

Systemy animacji komputerowej (SAK)

Marek Załęski (id.89264)

Laboratorium 1

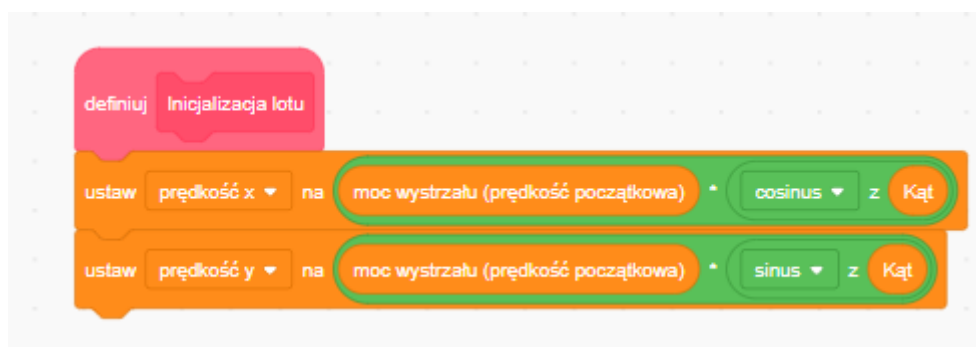
Zadanie A: Symulacja Rzutu Ukośnego (Fizyka)

Całą logikę rzutu ukośnego podzieliłem na trzy niezależne moduły. Dzięki temu kod jest czytelny, a każda funkcja odpowiada za konkretny etap symulacji.

1. Inicjalizacja lotu (Przygotowanie danych)

Zanim obiekt ruszy, muszę przeliczyć parametry ustawione na suwakach na wartości zrozumiałe dla silnika gry.

- **Co tu się dzieje:** Ten blok pobiera Moc wystrzału oraz Kąt i rozbija je na dwie składowe prędkości: poziomą (prędkość x) oraz pionową (prędkość y).
- **Matematyka:** Wykorzystuję tu funkcje trygonometryczne. Prędkość poziomą obliczam z cosinusa, a pionową z sinusa kąta wystrzału. To gwarantuje, że niezależnie od ustawionego kąta, całkowita energia strzału jest zgodna z tym, co wybrał użytkownik.

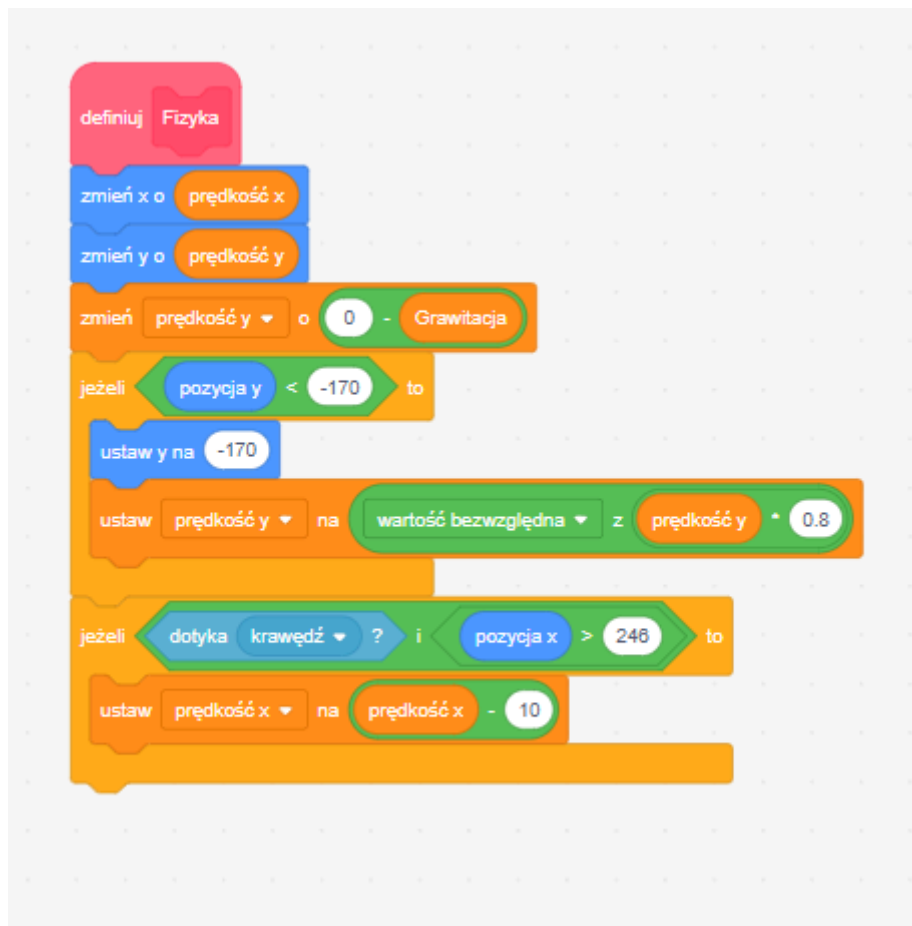


Rysunek 1. Inicjalizacja lotu. Wykonanie własne w Scratch.

2. Własny blok: Fizyka (Obliczenia w czasie rzeczywistym)

To jest "mózg" symulacji, który wywoływany jest w każdej klatce animacji.

- **Ruch i Grawitacja:** Skrypt najpierw przesuwa duszka, a potem modyfikuje jego prędkość pionową o wartość zmiennej Grawitacja. To właśnie tutaj realizowane jest wymaganie $v_y = v_y - g$.
- **Inteligentne Odbicie:** Zastosowałem mechanizm detekcji kolizji z podłożem ($y < -170$).
- **Utrata energii:** Zamiast prostego odbicia, użyłem mnożnika 0.8. Powoduje to, że po każdym uderzeniu o ziemię piłka traci część prędkości, co wygląda bardzo naturalnie i zapobiega "wiecznemu" skakaniu obiektu.

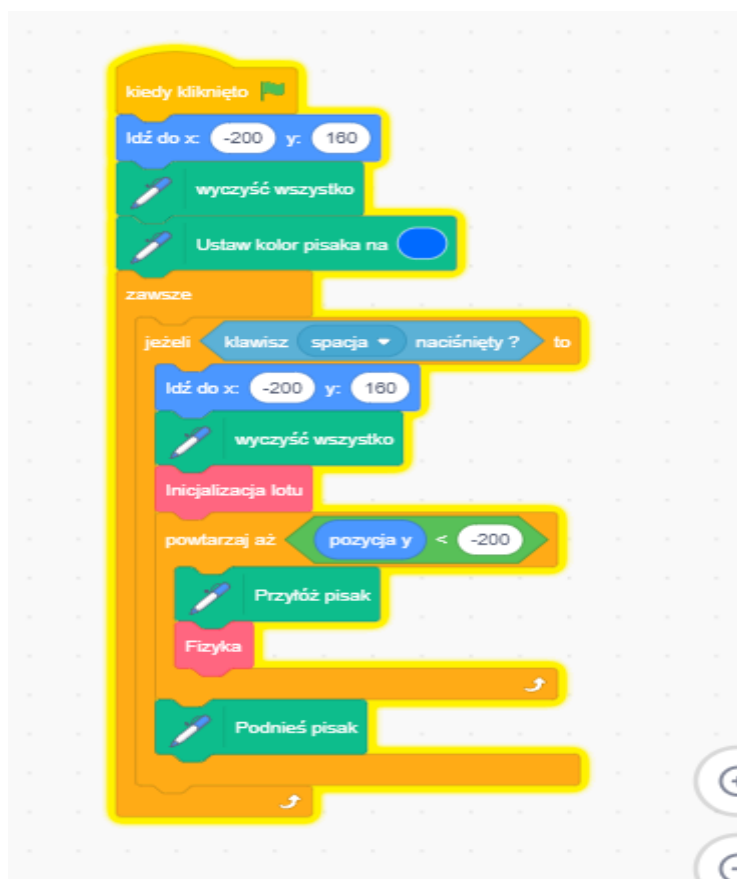


Rysunek 2. Fizyka. Opracowanie własne w Scratch.

3. Główna pętla sterująca (Zarządzanie czasem)

Ta sekcja odpowiada za to, kiedy symulacja ma wystartować i jak długo ma trwać.

- **Logika:** Program czeka na sygnał od użytkownika (np. naciśnięcie klawisza spacji).
- **Przebieg:** Po kliknięciu, skrypt najpierw resetuje pozycję duszka do punktu startowego, wywołuje blok Inicjalizacja lotu, a następnie w pętli powtarzaj aż wykonuje blok Fizyka.
- **Wizualizacja:** Wewnątrz tej pętli włączyłem również obsługę **Pióra**, dzięki czemu tor lotu jest rysowany na ekranie w czasie rzeczywistym, tworząc czytelną parabolę.



Rysunek 3. Główna pętla sterująca. Opracowanie własne w Scratch.

Wynik przeprowadzonej symulacji :



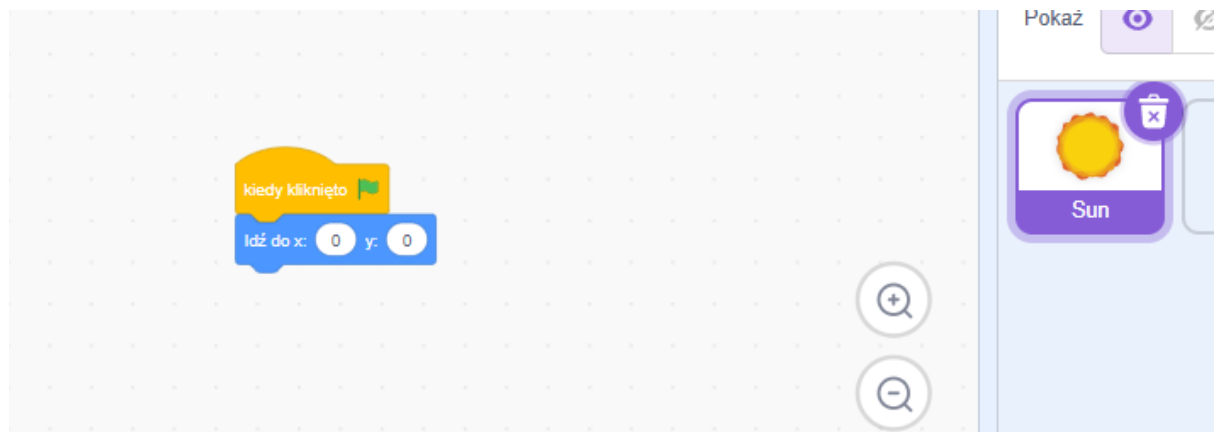
Rysunek 4. Wynik przeprowadzonej symulacji (przykładowy). Opracowanie własne w Scratch.

Zadanie B: Animacja Układu Słonecznego (Kinematyka)

Celem tej części projektu było stworzenie hierarchicznego modelu ruchu ciał niebieskich przy użyciu funkcji trygonometrycznych.

1. Pozycjonowanie Słońca

Słońce stanowi środek układu współrzędnych (punkt 0,0), wokół którego poruszają się pozostałe obiekty.

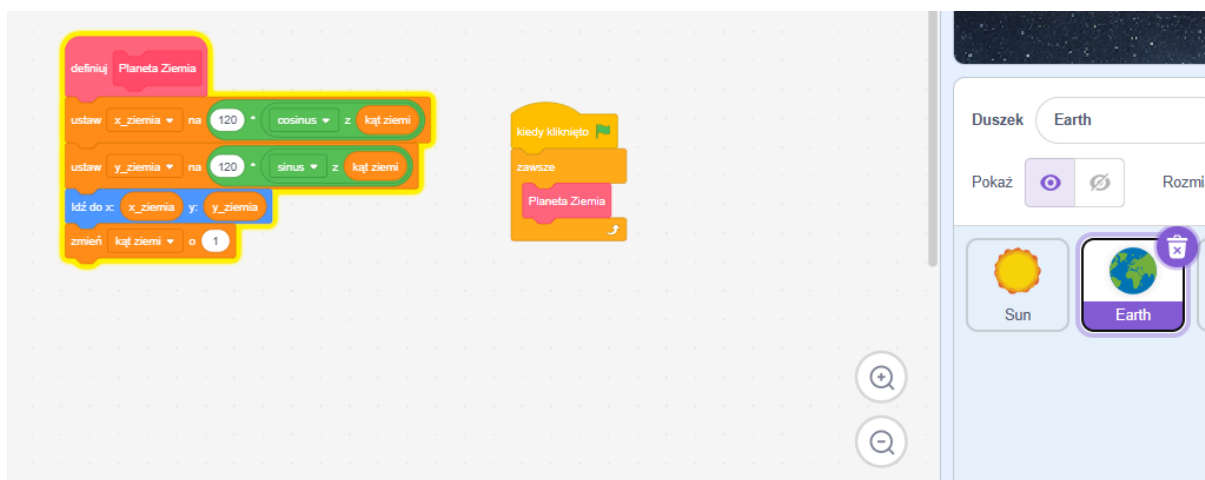


Rysunek 5. Przedstawia skrypt pozycjonujący Słońce w centrum sceny. Opracowanie własne w Scratch.

2. Własny blok: Planeta Ziemia (Orbita kołowa)

Logika ruchu Ziemi została zamknięta w bloku Planeta Ziemia.

- **Opis:** Pozycja Ziemi jest obliczana za pomocą funkcji cosinus (dla osi X) oraz sinus (dla osi Y). Pomnożenie wyniku przez 120 definiuje promień orbity. Zmienna kąt ziemi pełni rolę licznika czasu, który w każdej klatce zwiększa się o 1, zapewniając płynny ruch kołowy.

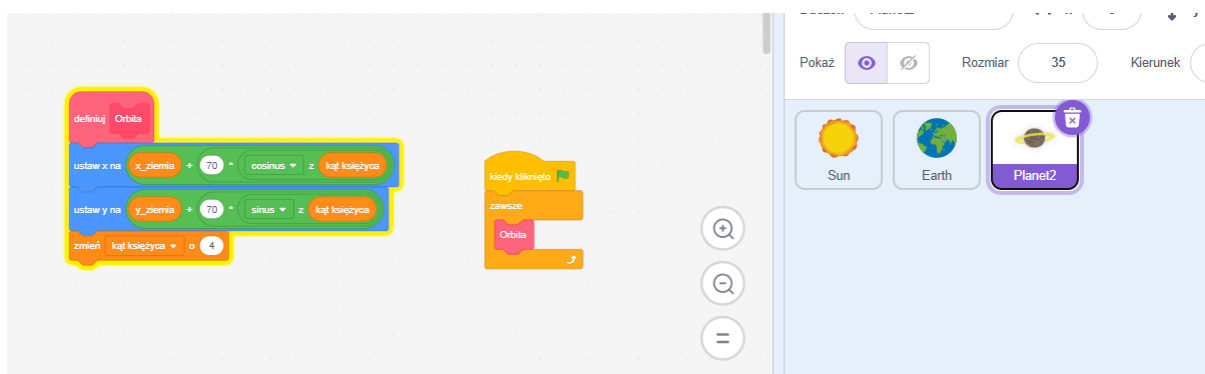


Rysunek 5. Przedstawia implementację bloku Planeta Ziemia oraz pętlę główną duszka, realizującą ruch po orbicie kołowej. Opracowanie własne w Scratch.

3. Własny blok: Orbita (Ruch Księżyca / Planet2)

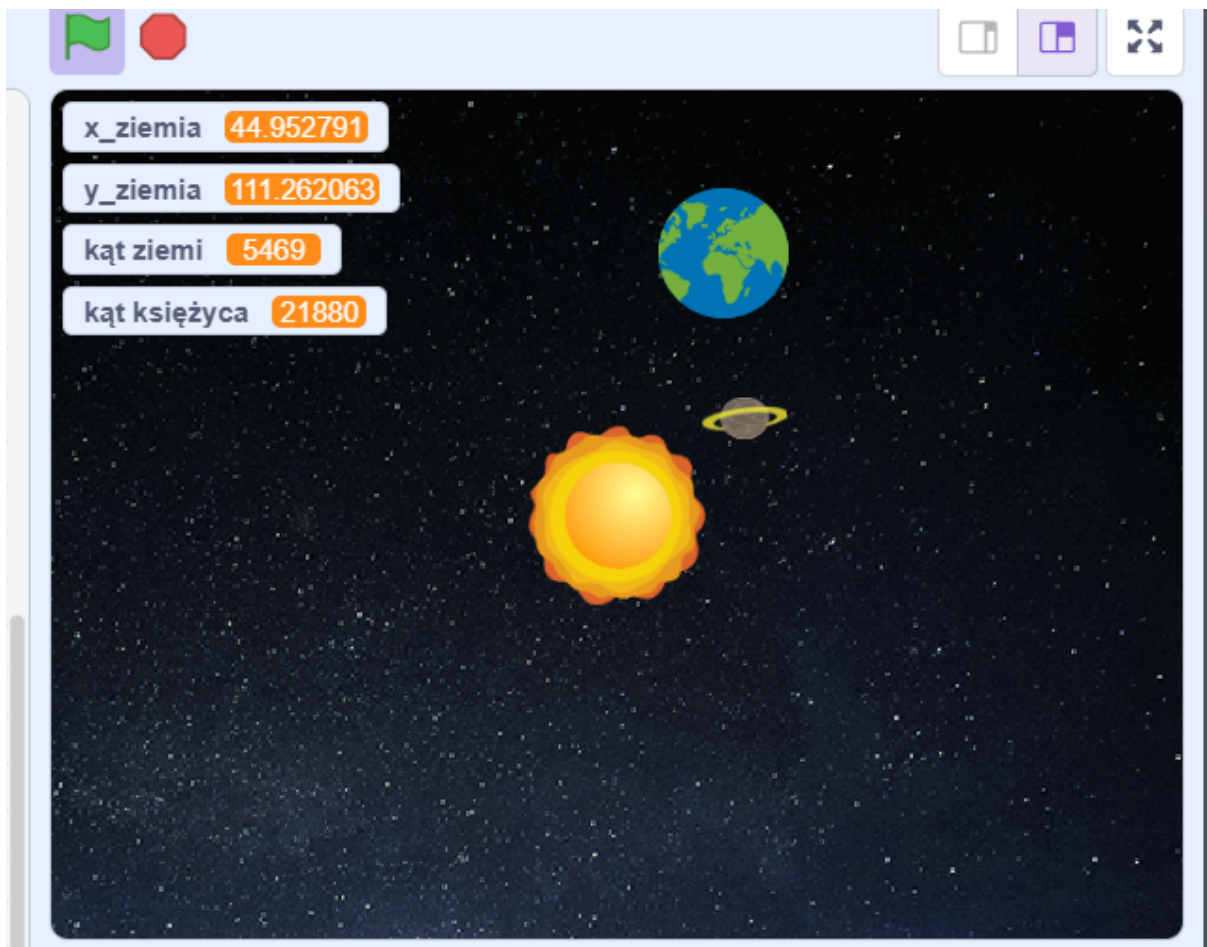
Księżyc (w projekcie nazwany Planet2) porusza się w sposób hierarchiczny – jego pozycja zależy od aktualnego położenia Ziemi.

- **Opis:** W bloku Orbita współrzędne obliczane są jako suma: [aktualna pozycja Ziemi] + [przesunięcie wynikające z trygonometrii]. Dzięki temu Księżyc zawsze "podąża" za planetą.
- **Dynamika:** Zastosowałem mnożnik promienia równy 70 oraz szybszy przyrost zmiennej kąt księżyca (o 4 w każdej klatce), co sprawia, że satelita krąży wokół Ziemi znacznie szybciej niż ona wokół Słońca.



Rysunek 6. Przedstawia kod bloku Orbita, implementujący hierarchiczny model ruchu Księżyca względem pozycji Ziemi.

Wynik przeprowadzonej symulacji :



Rysunek 7. Wynik przeprowadzonej symulacji (przykładowy). Opracowanie własne w Scratch.

Podsumowanie Techniczne

Oba zadania zostały zrealizowane zgodnie z zasadami Clean Code:

- Każdy kluczowy proces (fizyka, inicjalizacja, orbity) posiada własny blok funkcyjny.
- Logika jest oddzielona od pętli głównej zawsze.
- Program wykorzystuje zmienne globalne (czas/kąt) oraz suwaki do interakcji w czasie rzeczywistym.