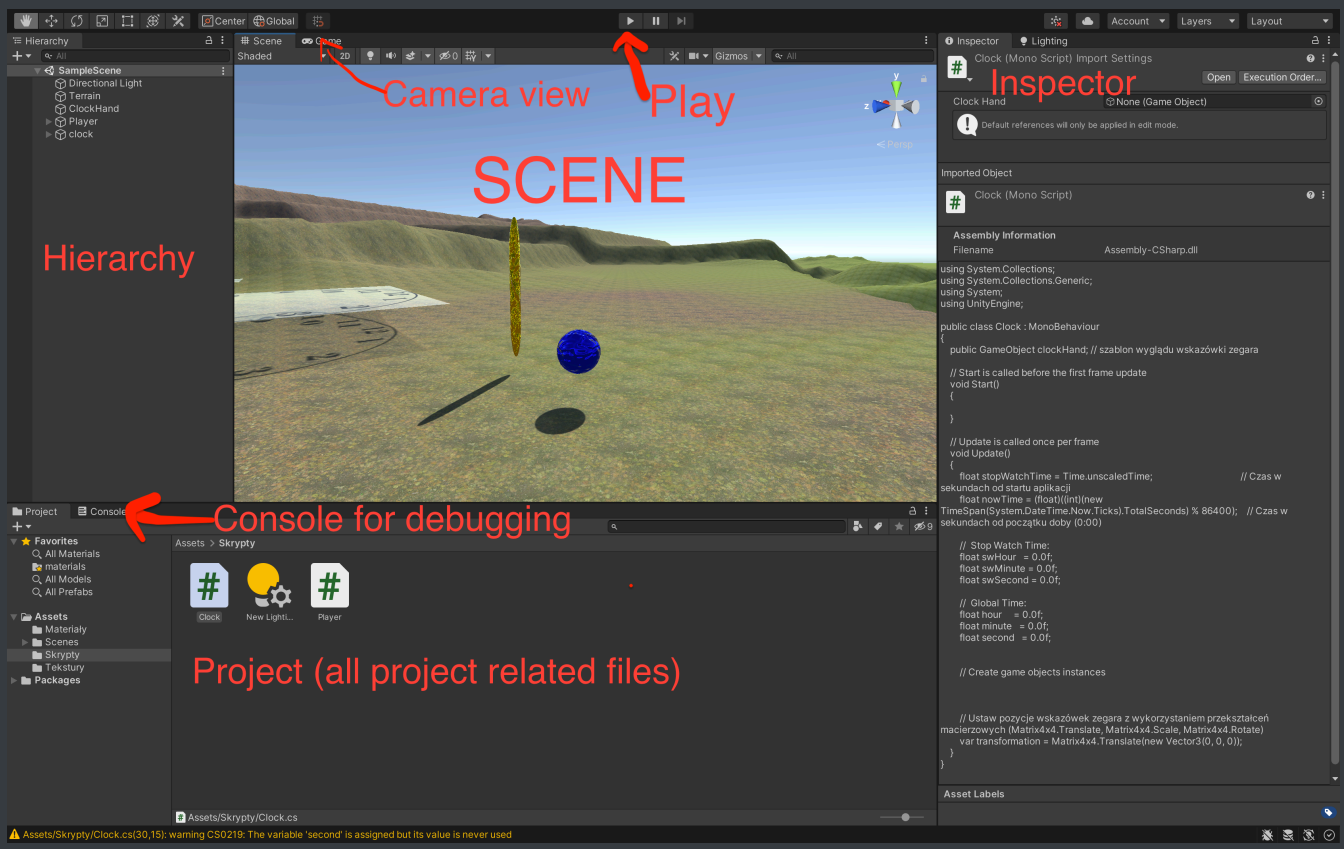
Ćwiczenia 14.03

Hello World w Unity

(Pliki https://git.wmi.amu.edu.pl/andkok/Unity_artifitial_world/src/branch/cw1/materiałyHelloWorld)

1. **Utwórz nowy projekt**
   - ☐ Otwórz Unity Hub
   - ☐ Wybierz NEW
   - ☐ 3D, wpisz nazwę i wybierz lokalizacje na twoim komputerze. Zatwierdź klikając CREATE
2. **Import materiałów, tworzenie terenu i prostych krztałtów**



Po środku ekranu jest podgląd sceny (karta Scene). Żeby się w nim przemieszczać klikamy Q i wtedy możemy przesuwać obraz myszką (kliknięcie na prawy przycisk myszy do obracania widoku). Klawisz W zmienia w tryb przesuwania obiektów, a klawisz E jest do obracania obiektów i R zmiany rozmiarów.

W dolnej części ekranu, w karcie Project przechowujemy wszystkie modele, tekstury, skrypty i inne potrzebne elementy

