Struktura książki	11
Lekcja 1. Programowanie w C# i Unity — szybki start	13
Instrukcja instalacji programów Unity i Visual Studio	13
Unity i Visual Studio — pierwsze uruchomienie	
i utworzenie pierwszego projektu	17
Informacje o Unity i Visual Studio	17
Utworzenie pierwszego projektu	17
Główne okna Unity	18
Uruchomienie Visual Studio	19
Okna Unity oraz wstawienie pierwszych obiektów na scenę	22
Podstawowe okna Unity	22
Zasoby i prefabrykaty	23
Wstawienie obiektów na scenę	23
Modyfikacja obiektów poprzez okno inspektora	24
Uruchomienie gry	25
Podstawowe komponenty: Transform, Collider, Renderer i Rigidbody	25
Komponenty ogólnie	25
Komponent Transform	26
Komponent Renderer	27
Komponent Collider	27
Komponent Rigidbody	27

Pierwszy własny skrypt C# i pierwszy własny komponent	28
Dodanie własnego komponentu	28
Kod źródłowy komponentu	28
Komentarze w kodzie programu	29
Dodanie parametru do komponentu	29
Komponent a klasa	30
Funkcje. Specjalne funkcje komponentów Unity.	
Użycie funkcji OnCollisionEnter	31
Obiekty i pobieranie komponentów jako obiektów.	
Zmiana parametrów i pól obiektu	32
Bardziej zaawansowany sposób wykonania zadania (opcjonalnie)	32
Kompilacja projektu w Unity	34
Porady i pomoc w poruszaniu się po programach Unity i Visual Studio	35
Mam otwarte Unity, ale nie widać sceny i obiektów	35
Nie umiem otworzyć Visual Studio tak,	
aby otwierało mój projekt i moje pliki źródłowe	35
Zadania do samodzielnego wykonania	36
Zadanie 1. Rozbudowa projektu ObjectClicker — część I	36
Zadanie 2. Rozbudowa projektu ObjectClicker — część II	37
Zadanie 3. Rozbudowa projektu ObjectClicker — część III	38
Zadanie 4. Rozbudowa projektu ObjectClicker — część IV	40
Lekcja 2. Informacja, algorytm, kompilacja	45
Informacja a algorytm	45
W świecie komputerów informacja jest wszystkim	45
W świecie komputerów programista jest najważniejszy	46
Sposób zapisu informacji to też informacja	46
Algorytm. Główna rola programisty polega	
na zapisaniu algorytmu w języku programowania	46
Do przemyślenia	47
Algorytmy	47
Algorytm Pieczenie ciasta	47
Algorytm obliczania pola powierzchni koła	50
Zapis algorytmu w schemacie blokowym na przykładzie algorytmu gry 3-5-8	51
Zapis algorytmu gry 3-5-8 w pseudokodzie	53
Dokumentacja oraz procesy tworzenia gier	53
Metody tworzenia gier i programów	53
Role w projekcie	54
Dokumentacja projektowa	56

Kompilacja projektu w Unity i Visual Studio, biblioteki, pliki projektu	58
Kompilacja projektu w Unity	58
Pliki bibliotek	59
Pliki projektu i rozwiązań	60
Inne pliki i katalogi	61
Kompilacja kodu źródłowego w Visual Studio	61
Zadania do samodzielnego wykonania	62
Samodzielne utworzenie dokumentacji projektowej gry	62
Lekcja 3. Typy danych, zmienne, funkcje i klasy	63
Jak manipulować obiektami na scenie 3D w Unity	63
Przybornik narzędzi	63
Przyciąganie podczas przesuwania	64
Szybkie debugowanie zmiennych w Unity i VS	64
Typy danych i zmienne	66
Typy danych i zmienne	66
Rzutowanie zmiennych	69
Funkcje	69
Klasy — część I	72
Lekcja 4. Instrukcje warunkowe	75
Instrukcje warunkowe — część I	75
Zadania do samodzielnego wykonania	77
Projekt JakimJestemTypem — część I	77
ProjektRosliny — część I	79
ProjektRosliny — część II	81
Projekt JakimJestemTypem — część II	85
Lekcja 5. Typy danych, klasy, instrukcje warunkowe — ciąg dalszy.	
Programowanie w Unity	89
Klasy — część II	89
Dziedziczenie klas	89
Tworzenie nowych obiektów	92
Złożenia wywołań metod i pól	92
Słowo kluczowe this	93
Rzutowanie typów obiektowych	94

