Міністерство освіти і науки України

Департамент освіти і науки Вінницької облдержадміністрації

Вінницьке територіальне відділення МАН України

Відділення: «Комп’ютерних наук»

Секція: «Технології програмування»

**Дослідження моделей зоряних систем**

Роботу виконав:

Кандидат у члени МАН

Мальований Денис Олегович,

учень 9 класу

комунального закладу

“Вінницький технічний ліцей”

Науковий керівник:

Єфременюк Ніна Василівна,

учитель інформатики

комунального закладу

“Вінницький технічний ліцей ”,

учитель-методист

Вінниця – 2019

ЗМІСТ

[ВСТУП 3](#_Toc1840379)

[РОЗДІЛ 1 ВИБІР МОВИ І СЕРЕДОВИЩА ПРОГРАМУВАННЯ 6](#_Toc1840380)

[РОЗДІЛ 2 СТВОРЕННЯ МОДЕЛІ 7](#_Toc1840381)

[2.1 Структура програми 7](#_Toc1840382)

[2.2. Клас Planet 8](#_Toc1840383)

[2.3. Клас Space 10](#_Toc1840384)

[2.4. Основна частина програми 11](#_Toc1840385)

[РОЗДІЛ 3 ОПИС РОБОТИ ПРОГРАМИ 12](#_Toc1840386)

[ВИСНОВКИ 16](#_Toc1840387)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 17](#_Toc1840388)

[Додаток А 18](#_Toc1840389)

[Додаток Б 23](#_Toc1840390)

[Додаток В 29](#_Toc1840391)

[Додаток Г 38](#_Toc1840392)

**ВСТУП**

Дослідження моделей зоряних систем доволі **актуально**, оскільки у наш час дуже стрімкого розвитку зазнає космічна галузь: запуск штучних супутників стає все буденнішим, дешевшим і простішим. Науковці планують колонізувати найближчі планети, досліджують найвіддаленіші закутки всесвіту. Але велика кількість людей не уявляє за якими законами рухаються космічні об’єкти.

**Основною метою** науково-дослідницької роботи є демонстрація руху та взаємодії космічних тіл, дослідження орбітальної механіки.

**Предметом дослідження** є зоряні системи та їх компоненти: планети й інші космічні тіла.

**Об’єктом дослідження** фізичні та математичні взаємозв’язки між космічними тілами й складовими зоряних систем.

**Прикладна цінність** науково-дослідницької роботи полягає у можливості використання програми для навчання, експериментів і розваг.

**Очікувані результати виконання роботи:**

* **дослідити математичну модель взаємодії космічних тіл та її закони;**
* навчитись працювати у середовищі програмування Processing;
* розробити прикладну програму для графічного відображення дослідженої моделі.

**Теоретичні відомості для побудови моделі**

Для побудови фізично-математичної моделі проекту мені знадобились підручник за 9 клас для поглибленого вивчення математики, а саме таких тем:

* вектори;
* рівняння фігур (еліпс);
* рівняння прямої на координатній площині;

і фізики:

* рівнозмінний рух;
* закони Ньютона;
* закон всесвітнього тяжіння;
* закони Кеплера.

Використаємо три закони **Ньютона**:

* “Існують такі системи відліку, в яких тіло перебуває у стані спокою або рівномірного прямолінійного руху, коли на нього не діють інші тіла або дія цих тіл скомпенсована”;
* “Прискорення, з яким рухається тіло у інерціальних системах відліку, прямопропорційне рівнодійній силі, що діє на тіло, і оберненопропорційне масі тіла. ”;
* “Усі тіла в інерціальних системах відліку взаємодіють між собою із силами, що направлені вздовж однієї прямої, рівні за модулем та протилежні за напрямком” (справедливо для пари тіл);

також **закон всесвітнього тяжіння**: “Усі тіла взаємодіють між собою із силою, що прямопропорційна добутку мас цих тіл і оберненопропорційна квадрату відстані між ними. Ця сила напрямлена вздовж прямої, що сполучає центри мас тіл. ”;

і перший із **трьох законів Кеплера**: “Всі планети обертаються навколо Сонця еліптичними орбітами, в одному з фокусів яких перебуває Сонце (всі орбіти планет і тіл Сонячної системи мають один спільний фокус, в якому, власне, і розташовано Сонце)”.

Найближча до Сонця точка орбіти називається **перигелієм**, а найдальша від нього точка — **афелієм**.

Прискорення, що спричинене гравітаційними силами, рівне . Формула доцентрового прискорення , звідки . Якщо розглядати орбіти планет, як еліпси, тоді кінцева формула набуває вигляду , де ah – афелій орбіти, а ph – перигелій.

Якщо два вектори перпендикулярні, то косинус між ними дорінює нулю, звідси скалярний добуток також дорівнює нулю. Отже, x1\*x2 = - y1\*y2, звідки y2 = x1/y2, якщо x1 = -1. Усе це використано у методі generate() класа Space для придання планетам початкової швидкості.

**РОЗДІЛ 1  
ВИБІР МОВИ І СЕРЕДОВИЩА ПРОГРАМУВАННЯ**

Для реалізації моделі я використав мову і середовище програмування Processing, що основана на Java і має вбудовану графічну бібліотеку OpenGL. Вона надає багато можливостей при роботі з графікою, а багата документація й різноманіття функцій робить процес розробки значно легшим.

Сильні сторони Processing:

* **легкий і простий;**
* **має широку вбудовану графічну бібліотеку;**
* **кросплатформний завдяки Java та її JVM;**
* **швидкий;**
* **гнучкий.**

Як було зазначено вище, Java використовує JVM (Java Virtual Machine) – віртуальну машину Java. Завдячуючи цьому з’являються перспективи використання програми на різних ОС. Код транслюється в байт-код віртуальної машини, завдяки чому він не залежить від виконавчої платформи, а дякуючи численним її оптимізаціям, програми на Java не поступаються місцем за швидкістю майже ні одному аналогу на інших мовах.

Графічну оболонку Processing IDE подано на рисунку Г.1 і Г.2.

**РОЗДІЛ 2  
СТВОРЕННЯ МОДЕЛІ**

**2.1 Структура програми**

Програма складається з трьох частин, класу **Planet**, що є абстракцією реальних планет, класу **Space**, що керує списком об’єктів попереднього класу, реалізує їх завантаження з XML-файлу, а також **основної частини**, звідки викликаються усі інші, реалізується керування симуляцією, відображення графічного інтерфейсу та інше. Також у корні папки з ПЗ присутні файл planets.xml і папка textures з текстурами планет. Графічно структуру програми можна представити так (Рис. 2.1).