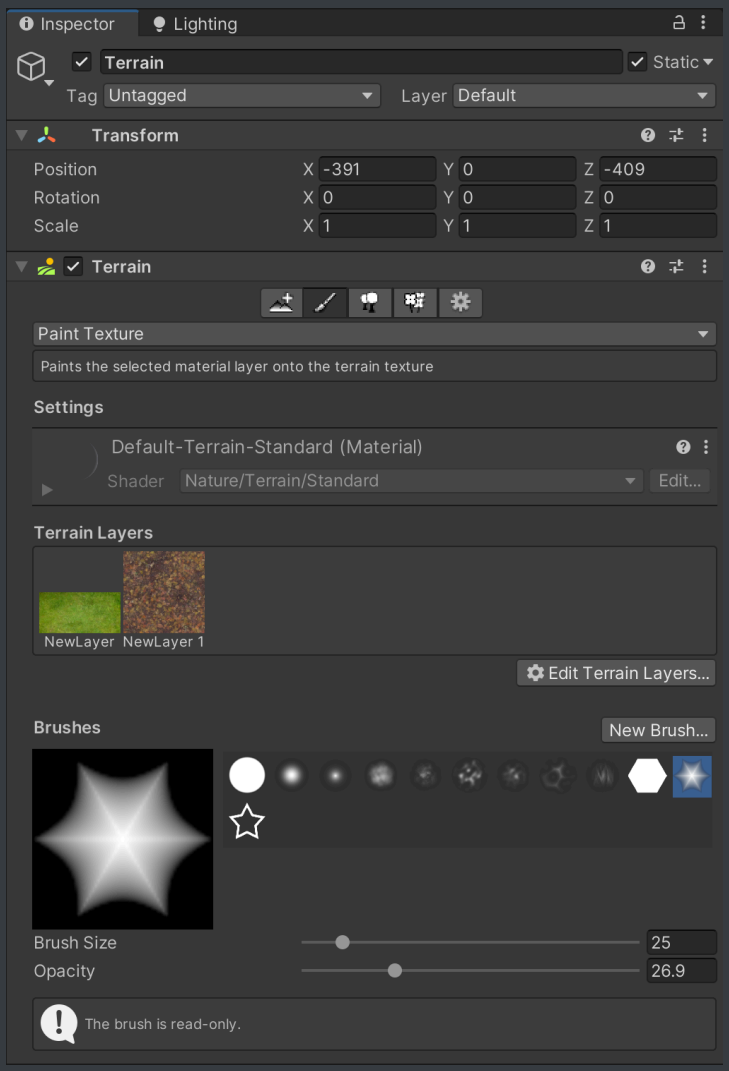
- [ ] W folderze Assets kliknij prawym przyciskiem myszy i z menu wybierz Create -> Folder. Nazwij folder tekstury
- [ ] Otwórz ten folder i przeciągnij do niego wcześniej pobrane tekstury

  Po lewej stronie jest karta Hierarchy. W niej znajdują się wszystkie obiekty, które są w tworzonej scenie. Tam się dodaje nowe elementy sceny i ustawia zależności pomiędzy elementami

  ## Teren

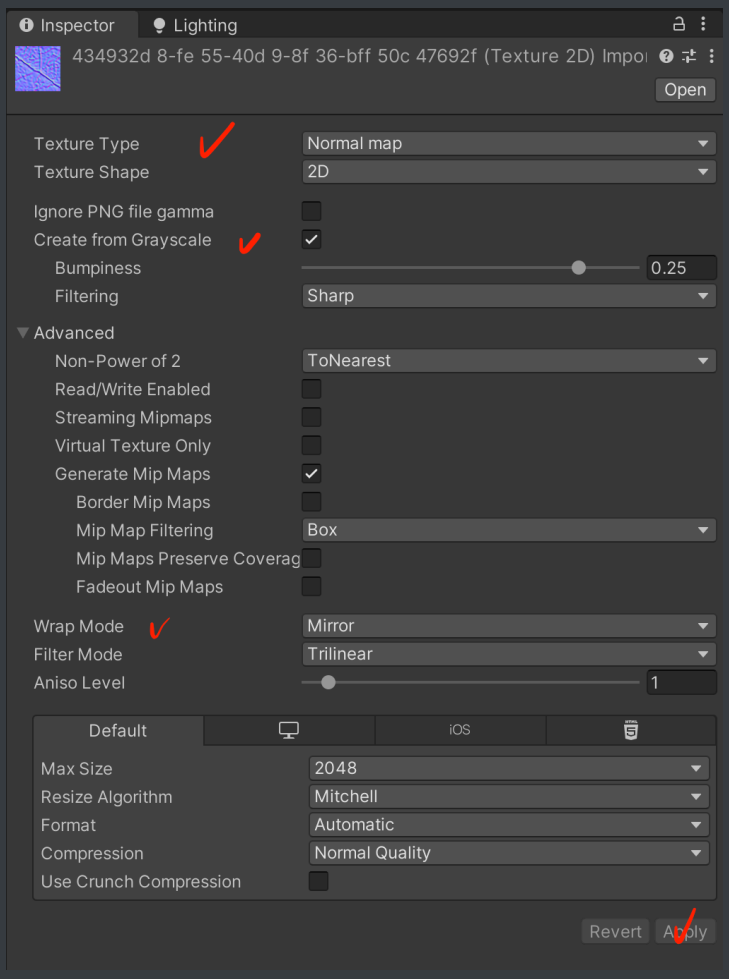
- [ ] Kliknij prawym przyciskiem myszy w polu Hierarchy i wybierz 3D Object -> Terrain

  Po prawej stronie mamy karte inspector w której są ustawienia zaznaczonego obiektu