Instrukcje warunkowe — część II	95
Zagnieżdżanie warunków	95
Kolejność obliczeń	96
Operator !	96
Zadanie do samodzielnego wykonania	97
Projekt JakimJestemTypem — część III	97
Programowanie komponentów Unity	100
Funkcje Update i OnGUI	100
Zmiana pozycji obiektu sceny w hierarchii obiektów	102
Aktywność obiektu	103
GameObject.Find	103
Zadania do samodzielnego wykonania	104
ProjektRosliny — część III	104
Gra 3-5-8	108
Lekcja 6. Kolekcje i pętle	113
Kolekcje	113
Kolekcje ogólnie	113
Deklaracja i używanie tablic	113
Ćwiczenie	114
Pętle for i foreach	115
Pętla for	115
Przykłady pętli for	116
Zadanie do samodzielnego wykonania	119
Ćwiczenie pętli for	119
Pętla foreach	119
Zadania do samodzielnego wykonania	121
Ćwiczenie pętli foreach	121
Projekt ObjectClicker— część V	122
Pętla while	123
Pętla while	123
Zadanie do samodzielnego wykonania	125
Projekt ObjectClicker — część VI — wersja z pętlą while.....	125
Podsumowanie tematyki pętli	125
Zadania do samodzielnego wykonania	126
ProjektRosliny — część IV	126
Gra MicroAbrix	130

Lekcja 7. Cykl życia obiektów, przestrzenie nazw, instrukcja switch, typ string, debugowanie	133
Konstruktory i przeciążanie metod	133
Konstruktory. Cykl życia obiektów	133
Przeciążanie metod	134
Widoczność zmiennych	136
Widoczność zmiennych wewnątrz bloków	136
Pola klasy	137
Inne sposoby przekazywania parametrów: out i ref	139
Pola statyczne klasy	141
Przestrzenie nazw	142
Instrukcja switch	143
Instrukcja switch	143
Instrukcje break i continue	144
Typ string	144
Typy danych string i char	144
Metody i pola klasy String	145
Klasa StringBuilder	145
Konwersja liczb na tekst i odwrotnie	146
Debugowanie w Visual Studio	146
Podłączenie Visual Studio do gry uruchomionej w Unity	146
Stos wywołań	148
Podgląd i modyfikacja wartości zmiennych	148
Kontynuacja wykonania programu po pułapce	149
Zadania do samodzielnego wykonania	150
Projekt FixMe1	150
Projekt FixMe2	152
Uszkodzony ProjektRosliny — część V	155
Lekcja 8. Typ enum. Użycie gotowego kodu	161
Typ wyliczeniowy — Enum	161
Typ enum	161
Rzutowanie typu enum	162
Wyszukiwanie i użycie dokumentacji technicznej.	
Wykorzystanie gotowego kodu	163
Pomoc techniczna Unity i Visual Studio	163
Uzyskiwanie pomocy w internecie	165

Zakup zasobów. Unity Asset Store	165
Wady i zalety używania gotowego kodu	167
Zadania do samodzielnego wykonania	169
Gotowy projekt Tanks!	169
Skorowidz	171

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 ObiektZPolemBool =
            ↳collision.collider.GetComponent<KomponentZPolemBool>();
    }
}
```

```

if(ObiektZLiczbaCalkowita != null)
{
    int wartosc = ObjektZLiczbaCalkowita.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 zanieśenie 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 podobieństwach tego obiektu;
            // szukamy renderera obiektu o nazwie RoslinaX_Objekt
            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;
            // powiadamy 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_Objekt
        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 zatrzymać 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 doZasadeniaRoslina = posadzonaSadzonka.Roslina;
        // włączamy roślinę
        Transform transformRosliny = doZasadeniaRoslina.GetComponent<Transform>();
        // GetComponentInChildren<Renderer>() szuka komponentu Renderer w podobiektach tego obiektu;
        // szukamy renderera obiektu o nazwie RoslinaX_Objekt
        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;
    // wylaczamy 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_Objekt
        Renderer rendererRosliny = transformRosliny.GetComponentInChildren<Renderer>();
        rendererRosliny.enabled = false;
    }
    private void OnCollisionEnter(Collision collision)
    {
        Gracz obiektGracz = collision.collider.GetComponent<Gracz>();
        obiektGracz.NiesionaSadzonka = this;
        // wylaczamy 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;
        //wyluczamy 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 KomponentZLiczbaCalokowita,
- 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 *KomponentZPolemBool.cs* są takie same jak w zadaniu „Projekt JakimJestemTypem — część I”.