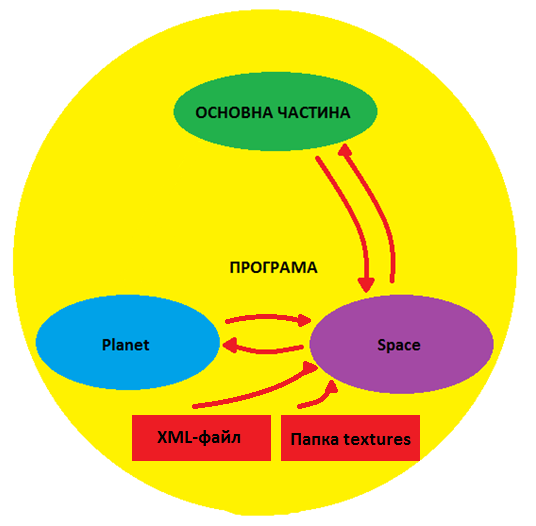


Рис.2.1 Структура програми

**2.2. Клас Planet**

Найцікавішими властивостями класу Planet є методи **collide()** і **gravity()**. Перший викликає функцію **detectCollision()** і, якщо та поверне істину, тобто планети зіткнулися, викликає **absorb()**. Останній оснований на поглинанні планети, що має меншу масу, розрахунку прискорення, що діє на поглинаючу планету і т.п. Функція **gravity()** є реалізацією закону всесвітнього тяжіння.

Крім того, клас має також багато інших функцій:

* конструктори;
* геттери і сеттери, із загальною назвою setAAA() або getAAA(), де AAA- назва поля;
* distance() – повертає відстань між точками/об’єктами;
* gravityForce() – повертає значення сили всесвітнього тяжіння, що діє на пару тіл;
* addMass() – додає масу і перераховує радіус за густиною;
* addTemperature() – додає температури і встановлює колір відповідно до неї;
* move() – “рухає” планету (додає до координат вектор);
* show() – малює текстуру планети із вказаним відтінком (r, g, b);
* mark() – малює точку на місці планети із вказаним відтінком (r, g, b);

і полів:

* x, y;
* mass;
* rad (радіус);
* r , g, b (кольори за схемою RGB);
* friction (коефіцієнт тертя, що використовується при обробці зіткнень);
* density (густина);
* temperature;
* specificHeatCapacity (питома теплоємність);
* helion (відстань від супутника до “хазяїна”);
* master (назва “хазяїна”);
* vect (вектор руху);
* delete (флаг видалення);
* texture (текстура об'єкта).

Увесь код класу Planet наведений в додатку А.

**2.3. Клас Space**

**Конструктор** класа Space завантажує у динамічну базу даних (список, або ArrayList) планети з вказаного в аргументах файлу XML-формату. Функція **generate()** створює нові об’єкти класа Planet, копіюючи завантажені, обчислює незаповнені поля і генерує світ симуляції. Клас має такі поля:

* planets - список, що містить об’єкти планет, які беруть участь у симуляції;
* planetDB – список, що містить планети із бази даних, заповнюється із створення екземпляра класа Space.

Також існують такі методи:

* два геттера раніше описаних полів;
* addPlanet() – додає вказівник на об'єкт планети у список planets;
* show() – викликає функцію show() у всіх об’єктів в списку planets;
* mark() – викликає функцію mark() у всіх об’єктів в списку planets;
* orbit() – будує орбіти планет;
* tick() – оброблює один “тік” (ітерацію) симуляції.

Увесь код класу Space наведений в додатку Б.

**2.4. Основна частина програми**

В головній частині існують такі функції, які властиві для Processing’у, як **setup()** – виконується один раз при старті, і **draw()** – циклічно повторюється до вимкнення. Написані і такі функції:

* keyboard() – оброблює введення з клавіатури;
* graphicface() – малює графічний інтерфейс програми;
* mousePressed() – виконується, коли будь-яка кнопка миші затиснута;
* mouseReleased() – виконується, коли кнопка миші відпущена.

Увесь код основної частини програми наведений в додатку В.

**РОЗДІЛ 3  
ОПИС РОБОТИ ПРОГРАМИ**

Можна налаштувати об’єкти симуляції перед її запуском, змінюючи файл planets.xml. Кожна планета у ньому має такі поля: маса, радіус, температура, дальність від тіла, супутником якого вона є, його і власна назви. Все інше розраховується при запуску програми. Таким чином можна додавати, конфігурувати і видаляти планети. Файл має такий вигляд (Рис. 3.1).



Рис.3.1 Вміст файлу planets.xml

Графічний інтерфейс програми такий має вигляд (Рис. 3.2).