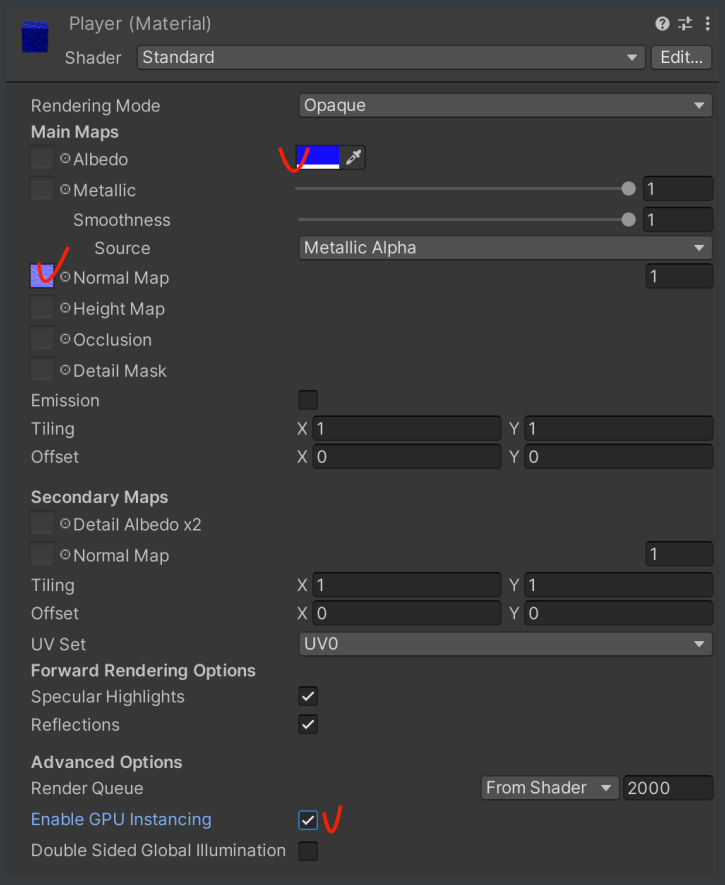


- ☐ Dodaj teksturę terenu
  - ☐ W Inspektorze przy zakładce Terrain wybierz drugą opcje (ikonka pędzelka)
  - ☐ Wybierz z menu Paint Texture
  - ☐ Kliknij Edit Terrain Layers -> Create Layer
  - ☐ Dodaj trzy tekstury
  - ☐ Urozmaić teren zaznaczając tekstury i malując przy pomocy wybranych poniżej pędzli
- ☐ Zmień wysokości terenu
  - ☐ W Inspektorze przy zakładce Terrain wybierz drugą opcje (ikonka pędzelka)
  - ☐ Wybierz z menu Raise or Lower Terrain
  - ☐ Podnosi się teren "malując" po obiekcie terenu, żeby teren obniżyć należy trzymać wciśnięty shift

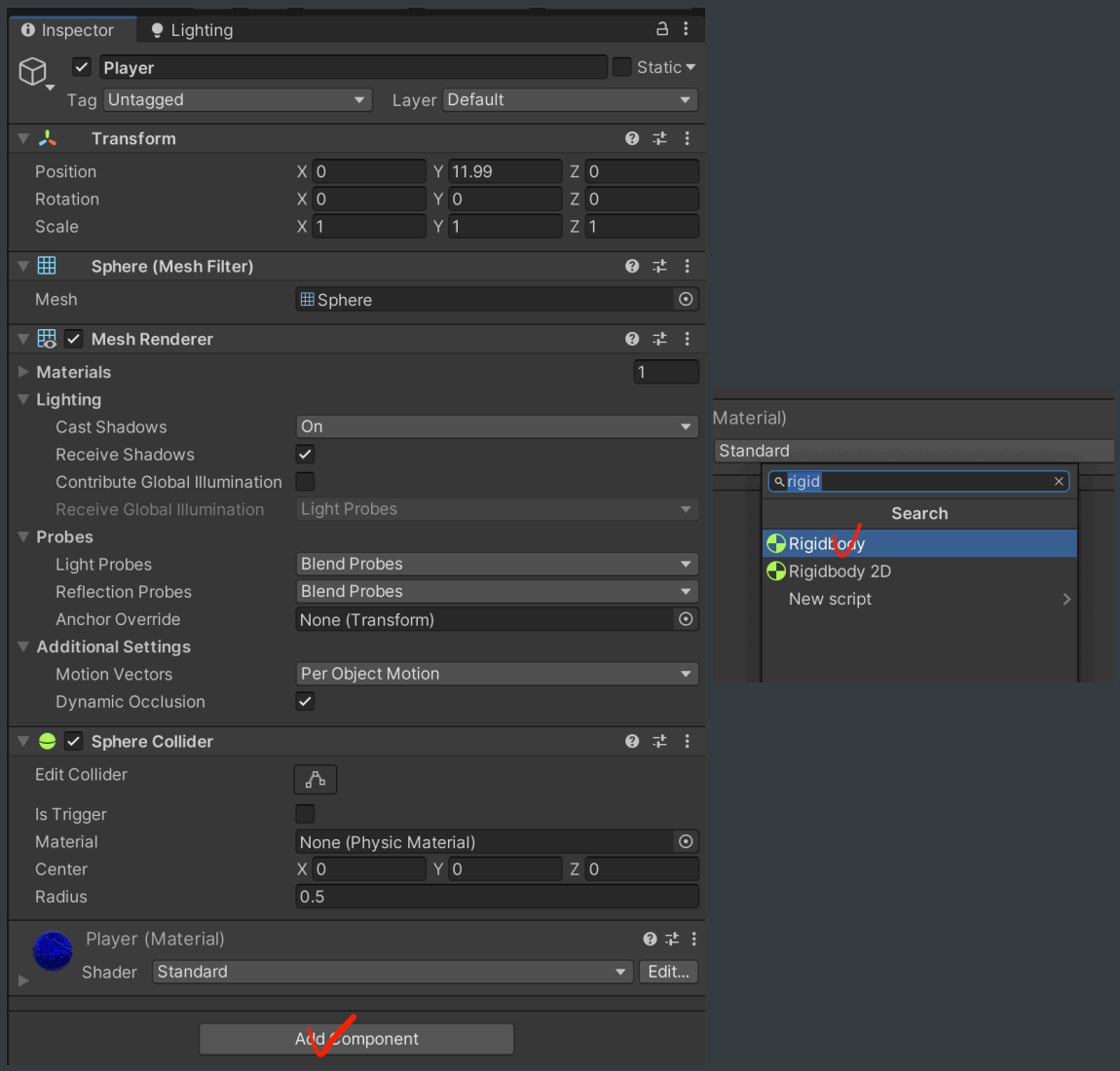
Proste kształty

- [ ] W karcie Hierarchy kliknij prawym przyciskiem myszy i wybierz 3D Object -> Sphere
- [ ] żeby nadać kuli koloru utwórz materiał
  - [ ] 

