Dokumentacja końcowa projektu

Temat: Symulacja ruchu obiektów w polu grawitacyjnym z obiektem centralnym.

Piotr Gierżatowicz-Sierpień, 331376

Link do repozytorium: https://gitlab-stud.elka.pw.edu.pl/PIPR_24Z_WW/pgierzat/pipr_24z_pgierzat

Commit: bafda99da20f7ce960ac7f58a6ecbc2042407f2f

Cel i opis projektu

Celem projektu była implementacja programu, który miał przeprowadzić symulację ruchu obiektów w polu grawitacyjnym. W symulacji przyjmujemy, że na obiekty niecentralne ma wpływ jedynie oddziaływanie pochodzące z obiektu centralnego. Program oblicza trajektorie obiektów, oraz je zapisuje. W przypadku zderzeniem między obiektami, dokonują one sklejenia i poruszają się dalej zgodnie z zasadą zachowania pędu. W przypadku zderzenia z obiektem centralnym, dodawana dochodzi tylko do zwiększenia masy obiektu centralnego. Po ilości podanych kroków przez użytkownika rysuje wynik symulacji. Pozwala na kontynuowanie symulacji po podaniu zapisanej konfiguracji wejścia programu. Użyte biblioteki: Matplotlib, Numpy, Json.

Podział programu na klasy i opis klas

Program składa się z czterech klas:

BaseObject – klasa zawierająca wspólne argumenty dla klasy Object i CentralObject. Umożliwa dziedziczenie wspólnych atrybutów, bez potrzeby tworzenia ich oddzielnie dla każdej z klas.

Object – klasa dziedzicząca po BaseObject, rozszerza konstruktor o velocity_x i velocity_y – prędkości w osi x i y. Reprezentuje obiekt w symulacji.

CentralObject – klasa również dziedzicząca po BaseObject, rozszerza konstruktor o radius – promień obiektu centralnego, wykorzystywany do sprawdzania kolizji z obiektem centralnym.

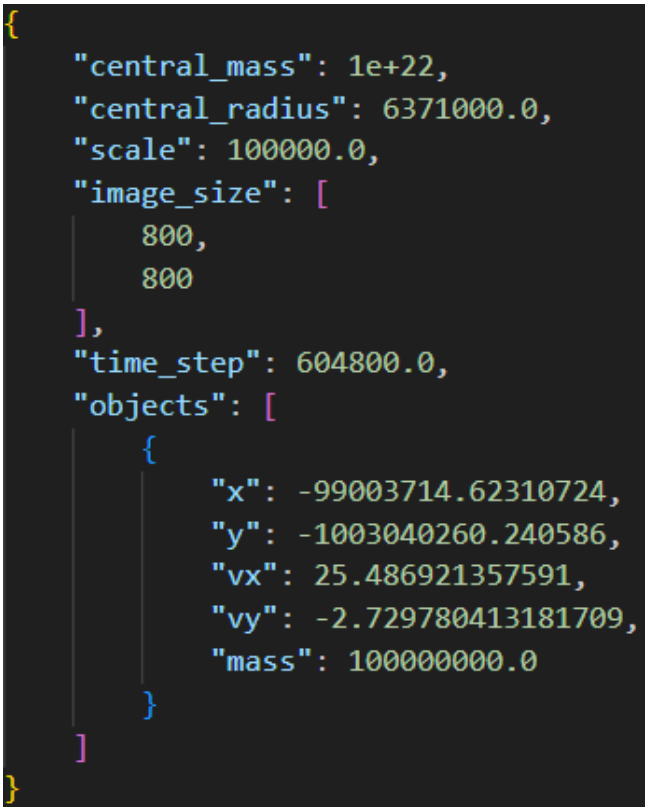
System – klasa odpowiedzialna za przeprowadzenie symulacji, przetworzenie przekazanych danych, wykonanie obliczeń oraz wizualizacje i zapisanie wyników symulacji.

Instrukcja użytkownika

Program uruchamiany z linii polecenia w formacie python3 main.py -s [liczba_kroków] -c [plik konfiguracyjny] -o [raport_kolizji] --state [stan]

Konfiguracja jest w formacie json, przez co wymaga odpowiedniej struktury pliku.

Format pliku konfiguracyjnego:



Symulacja wymaga podania parametrów dla obiektu centralnego, obiekty mogą lecz nie muszą być utworzone. Time_step mówi o kroku czasowym w symulacji, przy ustalaniu parametru warto zwrócić uwagę, że większy time_step wiąże się z mniejsza dokładnością symulacji.

Część refleksyjna

Do zakresu wykonanych prac należy implementacja klas reprezentujących obiekty oraz klasy odpowiedzialnej za symulacje. Obiekty poruszają się zgodnie z prawem powszechnego ciążenia. Siła pochodzącą od obiektu centralnego jest obliczana zgodnie ze wzorem $F = G\ \frac{M*m}{r^2}$ gdzie M to masa obiektu centralnego, m to masa obiektu, r odległość między nimi oraz G – stała grawitacyjna.

Problemem który napotkałem w czasie testowania programu jest niedokładność zmiennych typu float – przy inicjalizacji obiektów na tych samych koordynatach np. 1e8, obiekty nie mają dokładnie tych samych koordynatów przez co nie jest wykrywane to jako kolizja. Wykrywanie kolizji przy inicjalizacji działa poprawnie dla mniejszych koordynatów, co prezentują testy

jednostkowe. Próbowałem rozwiązać ten problem przy użyciu biblioteki Decimal, jednak nie przyniosło to oczekiwanych rezultatów. Zrealizowałem wszystkie wymagania przedstawione w projekcie.

Prezentowane rozwiązanie w większości pokrywa się z tym co planowałem, rzecz która się zmieniła to biblioteka użyta do wizualizacji symulacji. Ostatecznie postawiłem na wizualizację przy użyciu Matplotlib, zamiast początkowo planowanego Tkinter'a. Stwierdziłem, że będzie to rozwiązanie łatwiejsze do zrealizowania w przypadku przedstawiania symulacji wg. Iteracji.