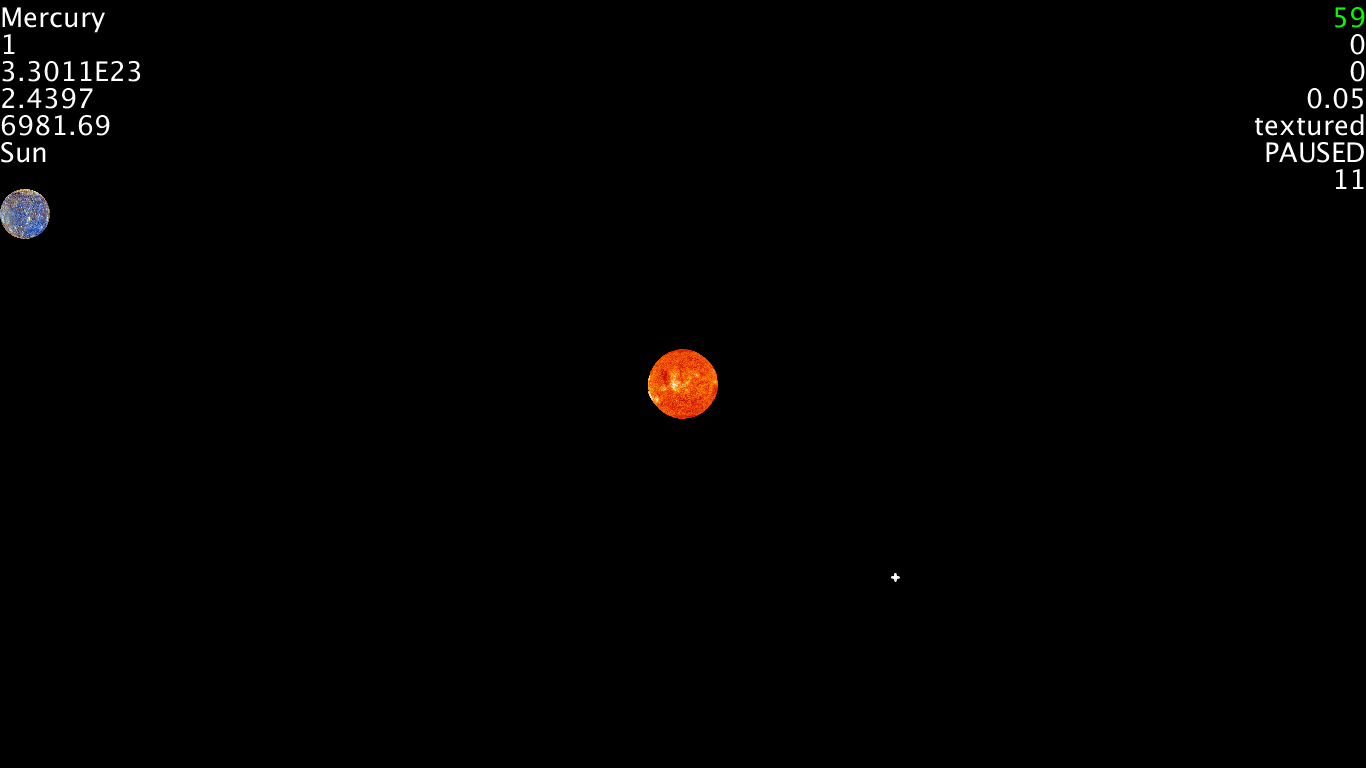


Рис.3.2 Графічний інтерфейс програми

Керування камерою спостерігача відбувається клавішами ‘w’, ‘a’, ‘s’, ‘d’ для руху вгору, вліво, вниз і вправо відповідно. Камера приближується клавішою ‘z’, а віддаляється – ‘x’. Клавішою ‘c’ можна повернутись у точку (0;0). Якщо затиснути ліву клавішу і простягнути мишу в будь-якому напрямку, а потім відпустити кнопку, то у точці, де спочатку перебувала миша з’явиться планета, що буде рухатися у напрямку миші з початковою швидкістю, що пропорційна її переміщенню. До речі, у лівому верхньому кутку вказано характеристики планети, яка вибрана зараз для вставки, а також її фото. Натиснувши лівою клавішею миші на космічному тілі, воно зникне. Кнопка ‘m’ перемикає відображення текстур планет, ‘i’ – інтерфейсу, а ‘o’ – орбіт планет (Рис.3.3). Кнопка ‘p’ зупиняє і продовжує хід симуляції. У правому верхньому кутку показано зверху донизу відповідно FPS (кількість кадрів за секунду), x- та y-координати, коефіцієнт приближення, а також режим перегляду, індикатор “ходу” симуляції і кількість космічних тіл у ній.

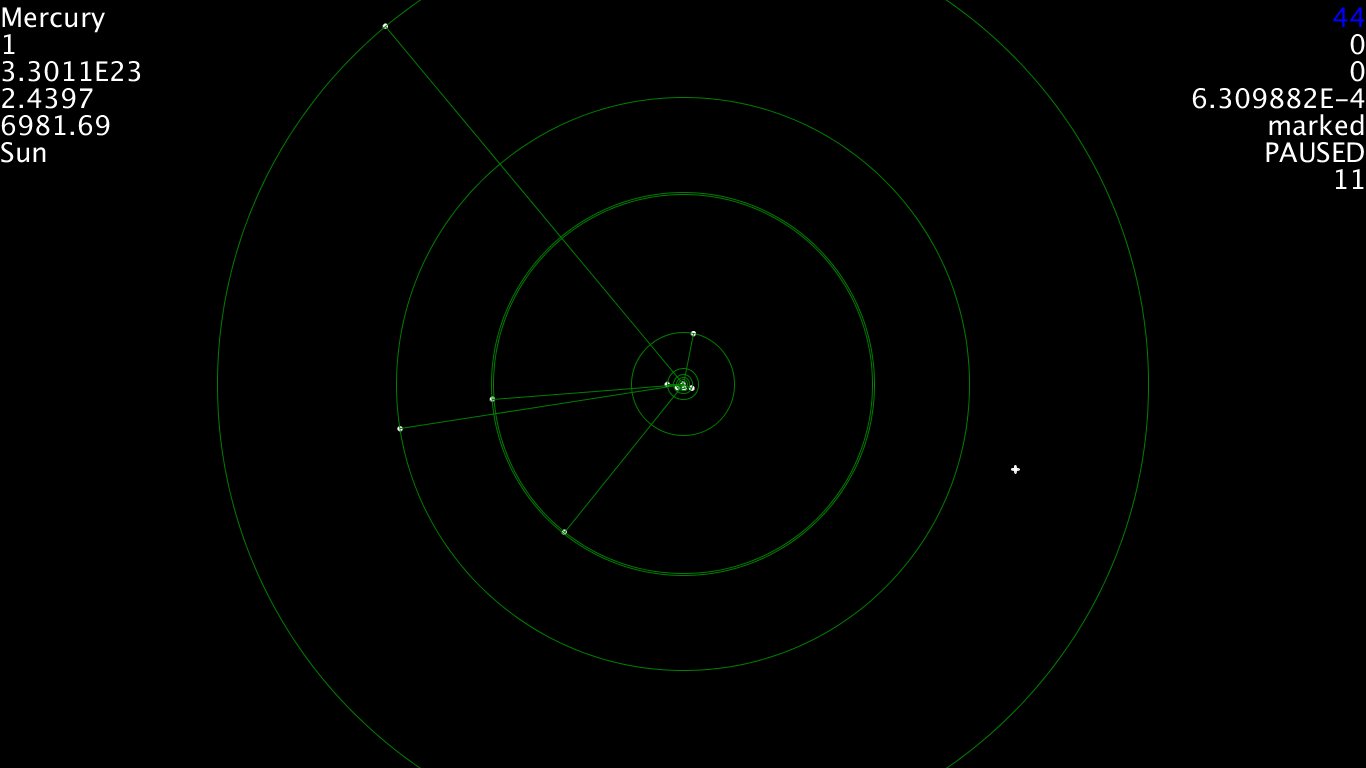


Рис.3.3 Відображення орбіт планет

Земля та Місяць взаємно рухаються навколо Сонця (Рис.3.4).

