Міністерство освіти і науки України

Департамент освіти і науки Вінницької облдержадміністрації

Вінницьке територіальне відділення МАН України

Відділення: «Комп’ютерні науки»

Секція: «Мультимедійні системи, навчальні програми та ігрові програми»

**МУЛЬТИМЕДІЙНИЙ ДОДАТОК ДЛЯ ЕМУЛЮВАННЯ ЗОРЯНИХ СИСТЕМ**

Роботу виконав:

дійсний член МАН

Мальований Денис Олегович,

учень 10 класу

комунального закладу

“Вінницький технічний ліцей”

Науковий керівник:

Єфременюк Ніна Василівна,

учитель інформатики

комунального закладу

“Вінницький технічний ліцей ”,

учитель-методист

**АНОТАЦІЯ**

Мальований Денис Олегович учень 10 класу комунального закладу “Вінницький технічний ліцей”

**МУЛЬТИМЕДІЙНИЙ ДОДАТОК ДЛЯ ЕМУЛЮВАННЯ ЗОРЯНИХ СИСТЕМ**

Науково-дослідна робота присвячена створенню мультимедійного кросплатформного додатку для емулювання зоряних систем та популяризації технічних наук, зокрема астрономії, фізики, математики та інформатики. Програму можна використовувати як для гри та дослідження Всесвіту, так і для наукових симуляцій. ПЗ позиціюється як одно- та багатокористувацька гра.

**Основна мета** **–** створення кросплатформного мультимедійного додатка для демонстрації руху й взаємодії космічних тіл, дослідження орбітальної механіки, наукових експериментів, а також гри із друзями.

**Об’єкт дослідження** **–** технології й методи створення ПЗ для фізичного моделювання.

**Предмет дослідження** **–** створення прикладного ПЗ для емулювання зоряних систем.

**Практична цінність** **–** можливість використання програми для навчання, експериментів і розваг.

Додаток **аналогів не має**.

**Ключові слова:** зоряні системи, кросплатформність, мультипотоковість, мережеві технології, довга арифметика, ООП, OpenGL, C++, Processing, Java.

**МОТИВАЦІЙНИЙ ЛИСТ**

Шановні члени журі!

Мені з раннього дитинства було цікаво – що, чому та як працює, зокрема як влаштований космос. Я мав більше десятка різних енциклопедій. Космічна тематика була однією з моїх найулюбленіших, разом із робототехнікою та комп’ютерною інженерією.

Три роки тому, влітку, я зацікавився програмуванням, зокрема мовою С++. Тоді я зрозумів – все намріяне можна реалізувати власноруч.

Згодом мені сподобалась гра “Kerbal Space Program”, де я провів не один десяток годин. У ній можна керувати спроєктованою власноруч технікою – наземною, льотною та космічною. Дуже цікаво реалізована система гравітаційних маневрів та керування часом.

Приблизно в той самий час натрапив у YouTube на відео Михайла Царькова про генетичні алгоритми мовою Pixilang. Відтак я почав слідкувати за його творчістю. Пізніше він виклав відео про орбітальну механіку. Автор показав, як виглядав би наш світ, якби закони фізики були б інші. Тоді я вперше побачив як відбуваються обрахунки в подібних додатках.

Рік тому ми із другом побачили ще один канал на YouTube – “Coding Train”. У ньому автор реалізує дуже прості й цікаві графічні програми в середовищі Processing. Мій товариш одразу ж завантажив його, що зацікавило й мене. Ми почали експериментувати й робити величезну кількість додатків. Тоді я згадав про свою пристрасть і почав втілювати її в життя.

Мета мого проєкту – зацікавити маси технічними науками, зокрема астрономією, фізикою, інформатикою та математикою, адже усе це мені було необхідне для моделювання та реалізації.

У майбутньому я бачу себе розробником ПЗ, спрямованого на комп’ютерні мережі, системні та ігрові додатки.

Минуло року я брав участь у МАН зі схожою тематикою, проте наразі програма змінена повністю. Я використав досвід, який здобув за цей час.

У цьому навчальному році збірна команда з моїх друзів брала участь у хакатоні “NASA International Space Apps Challenge 2019”. Ми писали дещо схожу програму. Нам вдалося побачити багато чудових робіт. Це мене надихнуло на продовження проєкту.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ПІБ підпис

ЗМІСТ

[**ВСТУП 6**](#_Toc34168908)

[**РОЗДІЛ 1 ВИБІР МОВИ І СЕРЕДОВИЩА ПРОГРАМУВАННЯ 9**](#_Toc34168909)

[**РОЗДІЛ 2 СТВОРЕННЯ МОДЕЛІ 10**](#_Toc34168910)

[**2.1. Структура програми 10**](#_Toc34168911)

[**2.2. XML-файли 12**](#_Toc34168912)

[**РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ІНТЕРФЕЙСУ 13**](#_Toc34168913)

[**ВИСНОВКИ 17**](#_Toc34168914)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 18**](#_Toc34168915)

[**ДОДАТОК А 20**](#_Toc34168916)

# ВСТУП

Ідеєю науково-дослідної роботи є створення програмного забезпечення для дослідження зоряних систем та взаємодії їх компонентів, що буде корисним як для звичайного користувача, так і науковця в даній галузі.

Дослідження 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-дослідної роботи є створення кросплатформного мультимедійного додатка для демонстрації руху й взаємодії космічних тіл, дослідження орбітальної механіки, наукових експериментів, а також гри із друзями.