  - [ ] Dodaj teksture SphereNormals do tekstur i Inspektorze wybierz texture type: Normal Map, zaznacz create from grayscale, wrap mode: mirror
  - [ ] W folderze Assets utwórz folder Materiały i kliknij prawym przyciskiem myszy i wybierz Create -> Material

- [ ] 

    - [ ] Wybierz kolor Albedo i dodaj teksture do "Normal Map"
    - [ ] Przeciągnij materiał na kulę
- [ ] żeby na kulę działała grawitacja i kolizja dodajemy do niej Rigid Body (Add Component -> Rigidbody), i w Constrains - Freeze rotation zaznacz wszystkie (żeby nam się kula nie stoczyła)

- [ ] 

3. Dodawanie skryptu C#

W Unity za większość akcji i różnych interakcji odpowiadają skrypty C#

- [ ] Utwórz folder skrypty i w nim Create -> C# Script, nazwij go Player.
- [ ] Otwórz ten skrypt i Przed funkcją Start dodaj zmienne:

```
public float speed     = 0.1f;
public float rotationSpeed = 0.1f;
```

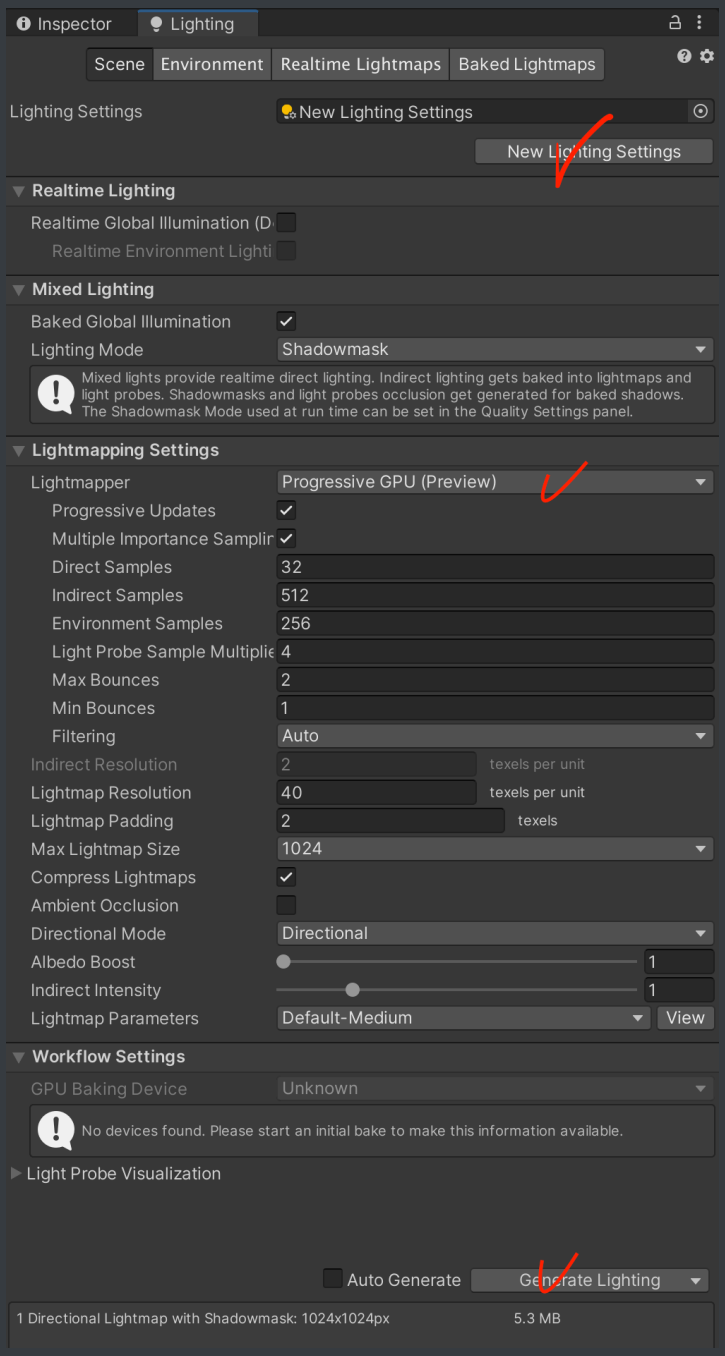
- [ ] wewnątrz funkcji Update wklej:

```
        float move  = Input.GetAxis("Vertical") * speed;
    // Pobierz dane od użytkownika (klawisze strzałek lub joystick)
    i przeskaluj wcześniej podaną wartością (przód tył)
        float rotate = Input.GetAxis("Horizontal") * rotationSpeed;
    // Pobierz dane od użytkownika (klawisze strzałek lub joystick)
    i przeskaluj wcześniej podaną wartością (obrót)
        Vector3 moveVector = new Vector3(PODAJ_X,PODAJ_Y,PODAJ_Z);
     // Wektor przesuwający objekt      DO EDYCJI
        gameObject.GetComponent<Transform>().Translate(moveVector,
    Space.Self);  // Funkcja przesuwająca objekt
        gameObject.GetComponent<Transform>
    ().Rotate(PODAJ_X,PODAJ_Y,PODAJ_Z, Space.Self); // Funkcja
    obracająca objekt DO EDYCJI
```

- [ ] Zamień wartości wektora ruchu i funkcji obrotu
- [ ] Przeciągnij skrypt na object Player lub w Inspektorze obiektu Player wybierz Add Component i dodaj Player

4. **Ustawianie oświetlenia (globalne)**

- [ ] Ustawienia oświetlenia są w Window -> Rendering -> Ligthning, można je sobie przypiąć w dowolnym miejscu (ja przesuwam je koło inspektora)