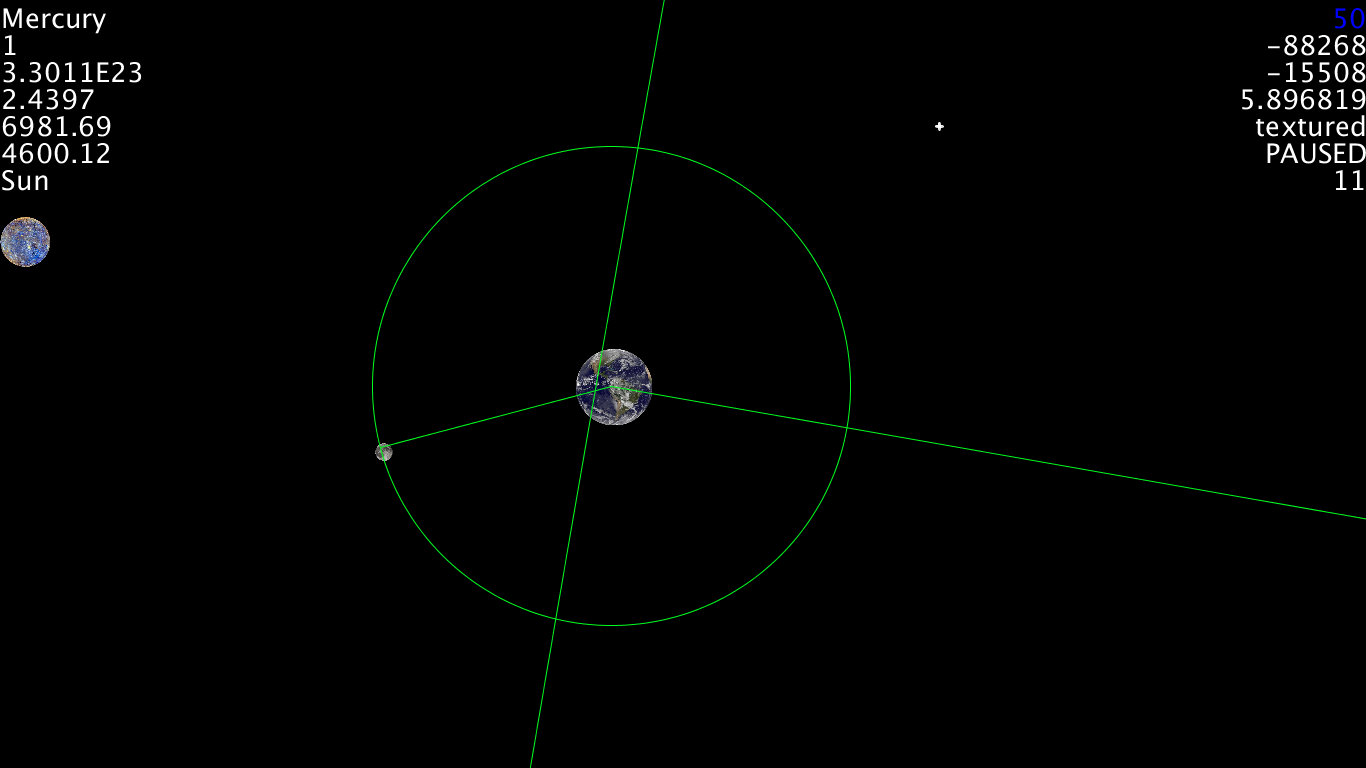
****

Рис.3.4 Земля та Місяць

Створена користувачем планета рухається навколо Сонця по еліптичній орбіті. Видно її афелій (Рис.2.5) та перигелій, що проходить “у міліметрі” від Сонця (Рис.2.6).

