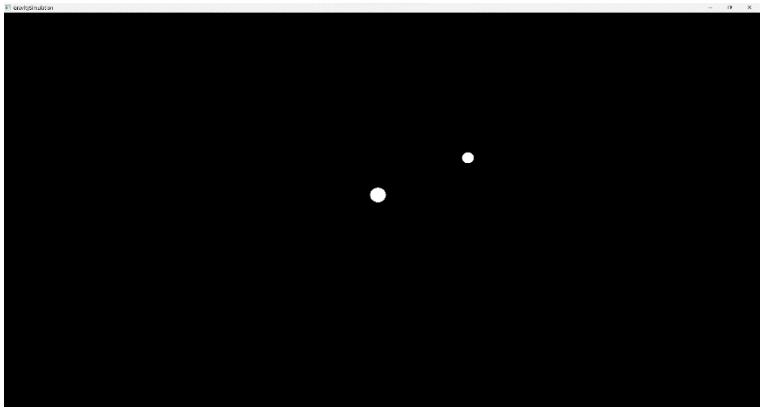
Felhasználói dokumentáció

GravitySimulation

Pontosított specifikáció:

A program célja égitestek mozgásának szimulálása Newton $F = G\frac{m_1 * m_2}{r^2}$ gravitációs törvénye alapján, 2 dimenzióban.



Bemenetek:

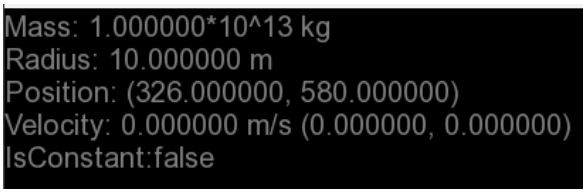
A program grafikus felületet nyit meg, amire a felhasználó sandbox-szerűen tud objektumokat lehelyezni.

A megjelenő ablak az x gombra kattintással, vagy az Esc gomb megnyomásával bármikor bezárható.

Új objektum felvitele:

Ha kurzorunkat a grafikus felületre visszük, észrevehetjük, hogy van mellette egy halványabb objektum. Ez a lerakni kívánt objektum előnézete, melyet a következőképpen állíthatunk:

A bal egérgombot nyomva tartva megjelenik egy menü, ami az objektumunk adatait tartalmazza.



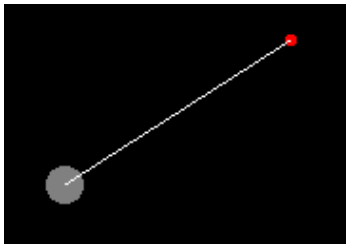
A megjelenő adatokat (a pozíció kivételével) kizárólag a bal egérgomb nyomva tartása közben tudjuk változtatni!

Pozíció változtatása: Az egerünk helyét lekövetve automatikusan állítódik.

Tömeg változtatása: A billentyűzeten a fel és le nyilak megnyomásával/nyomva tartásával történik.

Sugár változtatása: Az egér görgőjének fel-le tekerésével állítható.

Sebesség beállítása: Az *Angry Birds* játék csúzlis mechanikájához hasonló: A bal egérgomb lenyomására megjelenik egy piros célzópont. Az objektum az egérgomb elengedése után a célzópont irányába kapja meg a kezdősebességét. A sebesség annál nagyobb, minél messzebb húztuk az objektumot a célzóponttól.



Kimenet:

A felhasználó által kitöltött játéktér, mellyel folyamatosan interaktálni lehet.

Egyéb tudnivalók:

A program által kiírt adatok megfelelnek a valóságnak, vagyis ugyanakkora súlyú és sebességű objektumok a valóságban is ugyanolyan alakú pályára állnának, és ugyanúgy változtatnák sebességüket, mint ami a programban megjelenik.

Ez alóli kivétel a dolgok lezajlásának sebessége: Egy másodperc alatt a szimulációban 1 perc telik el.

Mivel sem Newton képlete, sem a valóságunk nem gátolja meg, a programban létrehozható negatív tömegű test. Ennek kitapasztalását a felhasználóra hagyom.

Programozói dokumentáció

GravitySimulation

Megvalósítás eszközei:

A program C++ nyelven írodott, a grafikus megjelenítéshez az SFML könyvtárat használtam.

