# Specyfikacja Sphere Drop

Mkołaj Strużykowski

27 czerwca 2023

## Spis treści

# 1 Założenie

Sphere Drop to silnik fizyczny skupiony na symulacji w czasie rzeczywistym obiektów umieszczonych w polu grawitacyjnym. Do obliczania kolejnych położeń obiektów używany jest algorytm Verleta:

$$r(t + \Delta t) = 2r(t) - r(t - \Delta t) - \frac{f(t)}{m}\Delta t^2$$

Gdzie:
$r(t)$ = położenie obietku w chwili t,
$f(t)$ = siła działająca na ciało w chwili t,
$m$ = masa obiektu,
$\Delta t$ = skok czasowy(dla programu jest to $\frac{1}{60}$ sekundy).

Na wszystkie generowane oddziałowuje siła grawitacji zwrócona pionowo w dół.

$$F_g = G\frac{mM}{r^2}$$

Gdzie:
$G$ = stała grawitacji,
$m$ = masa obiektum
$M$ = masa Planety, na której znajduje się obiekt,
$r$ = promień dzielący środek obiektu od środka Planety.

Powyższe zmienne podstawowo są zainicjowane wartościami z naszego Wszechświata, a Planetą jest Ziemia, jednak użytkownik może dowolnie je modyfikować. Jeden metr odpowiada jednemu pixelowi.

$G = 6.6743 \cdot 10^{-11} \frac{m^2}{kg \cdot ^2 s}$
$M = 5.97219 \cdot 10^{24} kg$
$r = 6.3781 \cdot 10^{6} m$

Podczas obliczania sił działjących na obiekty pomijany jest opór powietrza oraz tarcie podłoża, ponadto wszystkie zdeżenia są idealnie sprężyste.

# 2 Klasa Object

Klasa Object jest klasą bazową dla wszystkich symulowanych obiektów, każdy obiekt składa się z prywatnych pól:

- std::vector<float> oldPosition - przechowujący informacje o położeniu w osi x i y obiektu w poprzedniej chwili czasowej,
- std::vector<float> currentPosition - przechowujący informacje o położeniu w osi x i y obiektu,
- std::vector <float> forces - przechowujący informacje o siłach działajacych na obiekt w osiach x i y,
- float mass - przechowuje informacje o masie obiektu,
- int timeOfBirth - czas pojawienia się obiektu, używany do ustalania początkowej prędkości.

oraz publicznych metod:

- std::vector<float> getOldPosition() const,
- std::vector<float> getPosition() const,
- std::vector<float> getForces() const,
- float getMass() const,

- int getTimeOfBirth() const,
- virtual std::vector<float> info() const - funkcja wirtualna zwracająca wektor specjlanych cech zależnych od klasy pochodnej klasy object,
- virtual void draw(sf::RenderWindow & win) const - funkcja wirtualna pozwalająca wyświetlać obiekt na ekranie, jako wartość przyjmuje wskaźnik na okno, na którym obiekt ma być rysowany.
- void updatePosition(std::vector<float> newPosition) - funkcja pozwalająca zmienić położenie obiektu przyjmuje wektor położenia x i y,
- void updateOldPosition(std::vector<float> newPosition); - funkcja pozwalająca zmienić poprzednie położenie obiektu,
- void updateForces(std::vector<float> forcesChange, int axis) - funkcja przyjmuje wektor zmiany sił działających na ciało oraz oś, w której nastąpiła zmiana 0 - x, 1 - y,
- int spawn(int x, int y, int timeOfBirth) - funkcja przyjmuje początkowe położenie na osiach x i y oraz czas powstania obiektu potrzebny do ustalenia prędkości początkowej. Funkcja spawn ustala pozycję początkową według współrzędnych kursora myszy.

### 2.1 Circle

Circle jest klasą pochodną dla klasy object posiada dodatkowe pole radius określające długość promienia koła, a także metody draw, która przesłania wirtualną metodę klasy object oraz info() zwracającą jednoelementowy wektor przechowujacy informację o promieniu koła.

## 3 Klasa View

Rolą klasy View jest wyświetlanie interfejsu graficznego użytkownika oraz symulowanych obiektów skład się z prywatnych pól:

- std::vector<Object*> & o - wskaźnik na wektor obiektów,
- Controller &c - wskaźnik na kontroler,
- sf::Font - oraz listę czcionek użytych w programie.

oraz prywatnych metod:

- void drawObjects(sf::RenderWindow&) const - wywołuje funkcje draw klasy object dla wszystkich obiektów.
- void drawPointer(sf::RenderWindow& ) const - funkcja na podstawie pozycij mysz podczas tworzenia obiektu wyświetla wektor prędkości oraz wartości prędkości początkowej nadawanej przez użytkownika,
- void Menu(sf::RenderWindow &) const - funkcja wyświetla menu użytkownika zaraz po włączeniu programu,
- void menuPause(sf::RenderWindow &) const - wyświetla menu pauzy dostępne dla użytkownika już podczas symulacji,
- void options(sf::RenderWindow &) const - wyświetla możliwe do zmiany przez użytkownika opcje symulacji,
- void drawPause(sf::RenderWindow &) const - wyświetla ikonę pauzy,
- void inspectObjects(sf::RenderWindow ) const - wyświetla informację o poszczególnych obiektach, prędkość w obu kierunkach, siły działające na ciało w obu kierunkach, masę oraz informacje dodatkowe (np. promień).

]

# 4 Klasa Controller

Klasa Controller jest odpowiedzialna za komunikację między użytkownikiem i programem, a także za obliczanie i wyznaczania sił, prędkości oraz pozycji wszystkich obiektów i środowiska, w którym się znajdują.

W klasie Controller znajdują się trzy struktury enum opisujące stan programu.

- enum LEVEL
  - MENU - menu wyświetlane użytkownikowi po włączeniu programu,
  - GRAVITYFIELD - głowny ekran zajmujący się symulacją obiektów w obecności pola grawitacyjnego,
  - PLANETARYSYSTEM - jeszcze nie rozwinięty tryb.
- enum STATE
  - RUNNING - symulacja działa w czasie rzeczywistym,
  - PAUSE_POINTER - zatrzymanie symulacji na czas tworzenia nowego obiektu,
  - PAUSE_ - zwykła pauza pozwalająca się dokładnie przyjrzeć obiektom,
  - PAUSE_MENU - pauza pozwalająca na wyświetlenie menu pauzy, powrót do podstawowego menu lub włączenie menu opcij,
  - PAUSE_OPTIONS - pauza aktywna podczas przeglądania i zmieniania opcji symulacji
- enum HIGHLIGHT- wszystkie elementy listy HIGHLIGHT przekazują informacje do klasy View o tekście, na który użytkownik najechał kursorem, aby go podświetlić.

Prywatne pola klasy Controller to:

- std::vector<Object* >& o - wskaźnik na wektor obiektów,
- LEVEL level - aktualny poziom symulacji,
- HIGHLIGHT highlight - aktualnie podświetlany tekst,
- STATE state - akrualny stan symulacji,
- int globalTime - zmienna odliczająca czas od rozpoczecia symulacji,
- float G - przechowuje wartość stałej grawitacyjnej,
- float M - przechowuje wartość masy planety,
- float R - przechowuje wartość promienia planety.

Prywatne metody klasy Controller:

- void addObject(sf::Event &, sf::Window &) - funkcja po kliknięciu myszy jeśli poziomem nie jest MENU wzywa funkcje spawn dla każdego obiektu i przekazując jako współrzędne x, y aktualną pozycję myszy,
- void setStartingVelocity(sf::Event &) - funkcja współpracuje z funkcją addObject pozycja kursora po zwolnieniu przycisku myszy jest ustalana dla ostatnio powstałego obiektu co pozwala użytkownikowi nadać początkową prędkość obiektu,
- void addForces(Object *) - Funkcja dla każdego obiektu oblicza oraz sumuje wszystkie działające na niego siły (np. siłę grawitacji), następnie z przypisuje mu je,

- std::vector<float> calculateAcceleration(Object * ) - na podstawie działających na obiekt sił oblicza wypadkowe przyspieszenie:

$$a = \frac{F_{wyp} \cdot FRAMERATE}{m}$$

Gdzie:
$F_{wyp}-$ to wypadkowa sił działająca na obietk,
$FRAMERATE-$ to stała opisująca ilość skoków czasowych(klatek) przypadających na jedną sekundę symulacji, na stałe ustalona na 60,
$m-$ masa obiektu,

- int constrainEdges(Object *o) - funkcja dla każdego obiektu sprawdza czy dotyka on krawędzi okna programu i jeżeli tak oblicza sprężyste odbicie się obiektu od niej,
- void changePosition(Object *) - funkcja za pomocą algorytmu Verlet'a oblicza kolejną pozycję każdego obiektu, oraz zapisuje aktualną pozycję obiektu jako poprzednią,
- void changeLevel(sf::Event &, sf::Window &) - funkcja odpowiedzialna za wykrywanie pozycji myszy i zmienianie poziomu jeżeli użytkownik wybierze dany tryb, ponadto jest odpowiedzialna za zapisywanie zmian wprowadzonych przez użytkownika w ustawieniach,
- int checkCollision() - funkcja sprawdza czy dwa obiekty nachodzą na siebie za pomocą obliczania odległości pomiędzy środkami wszystkich obiektów klasy Circle i porównywania ich z sumą promieni. Jeśli odległość pomiędzy środkami dowolnych dówch obiektów jest mniejsza niż suma promieni wywołuje funkcje resolveCollision,
- void resolveCollision(Object*,Object*) - funkcja jako argumenty przyjmuje wskaźniki na dwa obiekty, których funkcja checkCollision wykryła zderzenie. Zderzenie pomiędzy obiektami są idealnie sprężyste, więc całkowita energia kinetyczna układu zostaje zachowana:

$$\frac{m_1 v_1^2}{2} + \frac{m_2 v_2^2}{2} = \frac{m_1 u_1^2}{2} + \frac{m_2 u_2^2}{2}$$

Gdzie:
$v_1, v_2-$ prędkości ciał po zderzeniu,
$u_1, u_2-$ predkości ciał przed zderzeniem,
$m_1, m_2-$ masy ciał.

Po przekształceniu oraz zastosowania zasady zachowania pędu otrzymujemy wzór na wartośc prędkości po zderzeniu:

$$v_1 = \frac{u_1(m_1 - m_2) + 2m_2 u_2}{m_1 + m_2}$$

Aby znaleźć wektor prędkości funkcja wyznacza wektor prostopadły do wektora łączącego oba środki obiektów z punktem zaczepienia w środku obiektu1.

$$\vec{p} = (1, -\frac{\vec{x_1}}{\vec{y_1}})$$

$\vec{x_1} = S_{x1} - S_{x2}$
$\vec{y_1} = S_{y1} - S_{y2}$

$S_{y1}, S_{y2}-$ Środki kół w osi y,
$S_{x1}, S_{x2}-$ Środki kół w osi x.

Następnie wykonuje rzut wektora prędkości na wektor $\vec{p}$:

$$\vec{W} = \frac{\vec{v} \circ \vec{p}}{\vec{p} \circ \vec{p}} \cdot \vec{p}$$

Następnie wartości prędkości sa odpowiednio skalowane według wzoru na prędkość po zderzeniu, a później obliczany jest punkt w którym obiekt powiniein znaleźć się po zderzeniu:

$$x_{nowy} = 2\vec{W_x} - \vec{\nu_x}$$
$$y_{nowy} = 2\vec{W_y} - \vec{\nu_y}$$

$\vec{\nu_x} = S_x + v_x$
$\vec{\nu_y} = S_y + v_y$

Na koniec za pomocą object.updatePosition() funkcja ustala nowe pozycje obiektów.

- void pause() - funkcja odpowiedzialna za zmianę stanu symulacji z RUNNING, na PAUSE_POINTER, używana do zatrzymywania symulacji podczas tworzenia nowego obiektu,
- void resume(sf::Event &) - funkcja zmienia stan symulacji z różnych pauz na RUNNING,
- void restart(sf::Event &) - funkcja resetuje symulację, za pomocą funkcj clear(), czyści wektor obietków oraz ustala wszystkie opcje na standardowe.
- void pausePlay(sf::Event &) - funkcja jest odpowiedzialna za ustalanie stanu programu na PAUSE_ lub RUNNING po kliknięciu przycisku 'P' przez użytkownika,
- void menu() - funkcja, która po kliknięciu przycisku 'ESC' ustala stan gry na PAUSE_MENU,
- void update() - funkcja w każdej klatce programu wywołuje: addForces(), changePosition(), constrainEdges() i checkCoillision dla każdego obiektu, zwiększa również int globalTime.
- void changeGravitationalConstant (float gravitationalConstant) -pozwala na zmianę stałej grawitacyjnej,
- void changePlanetRadius (float radius) - pozwala na zmianę promienia Planety,
- void changePlanetMass(float mass) -pozwala na zmianę masy Planety.

Publicznymi metodami klasy Controller są funkcje zwracające aktualny stan poszczególnych pól klasy:

- LEVEL getLevel() const,
- HIGHLIGHT getHighlight() const,
- STATE getState() const,
- std::vector<float> getMousePosition() const,
- float getGravitationalConstant() const,
- float getPlanetRadius() const,
- float getPlanetMass() const.

# 5 Bibliografia


- algorytm Verlet'a Zderzenie sprężyste
- Dokumentacja SFML
- Forum SFML