Lekcja 2. Wstęp do języka Lua

Cel lekcji

Celem lekcji będzie zapoznanie się z językiem programowania Lua oraz stworzenie i zaprogramowanie ucieczki przed kulkami.

Ważne informacje dotyczące przebiegu lekcji

Tempo lekcji

Pamiętaj, lekcja to nie są wyścigi, jeśli podczas zajęć odnosimy wrażenie że może zabraknąć nam czasu na realizację całości materiału to nie przyspieszamy tempa prowadzenia zajęć żeby za wszelką cenę zrealizować każdy punkt lekcji. Najważniejsze jest aby uczestnik zrozumiał materiał, nowe zagadnienia, które pojawiają się na danej lekcji. W zależności od prowadzonego kursu: pominięcie zadania, uproszczenie projektu, zrezygnowanie z jakiejś funkcjonalności będzie lepszym rozwiązaniem niż narzucanie szybkiego tempa i chaotycznego sposobu prowadzenia lekcji.

Osoby spóźnione

Lekcje zaczynamy punktualnie, nie czekamy 10 minut na spóźnialskich. Osoby, które dotrą na zajęcia spóźnione staramy się sprawnie wprowadzić w temat i cel danej lekcji. W zależności od czasu spóźnienia, osobę taką wprowadzamy poprzez krótką rozmową lub udostępnienie projektu z aktualnym stanem.

Początek i podsumowanie zajęć

Na początku lekcji powinno odbyć się przypomnienie materiału z poprzednich zajęć, należy zadać pytania, przypomnieć uczestnikom zagadnienia, popytać o projekt, jego działanie. Na koniec lekcji podsumowujemy ją. Bazujemy na

pytaniach podanych w konspekcie, ale oczywiście możemy też zadać dodatkowe, własne dostosowane do danej grupy.

Agenda

1. Pytania początkowe
2. Tworzenie nowego projektu
3. Tworzenie nowego skryptu i wstęp do języka Lua
4. Ucieczka przed kulkami
5 Zadanie dodatkowe
6. Pytania końcowe i podsumowanie lekcji

Koniecznie do wykonania przed zajęciami

Przed rozpoczęciem zajęć pamiętaj o tym aby zapoznać się z instrukcją instalacji i obsługi programów. Sprawdź również jakie są najczęstsze problemy w działaniu środowiska. Dzięki temu, gdy uczestnik ma problem możesz zaproponować szybkie rozwiązanie. Link do wszelkich potrzebnych elementów znajdziesz poniżej:

https://drive.google.com/drive/folders/1vaEMCR5lGl8NCKJgiErSM7KQrYO7IDb7

W konspekcie znajdziesz zdania zapisane *kursywą* są to wskazówki dla Ciebie, jakie zadać pytanie, co dopowiedzieć, korzystaj z tego, parafrazuj, lekcja na pewno będzie ciekawsza.
NIE sprowadzaj się do tego że napiszesz linijkę kodu i przejdziesz dalej bez wytłumaczenia za co odpowiada, zawsze można coś dodatkowo powiedzieć. Konspekt jest, rozpisany szczegółowo, natomiast jesteś zobowiązany/a przerobić go w całości wcześniej przed zajęciami. Przerabiając sobie projekt z zajęć krok po kroku zgodnie z konspektem będziesz mieć pełną kontrolę, swobodę w tłumaczeniu i nic Cię nie zaskoczy podczas lekcji. Nasze konspekty nie są idealne (chociaż cały czas staramy się żeby takie były ☺), przerabiając je wcześniej możesz zlokalizować ewentualne błędy i je zgłosić. Czasami błędy będą wynikać z aktualizacji danej platformy więc na bieżąco musimy reagować.

1.Pytanie początkowe

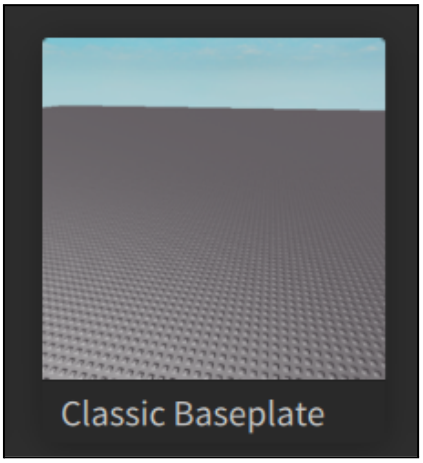
- Jakie znacie języki programowania?

- Czy ktoś z was wcześniej programował?

- Czy próbowaliście tworzyć skrypty w Roblox Studio?

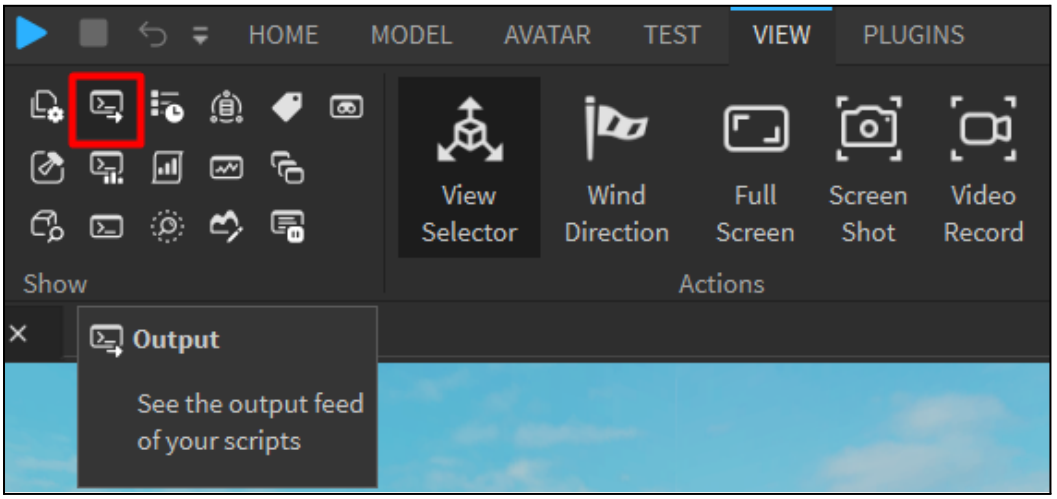
2.Tworzenie nowego projektu

Tworzymy nowy projekt z wykorzystaniem szablonu **Classic Baseplate**.