- [ ] 

  - [ ] Dodajemy nowe ustawienia (New Ligthning Settings)
  - [ ] Można zmienić Ligthmapper na Progressive GPU
  - [ ] Po dostosowaniu ustawień klikamy Generate Lightning (to może chwilę potrwać, nawet do kilku minut)

5. **Tworzenie wielu elementów skryptem z ustawianiem ich za pomocą macierzy (Clock)**

- [ ] Utwórz objekt tarczy zegara (tekstura  , zamień przy tworzeniu materiału Rendering mode na Transparent)
- [ ] Utwórz obiekt, który posłuży jako szablon wskazówek zegara
- [ ] Dodaj kod ustawiający zegar do obiektu tarczy zegara

```
using System.Collections;
using System.Collections.Generic;
using System;
using UnityEngine;
public class Clock : MonoBehaviour
{
  public GameObject clockHand; // szablon wyglądu wskazówki
zegara
  void Start()
  {
  }
  void Update()
  {

        DateTime centuryBegin = new DateTime(2001, 1, 1);
        DateTime currentDate = DateTime.Now;
        long elapsedTicks = currentDate.Ticks -
centuryBegin.Ticks;
        TimeSpan elapsedSpan = new TimeSpan(elapsedTicks);

        float stopWatchTime = Time.unscaledTime;
                    // Czas w sekundach od startu aplikacji
(mały zegar)
```

```
        float nowTime = (float)(((int)
(elapsedSpan.TotalSeconds)) % 86400);    // Czas w sekundach od
początku doby (0:00) (duży zegar)
    // Stop Watch Time:

    float swHour  = 0.0f;
    float swMinute = 0.0f;
    float swSecond = 0.0f;
    // Global Time:
    float hour   = 0.0f;
    float minute  = 0.0f;
    float second  = 0.0f;
// Ustaw pozycje, rozmiar i obrót wskazówek zegara z
wykorzystaniem przekształceń macierzowych:
    //    (Matrix4x4.Translate, Matrix4x4.Scale,
Matrix4x4.Rotate) dwa pierwsze przyjmują Vector3, a obroty
przyjmują Quaternion.Euler( X, Y, Z) (już bez new, bo to
funkcja)

    var transformationHsw = Matrix4x4.Translate(new Vector3(0,
0, 0));
    var transformationMsw = Matrix4x4.Translate(new Vector3(0,
0, 0));
    var transformationSsw = Matrix4x4.Translate(new Vector3(0,
0, 0));

    var transformationH = Matrix4x4.Translate(new Vector3(0, 0,
0));
    var transformationM = Matrix4x4.Translate(new Vector3(0, 0,
0));
    var transformationS = Matrix4x4.Translate(new Vector3(0, 0,
0));

// Create game objects instances
```

```
 Graphics.DrawMeshInstanced(clockHand.GetComponent<MeshFilter>
().mesh, 0, clockHand.GetComponent<MeshRenderer>().material, new
Matrix4x4[6]{transformationHsw, transformationMsw,
transformationSsw, transformationH, transformationM,
transformationS});

  }

}
```

- [ ] Zaznacz w inspektorze przy Clock Hand wcześniej przygotowany objekt
- [ ] Dodaj odpowiednie przekształcenia, tak aby duży zegar pokazywał czas i mały zegar pokazywał czas od uruchomienia aplikacji

L-System

W tej części zajęć skupimy się na tworzeniu L-Systemów i wykorzystaniu ich do reprezentacji zjawisk przyrodniczych.

> **Przypomnienie**
> **L-System** to system przepisywania i gramatyka formalna. Składa się z: symboli, które tworzą ciągi znaków; reguł produkcyjnych, które opisują na co należy przepisać dany znak; aksjomatu, czyli początkowego ciągu znaków; i mechanizmu, który tłumaczy ciąg znaków na reprezentacje geometryczną

Bazą dla tych zajęć jest książka Algorithmic Beauty of Plants dostępna za darmo pod linkiem lub w wyższej jakości

Składnia L-Systemów

Projekt zawiera bibliotekę, która interpretuje L-Systemy. Ich definicję pobiera z oddzielnego pliku. Ich składnie opisuję definicja L-Systemu opisującego rozwój bakterii Anabaena znajduje się w projekcie pod ścieżką `Assets\LSystem\Anabaena.txt` i wygląda następująco:

```
#axiom
L
#rules
L->lR
R->Lr
l->L
r->R
#end rules
```

Plik należy zacząć od linii `#axiom`, następnie w następnej linii zamieścić ciąg początkowy. Później pomiędzy liniami `#rules` i `#end rules` umieścić instrukcje przepisywania według zasady:
`<znak przepisywany>-><wynik przepisania>`
każdy znak przed strzałką i po strzałce (z wyjątkiem reguł o których później) jest traktowany jako następny symbol. W przypadku kilku reguł, które dotyczą tego samego symbolu wykona się ta, która jest wyżej w pliku. Między reguły można dodawać komentarze, znakiem komentującym jest `#`. Jeżeli znak nie posiada żadnej reguły, która by go opisywała, to nie jest on zmieniany.

Odpal scenę **LSystemFromFile**, zaznacz **LSystemController** w panelu po prawej. Po lewej w polu **L System Path** wpisz `Assets\LSystem\Anabaena.txt` kliknij **Load File**, by załadować LSystem. Następnie Evaluate, by wykonać przepisanie. W scenie wyświetlą się obiekty reprezentujące symbole a w konsoli wyświetli się wynik przepisania.

Składnia wszystkich rozszerzeń jest zaprezentowana w pliku `SampleLSystem.txt` w tej chwili niektóre reguły mogą byc niezrozumiałe, ale może się on przydać później jako wzorzec.