**Об’єктом дослідження** єтехнології й методи створення ПЗ для фізичного моделювання.

**Предметом дослідження** є створення прикладного ПЗ для емулювання зоряних систем.

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

**Очікувані результати виконання роботи:**

* дослідити математичну модель взаємодії космічних тіл та її закони;
* покращити вміння роботи з графічною бібліотекою OpenGL на С++ та Processing (Java);
* покращити вміння роботи із системою контролю версій Git на базі веб-платформи gitlab.com;
* покращити вміння застосовувати ООП;
* покращити вміння застосовувати мультипотокові застосунки;
* покращити вміння застосовувати мережеві технології;
* покращити вміння застосовувати алгоритми довгої арифметики;
* покращити вміння розробки кросплатформних додатків мовою програмування С++;
* розробити прикладну програму для мультимедійного зображення дослідженої моделі й взаємодії із нею.

Прикладом для створення проєкту стала гра Kerbal Space Program. На відміну від неї, даний проєкт не призначений для створення власної льотної техніки та виконання місій, має менше різноманітних механік, проте дозволяє досліджувати зоряні системи разом із друзями, у мультиплеєрі. Крім того, мені стало в пригоді відео Михайла Царькова, що вказане у списку використаних джерел.

**Теоретичні відомості для побудови моделі**

Для побудови фізично-математичної моделі проєкту 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;

З наукових джерел візьмемо:

* Кеплерові елементи орбіти;
* Закони Кеплера (наслідки законів Ньютона).

Використаємо три закониНьютона:

* *“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 в інерціальних системах відліку, прямопропорційне рівнодійній силі, що діє на тіло, і обернено пропорційне 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 обернено пропорційна 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 (перицентром)**, а найдальша від нього точка — **афелієм (апоцентром)**.

Для задання початкових векторів положення та руху небесних тіл використаємо **Кеплерові елементи орбіти**, які конвертуємо в **декартові вектори** за алгоритмом, що вказаних у посиланнях.

# РОЗДІЛ 1 ВИБІР МОВИ І СЕРЕДОВИЩА ПРОГРАМУВАННЯ

Для прототипування архітектури ПЗ я використав мову й середовище програмування Processing, що базується 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:

* легке й просте;
* містить вбудовану графічну бібліотеку OpenGL, проте назви й використання функцій значно простіші;
* кросплатформне;
* швидке;
* гнучке.

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, тому він не залежить від виконавчої платформи, а завдяки численним оптимізаціям, програми на Java працюють з чудовою швидкодією для властивого рівня абстракції.

Для кінцевої реалізації архітектури ПЗ використана мова програмування C++ разом із графічною бібліотекою OpenGL. Такий варіант використовує в рази менше пам’яті, має в рази більшу швидкодію, а також графічні покращення. Крім того, архітектура коду має кращий вигляд, оскільки Processing багато в чому обмежує розробника.

# РОЗДІЛ 2 СТВОРЕННЯ МОДЕЛІ

## **2.1. Структура програми**

Архітектура ПЗ складається з таких класів: **Interface** (**Camera**, **Telescope**,  **ButtonServer** (**Button** всередині), **KeyServer** (**Key** всередині), **SoundtrackThread** та **TextureLoaderThread** всередині), **Space** (**Body** всередині**)**, **Client** та **Server**. Графічно її можна представити так (Рис. 2.1).

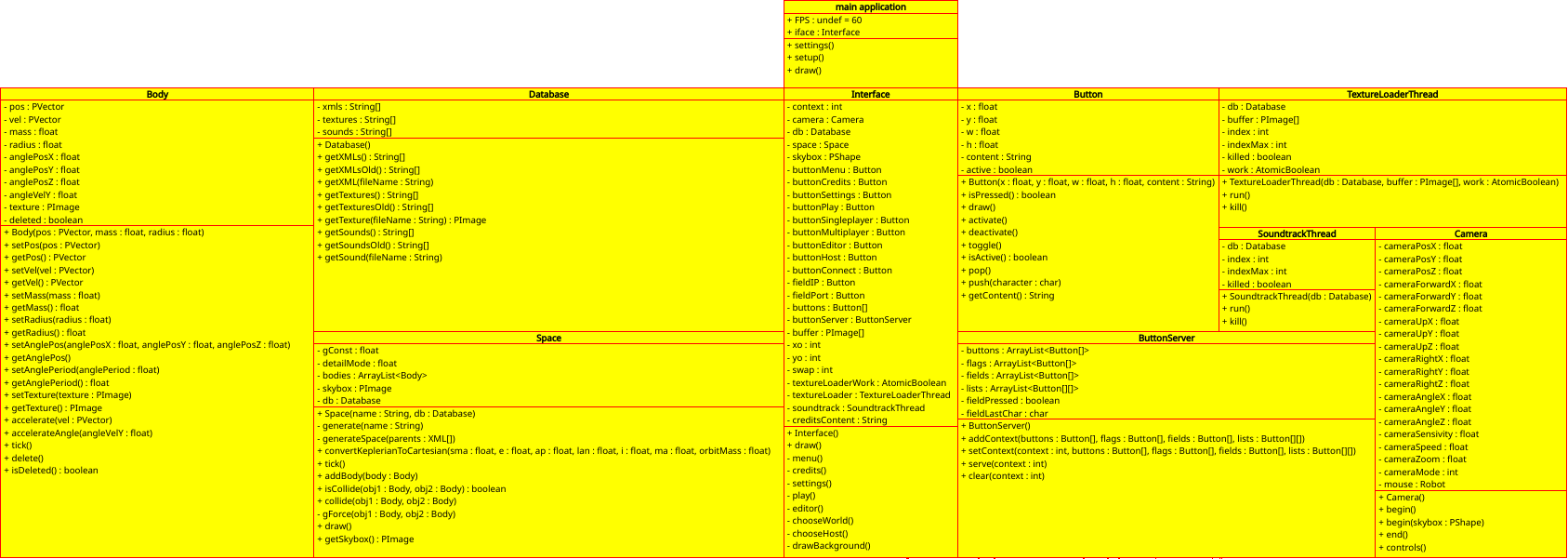


Рис.2.1 UML-діаграма. Взаємодія класів

Клас **Space** створений для керування симуляцією. Містить підклас **Body** – модель небесного тіла. Найцікавішим методом класу **Space** є **generate()**, який викликає **generateSpace()** – рекурсивна функція генерування об’єкта класу **Body** з файлу XML. Останній метод, своєю чергою, викликає **convertKeplerianToCartesian()**, що конвертує Кеплерові елементи орбіти в декартові вектори.

Клас **Database** створений для роботи зі статичними файлами: текстури, аудіо та XML. Цікаво, що для збільшення швидкодії об’єкт класу може запам’ятовувати список файлів.

Клас **Interface** створений для взаємодії з користувачем. Його підклас **Button** реалізує кнопку та поле введення, а **ButtonServer** обслуговує ці кнопки. Аналогічно **Key** (клавіша) та **KeyServer**. Клас **Camera** реалізує камеру, **Telescope** – телескоп, а **SoundtrackThread** та **TextureLoaderThread** – потоки програвання музики та завантаження текстур для анімованого фону.

Клас **Client** створений для комунікації у комп’ютерних мережах, тобто багатокористувацької гри.

Клас **Server** створений для керування багатокористувацькою грою. Він наслідує **Client**.

Код деяких класів наведений у додатку А.

Перевірка показала надійну й стабільну роботу програми. Отже, класи працюють правильно і взаємодіють між собою коректно.

## **2.2. XML-файли**

Початкові дані про світи зберігаються у відповідних XML-файлах ієрархічно. Тобто між тегами одного небесного тіла розташовується інформація про його супутники. Приклад XML-файлу Сонячної системи (Рис. 2.2).







Рис. 2.2

# РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ІНТЕРФЕЙСУ

У головній функції програми створюється об’єкт класу **Interface**, який реалізує головне меню програми: гра, редактор та титри. За вибраним користувачем файлом він створює об’єкт класу **Space**, що генерує ігровий світ, створюючи об’єкти класу **Body**. Цей же клас ініціалізує сервер для гри у мультиплеєрі. Після завершення гри чи редагування світу можна повернутися в головне меню.

Після запуску гра зустрічає Вас у головному меню з анімованим фоном (Рис. 3.1). Ви можете перейти до екрана вибору світу (Рис. 3.3) та хоста (Рис. 3.4) для гри або редагування, а також титрів (Рис. 3.2). Ви можете обрати режим поодинокої або багатокористувацької гри.

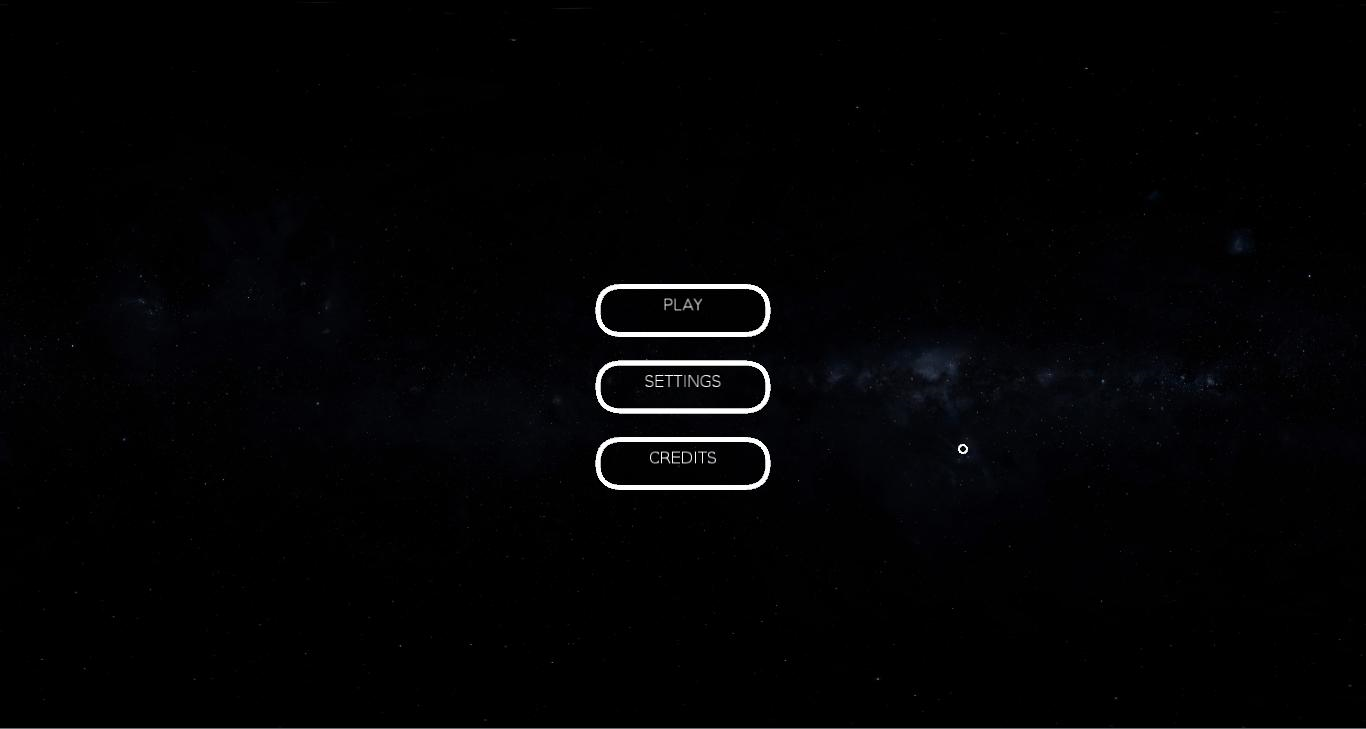


Рис. 3.1 Головне меню

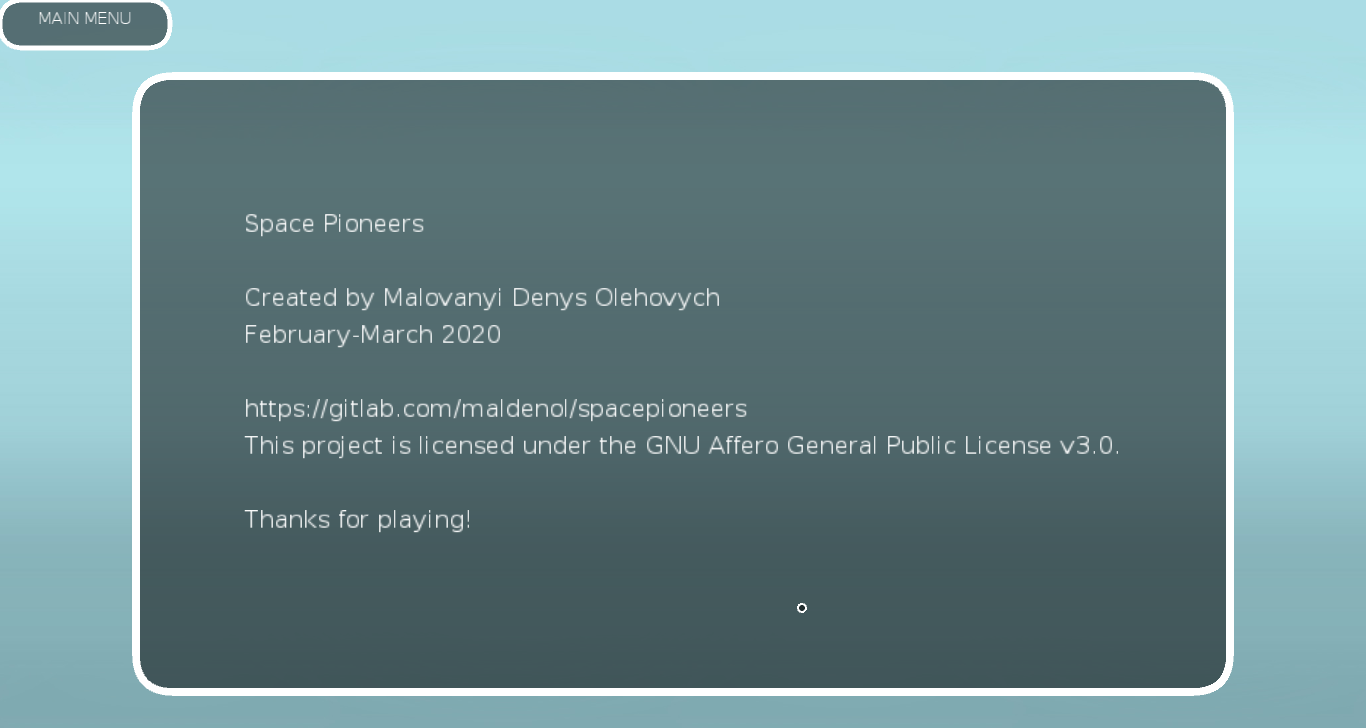


Рис. 3.2 Титри

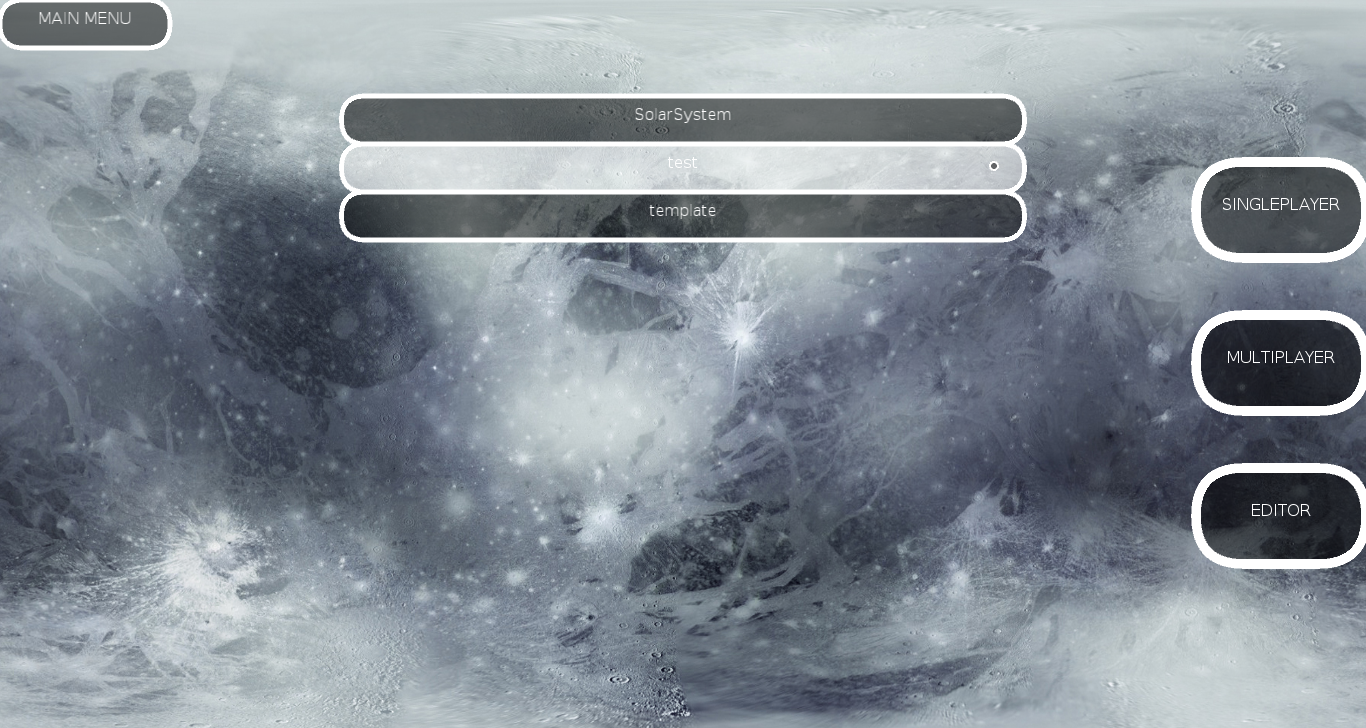


Рис. 3.3 Екран вибору світу

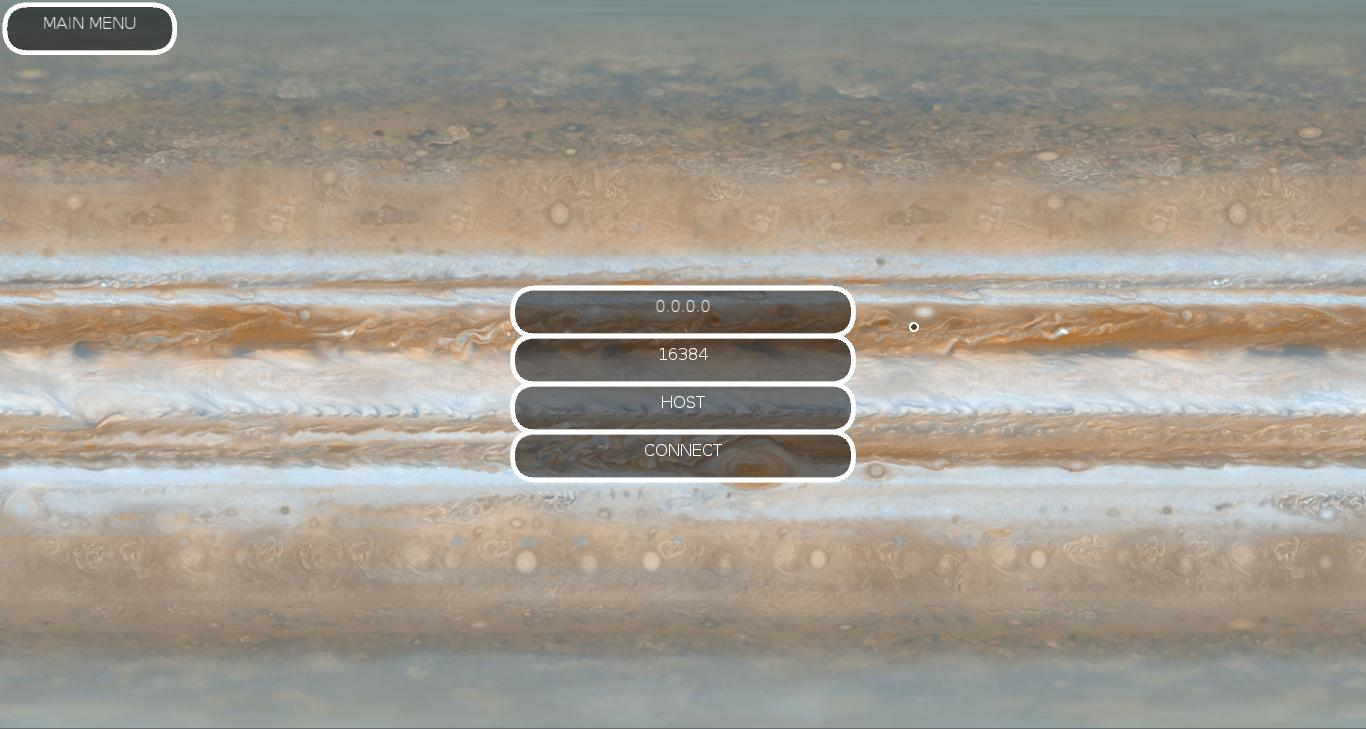


Рис. 3.4 Екран вибору хоста

Керування положенням камери спостерігача відбувається клавішами ‘w’, ‘s’, ‘a’, ‘d’, SPACE, SHIFT, ‘e’, ‘q’ для руху вперед, назад, вліво, вправо, вгору, вниз, нахилу вліво й вправо відповідно. Клавішею ‘c’ можна перемикатись між різними видами: вільна камера, об’єктив телескопа, від третьої особи. Кнопка ‘m’ перемикає зображення текстур планет, ‘i’ – інтерфейсу, а ‘o’ – орбіт планет. Кнопка ‘p’ зупиняє і продовжує хід симуляції. У правому верхньому кутку показано зверху донизу відповідно FPS (кількість кадрів за секунду).

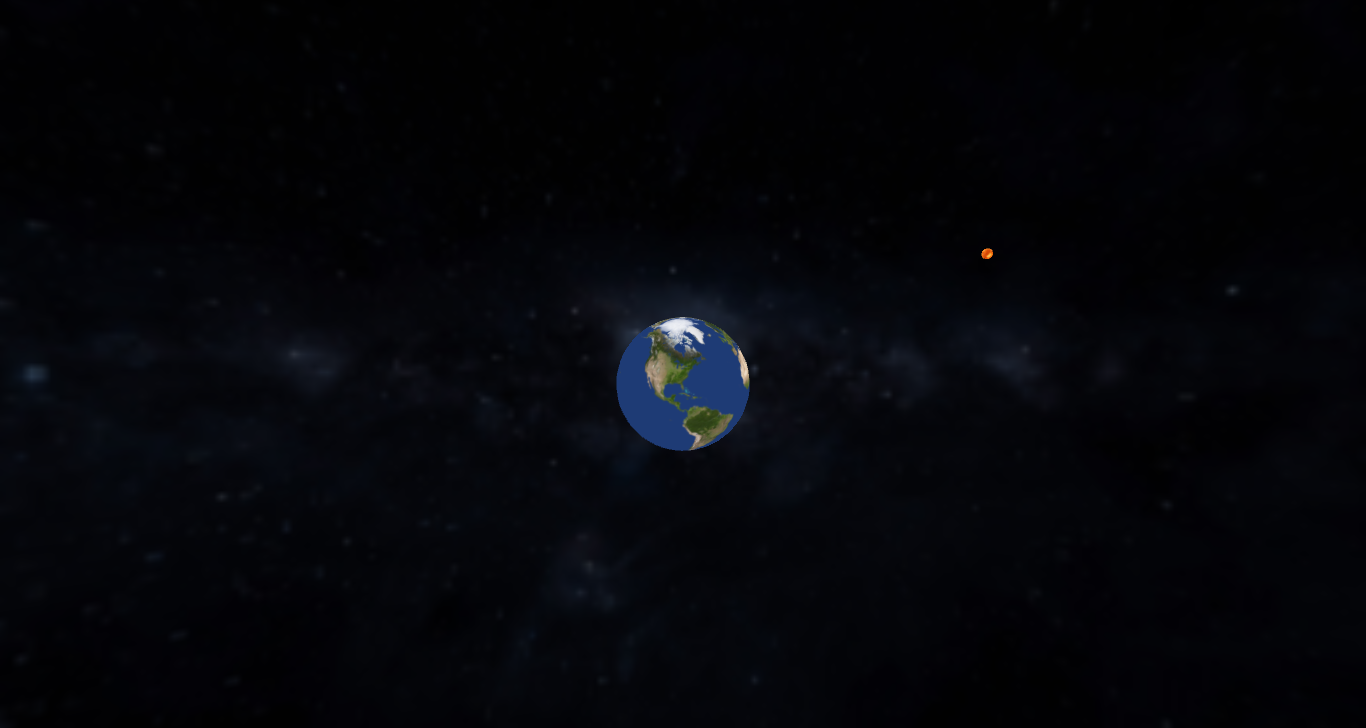


Рис. 3.5 Приклад роботи програми



Рис. 3.6 Приклад роботи програми

# ВИСНОВКИ

Під час виконання проєкту я набув навичок дослідження фізичних та математичних моделей Всесвіту, розробки та налагодження програм, здобув досвід роботи з СКВ Git, мережевими технологіями, довгою арифметикою, мультимедійними бібліотеками, мовами програмування різного рівня абстракції.

Була ретельно досліджена та програмно емульована модель зоряних систем. Створено інтерфейс для взаємодії з симуляцією, з можливістю завантаження, генерації й редагування власної зоряної системи. Реалізовано можливість керування власним космічним телескопом та гри у режимі мультиплеєру.

Було створено такі класи: Interface, Camera, Button, ButtonServer, SoundtrackThread, TextureLoaderThread, Space, Body, Client та Server. Перевірка показала надійну й стабільну роботу програми. Отже, класи працюють правильно і взаємодіють між собою коректно.

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

Додаток аналогів не має.

Результат виконаної роботи можна завантажити за посиланням <https://gitlab.com/maldenol/spacepioneers/>. Початковий код програми ліцензований під GNU AGPLv3.

# 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. Schwarz R. Keplerian Orbit Elements → Cartesian State Vectors [Електронний ресурс] / René Schwarz // 1. – 2017. – Режим доступу до ресурсу: <https://downloads.rene-schwarz.com/download/M001-Keplerian_Orbit_Elements_to_Cartesian_State_Vectors.pdf>.
4. [Електронний ресурс] / C. Reas, B. Fry, D. Shiffman. – 2019. – Режим доступу до ресурсу: <https://processing.org/reference/>.
5. Клімов О. Освой программирование играючи [Електронний ресурс] / Олександр Клімов. – 2019. – Режим доступу до ресурсу: <http://developer.alexanderklimov.ru/>.
6. ORACLE Java Documentation [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/.

# ДОДАТОК А

**Функція класу Space для переводу Кеплерових елементів орбіти в декартові вектори**

public float[][] convertKeplerianToCartesian(float sma, float e, float ap, float lan, float i, float ma, float orbitMass) {

float ea, ta, distance, prePosX, prePosY, preVelX, preVelY, posX, posY, posZ, velX, velY, velZ;

ea = ma;

float diff = abs(ma - (ea - e \* sin(ea))), lastDiff, lastEA;

while(true) {

lastEA = ea;

lastDiff = diff;

ea = ea - (ea - e \* sin(ea) - ma) / (1 - e \* cos(ea));

diff = abs(ma - (ea - e \* sin(ea)));

if(diff == 0.0)

break;

if(diff > lastDiff) {

ea = lastEA;

break;

}

}

ta = 2 \* atan2(sqrt(1 + e) \* sin(ea / 2), sqrt(1 - e) \* cos(ea / 2));

distance = sma \* (1 - e \* cos(ea));

prePosX = distance \* cos(ta);

prePosY = distance \* sin(ta);

preVelX = sqrt(this.gConst \* sma \* orbitMass) / distance \* -sin(ea);

preVelY = sqrt(this.gConst \* sma \* orbitMass) / distance \* sqrt(1 - e\*e) \* cos(ea);

posX = prePosX \* (cos(ap) \* cos(lan) - sin(ap) \* cos(i) \* sin(lan)) - prePosY \* (sin(ap) \* cos(lan) + cos(ap) \* cos(i) \* sin(lan));

posY = prePosX \* (cos(ap) \* sin(lan) + sin(ap) \* cos(i) \* cos(lan)) - prePosY \* (sin(ap) \* sin(lan) - cos(ap) \* cos(i) \* cos(lan));

posZ = prePosX \* (sin(ap) \* sin(i)) + prePosY \* (cos(ap) \* sin(i));

velX = preVelX \* (cos(ap) \* cos(lan) - sin(ap) \* cos(i) \* sin(lan)) - preVelY \* (sin(ap) \* cos(lan) + cos(ap) \* cos(i) \* sin(lan));

velY = preVelX \* (cos(ap) \* sin(lan) + sin(ap) \* cos(i) \* cos(lan)) - preVelY \* (sin(ap) \* sin(lan) - cos(ap) \* cos(i) \* cos(lan));

velZ = preVelX \* (sin(ap) \* sin(i)) + preVelY \* (cos(ap) \* sin(i));

return new float[][]{{posX, posY, posZ}, {velX, velY, velZ}};

}

**Рекурсивна функція класу Space для генерування об’єктів класу Body**

private void generateSpace(XML[] parents) {

float posX, posY, posZ, velX, velY, velZ, angPosX, angPosY, angPosZ, angPeriod, mass, radius;

float orbitMass, orbitPosX, orbitPosY, orbitPosZ, orbitVelX, orbitVelY, orbitVelZ;

float semiMajorAxis, eccentricity, argumentOfPeriapsis, longitudeOfAscendingNode, inclination, meanAnomaly;

String name;

PImage texture;

for (int i = 0; i < parents.length; i++) {

posX = parents[i].getFloat("posX");

posY = parents[i].getFloat("posY");

posZ = parents[i].getFloat("posZ");

velX = parents[i].getFloat("velX");

velY = parents[i].getFloat("velY");

velZ = parents[i].getFloat("velZ");

angPosX = radians(new Float(parents[i].getFloat("angPosX")));

angPosY = radians(parents[i].getFloat("angPosY"));

angPosZ = radians(parents[i].getFloat("angPosZ"));

angPeriod = parents[i].getFloat("angPeriod");

mass = parents[i].getFloat("mass");

radius = parents[i].getFloat("radius");

XML parent = parents[i].getParent();

orbitMass = parent.getFloat("mass");

orbitPosX = parent.getFloat("posX");

orbitPosY = parent.getFloat("posY");

orbitPosZ = parent.getFloat("posZ");

orbitVelX = parent.getFloat("velX");

orbitVelY = parent.getFloat("velY");

orbitVelZ = parent.getFloat("velZ");

semiMajorAxis = parents[i].getFloat("semiMajorAxis");

eccentricity = parents[i].getFloat("eccentricity");

argumentOfPeriapsis = radians(parents[i].getFloat("argumentOfPeriapsis"));

longitudeOfAscendingNode = radians(parents[i].getFloat("longitudeOfAscendingNode"));

inclination = radians(parents[i].getFloat("inclination"));

meanAnomaly = radians(parents[i].getFloat("meanAnomaly"));

name = parents[i].getString("name");

texture = this.db.getTexture(name);

texture.resize((int)(texture.width \* this.detailMode), 0);

Body body = new Body(posX, posY, posZ, mass, radius);

body.setAnglePos(angPosX, angPosY, angPosZ);

body.setAnglePeriod(angPeriod);

body.setTexture(texture);

body.setPos(posX, posY, posZ);

body.setVel(velX, velY, velZ);

if(orbitMass != 0.0) {

float[][] result = convertKeplerianToCartesian(semiMajorAxis, eccentricity, argumentOfPeriapsis, longitudeOfAscendingNode, inclination, meanAnomaly, orbitMass);

posX = result[0][0];

posY = result[0][1];

posZ = result[0][2];

velX = result[1][0];

velY = result[1][1];

velZ = result[1][2];

posX += orbitPosX;

posY += orbitPosY;

posZ += orbitPosZ;

body.setPos(posX, posY, posZ);

parents[i].setFloat("posX", posX);

parents[i].setFloat("posY", posY);

parents[i].setFloat("posZ", posZ);

velX += orbitVelX;

velY += orbitVelY;

velZ += orbitVelZ;

body.setVel(velX, velY, velZ);

parents[i].setFloat("velX", velX);

parents[i].setFloat("velY", velY);

parents[i].setFloat("velZ", velZ);

}

this.bodies.add(body);

generateSpace(parents[i].getChildren("body"));

}

}

**Підклас Button класу Interface**

private class Button {

private float x, y, w, h;

private String content;

private boolean active;

public Button(float x, float y, float w, float h, String content) {

this.x = x;

this.y = y;

this.w = w;

this.h = h;

this.content = content;

this.active = false;

}

public boolean isPressed() {

return (mousePressed && mouseX >= this.x && mouseX <= this.x + this.w && mouseY >= this.y && mouseY <= this.y + this.h);

}

public void draw() {

if(this.active)

fill(255, 127);

else

fill(0, 127);

stroke(255);

strokeWeight(this.h / 10);

rect(this.x, this.y, this.w, this.h, this.h / 2, this.h / 2, this.h / 2, this.h / 2);

if(this.active)

fill(0);

else

fill(255);

stroke(0);

strokeWeight(1);

textSize(16);

text(this.content, this.x + (this.w - textWidth(this.content)) / 2, this.y + this.h / 2);

noFill();

}

public void activate() {

this.active = true;

}

public void deactivate() {

this.active = false;

}

public void toggle() {

this.active = !this.active;

}

public boolean isActive() {

return this.active;

}

public void pop() {

if(this.content.length() >= 1)

this.content = this.content.substring(0, this.content.length() - 1);

}

public void push(char character) {

this.content += character;

}

public String getContent() {

return this.content;

}

}

**Підклас ButtonServer класу Interface**

private class ButtonServer {

private ArrayList<Button[]> buttons;

private ArrayList<Button[]> flags;

private ArrayList<Button[]> fields;

private ArrayList<Button[][]> lists;

private boolean fieldPressed;

private char fieldLastChar;

public ButtonServer() {

this.buttons = new ArrayList<Button[]>();

this.flags = new ArrayList<Button[]>();

this.fields = new ArrayList<Button[]>();

this.lists = new ArrayList<Button[][]>();

this.fieldPressed = false;

this.fieldLastChar = ' ';

}

public void addContext(Button[] buttons, Button[] flags, Button[] fields, Button[][] lists) {

this.buttons.add(buttons.clone());

this.flags.add(flags.clone());

this.fields.add(fields.clone());

this.lists.add(lists.clone());

}

public void setContext(int context, Button[] buttons, Button[] flags, Button[] fields, Button[][] lists) {

this.buttons.set(context, buttons.clone());

this.flags.set(context, flags.clone());

this.fields.set(context, fields.clone());

this.lists.set(context, lists.clone());

}

public void serve(int context) {

for(Button button : this.buttons.get(context)) {

button.draw();

if(button.isPressed()) {

button.activate();

for(Button b : this.buttons.get(context))

if(b != button)

b.deactivate();

}

}

for(Button flag : this.flags.get(context)) {

flag.draw();

if(flag.isPressed()) {

if(!this.fieldPressed)

flag.toggle();

this.fieldPressed = true;

} else

this.fieldPressed = false;

}

for(Button field : this.fields.get(context)) {

field.draw();

if(field.isPressed()) {

field.activate();

for(Button f : this.fields.get(context))

if(f != field)

f.deactivate();

}

if(field.isActive())

if(keyPressed) {

if(!this.fieldPressed || this.fieldLastChar != key) {

if(keyCode == BACKSPACE)

field.pop();

else if(key != CODED)

field.push(key);

this.fieldLastChar = key;

}

this.fieldPressed = true;

} else

this.fieldPressed = false;

}

for(Button[] list : this.lists.get(context)) {

for(Button element : list) {

element.draw();

if(element.isPressed()) {

element.activate();

for(Button e : list)

if(e != element)

e.deactivate();

}

}

}

}

public void clear(int context) {

for(Button button : this.buttons.get(context))

button.deactivate();

for(Button flag : this.flags.get(context))

flag.deactivate();

for(Button field : this.fields.get(context))

field.deactivate();

for(Button[] list : this.lists.get(context))

for(Button element : list)

element.deactivate();

}

}

**Функція класу Interface для зображення анімованого фону**

private void drawBackground() {

camera();

image(this.buffer[0], -width + this.xo, -height + this.yo);

image(this.buffer[0], this.xo, -height + this.yo);

image(this.buffer[1], -width + this.xo, this.yo);

image(this.buffer[1], this.xo, this.yo);

this.xo = (this.xo + 1) % width;

if(this.xo == 0 && this.swap == 0)

this.swap = 1;

if(this.swap == 1) {

this.yo = (this.yo + height / FPS) % height;

if(this.yo == 0) {

this.swap = 0;

this.buffer[1] = this.buffer[0];

this.buffer[0] = this.db.getTexture(this.db.getTexturesOld()[this.textureIndex]);

this.buffer[0].resize(width, height);

this.textureIndex = (this.textureIndex + 1) % this.textureIndexMax;

}

}

}