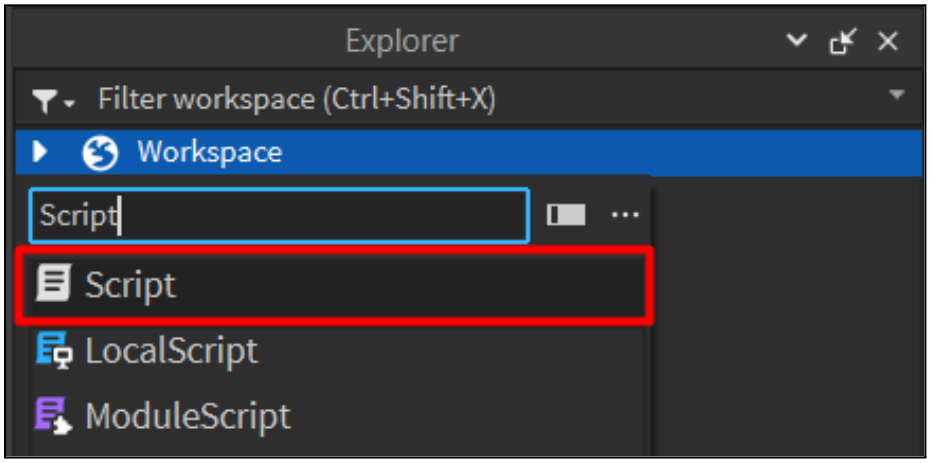


3.Tworzenie nowego skryptu i wstęp do języka Lua

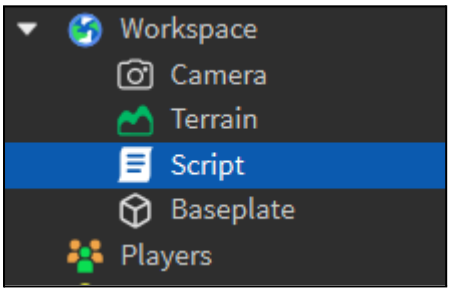
Przed rozpoczęciem programowania należy włączyć okno **Output**, będzie ono przydatne do sprawdzania czy program działa poprawnie oraz czy nie występują błędy.



Dodajemy nowy skrypt *(Script)* do **Workspace**.



Tak powinien wyglądać dodany nowy skrypt do **Workspace**:



Dodatkowe informacje:

Script – podstawowy skrypt wykonywany na serwerze, oznacza to, że czynności wykonywane przez ten skrypt, będą widoczne przez innych graczy, będziemy go wykorzystywać na dzisiejszych zajęciach.

LocalScript – ten skrypt wykonywany jest na kliencie, czyli np. na komputerze gracza, tego skryptu użyjemy na późniejszych zajęciach.

ModuleScript – pisanie skryptów modułowych przydaje się przy wielokrotnym wykorzystaniu kodu, tego rodzaju skrypty są wykorzystywane przez zaawansowanych programistów.

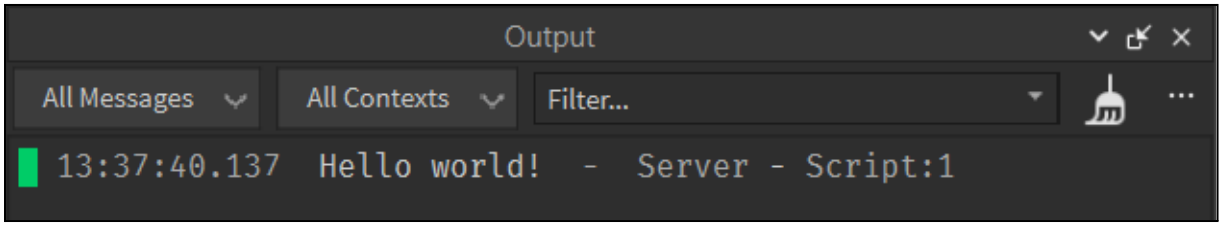
Edycję skryptu uruchamiamy poprzez dwukrotne naciśnięcie lewego przycisku na myszce na element **Script**, który znajduje się w Explorer.

Po uruchomieniu programu mamy klasyczny program dla początkującego programisty.

```
print("Hello world!")
```

Uruchomienie projektu spowoduje wyświetlenie tego tekstu w **Output**.

Na początku mamy informacje o czasie uruchomienia programu, następnie komunikat, który wyświetla się dzięki funkcji **print()**, która pokazuje dowolny ciąg znaków w polu wyjściowych **Roblox Studio**, kolejną informacją jest miejsce, w którym wykonuje się program, w tym przypadku jest to **Server**, ostatni komunikat dotyczy linii kodu dzięki, której wykonał się ten komunikat.



Wracamy teraz do elementu **Script** i modyfikujemy program. Zadajemy pytanie uczestnikom czy wiedzą co to jest **zmienna** i do czego służy (dzieci powinny je kojarzyć z innych semestrów np. ze scratcha, jeżeli nie będą w stanie wyjaśnić, opowiadamy czym są zmienne i do czego służą).

Tworzymy zmienną **imie** a następnie w kodzie programu do wartości zmiennej każdy przypisuje swoje imię. Możemy również wyjaśnić budowę zmiennej na poniższym przykładzie:

```
local imie = "Przemek"
```

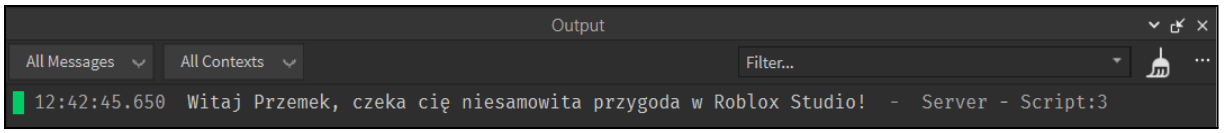
Słówko **local** oznacza, że dana zmienna jest dostępna tylko dla danego fragmentu kodu. Brak słówka **local** przy tworzeniu zmiennej spowoduje, że dana zmienna będzie globalna, czyli będzie dostępna w wielu miejscach.

Przy nazwach zmiennych nie korzystamy z polskich znaków typu „ą, ę, ó” itp. W tym przypadku wartość zmiennej jest typu (string) czyli tekstowy, jednak możemy zauważyć, że w języku Lua nie musimy określać typu danych, podobnie jak np. w Pythonie.

Kropki użyte w funkcji print(), służą do łączenia łańcuchów znaków to oznacza, że możemy wykorzystać zmienne do wplatania ich w teksty.

Kod programu:

```
local imie = "Przemek"
print("Witaj " .. imie .. ", czeka cię niesamowita przygoda w Roblox Studio!")
```



W tym miejscu możemy zapytać uczestników, czy widzą zastosowanie w grach gdzie możemy wpleść zmienne do tekstu.