Turtle Graphics

Turtle Graphics jest metodą tworzenia grafiki komputerowej, wykorzystuje kursor (tytułowego żółwia) wykonujący instrukcje w przestrzeni lokalnej.

L-Systemy można interpretować za pomocą Turtle Graphics, poprzez przypisanie każdemu symbolowi instrukcji jaką ma wykonać żółw. Następnie żółw będzie wykonywał kolejne instrukcje czytając napis od lewej do prawej.

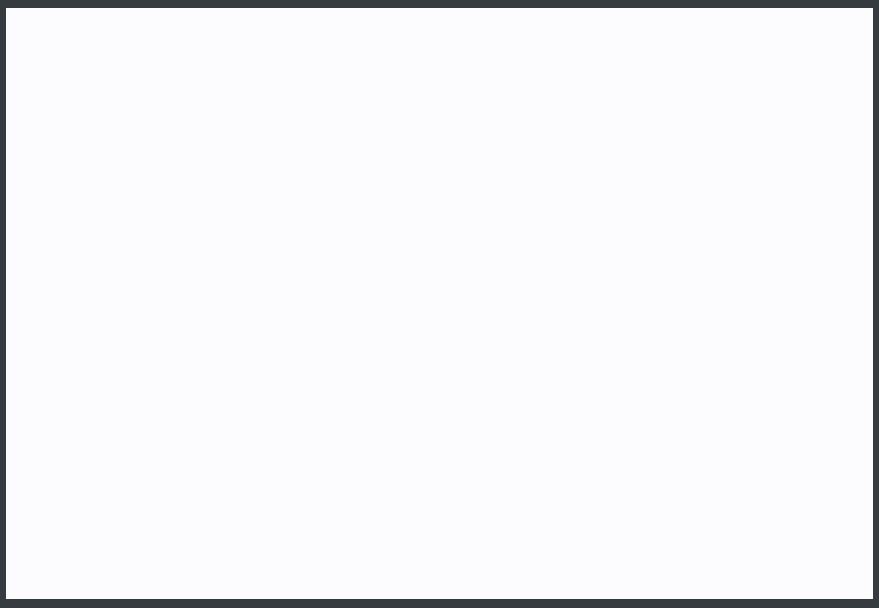
Na początek zaczniemy od prostej reprezentacji, gdzie `+` będzie oznaczał w kierunku zgodnym z ruchem wskazówek zegara o wskazany kąt, natomiast `-` w przeciwnym. Kąt zwyczajowo oznacza się grecką literą δ. Każdy inny symbol będzie oznaczał idź prosto o 1.

Odpal Scenę LSystem2D, załaduj plik `Sierpinski.txt`, ustaw kąt na 60 stopni i wykonaj kilka kroków.

Zadanie

napisz Lsystem, który będzie rysował gwiazdkę kocha

1. Krzywa kocha:

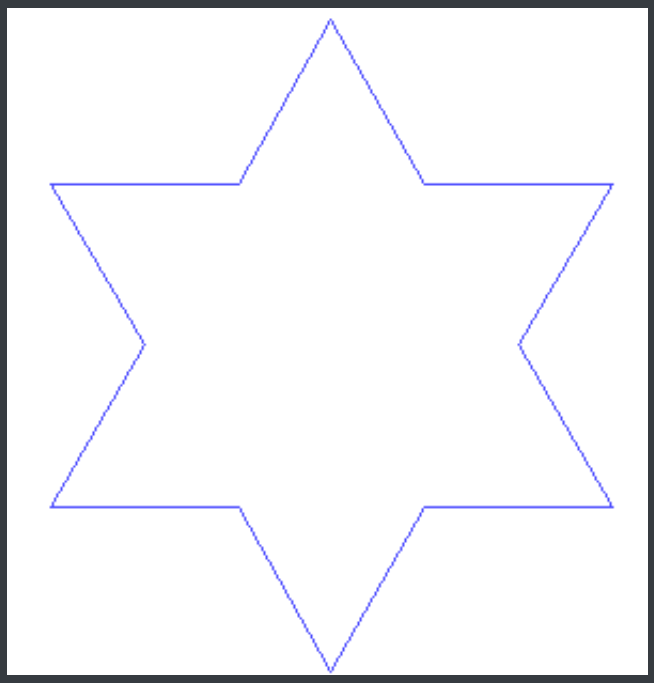


Opis:

1. Podziel linię na 3 równę części
2. Przy środkowej części narysuj równoboczny trójkąt zwrócony na zewnątrz
3. Usuń środkową część pierwotnej lini

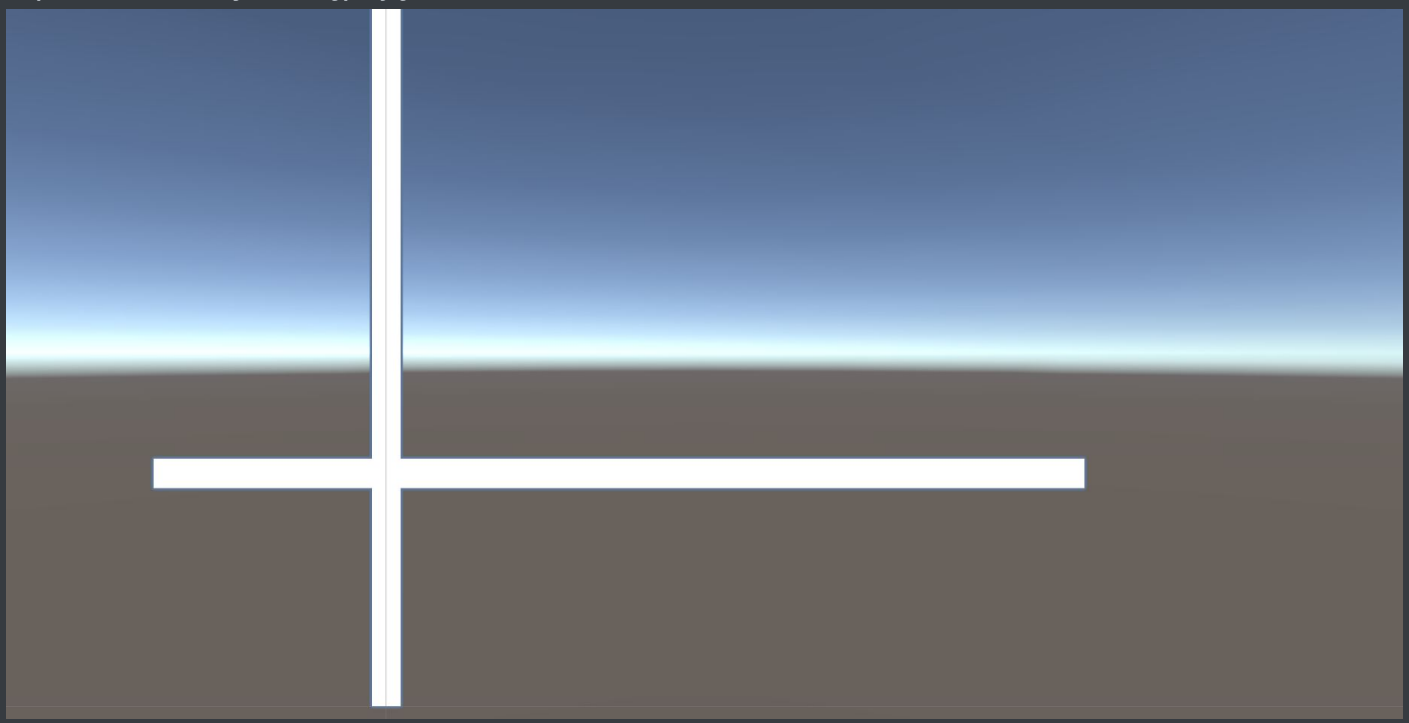
Musimy z jednej lini zrobić 4 nowe, z czego pierwsza i ostatnia idą w tym samym kierunku, a dwie środkowe idą pod innym kątem (podpowiedź: dając dwa razy + lub - możesz zwiększyć kąt)

2. W pierwszym kroku L-Systemu utwórz trójkąt



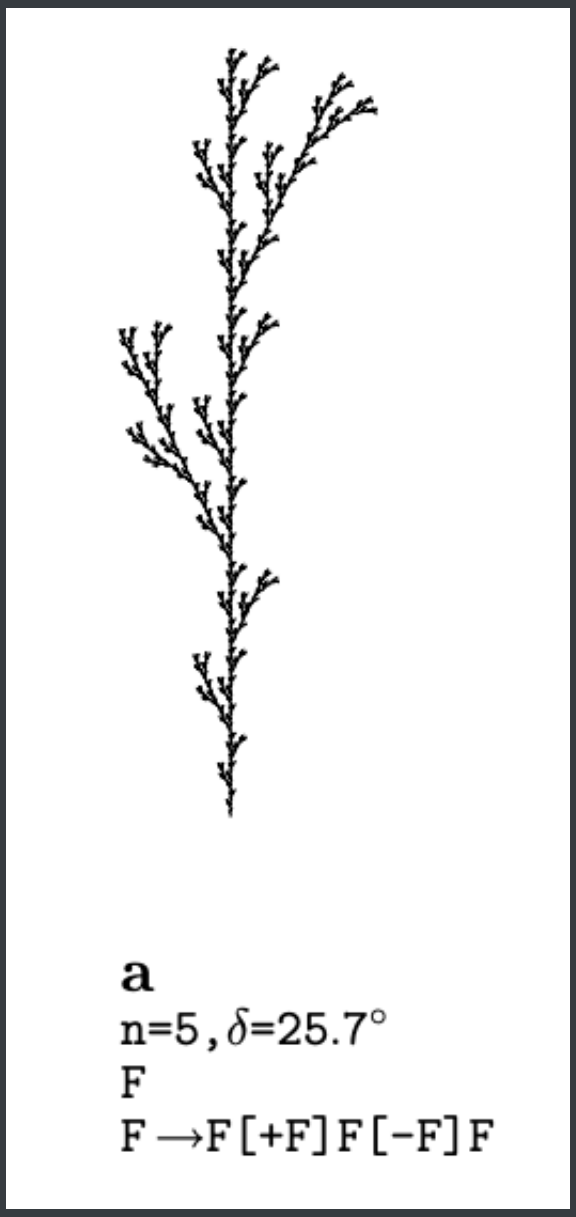
Bracketed L-systems

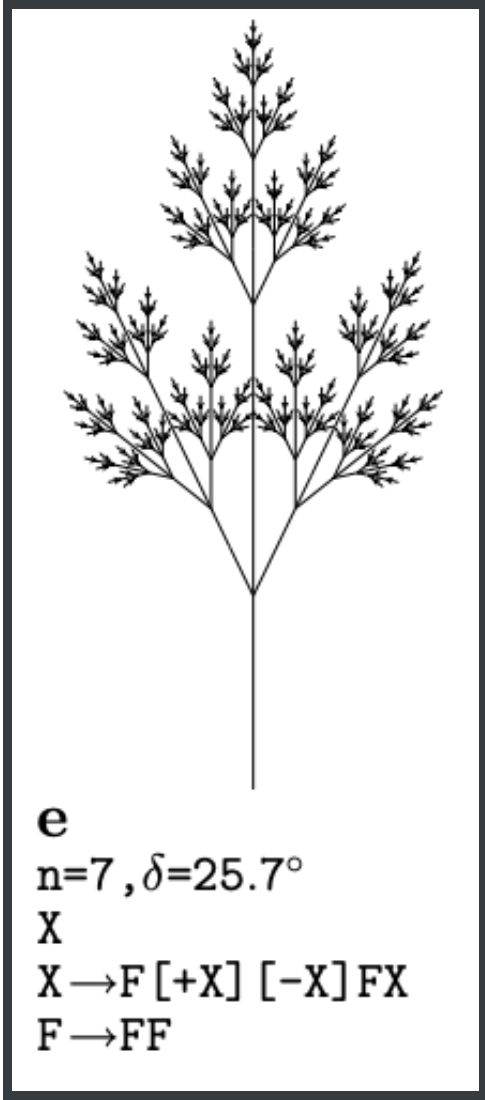
W podstawowej wersji L-Systemy są pojedyńczym ciągiem znaków, by uzyskać możliwość tworzenia rozgałęzień wprowadzamy dwa specjalne znaki `[` oraz `]` pierwszy mówi, żeby zapamiętać obecny stan, drugi oznacza by wrócić do stanu zapamiętanym przy ostatnim znaku `[`. Przykładowo ciąg symboli `F[+FFF][-F]FF` dla $\delta=90$ będzie reprezentowany następująco



Zadanie

Stwórz systemy a i e:





Pisanie własnej interpretacji LSystemów

Wróćmy do sceny LSystemFromFile W tej scenie zamiast kresek pojawiają się figury reprezentujące komórki (czerwona lewa, zielona prawa, niska młoda, wysoka dorosła). Otwórz skrypt AnabeanaTurtle.cs, który odpowiada za rysowanie. Zawiera on klasę `AnabeanaTurtle` dziedziczącą po `TurtleLSystem`. `TurtleLSystem` jest klasą abstrakcyjną, wymaga zdefiniowania funkcji `initLiteralInterpretation`, w której należy opisać jak interpretować symbole.

```
    protected override void initLiteralInterpretation() {
        turtleInterpretation = new Dictionary<string, Func<float[],
Tuple<GameObject, Matrix4x4>>>();
        //turtleInterpretation
        var transformation = Matrix4x4.Translate(new Vector3(0.0f, 0.1f,
0)) * Matrix4x4.Scale(new Vector3 (0.05f, 0.1f, 0.05f));

        turtleInterpretation.Add("+", (float[] args) => new
Tuple<GameObject, Matrix4x4>(null, Matrix4x4.Rotate(Quaternion.Euler(0,
0, -angle))));
        turtleInterpretation.Add("-", (float[] args) => new
Tuple<GameObject, Matrix4x4>(null, Matrix4x4.Rotate(Quaternion.Euler(0,
0, angle))));

        //Wildcard how to represent any other symbol
        turtleInterpretation.Add("*.*", (float[] args) => new
Tuple<GameObject, Matrix4x4>(obj, transformation));
    }
```

Żeby tego dokonać należy uzupełnić słownik `turtleInterpretation`, którego kluczami są opisywane symbole jako stringi. Natomiast wartościami są funkcje, które przyjmują jako argument tablicę parametrów danego symbolu (o tym później) a zwracają Krotkę, której pierwszym elementem jest rysowany obiekt, natomiast drugim transformacja, jaką wykona żółw. Powyższym przykładzie obiekty są czytane z modeli a transformacja zawsze jest taka sama, czyli translacja o wektor (0.1,0,0) i skalowanie o wektor (0.1,0.1,0.1). (Skalowania nie są pamiętane przez żółwia)

Te funkcje są wykorzystywane przez żółwia do interpretacji ciągu symboli. Przykładowo `LRr` zostanie zinterpretowany następująco:

- żółw przesuwa się o 0.1 w osi X umieszcza bigL w punkcie (0.1,0,0)
- żółw przesuwa się o 0.1 w osi X umieszcza bigR w punkcie (0.2,0,0) (ponieważ (0.1,0,0)+(0.1,0,0)=(0.2,0,0))
- żółw przesuwa się o 0.1 w osi X umieszcza bigL w punkcie (0.3,0,0)

Zadanie

Jak wyszukasz w internecie obrazki Anabaeny, zobaczysz, że są one często powykręcane, dodaj do macierz przekształceń obroty w osi Y o losowy kąt pomiędzy -20 a 20 stopni.

Parametryczne L-Systemy

Parametryczne L-Systemy operują na symbolach parametrycznych znakach, czyli takich, które posiadają 0 lub więcej parametrów rzeczywistych. Pozwala to przechowywać różne wewnętrzne stany obiektów. Przykładowo dla modelu Anabeany powyżej rozróżniamy tylko 2 stany, młody i dorosły. Dzięki parametrycznym L-Systemom możemy opisać wiek w sposób bardziej ciągły. Przykładowo poniższy L-System komórki która rośnie i jak osiągnie odpowiedni wiek rozdziela się na dwie młode komórki

```
#axiom
B(1)
#rules
B(a) : a<2 -> B(a+0.1)
B(a) : a>=2 -> B(1)B(1)
#end rules
```

Parametry zapisuje się wewnątrz nawiasów po przecinku. W aksjomacie muszą one mieć wartości liczbowe. Po lewej stronie trzeba nadać parametrom nazwy (mogą mieć one więcej niż jeden znak). Symbole są identyfikowane po nazwie i liczbie znaków. Po dwukropku można podać warunki logiczne jakie musi spełnić symbol. można w tym umieszczać operacje matematyczne, dozwolone jest mnożenie, dzielenie, dodawanie i odejmowanie. podobnie w parametrach po prawej stronie

Zadanie

Napisz dla parametrycznej wersji Anabeny taką interpretację, żeby komórki rosły wraz z wiekiem.