Program által használt adatszerkezetek:

Position osztály:

```
class Position {
	double x;
	double y;
};
```

Tagfüggvényei:

- két paraméteres konstruktor
- double getX(), double getY()
- bool operator== (Position p)
- Position operator+ (Position p)
- Position operator+= (Position p)
- Position operator- (Position p)

Vector osztály:

```
class Vector {
	Position direction;
};
```

Tagfüggvényei:

- default konstruktor, egy paraméteres konstruktor
- setDirection(Position p)
- Position getDirection()
- bool operator== (Vector v)
- Vector operator+ (Vector v)
- Vector operator+= (Vector v)

PlanetaryObject osztály:

```
class PlanetaryObject {
protected:
	Position position;
	Vector velocity;
	Vector acceleration;
	double mass;
	double radius;
	bool isConstant;
};
```

Tagfüggvényei:

- default konstruktor, 5 paraméteres konstruktor
- minden adattagjához egy getter és egy setter tagfüggvény

- operator==, operator!=
- double getDistance (PlanetaryObject p) – két objektum középpontja közti távolságot számolja ki.
- Vector ParticularAcceleration(PlanetaryObject p) – kiszámolja a newtoni képlet alapján, hogy az objektum mekkora és milyen irányú gyorsulást kap egy másiktól (vagyis hogy mekkora erővel húzzák).
- bool isOutOfScreen() – visszaadja, hogy az objektum elhagyta-e a játékteret

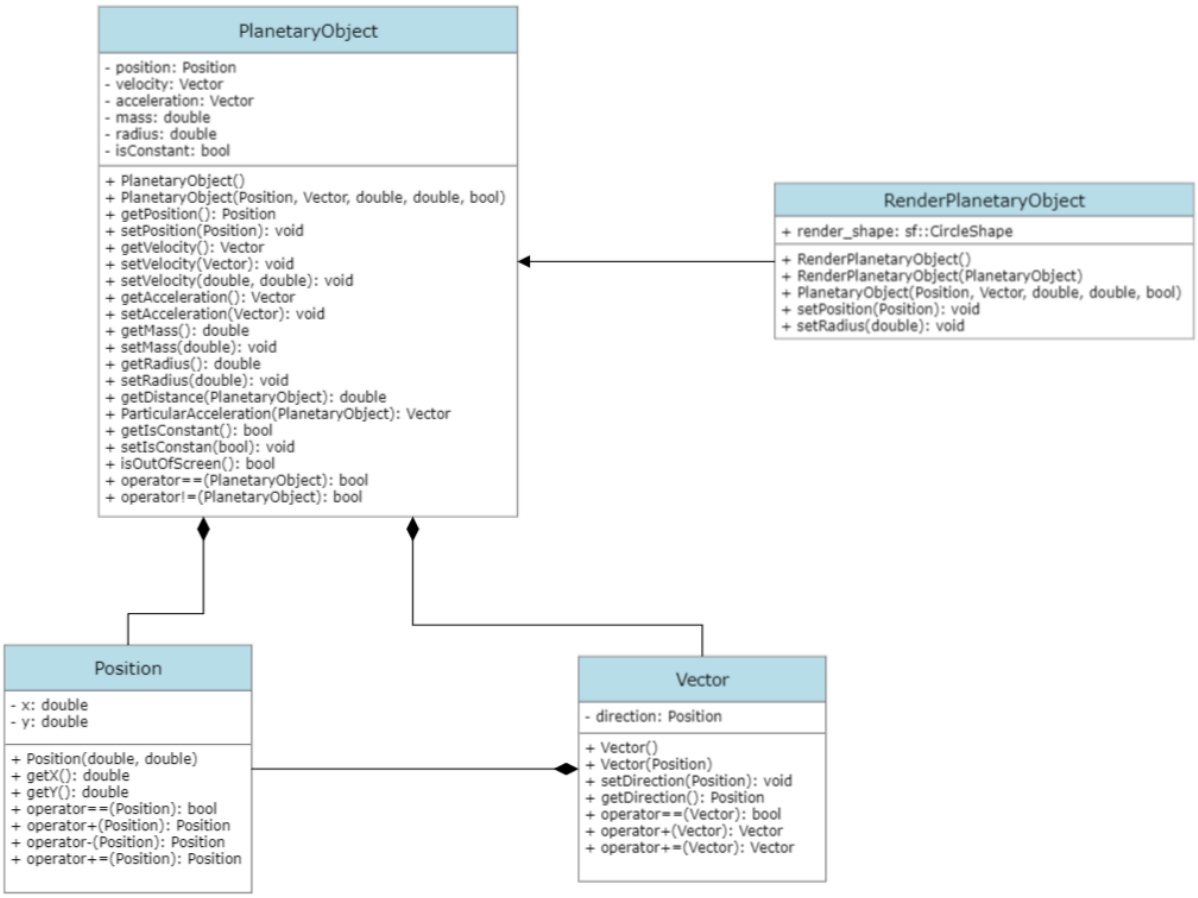
RenderPlanetaryObject osztály: PlanetaryObject gyerek osztálya, annak megjelenítéséért felel.

```
class RenderPlanetaryObject : public PlanetaryObject {
	sf::CircleShape render_shape;
};
```