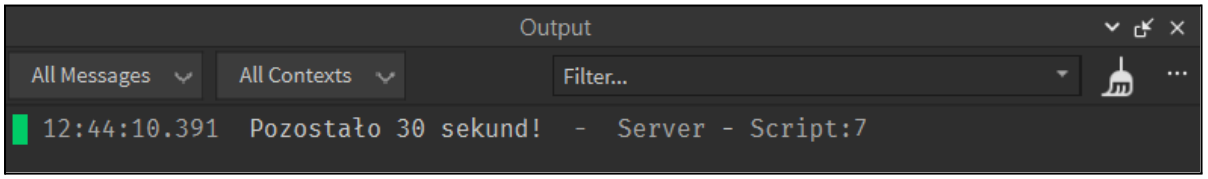
Jakie zastosowanie w grach ma wplatanie zmiennych do tekstu?

Przykładowe zastosowanie przy odliczaniu czasu: *Pozostało 30 sekund!*

Następnie dopisujemy kod programu i sprawdzamy działanie.

Kod programu:

```
local czas = 30
print("Pozostało " .. czas .. " sekund!")
```

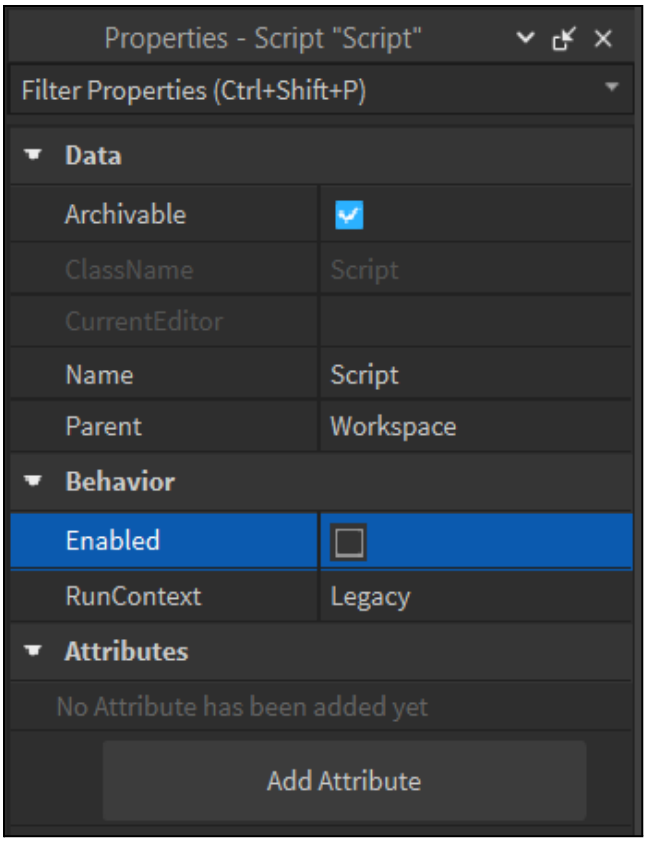


Po zakończeniu ćwiczenia możemy zakomentować linijki kodu lub wyłączyć skrypt poprzez odznaczenie opcji **Enabled** we właściwościach skryptu.

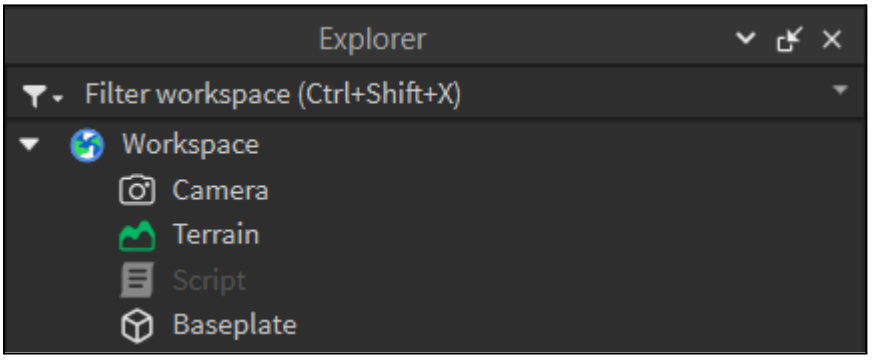
Dodawanie komentarzy

```
-- local imie = "Przemek"
-- print("Witaj " .. imie .. ", czeka cię niesamowita przygoda w Roblox Studio!")

-- local czas = 30
-- print("Pozostało " .. czas .. " sekund!")
```

Dezaktywowanie skryptu



Script w **Explorer** powinien stać się wyszarzony.



Funkcje

W praktycznym projekcie będziemy również korzystać z funkcji. Czy ktoś z was wie czym są funkcje w programowaniu? Funkcje to tak naprawdę zestaw instrukcji do wykonania, które można wykorzystać wielokrotnie w skrypcie. Funkcje można porównać w realnym życiu na przykład do przepisu na przygotowanie jedzenia. Przykładowo możemy nauczyć robota jak przygotować jedzenie.

Przykład do pokazania i omówienia uczestnikom.

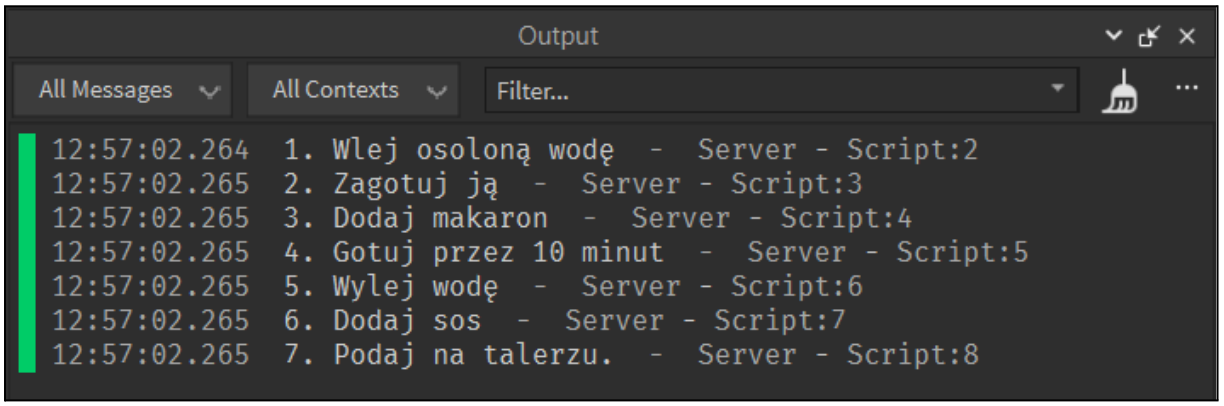
```
local function przygotujPosilek()
    print("1. Wlej osoloną wodę")
    print("2. Zagotuj ją")
    print("3. Dodaj makaron")
    print("4. Gotuj przez 10 minut")
    print("5. Wylej wodę")
```

```
    print("6. Dodaj sos")
    print("7. Podaj na talerzu.")
end
```

Funkcje tak naprawdę nie będą działać dopóki nie zostaną wywołane.

```
-- Powiedz programowi, aby uruchomił funkcję (czyli przygotował posiłek)
przygotujPosilek()
```

*Następnie pokaż co pojawiło się w oknie **Output**.*



Istnieją gotowe funkcje np. print(), wait() itd. Często również nazwy funkcji zapisujemy według zasady camelCase, przy czym pierwsza litera jest mała, a następne słowa pisane są wielką literą. Zasada ta istnieje w wielu językach programowania, nazwa wywodzi się z faktu, że przy jej zastosowaniu wielkie litery w połączonych ze sobą słowach przypominają kształtem garby wielbłąda.

Funkcje można łączyć ze zdarzeniami. Dzięki temu funkcja wykona się kiedy zostanie wywołane konkretne zdarzenie.

Zdarzenia są specjalnymi sygnałami, które aktywują się podczas gry.

Przykładowe zdarzenia:
- Dotknięcie przedmiotu
- Kliknięcie wskaźnikiem myszki na przedmiot

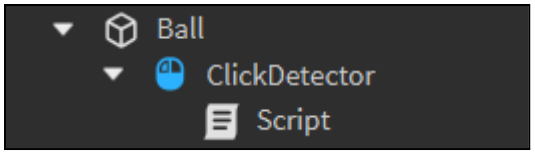
4. Ucieczka przed kulkami

Przyszedł czas na praktyczny projekt, naszym zadaniem będzie wydostanie się z pojemnika z kulkami. Kulki będą losowo spadać z góry, a my będziemy mieć ograniczony czas, jeżeli czas się skończy, kulki wygenerują się na nowo, a my spadniemy na dół (jeżeli będziemy podczas wspinaczki), dodatkową pomocą będzie możliwość usuwania pojedynczych kulek.

Pokazujemy gotowy projekt i tłumaczymy jak będzie wyglądać rozgrywka:
https://cutt.ly/MP0wqxh

Udostępniamy projekt starter: https://cutt.ly/oP0wtRt

Omawiamy **ClickDetector**

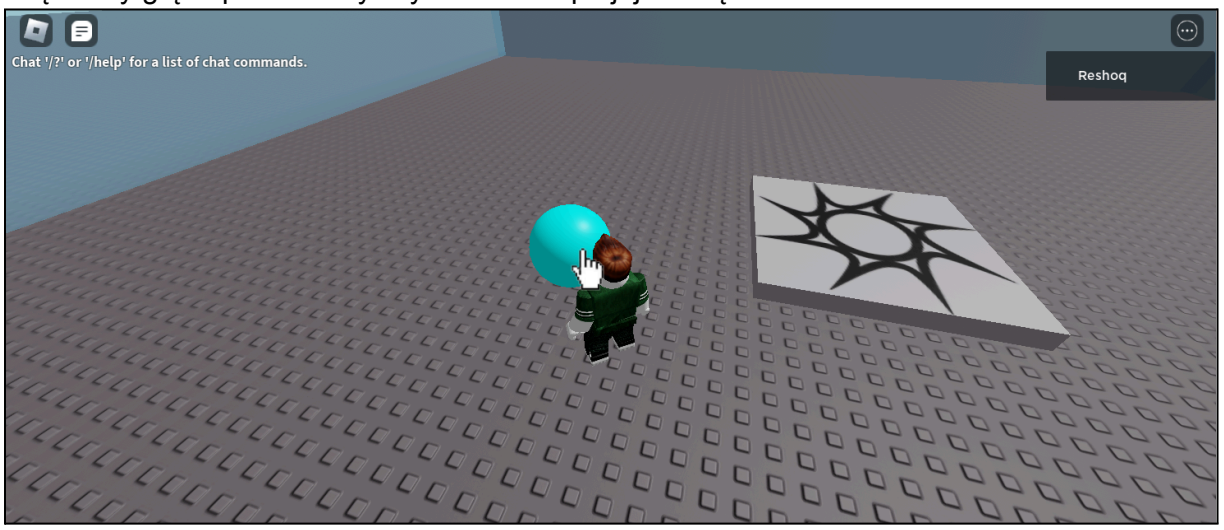


ClickDetector pozwala na wykrywanie podstawowych zdarzeń myszy takich jak:
- kliknięcie lewym przyciskiem myszy (zdarzenie **MouseClick**)
- kliknięcie prawym przyciskiem myszy (zdarzenie **RightMouseClick**)
- najechanie myszką na obiekt (zdarzenie **MouseHoverEnter**)
- odsunięcie myszką z obiektu (zdarzenie **MouseHoverLeave**)

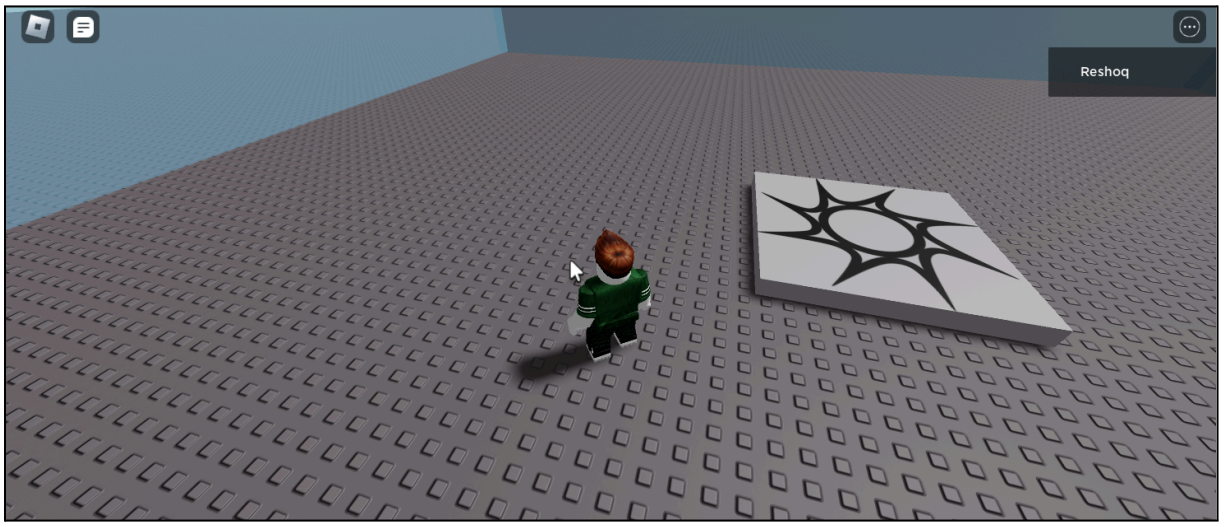
Tworzymy program do **Script** dołączonego do obiektu **ClickDetector**:

```
-- Odwołanie do obiektu ClickDetector
local clickDetector = script.Parent
-- Odwołanie do obiektu Ball
local ball = clickDetector.Parent

-- Funkcja Destroy() wykonująca program
function Destroy()
    -- Zniszczenie obiektu Ball
    ball:Destroy()
end
-- Kiedy obiekt ClickDetector wykryje kliknięcie, wykonaj funkcję Destroy()
clickDetector.MouseClick:Connect(Destroy)
```

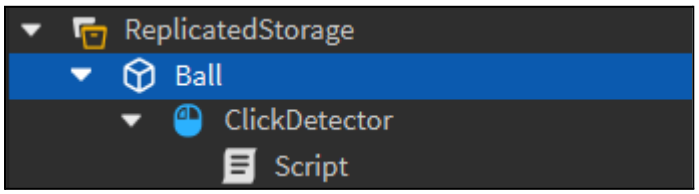
Czas na testy!
Włączamy grę i sprawdzamy czy kula znika po jej kliknięciu.



Kula znika!

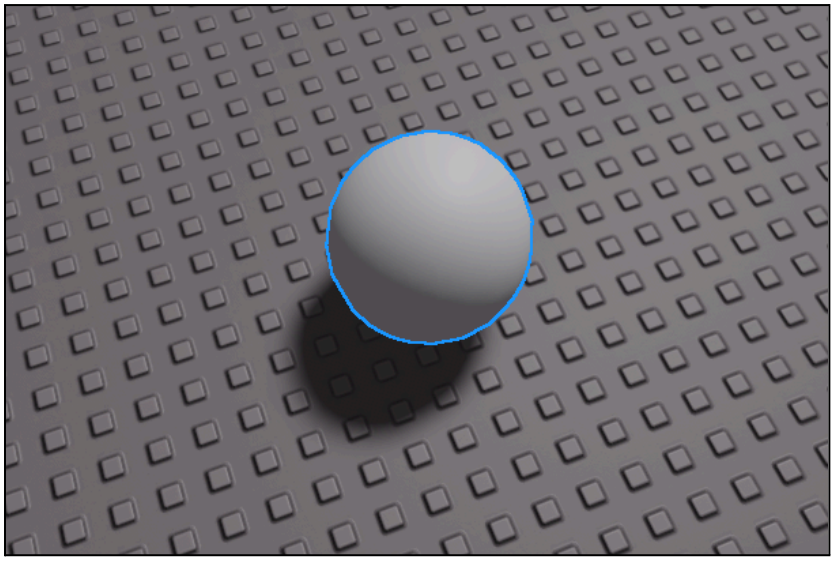


Jeżeli wszystko działa przenosimy **Ball** to **ReplicatedStorage**.



Najlepiej uczyć się na praktycznych przykładach:

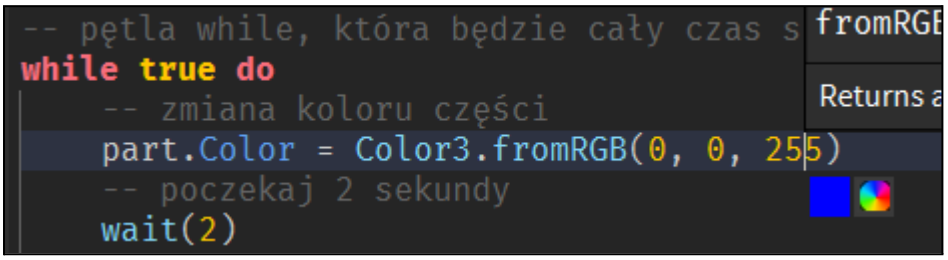
Wspólnie dodajcie dodatkową kulę do gry (**HOME/MODEL** > **Part** > **Sphere**), a następnie należy umieścić w niej nowy **Script**.



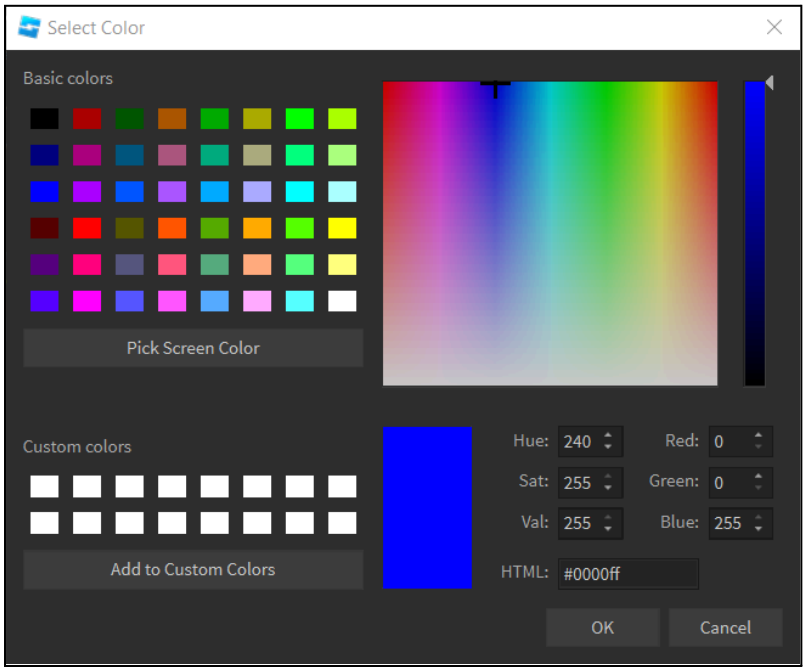
Tworzymy wspólnie skrypt, pamiętając aby dokładnie wytłumaczyć instrukcję **while** (np. nawiązując do poprzednich semestrów).

```lua
-- utworzenie zmiennej dla rodzica skryptu
local part = script.Parent

-- pętla, która będzie cały czas się powtarzać, ponieważ warunek jest prawdą
while true do
    -- zmiana koloru części
    part.Color = Color3.fromRGB(0, 0, 255)
    -- poczekaj 2 sekundy
    wait(2)
    part.Color = Color3.fromRGB(255, 0, 0)
    -- poczekaj 2 sekundy
    wait(2)
end
```

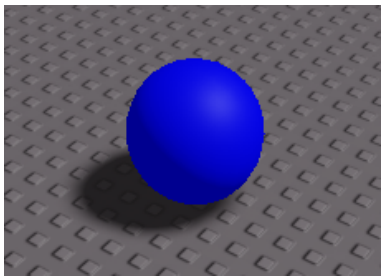
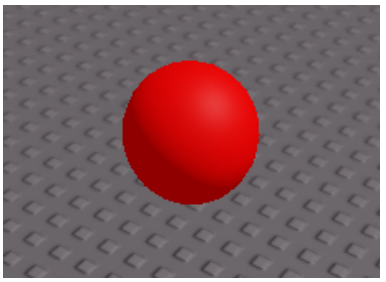
Wskazówka:
Wybór innych kolorów można wykonać w edytorze poprzez kliknięcie w liczby w nawiasie oraz następnie gdy pojawi się paleta kolorów klikamy w nią.



Po otwarciu okna **Select Color** możemy wybrać dowolny kolor, który nam się podoba.



Kiedy program będzie gotowy, uruchamiamy projekt i sprawdzamy czy kula zmienia kolory.

	Po uruchomieniu gry kula powinna zmieniać kolor co 2 sekundy.	

Następnie zmienimy warunek w pętli while na false, co będzie oznaczało, że warunek jest fałszywy i instrukcje znajdujące się wewnątrz pętli nie zostaną wykonane.

```
while false do
```

Pytanie do uczniów: *Czy kula dalej zmienia kolory?*

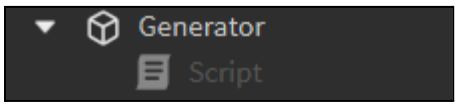
Oprócz niekończącego się powtarzania fragmentów programu, możemy również sterować ile razy dany fragment kodu powinien się wykonać poprzez wprowadzenie dodatkowej zmiennej oraz operatorów porównania.

Przykłady:
== jest równe
~= nie jest równe
< lub > mniejsze lub większe
<= oraz >= mniejsze lub równe oraz większe lub równe

```
-- utworzenie zmiennej dla rodzica skryptu
local part = script.Parent
local i = 1
```

```
-- pętla while, która sprawdza czy wartość zmiennej i jest mniejsza lub równa 5
while i <= 5 do
    -- zmiana koloru części
    part.Color = Color3.fromRGB(0, 0, 255)
    -- poczekaj 2 sekundy
    wait(2)
    part.Color = Color3.fromRGB(255, 0, 0)
    -- poczekaj 2 sekundy
    wait(2)
    -- zwiększ wartość zmiennej i o jeden
    i += 1
end
```

Po omówieniu pętli while wchodzimy do skryptu dołączonego do **Generator** i omawiamy go.

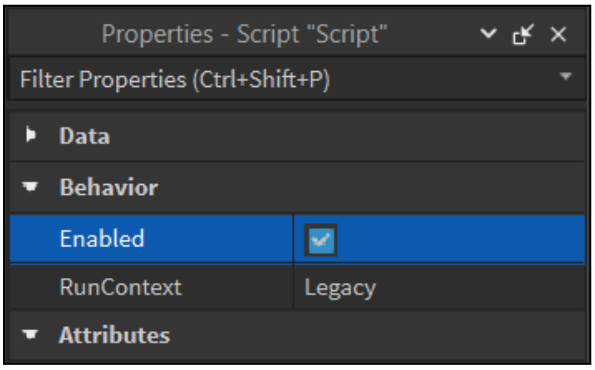


Program do **Script** dołączonego do obiektu **Generator**:

```
-- Odwołanie się do ReplicatedStorage
local replicatedStorage = game.ReplicatedStorage
-- Oczekiwanie na obiekt Ball
local ball = replicatedStorage:WaitForChild("Ball")
-- Odwołanie do rodzica skryptu czyli obiektu Generator
local generator = script.Parent

-- Utworzenie zmiennych
local newBall
local size
local counter
local x
local z
local i

-- Funkcja Generate wykonuje ciąg programów
function Generate()
    -- Klonujemy obiekt Ball i przypisujemy do zmiennej newBall
    newBall = ball:Clone()
    -- math.random służy do losowania liczb
    -- W tym przypadku jest to wartość, która będzie ustawiana jako wielkość kuli
    size = math.random(4,40)
    -- Ustawienie wielkości kuli
    newBall.Size = Vector3.new(size,size,size)
    -- Losowanie liczb x oraz z, które zostaną dodane do pozycji sklonowanej kuli
    x = math.random(-10,10)
    z = math.random(-10,10)
    -- Ustawienie pozycji kuli z przesunięciem
    newBall.Position = generator.Position + Vector3.new(x,0,z)
    -- Przypisanie generatora jako rodzic nowej kuli
    newBall.Parent = generator
```

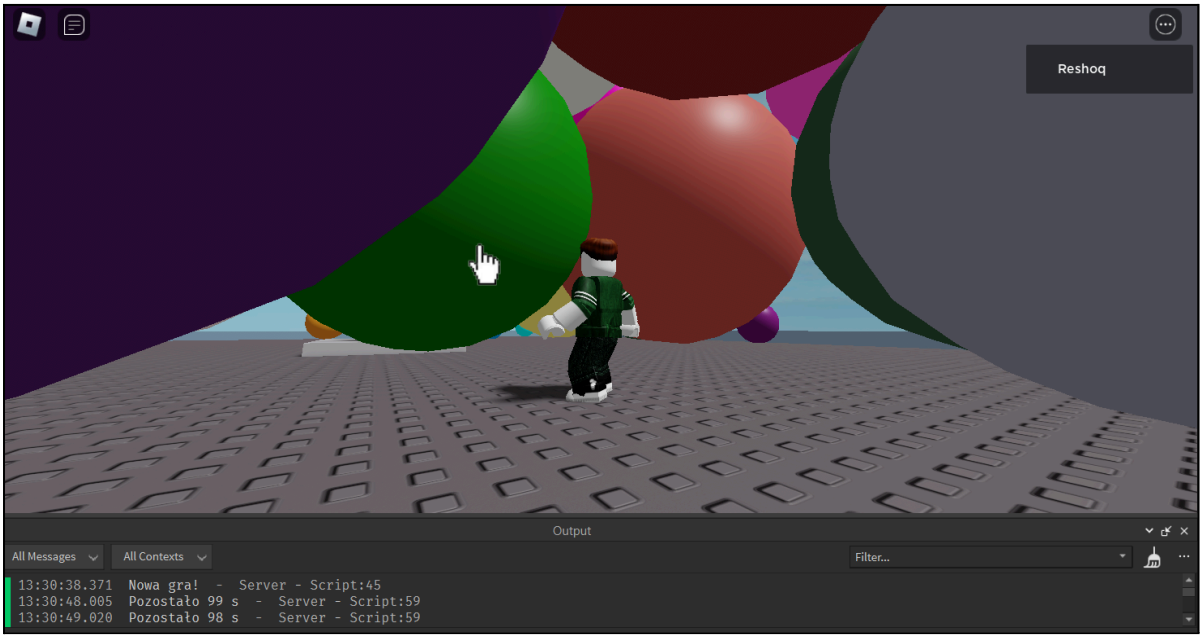
```
	-- Ustawienie losowego koloru
	newBall.BrickColor = BrickColor.Random()
end

-- Nieskończona pętla
while true do
	-- Ustawienie wartości startowych zmiennych pomocniczych
	i = 1
	-- Ustawienie wartości licznika na 100
	counter = 100
	-- Poczekaj 10 sekund
	wait(10)
	-- Wydrukuj
	print("Nowa gra!")
	-- Utworzenie 40 kul
	while i <= 40 do
		Generate()
		-- Poczekaj 0.2 sekundy
		wait(0.2)
		i += 1
	end
	-- Dodawanie kolejnych kul co jedną sekundę
	while counter  >= 1 do
		Generate()
		wait(1)
		-- zmniejsz wartość zmiennej counter o jeden
		counter -= 1
		print("Pozostało " .. counter .. " s")
	end
	-- Wydrukuj
	print("Koniec czasu!")
	-- Usuń wszystkie dzieci,
	-- czyli wszystkie kule które zostały dodane do Generatora
	generator:ClearAllChildren()
end
```

Po omówieniu programu włączamy skrypt w **Generator** poprzez zaznaczenie właściwości **Enabled**.

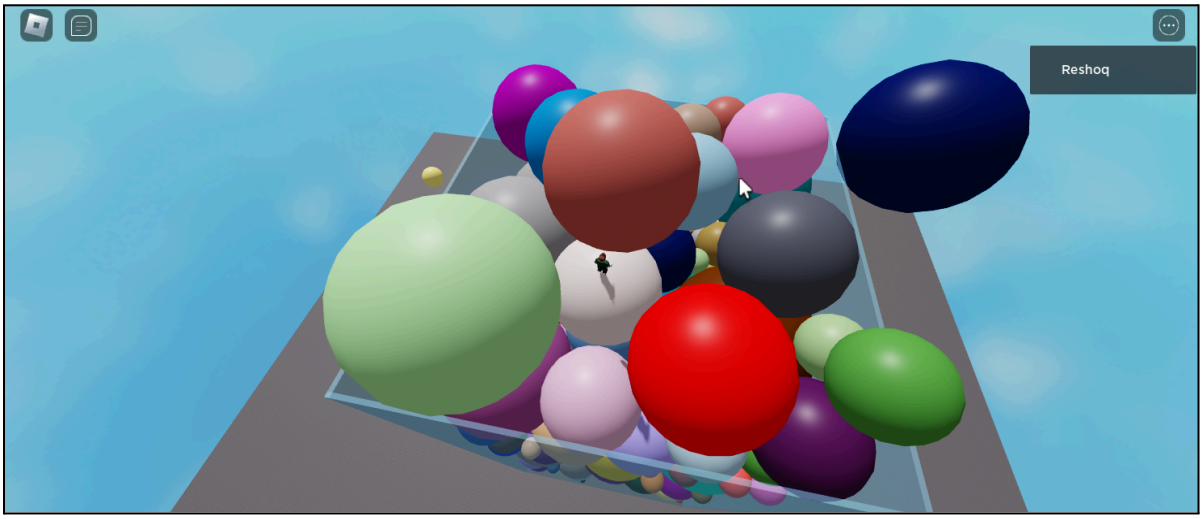


Czas na testy!

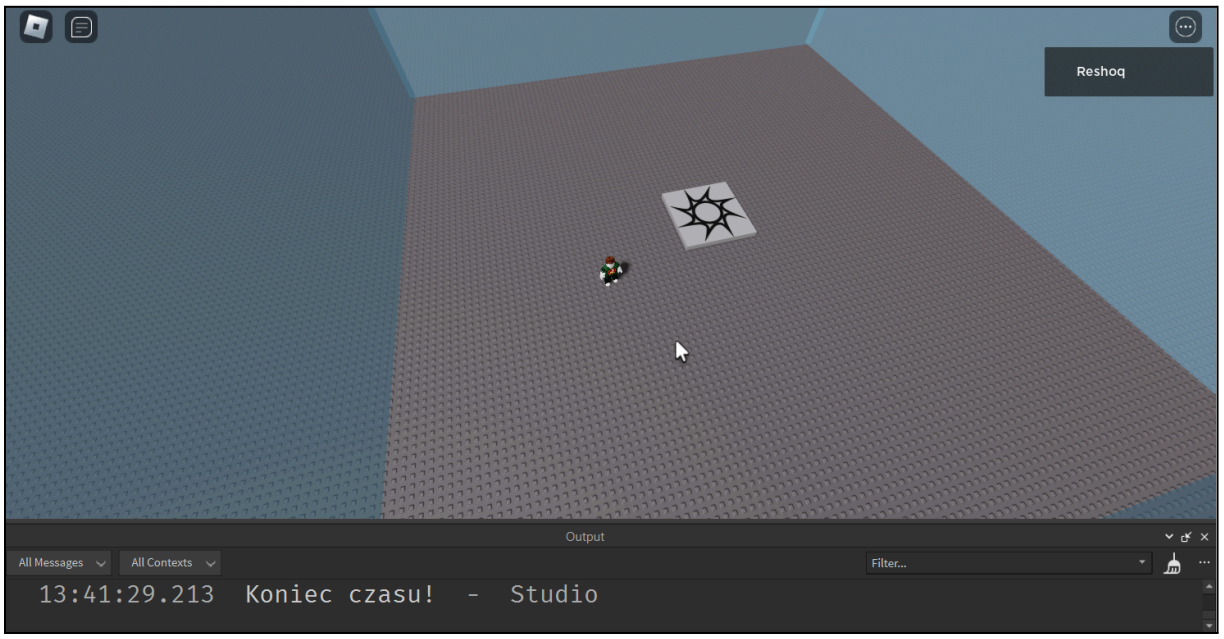
Po uruchomieniu gry i odczekaniu 10 sekund (czas ten może być różny, ze względu na czas załadowania projektu), kule powinny zostać wygenerowane oraz powinna być możliwość ich usuwania.



Próbujemy wydostać się z pudełka!



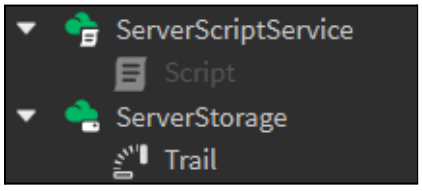
Jeżeli nie zdążymy kule zostaną usunięte.



Udało się!

5. Zadanie dodatkowe – ozdobna wstążka - Trail

Omawiamy **Script** do **ServerScriptService** oraz **Trail** w **ServerStorage**



Możemy dostosować właściwości **Trail** według własnego uznania, przykładowe ustawienia:

Appearance	
Brightness	1
Color	<ColorSequence>
FaceCamera	☐
LightEmission	0
LightInfluence	0
Texture	
TextureLength	1
TextureMode	Stretch
Transparency	0.5
Data	
Emission	
Enabled	☑
Lifetime	1
MaxLength	0
MinLength	0
WidthScale	2

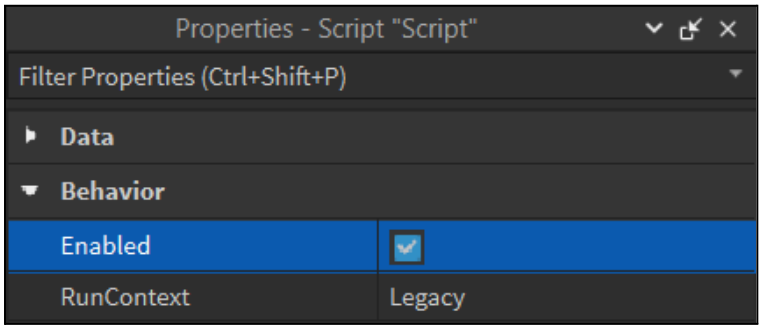
Przykładowy **ColorSequence**

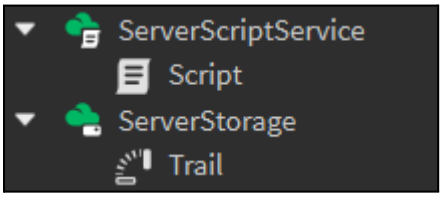
Program do **Script** w **ServerScriptService**

```
game.Players.PlayerAdded:Connect(function(player)
	-- Kiedy charakter zostanie dodany połącz się z funkcją
	player.CharacterAdded:Connect(function(character)
		-- Odwołanie się do obiektu HumanoidRootPart
		-- Który występuje w modelu character
		local r = character.HumanoidRootPart
		-- Sklonowanie obiektu Trail
		local t = game.ServerStorage.Trail:Clone()
		-- Stworzenie obiektu Attachment do określenia zerowej pozycji
		local a0 = Instance.new("Attachment", r)
		-- Ustawienie pozycji obiektu Attachment
		a0.Position = Vector3.new(-0.7, 0, 0)
		-- Stworzenie obiektu Attachment do określenia pierwszej pozycji
		local a1 = Instance.new("Attachment", r)
		-- Ustawienie pozycji obiektu Attachment
```

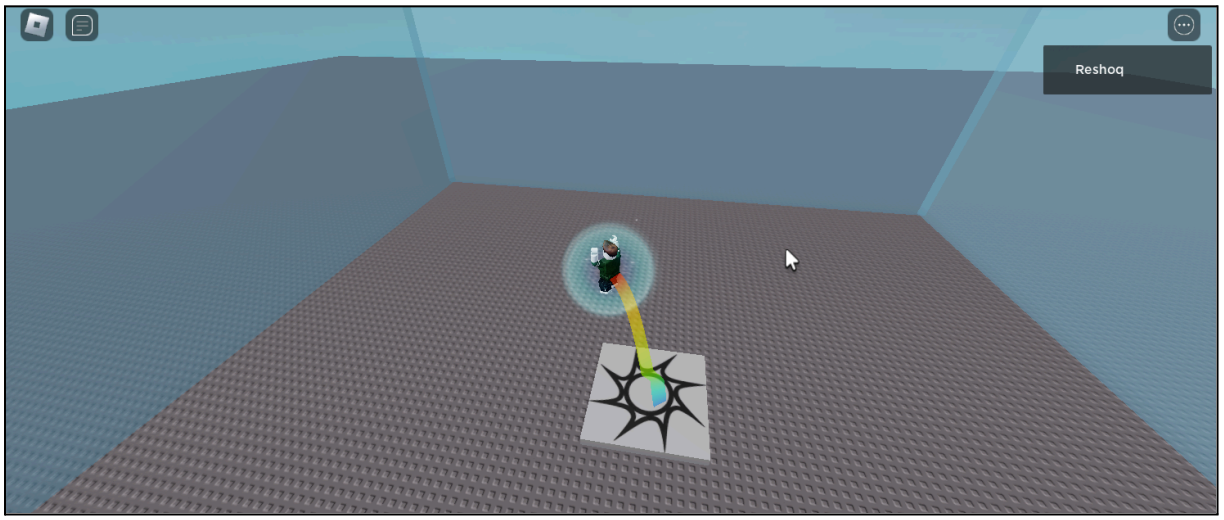
```
            a1.Position = Vector3.new(0.7, 0, 0)
            -- Przypisanie obiektu Attachment do właściwości obiektu Trail
            t.Attachment0 = a0
            -- Przypisanie obiektu Attachment do właściwości obiektu Trail
            t.Attachment1 = a1
            -- Ustawienie rodzica Trail na model character
            t.Parent = r
        end)
end)
```

Po omówieniu programu włączamy skrypt w **ServerScriptService** poprzez zaznaczenie właściwości **Enabled**.





Efekt końcowy!



6. Pytania końcowe i podsumowanie lekcji.

Pytania końcowe:
1. Czego dzisiaj się nauczyliście?
2. Jak odwołać się do rodzica skryptu? *(script.Parent)*
3. Jak działa pętla while?
4. Do czego służy zdarzenie MouseClick() i jak możemy wykorzystać je w grach?
(Do wykrycia kliknięcia na obiekt. Możemy je wykorzystać przykładowo do otwierania drzwi w pomieszczeniu).