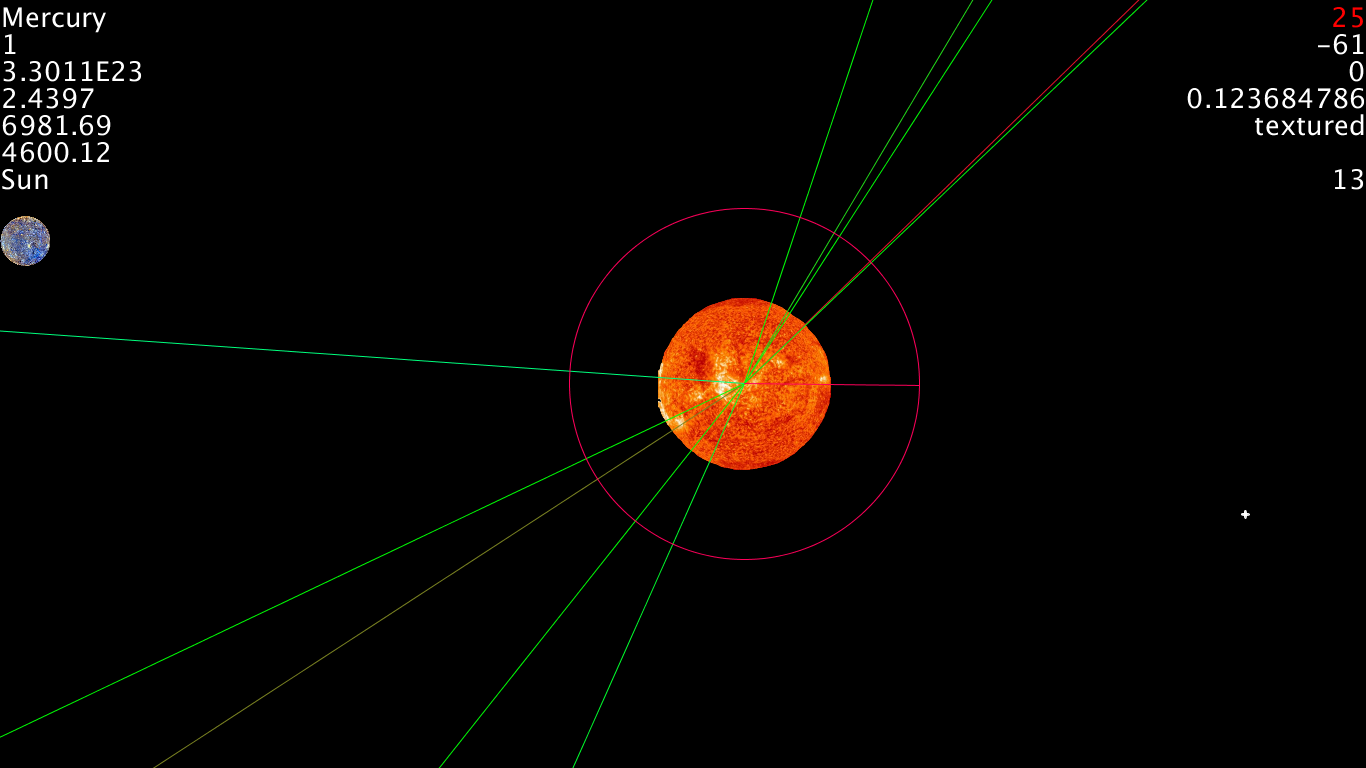


Рис.3.5 Сонце та планета у своєму апогеї

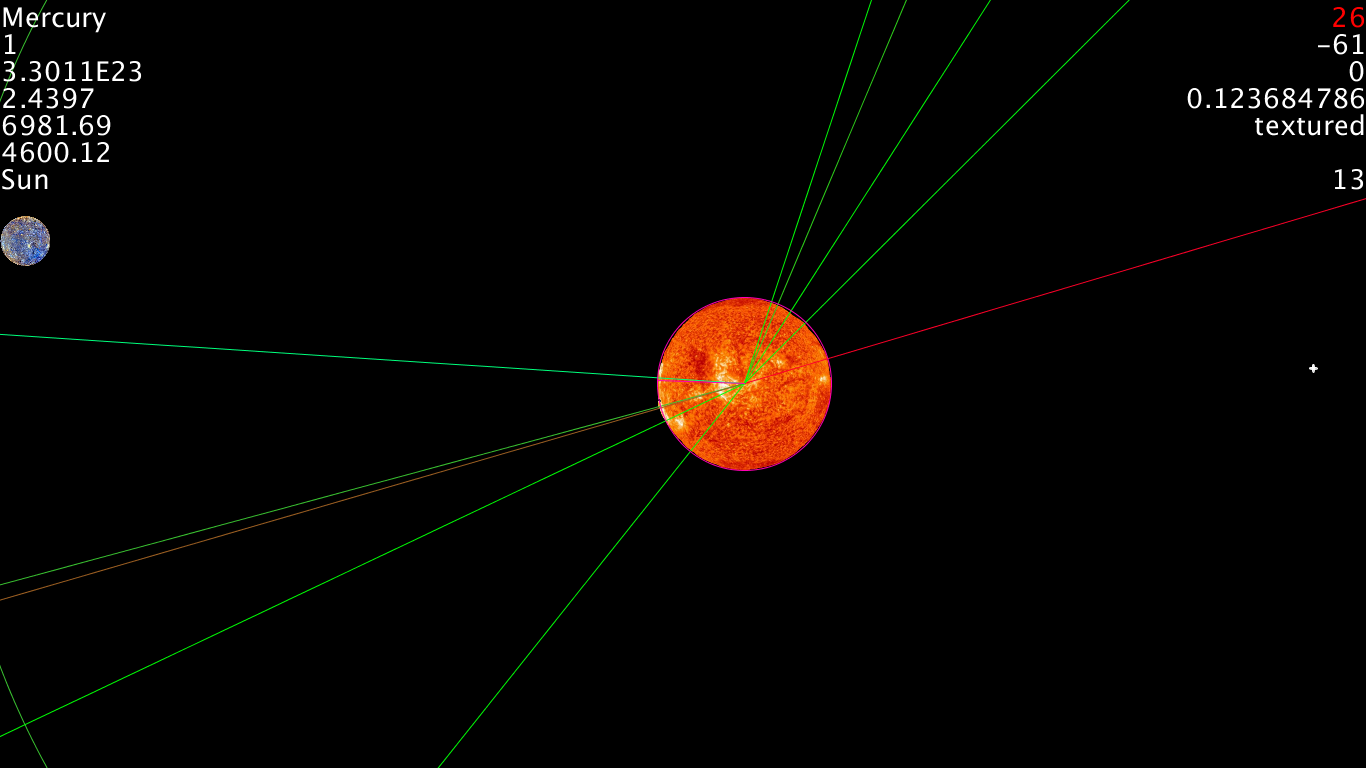


Рис.3.6 Сонце та планета у своєму перигеї

**ВИСНОВКИ**

Під час виконання проекту я набув навичок написання коду, відлагодження програми, аналізу проблем, що можуть виникати в процесі розробки програмного продукту. Набув значного досвіду написання коду з використанням ООП: навчився працювати з класом BigDecimal, що входить до стандартної бібліотеки Java, яка є реалізацією алгоритму довгої арифметики. Також написав власний клас BigVector, що використовує довгу арифметику при роботі з векторами. Нажаль ці два класи не вийшли з експериментальної версії ПЗ за браком часу.

Була ретельно досліджена та програмно описана модель зоряних систем та планет за вивченими законами фізики і математики. Космічні тіла притягуються за законом всесвітнього тяжіння, їх прискорення обраховуються відповідно до трьох законів Ньютона. Було реалізовано можливість зіткнення планет, зміни їх температури й вектора руху. Створено інтерфейс для взаємодії із симуляцією з можливістю завантаження й генерації власної зоряної системи, модифікації планет і супутників.

Використовуючи програму Processing, люди навчаються, експериментують, набувають знань у сфері астрономії і фізики. Крім того, зазначене ПЗ можна використовувати для точної симуляції зоряних систем.