Tagfüggvényei:

- default, 1 változós (PlanetaryObject-ből) és 5 változós konstruktor
- void setPosition(Position p) – a PlanetaryObject egyik virtual tagfüggvényének felülírása, hogy az ectra adatok is értéket kapjanak.
- void setRadius(double r) – az előzőhöz hasonlóan egy felüldefiniálás

Osztályok UML diagramja:



Program fájljai, és tartalmuk:

Position.h: Position osztály adattagjainak, és tagfüggvényeinek deklarációja, valamint a `bool doubleEQ(double a, double b);` deklaráció, amely a double értékek összehasonlítását végzi.

Mivel ez egy olyan alapvető osztály, ami közvetlenül, vagy közvetetten a projekt minden fájljába be van inkludálva, ezért itt kapott helyet a `#define GraphicRender true` definíció, amellyel változtatható a program viselkedése. Az igaz érték a grafikus felületet, a hamis érték pedig a csupán JPORTA-ra szánt console-os felületet futtatja.

Position.cpp: A doubleEQ függvény definíciója.

Vector.h: Vector osztály definiálása

PlanetaryObject.h: PlanetaryObject osztály definíciója, és bonyolultabb tagfüggvényeinek deklarációja.

Itt található a gravitációs konstans programbeli értékének definiálása: `#define Grav_const 0.000000000066743`

PlanetaryObject.cpp: A PlanetaryObject tagfüggvényeinek definiálása. Itt található a gravitációs vonzást kiszámoló függvény is.

DisplayOnConsole.h: A JPORTA-s megjelenítéshez használt függvények deklarációja.

DisplayOnConsole.cpp: A DisplayOnConsole.h függvényeinek definíciója.

Rendering.h: A RenderPlanetaryObject definiálása, illetve a program grafikus megjelenítéséért felelős függvényeinek deklarációja.

Itt található a program két legfontosabb függvényének deklarációja:

`void UpdatePositions(std::vector<RenderPlanetaryObject>& rendered_space):`

Minden függvényhíváskor végigiterál az összes objektumon, és mindegyik objektumra megnézi a többi objektum által rá ható erőket. Ezeket összeadja, majd frissíti az objektumok jelenlegi gyorsulását, sebességét, és pozícióját.

Objektumok ütközésekor, illetve az objektum játéktérből való kilépésekor a megfelelő objektumot törli.

`void RenderObjects(std::vector<PlanetaryObject> s):`

A játéktér létrehozásáért felelős függvény, melynek hasában nagyrészt az SFML által igényelt és szolgáltatott dolgokat használjuk. Itt történik az események (kattintás, billentyűleütés) érzékelése, az előnézet adatainak kiírása, az objektumok kirajzolása, illetve az előbb leírt UpdatePositions() függvény hívása is.

Rendering.cpp: A RenderPlanetaryObject osztály tagfüggvényeinek, illetve az előbb említett függvényeknek a definiálása itt kap helyet.

GravitySimulation.cpp: A program „main” fájlja, ez felel az előzőleg említett függvények hívásáért. Ez az egyetlen fájl, ahol a grafikáért felelős és az attól teljesen elkülöníthető kódrészletek egymással találkoznak, így itt történik meg a kettő elkülönítése.

Specifikációra leadott fájl

Gravity Simulation

A program célja:

„Égitestek” mozgásának szimulálása Newton $F = G\frac{m_1 * m_2}{r^2}$ gravitációs törvénye alapján, 2 dimenzióban.

Felhasználóval való kommunikáció:

A szimulációs program végső változata a felhasználóval (reményeim szerint) grafikus felületen kommunikál.

Bemenetek:

A lehelyezni kívánt égitest adatait a felhasználó az egér és a billentyűzet segítségével állíthatja majd. A lehelyezés előtt a képernyőn megjelenő prototípus mutatja majd, hogy hogyan fog kinézni és milyen tulajdonságokkal rendelkezik majd az objektum.