Skorowidz

A

- agile, 54
- aktywność obiektu, 103
- algorytm, 46
 - obliczania pola powierzchni koła, 50
 - Pieczenie ciasta, 47
- architektura projektu, 58

B

- biblioteka, 59
- błędy, 147
- breakpoint, 147

C

- cykl życia
 - obiektu, 133
 - projektu, 54

D

- debugowanie, 64
 - breakpoint, 147
 - kontynuacja wykonania programu, 149
 - modyfikacja wartości zmiennych, 148
 - podgląd wartości zmiennych, 148
 - w Visual Studio, 146
- deklaracja
 - tablic, 113
 - zmiennych, 67
- destruktor, 134
- diagram klas, 57
- dodanie
 - parametru do komponentu, 29
 - własnego komponentu, 28
- dokumentacja, 53
 - fabuła gry, 57
 - mechanika gry, 56
 - projektowa, 56

- techniczna, 163
- wizja projektu, 56
- wymagań, 57
- dostęp
 - do metod, 91
 - do renderera, 32
- dziedziczenie klas, 89

F

- FPS, frame per second, 100
- funkcja, 31, 69
 - OnCollisionEnter, 31
 - OnGUI, 100
 - Parse, 146
 - Update, 100, 101
- funkcje
 - prywatne, 71
 - publiczne, 71

G

- gra MicroAbrix, 130
- GUI, 102

H

- hierarchia obiektów, 102

I

- indeks, 113
- informacja, 45
- instalacja
 - Unity, 13
 - Visual Studio, 13
- instrukcja
 - break, 143
 - continue, 144
 - switch, 143
- instrukcje warunkowe, 75, 95

K

klasa, 30, 32, 89
 GameObject, 103
 OpelAstra, 72
 String, 145
 StringBuilder, 145
 klasy
 dziedziczenie, 89
 ogólne, 89
 szczegółowe, 89
 kolejność obliczeń, 96
 kolekcje, 113
 komentarze, 29
 kompilacja projektu, 34
 w Unity, 58
 w Visual Studio, 61
 komponent, 30, 32
 Collider, 27
 Renderer, 27
 Rigidbody, 27
 Transform, 26
 komponenty
 ogólne, 25
 Unity, 100
 konstruktor, 133
 domyślny, 134
 przeciążanie, 135
 konwersja liczb, 146

M

metoda ToString, 146
 metodologia
 Scrum, 54
 tradycyjna, 54
 zwinna, 54
 metody, 71
 przeciążanie, 134
 klasy String, 145
 modyfikacja
 obiektów, 24
 wartości zmiennych, 148
 modyfikator
 out, 139, 140
 ref, 140

O

obiekty, 32, 72
 aktywność, 103
 na scenie, 32
 obracanie, 64
 tworzenie, 92
 zwracanie, 139
 okna Unity
 główne, 18
 podstawowe, 22
 okno
 hierarchii obiektów, 18
 inspektora, 18, 24
 kompilacji, 59
 widoku sceny, 18
 operator
 !, 96
 !=, 76
 ||, 68
 <, 76
 <=, 76
 ==, 76
 >, 76
 >=, 76
 &&, 68
 as, 94
 is, 95
 otwieranie projektu, 20

P

pętla
 for, 115, 116, 125
 foreach, 119, 126
 while, 123, 126
 pętle
 instrukcja break, 144
 instrukcja continue, 144
 pierwszy
 komponent, 28
 projekt, 17
 skrypt, 28
 platforma .NET, 17
 plik
 DoDestrukcji.cs, 131
 Doniczka.cs, 80, 106, 128, 159
 GłównySkrypt.cs, 150, 152
 GłównySkryptPoprawnie.cs, 151, 154

Gracz.cs, 77, 79, 82, 86, 98, 105, 127, 131
 Jablko.cs, 99
 Klikacz.cs, 122, 125
 KomponentGracza.cs, 109, 119, 121
 KomponentZLiczbaCalkowita.cs, 78
 KomponentZLiczbaZmiennoprzecinkowa.cs, 78
 KomponentZPolemBool.cs, 78
 KomponentZPolemObiektowym.cs, 87
 Konewka.cs, 85, 108
 Owoc.cs, 99
 Pojemnik.cs, 110
 Sadzonka.cs, 80, 84, 107, 155
 pliki
 bibliotek, 59
 projektu, 60
 podgląd wartości zmiennych, 148
 pola klasy, 71
 dostępność, 138
 statyczne, 141
 String, 145
 użycie, 137
 widoczność, 138
 pomoc techniczna, 163
 prefabrykaty, 23, 32
 programowanie
 ekstremalne, 54
 komponentów Unity, 100
 projekt
 FixMe1, 150
 FixMe2, 152
 ObjectClicker, 122
 ProjektRosliny, 79, 104, 126, 155
 Tanks!, 169
 przeciążanie
 konstruktorów, 135
 metod, 134
 przekazanie parametrów do funkcji, 70, 139
 modyfikator out, 139
 modyfikator ref, 140
 przestrzenie nazw, 142
 przesuwanie
 obiektu, 63
 sceny, 63
 przybornik narzędzi, 63
 przyciąganie podczas przesuwania, 64
 przypadki użycia, 57

pseudokod, 53
 pułapka w kodzie, 147, *Patrz także*
 debugowanie

R

ramki graficzne, 100
 refaktoryzacja, 82
 rola
 Analityk, 54
 Designer, 55
 Dokumentalista, 55
 Lider zespołu, 56
 Menedżer, 56
 Programista, 55
 Projektant, 55
 Tester, 55
 Wdrożeniowiec, 55
 rozciąganie obiektu, 64
 rozwiązania, 60
 rzutowanie
 typów, 66
 typów obiektowych, 94
 typu enum, 162
 zmiennych, 69

S

scena, 23
 3D, 63
 otwieranie, 36
 przesuwanie, 63
 zmiana pozycji obiektu, 102
 schemat blokowy, 51, 52
 Scrum, 54
 skalowanie obiektu, 64
 skok warunkowy, 143
 słowo kluczowe
 break, 143
 continue, 144
 if, 75
 new, 92
 null, 76, 93
 private, 91
 protected, 91
 public, 91
 static, 141
 this, 93
 using, 142

sortowanie, 117
 specyfikacja, *Patrz także* dokumentacja
 wewnętrzna programu, 56
 zewnętrzna programu, 57
 stos wywołań, 148
 szybkie GUI, 102

T

tablice, 113
 deklaracja, 113
 sortowanie elementów, 117
 użycie, 114
 tworzenie
 gier i programów, 53
 obiektów, 92
 projektu, 17
 typ danych
 bool, 66, 68
 float, 66, 67
 enum, 161
 int, 66
 obiektyw, 92
 string, 144
 typy
 proste, 66
 wyliczeniowe, 161

U

UML, Unified Modeling Language, 57
 Unity, 17
 Unity Asset Store, 165
 uruchomienie
 gry, 25
 Visual Studio, 19
 uzyskiwanie pomocy, 165
 użycie
 dokumentacji technicznej, 163
 funkcji OnCollisionEnter, 31
 gotowego kodu, 167
 instrukcji break, 144
 instrukcji continue, 144
 instrukcji switch, 143
 klasy StringBuilder, 146
 modyfikatora out, 140
 pętli for, 117
 pętli while, 123
 pola klasy, 72, 137

pola statycznego, 141
 przestrzeni nazw, 142
 tablic, 113
 typu string, 145
 typu wyliczeniowego, 162
 zmiennych, 67, 68

V

Visual Studio, 17

W

wartość null, 76, 93
 warunek, 75
 widoczność
 metod i pól, 91
 zmiennych, 136
 wyszukiwanie błędów, 147
 wyszukiwarka zasobów, 166
 wywołania
 przeciążonej metody, 135
 wielokrotne, 92

Z

zagnieżdżanie warunków, 95
 zakładka Project, 19
 zakup zasobów, 165
 zasoby, 23, 32
 zmienne, 66
 modyfikacja wartości, 148
 podgląd wartości, 148
 typu obiektowego, 73
 użycie, 67
 zwracanie obiektu, 139

PROGRAM PARTNERSKI

— GRUPY HELION —



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion

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.pl HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	<i>Sprawdź nasze szkolenia!</i> SZKOLENIA  AKADEMIA IT & BUSINESS HELIONSZKOLENIA.PL	KOD KORZYŚCI Sięgnij po więcej!   ISBN 978-83-283-9237-3  9 788328 392373
INFORMATYKA W NAJLEPSZYM WYDANIU		Cena: 49,90 zł