Отже, Processing – це зручне і просте середовище програмування для візуалізації різних фізично-математичних моделей. Завдячуючи кросплатформності, програма написана у цьому середовищі може бути запущена на будь-якому пристрої, на який портована JVM.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. Царьков М. небесная механика [Електронний ресурс] / Михайло Царьков – Режим доступу до ресурсу: https://www.youtube.com/watch?v=fBC3cCsCEzA;
2. Alpha Centauri. Всё об орбитальной механике | Как запускают спутники [Електронний ресурс] / Alpha Centauri – Режим доступу до ресурсу: https://www.youtube.com/watch?v=YvbB4S5NiX8.
3. Shiffman D. The Nature of Code: Simulating Natural Systems with Processing / Daniel Shiffman;
4. Shiffman D. Learning Processing, Second Edition: A Beginner's Guide to Programming Images, Animation, and Interaction / Daniel Shiffman;
5. Reas C. Processing: A Programming Handbook for Visual Designers, Second Edition / C. Reas, B. Fry;
6. Reas C. Make: Getting Started with Processing, Second Edition / C. Reas, B. Fry;
7. *“* Schildt H. Java - The Complete Reference / Herbert Schildt., 2014. –Reas C. Reference. Processing was designed to be a flexible software sketchbook. [Електронний ресурс] / C. Reas, B. Fry, D. Shiffman. – 2019. – Режим доступу до ресурсу: https://processing.org/reference/.
8. Клімов О. Освой программирование играючи [Електронний ресурс] / Олександр Клімов. – 2019. – Режим доступу до ресурсу: http://developer.alexanderklimov.ru/;
9. ORACLE Java Documentation [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/;

**Додаток А**

Код класу Planet:

import static java.lang.Math.\*;

final double gravConst = 6.6740831 \* Math.pow(10, -25);

final float maxTemperature = 60000;

class Planet {

private String name = "";

private float x, y;

private float mass;

private float rad;

private float r = 255, g = 255, b = 255;

private float friction = 0.0001;

private float density;

private float temperature = 0;

private float specificHeatCapacity = 1;

private float producedHeat;

private float aphelion, perihelion;

private String master = "";

private PVector vect = new PVector(0, 0);

private boolean delete = false;

private PImage texture;

public Planet (float x, float y, float mass, float rad) {

this.x = x;

this.y = y;

this.mass = mass;

this.rad = rad;

this.density = 3F / 4F / PI \* mass / pow(rad, 3);

};

public Planet (Planet planet)

{

this.name = planet.name;

this.x = planet.x;

this.y = planet.y;

this.mass = planet.mass;

this.rad = planet.rad;

this.r = planet.r;

this.g = planet.g;

this.b = planet.b;

this.friction = planet.friction;

this.density = planet.density;

this.temperature = planet.temperature;

this.specificHeatCapacity = planet.specificHeatCapacity;

this.producedHeat = planet.producedHeat;

this.aphelion = planet.aphelion;

this.perihelion = planet.perihelion;

this.master = planet.master;

this.vect = planet.vect;

this.delete = planet.delete;

this.texture = planet.texture;

};

public float distance(float x, float y) {

return sqrt(pow(x - this.x, 2) + pow(y - this.y, 2));

};

public float distance(float xa, float ya, float xb, float yb) {

return sqrt(pow(xa - xb, 2) + pow(ya - yb, 2));

};

public float distance(Planet plan) {

return distance(plan.getX(), plan.getY());

};

public float distance(Planet a, Planet b) {

return distance(a.getX(), a.getY(), b.getX(), b.getY());

};

public boolean detectCollision(Planet a, Planet b) {

return distance(a, b) <= a.getRad() + b.getRad() && a != b;

};

public boolean detectCollision(Planet plan) {

return detectCollision(this, plan);

};

public void absorb(Planet plan) {

PVector result = new PVector(vect.x, vect.y);

result.sub(plan.getVector());

float energy = plan.getMass() \* (result.x \* result.x + result.y \* result.y) / 2;

addTemperature(energy \* (1 - friction) / (mass \* specificHeatCapacity));

this.addMass(plan.getMass());

PVector vect = new PVector(this.x - plan.getX(), this.y - plan.getY());

PVector planVect = plan.getVector();

planVect.normalize();

planVect.div(rad);

if(distance(this, plan) < distance(x, y, plan.getX() + planVect.x, plan.getY() + planVect.y))

vect.mult(-1);

vect.normalize();

vect.mult(energy \* friction / mass);

this.addVector(vect);

plan.delete();

};

public void collide(Planet a, Planet b) {

if(detectCollision(a, b))

{

if(a.mass > b.mass)

a.absorb(b);

else

b.absorb(a);

}

};

public void collide(Planet plan) {

collide(this, plan);

};

public float gravityForce(Planet a, Planet b) {

float dist = distance(a, b);

if((int)dist == 0) return 0;

dist \*= dist;

return (float)gravConst \* a.mass \* b.mass / dist;

};

public float gravityForce(Planet plan) {

return gravityForce(this, plan);

};

public void gravity(Planet plan) {

float dist = this.distance(plan);

if((int)dist == 0) return;

dist \*= dist;

PVector vect = new PVector(plan.x - this.x, plan.y - this.y);

vect.normalize();

vect.mult((float)gravConst \* plan.mass / dist);

this.addVector(vect);

};

public void addVector(PVector vect) {

this.vect.add(vect);

};

public void addMass(float mass) {

this.mass += mass;

this.rad = (float)Math.cbrt(3F / 4F / PI \* this.mass / density);

};

public void addTemperature(float temper) {

temperature += temper;

if(temperature >= 2000)

{

if(temperature >= 60000) { r = 127; g = 170; b = 255; }

else

{

if(temperature <= 3500) { r = 255; g = 0; b = 0; }

else if(temperature <= 5000) { r = 255; g = 128; b = 0; }

else if(temperature <= 6000) { r = 255; g = 255; b = 0; }

else if(temperature <= 7500) { r = 255; g = 245; b = 230; }

else if(temperature <= 10000) { r = 255; g = 255; b = 255; }

else if(temperature <= 30000) { r = 200; g = 215; b = 255; }

else if(temperature <= 60000) { r = 150; g = 170; b = 255; }

}

}

else { r = 255; g = 255; b = 255; }

};

public void move() {

x += vect.x;

y += vect.y;

};

public void show() {

tint(r, g, b);

image(texture, x - rad, y - rad, rad \* 2, rad \* 2);

};

public void mark(float zoom) {

stroke(r, g, b);

fill(r, g, b);

ellipse(x, y, 5 / zoom, 5 / zoom);

};

public float getSpeed() {

return sqrt(vect.x \* vect.x + vect.y \* vect.y);

};

public void setName(String name) { this.name = name; };

public void setX(float x) { this.x = x; };

public void setY(float y) { this.y = y; };

public void setPos(float x, float y) { this.setX(x); this.setY(y); };

public void setMass(float mass) { this.mass = 0; addMass(mass); };

public void setRad(float rad) { this.rad = rad; this.density = 3F / 4F / PI \* mass / pow(rad, 3); };

public void setFriction(float friction) { this.friction = friction; };

public void setDensity(float density) { this.density = density; this.rad = (float)Math.cbrt(3F / 4F / PI \* this.mass / density); };

public void setAphelion(float aphelion) { this.aphelion = aphelion; };

public void setPerihelion(float perihelion) { this.perihelion = perihelion; };

public void setVector(float x, float y) { this.vect = new PVector(x, y); };

public void setVector(PVector vect) { this.vect = vect; };

public void setTemperature(float temperature) { this.temperature = 0; addTemperature(temperature); };

public void setSpecificHeatCapacity(float specificHeatCapacity) { this.specificHeatCapacity = specificHeatCapacity; };

public void setTexture(PImage texture) { this.texture = texture.copy(); };

public void setMaster(String master) { this.master = master; };

public String getName() { return this.name; };

public float getX() { return this.x; };

public float getY() { return this.y; };

public float getMass() { return this.mass; };

public float getRad() { return this.rad; };

public float getFriction() { return this.friction; };

public float getDensity() { return this.density; };

public float getAphelion() { return this.aphelion; };

public float getPerihelion() { return this.perihelion; };

public PVector getVector() { return this.vect; };

public float getTemperature() { return this.temperature; };

public float getSpecificHeatCapacity() { return this.specificHeatCapacity; };

public PImage getTexture() { return this.texture; };

public String getMaster() { return this.master; };

public void delete() { this.delete = true; };

public boolean isDelete() { return this.delete; };

};

**Додаток Б**

Код класу Space:

import java.util.Iterator;

class Space {

private ArrayList<Planet> planets = new ArrayList<Planet>();

private ArrayList<Planet> planetDB = new ArrayList<Planet>();

public Space(String datafile) {

XML database;

database = loadXML(datafile + ".xml");

XML[] children = database.getChildren("planet");

int len = children.length;

for (int i = 0; i < len; i++)

{

String name = children[i].getContent();

float mass = children[i].getFloat("mass");

float rad = children[i].getFloat("rad") / (float)Math.pow(10, 6);

float temperature = children[i].getFloat("tempature");

float aphelion = children[i].getFloat("aphelion") / (float)Math.pow(10, 7);

float perihelion = children[i].getFloat("perihelion") / (float)Math.pow(10, 7);

String master = children[i].getString("master");

PImage texture = loadImage("textures/" + name.toLowerCase() + ".png");

Planet planet = new Planet(0, 0, mass, rad);

planet.setName(name);

planet.setTemperature(temperature);

planet.setAphelion(aphelion);

planet.setPerihelion(perihelion);

planet.setTexture(texture);

planet.setMaster(master);

planetDB.add(planet);

}

};

public void generate() {

int len = planetDB.size();

planets.add(new Planet(planetDB.get(len - 1)));

for(int i = 0; i < len - 1; i++)

{

Planet plan = new Planet(planetDB.get(i));

Planet master = new Planet(0, 0, 1, 1);

for(Planet p : planets)

if(p.getName().equals(plan.getMaster()))

master = new Planet(p);

float angle = random(0, TWO\_PI);

float r = plan.getAphelion();

float x = master.getX() + r \* cos(angle), y = master.getY() + r \* sin(angle);

plan.setPos(x, y);

PVector go;

go = new PVector(-1, (master.getX() - x) / (master.getY() - y));

go.normalize();

double startSpeed = Math.sqrt(gravConst \* master.getMass() / r \* (1 + plan.getPerihelion() / r));

go.mult((float)startSpeed);

plan.setVector(go);

plan.addVector(master.getVector());

planets.add(plan);

}

};

void tick() {

Iterator<Planet> iter = planets.iterator();

while(iter.hasNext() == true)

{

Planet a = iter.next();

a.move();

if(a.isDelete())

iter.remove();

else

{

String master = null;

float minHelion = 0, helion = 0;

for(Planet b : planets)

{

a.collide(b);

a.gravity(b);

b.gravity(a);

helion = a.distance(b);

if(helion < minHelion && !a.getMaster().equals("null"))

{

minHelion = helion;

master = b.getName();

}

}

if(master != null && !master.equals(a.getMaster()))

{

a.setMaster(master);

a.setPerihelion(minHelion);

a.setAphelion(minHelion);

}

}

}

};

void show() {

for(Planet planet : planets)

planet.show();

};

void mark(float zoom) {

for(Planet planet : planets)

planet.mark(zoom);

};

void orbit(float zoom) {

strokeWeight(1 / zoom);

noFill();

for(Planet planet : planets)

{

Planet master = null;

for(Planet mast : planets)

if(mast.getName().equals(planet.getMaster()))

master = mast;

if(master == null) continue;

float dist = planet.distance(master);

float ah = planet.getAphelion(), ph = planet.getPerihelion();

float speed = planet.getSpeed(), maXspeed = 100;

float r = map(dist, ph, ah, 255, 0);

float g = map(dist, ph, ah, 0, 255);

float b = map((speed > maXspeed) ? maXspeed : speed, 0, maXspeed, 0, 255);

stroke(r, g, b);

if(planet.distance(planet.getX(), planet.getY(), master.getX(), master.getY()) >= 1 / zoom)

{

line(planet.getX(), planet.getY(), master.getX(), master.getY());

ellipse(master.getX(), master.getY(), dist \* 2, dist \* 2);

}

}

strokeWeight(1);

};

void addPlanet(Planet plan) { planets.add(plan); };

public ArrayList<Planet> getPlanets() { return this.planets; };

public ArrayList<Planet> getPlanetDatabase() { return this.planetDB; };

};

**Додаток В**

Код основної частини програми:

int fps = 60;

float zoom = 0.05;

float seconds = 0.35;

boolean marked = false, orbit = false, pause = true, inter = true;

int timePress = 0;

float viewX = 0, viewY = 0;

float userX = 0, userY = 0;

Space world;

boolean first = true;

int dbindex = 0;

void setup()

{

fullScreen(JAVA2D);

frameRate(fps);

noCursor();

world = new Space("planets");

world.generate();

background(0);

};

void draw()

{

background(0);

translate(width / 2 - viewX, height / 2 - viewY);

scale(zoom);

if(!pause)

world.tick();

if(!marked)

world.show();

else

world.mark(zoom);

if(orbit)

world.orbit(zoom);

graphicface();

keyboard();

};

void keyboard()

{

if(!keyPressed) return;

if(key == 'z')

{

zoom \*= 1.05;

viewX \*= 1.05;

viewY \*= 1.05;

}

if(key == 'x')

{

zoom \*= 0.95;

viewX \*= 0.95;

viewY \*= 0.95;

}

if(key == 'c')

{

viewX = 0;

viewY = 0;

}

if((key == 'm' || key == 'M') && frameCount - timePress >= fps \* seconds)

{

marked = !marked;

timePress = frameCount;

}

if((key == 'p' || key == 'P') && frameCount - timePress >= fps \* seconds)

{

pause = !pause;

timePress = frameCount;

}

if((key == 'o' || key == 'O') && frameCount - timePress >= fps \* seconds)

{

orbit = !orbit;

timePress = frameCount;

}

if((key == 'i' || key == 'I') && frameCount - timePress >= fps \* seconds)

{

inter = !inter;

timePress = frameCount;

}

if(key == 'w')

viewY -= width / 100;

if(key == 'a')

viewX -= width / 100;

if(key == 's')

viewY += width / 100;

if(key == 'd')

viewX += width / 100;

if(key == '+' && frameCount - timePress >= fps \* seconds)

{

dbindex = (dbindex + 1) % world.getPlanetDatabase().size();

timePress = frameCount;

}

if(key == '-' && frameCount - timePress >= fps \* seconds)

{

dbindex = (dbindex + world.getPlanetDatabase().size() - 1) % world.getPlanetDatabase().size();

timePress = frameCount;

}

};

void graphicface() {

scale(1 / zoom);

translate(- width / 2, - height / 2);

if(inter)

{

float size = width / 50;

textSize(size);

int fpsdisp = (int)frameRate;

int x = (int)viewX;

int y = (int)viewY;

String viewMode;

int planetsCount = world.getPlanets().size();

if(fpsdisp >= 55)

fill(0, 255, 0);

else if(fpsdisp >= 40)

fill(0, 0, 255);

else if(fpsdisp >= 30)

fill(255, 255, 0);

else

fill(255, 0, 0);

if(marked)

viewMode = "marked";

else

viewMode = "textured";

text(fpsdisp, width - textWidth(String.valueOf(fpsdisp)) + viewX, size + viewY);

fill(255);

text(x, width - textWidth(String.valueOf(x)) + viewX, size \* 2 + viewY);

text(y, width - textWidth(String.valueOf(y)) + viewX, size \* 3 + viewY);

text(String.valueOf(zoom), width - textWidth(String.valueOf(zoom)) + viewX, size \* 4 + viewY);

text(viewMode, width - textWidth(viewMode) + viewX, size \* 5 + viewY);

if(pause) text("PAUSED", width - textWidth("PAUSED") + viewX, size \* 6 + viewY);

text(String.valueOf(planetsCount), width - textWidth(String.valueOf(planetsCount)) + viewX, size \* 7 + viewY);

Planet plan = new Planet(world.getPlanetDatabase().get(dbindex));

text(plan.getName(), viewX, size + viewY);

text(String.valueOf(dbindex + 1), viewX, size \* 2 + viewY);

text(String.valueOf(plan.getMass()), viewX, size \* 3 + viewY);

text(String.valueOf(plan.getRad()), viewX, size \* 4 + viewY);

text(String.valueOf(plan.getAphelion()), viewX, size \* 5 + viewY);

text(String.valueOf(plan.getPerihelion()), viewX, size \* 6 + viewY);

text(String.valueOf(plan.getMaster()), viewX, size \* 7 + viewY);

if(!marked) image(plan.getTexture(), viewX, size \* 8 + viewY, 50, 50);

}

if(mousePressed)

{

stroke(0, 0, 255);

fill(0, 0, 255);

ellipse(userX, userY, 4, 4);

stroke(0, 255, 0);

line(userX, userY, mouseX + viewX, mouseY + viewY);

stroke(255);

fill(255);

ellipse(mouseX + viewX, mouseY + viewY, 4, 4);

}

else

{

stroke(255);

fill(255);

ellipse(mouseX + viewX, mouseY + viewY, 2, 8);

ellipse(mouseX + viewX, mouseY + viewY, 8, 2);

}

};

void mousePressed()

{

userX = mouseX + viewX;

userY = mouseY + viewY;

};

void mouseReleased()

{

if(first) { first = false; return; };

if(mouseButton == RIGHT)

{

for(Planet planet : world.getPlanets())

if(planet.distance((userX - width / 2) / zoom, (userY - height / 2) / zoom) <= planet.getRad()) { planet.delete(); break; }

}

else if(mouseButton == LEFT)

{

PVector vect = new PVector(mouseX + viewX - userX, mouseY + viewY - userY);

Planet plan = new Planet(world.getPlanetDatabase().get(dbindex));

plan.setPos((userX - width / 2) / zoom, (userY - height / 2) / zoom);

plan.setVector(vect);

Planet master = null;

for(Planet mast : world.getPlanets())

if(plan.getMaster().equals(mast.getName())) { master = mast; break; }

if(master != null)

{

plan.setAphelion(plan.distance(master));

plan.setPerihelion(plan.distance(master));

}

world.addPlanet(plan);

}

};

**Додаток Г**

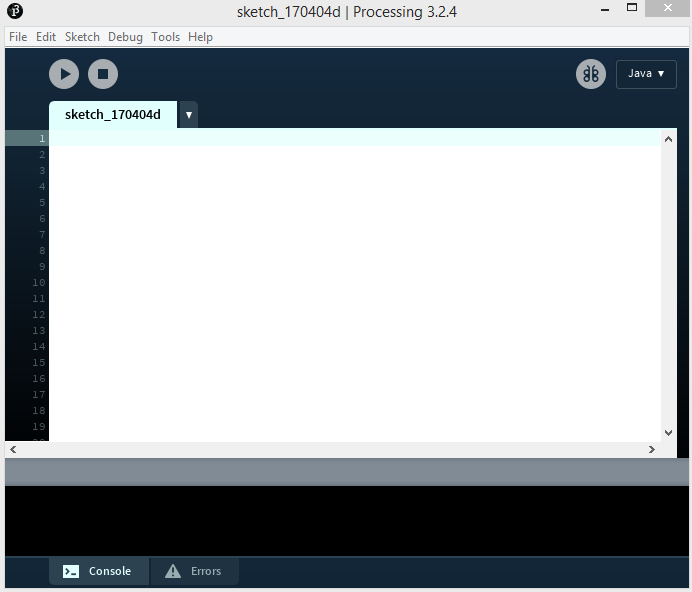


Рис.Г.1 Графічна оболонка середовища Processing



Рис.Г.2 Логотип Processing