Az állítható tulajdonságok a következők:

- Égitest helye – az egér bal klikkjével helyezhető le az objektum a játéktérre
- Égitest mérete (sugara) – egér görgőjével
- Égitest súlya – fel-le nyilakkal a billentyűzeten
- Elpusztulhat-e az égitest egy ütközésben – ’i’ betűvel váltogatható az állapot
- Égitest kezdeti sebességének nagysága és iránya – egér segítségével, Angry Birds-szerű célzással.

Kimenetek:

A program által generált játéktér, melyen érvényesülnek a kölcsönhatások. Erre futás során bármikor lehelyezhető egy új objektum, így ez az input is egyben.

Egyéb megjegyzés:

A program csak Newton képlete alapján számol, ezt a kölcsönhatást azonban ideális esetben mindegyik test kifejti az összes többire. Ez egy határon túl rengeteg számolást jelent majd, így az optimalizációra is nagy hangsúlyt kell fektetni.

Terv

Gravity Simulation

A program célja:

„Égitestek” mozgásának szimulálása Newton $F = G\frac{m_1 * m_2}{r^2}$ gravitációs törvénye alapján, 2 dimenzióban.

Felhasználóval való kommunikáció:

A szimulációs program végső változata a felhasználóval (reményeim szerint) a standard input/output-on kívül grafikus felületen is kommunikál (ez lesz az elsődleges felület).

Bemenetek:

A lehelyezni kívánt égitest adatait a felhasználó az egér és a billentyűzet segítségével állíthatja majd. A lehelyezés előtt a képernyőn megjelenő prototípus mutatja majd, hogy hogyan fog kinézni és milyen tulajdonságokkal rendelkezik majd az objektum.

Az állítható tulajdonságok a következők:

- Égitest helye – az egér bal klikkjével helyezhető le az objektum a játéktérre
- Égitest mérete (sugara) – egér görgőjével
- Égitest súlya – fel-le nyilakkal a billentyűzeten
- Égitest kezdeti sebességének nagysága és iránya – egér segítségével, Angry Birds-szerű célzással.
- Égitest gyorsulásának nagysága és iránya – nem megadható, program számolja
- Elpusztulhat-e az égitest egy ütközésben – ’i’ betűvel váltogatható az állapot

Kimenetek:

A program által generált játéktér, melyen érvényesülnek a kölcsönhatások. Erre futás során bármikor lehelyezhető egy új objektum, így ez az input is egyben.

Fontosabb függvények:

```
void RenderObjects(std::vector<PlanetaryObject> s)
```

A grafikus felületért (megjelenés és interakciók) felelős függvény. Itt van meghívva a legtöbb függvény. A grafikus ablak megnyitásán és bezárásán kívül a bolygók sebességének, gyorsulásának, pozíciónak állításáért felelős függvények is itt hívódnak meg.

Osztályok:

```
class Position {
	double x;
	double y;

};
```

Tagfüggvényei:

- két paraméteres konstruktor
- double getX(), double getY()
- bool operator== (Position p)
- Position operator+ (Position p)
- Position operator+= (Position p)

```
class Vector {
	Position direction;

};
```

Tagfüggvényei:

- default konstruktor, egy paraméteres konstruktor
- setDirection(Position p)
- Position getDirection()
- bool operator== (Vector v)
- Vector operator+ (Vector v)
- Vector operator+= (Vector v)

```
class PlanetaryObject {
protected:
	Position position;
	Vector velocity;
	Vector acceleration;
	double mass;
	double radius;

};
```

Tagfüggvényei:

- default konstruktor, 4 paraméteres konstruktor
- minden adattagjához egy getter és egy setter tagfüggvény
- operator==, operator!=
- double getDistance (PlanetaryObject p) – két objektum középpontja közti távolságot számolja ki.
- Vector ParticularAcceleration(PlanetaryObject p) – kiszámolja a newtoni képlet alapján, hogy az objektumot mekkora erővel húzza egy másik.

A grafikus kezelés bizonytalansága miatt, és mivel nagyrészt SFML beépített osztályok és függvények használata is szükséges a megvalósításhoz, ezért ezekből egyelőre csak egyet írok a specifikációba.

```
class RenderPlanetaryObject : public PlanetaryObject {
	sf::CircleShape render_shape;
```

```
};
```

Tagfüggvényei: default, 1 változós (PlanetaryObject-ből) és 4 változós konstruktor

A render_shape a grafikus megjelenítés során magát az objektumot jelzi, az SFML-lel ennek tagfügvényeit használva lehet kirajzolni az objektumot.

UML osztálydiagram: