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 мультимедійного, ігрового та навчального ПЗ для фізичного моделювання.

**Предмет дослідження** **–** створення мультимедійного, ігрового та навчального 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. Згодом я знайшов канал Renesco Rocketman, де побачив цікаве пояснення роботи з векторами в комп’ютерних іграх.

Рік тому ми із другом побачили ще один канал на YouTube – “The 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 мультимедійного, ігрового та навчального ПЗ для фізичного моделювання.

**Предметом дослідження** є створення прикладного мультимедійного, ігрового та навчального 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;
* Здобути досвід роботи з графічною бібліотекою OpenGL на С++ та Processing (Java);
* покращити вміння роботи із системою контролю версій Git на базі вебплатформи GitLab (<https://gitlab.com>);
* покращити вміння роботи з ООП;
* покращити вміння роботи з потоками;
* покращити вміння роботи з мережевими технологіями;
* покращити вміння роботи з довгої арифметики;
* здобути досвід 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;
* кватерніони;

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 спрямована 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** для довільного тіла**)**, а найдальша від нього точка — **афелієм (апоцентром** для довільного тіла**)**.

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

# РОЗДІЛ 1 ВИБІР МОВИ І СЕРЕДОВИЩА ПРОГРАМУВАННЯ

Для прототипування архітектури ПЗ я використав середовище програмування Processing, що базується на мові Java і має вбудовану графічну бібліотеку OpenGL. Воно 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.

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, тому доводиться використовувати “милиці”.

# РОЗДІЛ 2 СТВОРЕННЯ МОДЕЛІ

## **2.1. Структура програми**

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

Рис.2.1 UML-діаграма. Взаємодія класів

Клас **Game** є “головним”, він викликає усі інші. Його підклас **Telescope** реалізує телескоп, а **SoundtrackThread** та **TextureLoaderThread** – потоки програвання музики та завантаження текстур для анімованого фону.

Клас **Space** створений для керування симуляцією. Містить підклас **Body** – модель небесного тіла, що наслідує **SphericalBody**.

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, а також хешувати.

Клас **Interface** створений для взаємодії з користувачем. Клас **Button** реалізує кнопку та поле введення, а **ButtonServer** обслуговує ці кнопки. Аналогічно **Key** (клавіша) та **KeyServer**. Клас **Camera** реалізує камеру.

Клас **Mathematics** є статичним і містить корисні математичні методи, а також статичний клас **Vector**.

Клас **Physics** є статичним і містить клас **SphericalBody**, що реалізує модель сферичного пружного фізичного тіла.

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

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

Найцікавішими функціями програми я вважаю Space.generateSpace, Mathematics.convertKeplerianToCartesian, Game.drawBackground, Space.gForce, Space.tick та Space.draw. Їх код наведений у додатку А.

Я вважаю, що застосування саме ООП для розробки є вдалим. Перевірка показала надійну й стабільну роботу програми. Отже, класи працюють правильно і взаємодіють між собою коректно.

Крім того, класи **Interface**, **Mathematics**, **Physics**, **Client** та **Server** я планую використати для розробки власного ігрового рушія в майбутньому.

Під час реалізації стало очевидним те, що використання стандартного типу float для моделювання таких точних систем, як зоряних, не дає достатньої точності. Тому використання алгоритмів довгої арифметики є необхідним.

Слід наголосити, що додаток саме емулює взаємодію компонентів зоряних систем. Тобто всі розрахунки між будь-якими космічними тілами відбуваються у кожен такт виконання за реальними фізичними законами: всесвітнього тяжіння та Ньютона. Саме у цьому полягає науковість проєкту.

## **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

Як виявилося, формат XML дуже вдалий для представлення ієрархічних систем (багаторівневих систем супутників у моєму випадку).

# РОЗДІЛ 3 ОПИС ПРОГРАМНОГО ІНТЕРФЕЙСУ

У головній функції програми створюється об’єкт класу **Game**, 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 зустрічаєЯк виявилося, формат XML дуже вдалий для представлення ієрархічних систем (багаторівневих систем супутників у моєму випадку). Вас у головному меню з анімованим фоном та саундтреком (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.

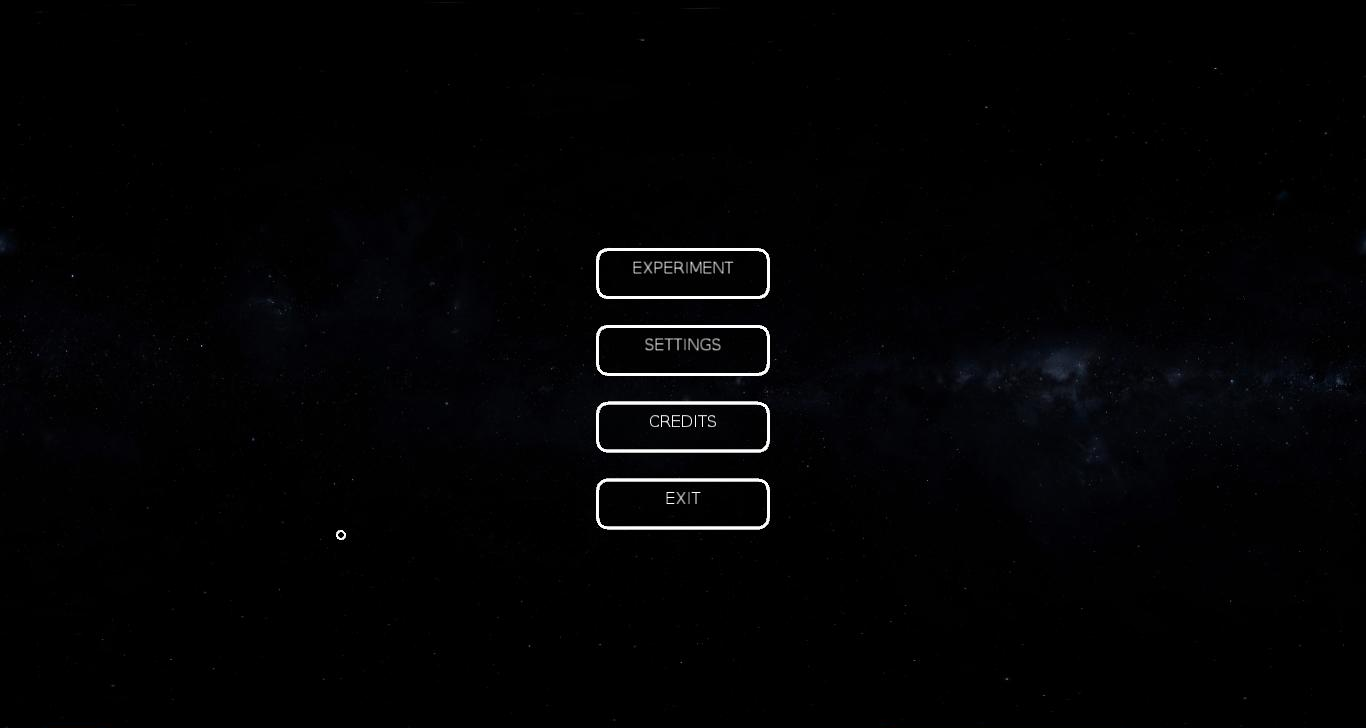


Рис. 3.1 Головне меню

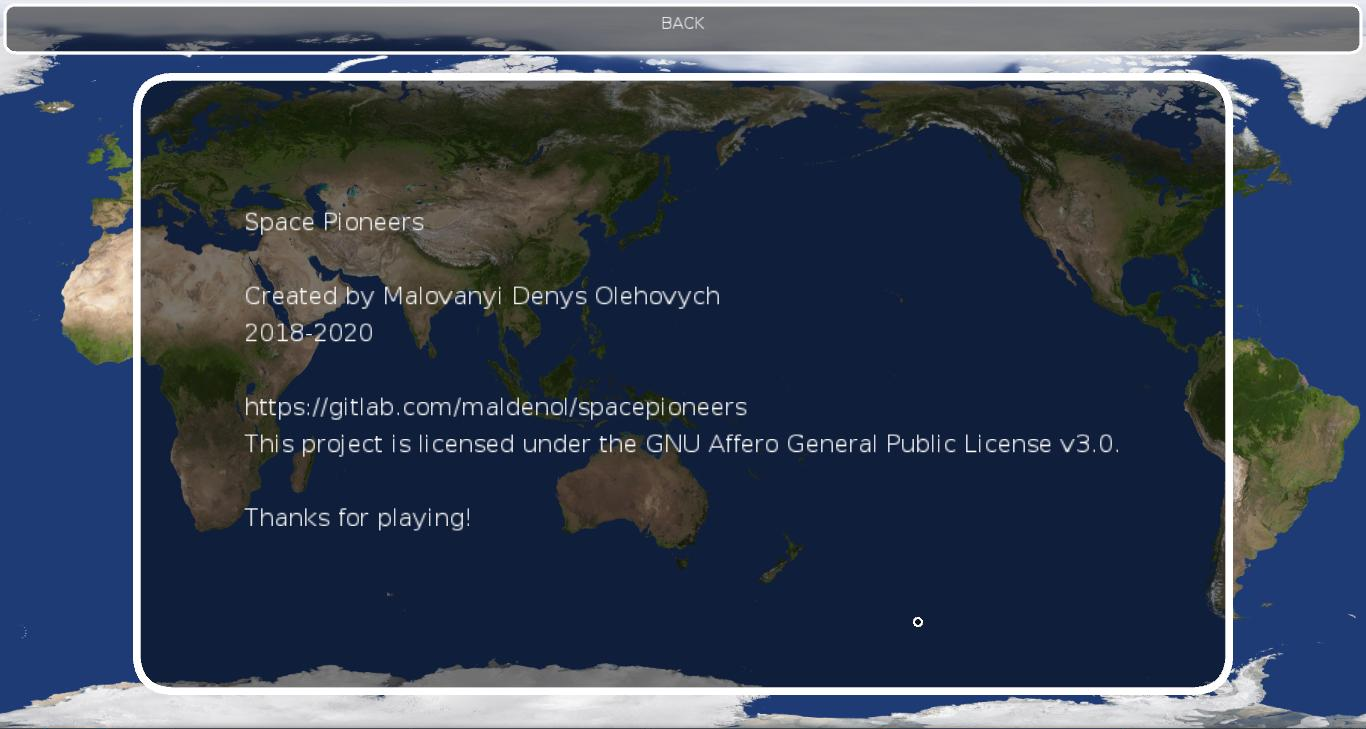


Рис. 3.2 Титри

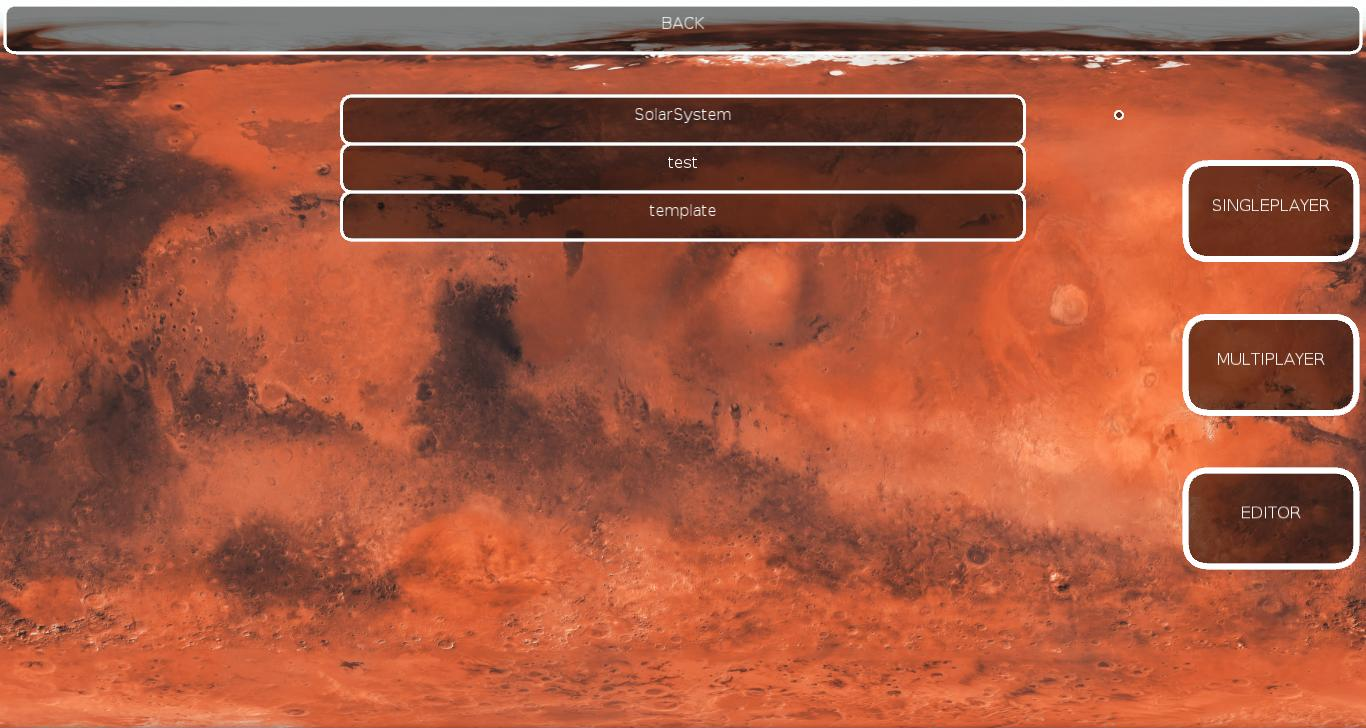


Рис. 3.3 Екран вибору світу

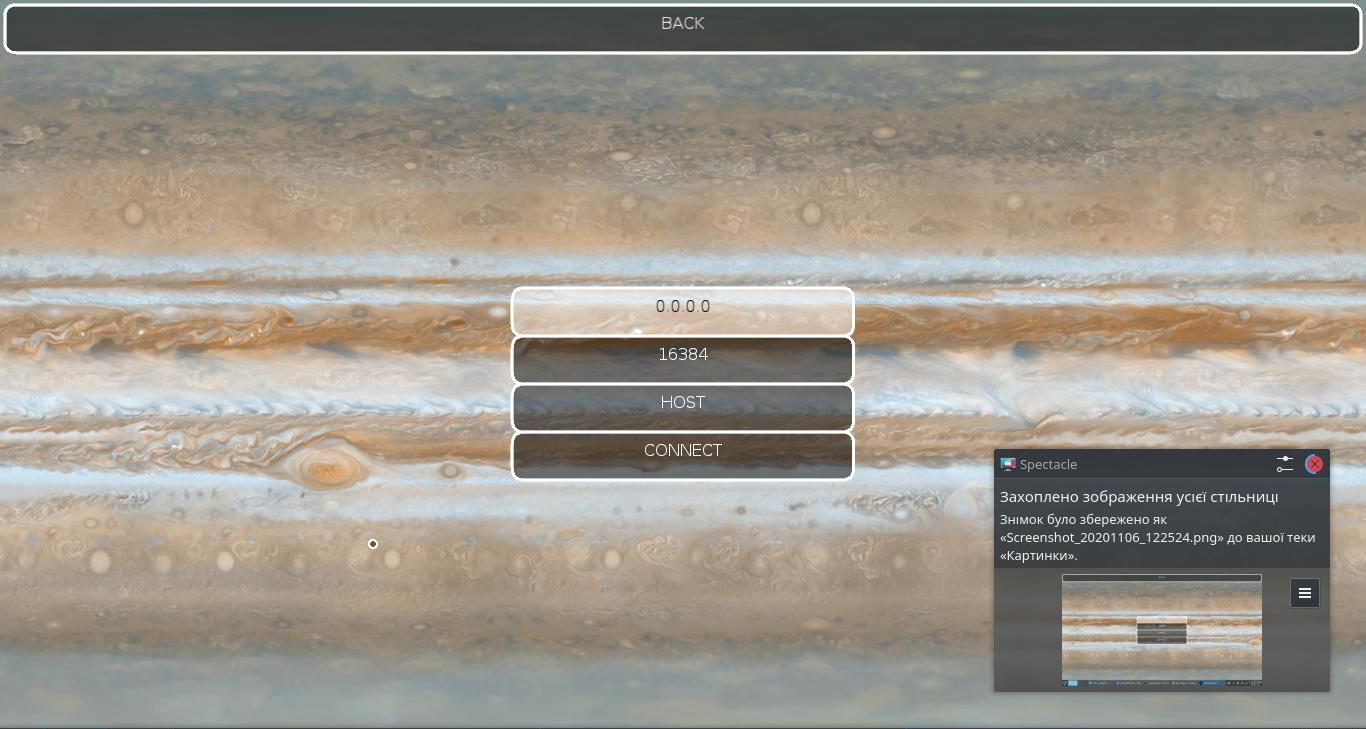


Рис. 3.4 Екран вибору хоста

Керування положенням камери спостерігача відбувається клавішами ‘w’, ‘s’, ‘d’, ‘a’, SPACE, SHIFT, ‘q’, ‘e’ для руху вперед, назад, вліво, вправо, вгору, вниз, крену вліво й вправо відповідно. Миша використовується для нишпорення та тангажу. Клавішами ‘7’, ‘8’ і ‘9’ можна перемикатись між різними видами, ‘4’, ‘5’ і ‘6’ – змінювати відстань від спостерігача до телескопа, а ‘1’, ‘2’ і ‘3’– керувати лінзами телескопа. Клавіші ‘-’ та ‘+’ перемикають степені свободи спостерігача на 5 і 6 відповідно, а ‘\*’ і ‘/’ – керування камерою та телескопом. Клавіша ‘i’ перемикає зображення інтерфейсу, а ‘o’ – орбіт планет. Кнопка ‘p’ зупиняє чи продовжує хід симуляції, а ‘[’ та ‘]’ – сповільнюють та пришвидшують. Клавіша ENTER робить знімок.

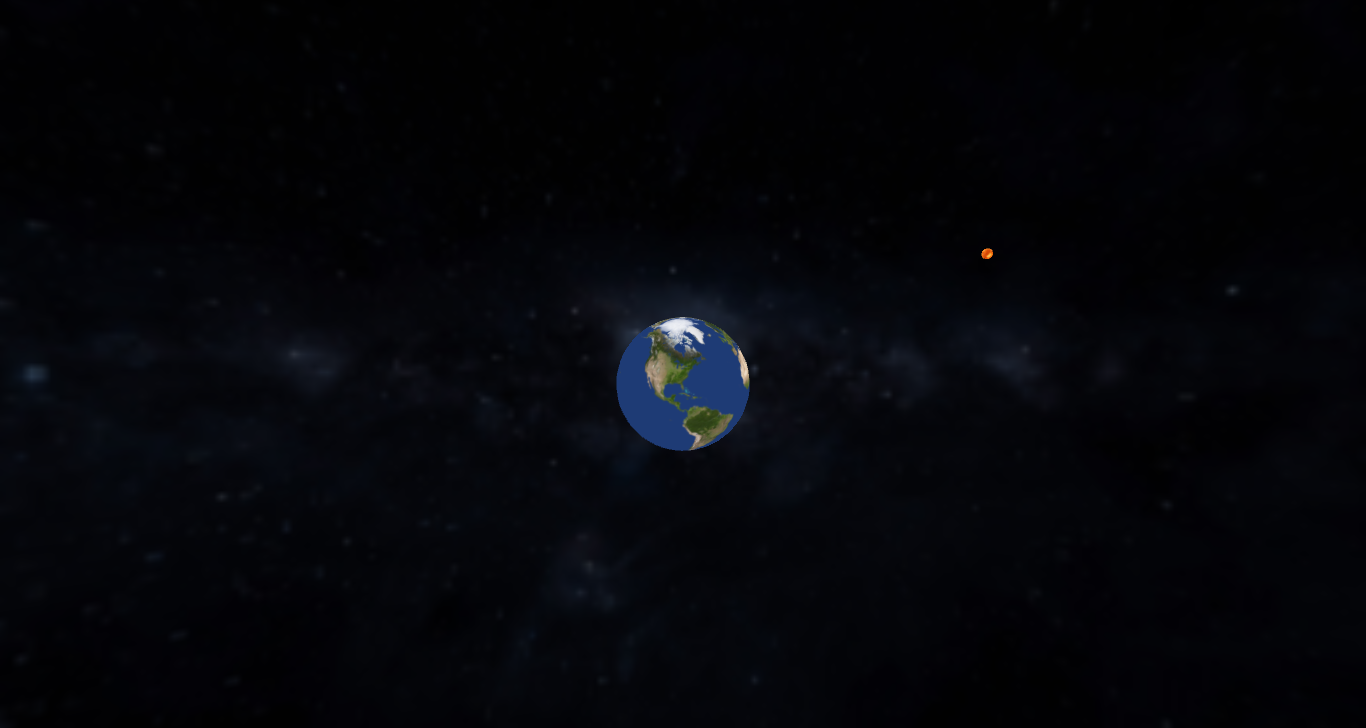


Рис. 3.5 Приклад роботи програми



Рис. 3.6 Приклад роботи програми



Рис. 3.7 Приклад роботи програми



Рис. 3.8 Приклад роботи програми



Рис. 3.9 Приклад роботи програми

Під час написання коду виявилося, що розробка гарного інтерфейсу займає в десятки (а то і сотні) раз більше часу, ніж самої математичної моделі. Найбільш цікаво програмувати камеру та всілякі графічні “родзинки”, а обробники натиснень кнопок та клавіш – найменш.

# ВИСНОВКИ

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

Зроблено такі висновки:

* Перед реалізацією надзвичайно важливо вдало й детально продумати архітектуру ПЗ, оскільки для її зміни надалі витрачається купа часу.
* Використання стандартного типу float для моделювання таких точних систем, як зоряних, не дає достатньої точності.
* Розробка гарного інтерфейсу займає в десятки (а то і сотні) раз більше часу, ніж самої математичної моделі.
* Processing – зручне середовище для створення прототипів ПЗ, оскільки не дозволяє низькорівневим проблемам, як у С++, наприклад, заважати процесу розробки.
* Формат XML дуже вдалий для представлення ієрархічних систем (багаторівневих систем супутників у моєму випадку).
* Застосування ООП для розробки подібних систем є вдалим рішенням.

Було створено такі класи: Game (Telescope, TextureLoaderThread, SoundtrackThread), Space (Body), Interface (ButtonServer (Button), KeyServer (Key), Camera), Mathematics (Vector), Physics (SphericalBody), Client та Server. Перевірка показала надійну й стабільну роботу програми. Отже, класи працюють правильно і взаємодіють між собою коректно.

Крім того, класи Interface, Mathematics, Physics, Client та Server я планую використати для розробки власного ігрового рушія в майбутньому.

Усі розрахунки між будь-якими космічними тілами відбуваються у кожен такт виконання за реальними фізичними законами: всесвітнього тяжіння та Ньютона, у чому й полягає науковість проєкту.

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. Небесна механіка [Електронний ресурс] / Михайло Царьков – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=fBC3cCsCEzA>.
2. Renesco Rocketman. Векторна алгебра на прикладі комп'ютерних ігор [Електронний ресурс] / Renesco Rocketman – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=IB2M8Qz35lc&list=PL6ZsIWeUKQCrpiQB521x-FWxhugOpZwfY>.
3. The Coding Train. Coding Challenge [Електронний ресурс] / The Coding Train – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=17WoOqgXsRM&list=PLRqwX-V7Uu6ZiZxtDDRCi6uhfTH4FilpH>.
4. Alpha Centauri. Все про орбітальну механіку | Як запускають супутники [Електронний ресурс] / Alpha Centauri – Режим доступу до ресурсу: <https://www.youtube.com/watch?v=YvbB4S5NiX8>.
5. 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>.
6. [Електронний ресурс] / C. Reas, B. Fry, D. Shiffman. – 2019. – Режим доступу до ресурсу: <https://processing.org/reference/>.
7. Клімов О. Опануй програмування граючи [Електронний ресурс] / Олександр Клімов. – 2019. – Режим доступу до ресурсу: <http://developer.alexanderklimov.ru/>.
8. ORACLE Java Documentation [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/.

# ДОДАТОК А

**Метод convertKeplerianToCartesian класу Mathematics 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}};

}

**Рекурсивний метод generateSpace класу Space для генерування об’єктів класу Body з XML-файлу**

private void generateSpace(XML[] parents) {

float positionX, positionY, positionZ, velocityX, velocityY, velocityZ, angPositionX, angPositionY, angPositionZ, angPeriod, mass, radius;

int bright;

float orbitMass, orbitPositionX, orbitPositionY, orbitPositionZ, orbitVelocityX, orbitVelocityY, orbitVelocityZ;

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

String name;

PImage texture;

XML parent;

float[][] result;

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

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

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

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

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

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

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

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

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

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

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

mass = parents[i].getFloat("mass") \* this.valuesKoefficient;

radius = parents[i].getFloat("radius") \* this.valuesKoefficient;

bright = parents[i].getInt("bright");

parent = parents[i].getParent();

orbitMass = parent.getFloat("mass");

orbitPositionX = parent.getFloat("positionX");

orbitPositionY = parent.getFloat("positionY");

orbitPositionZ = parent.getFloat("positionZ");

orbitVelocityX = parent.getFloat("velocityX");

orbitVelocityY = parent.getFloat("velocityY");

orbitVelocityZ = parent.getFloat("velocityZ");

semiMajorAxis = parents[i].getFloat("semiMajorAxis") \* this.valuesKoefficient;

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);

if(this.detailMode < 1) {

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

}

Body body = new Body(positionX, positionY, positionZ, mass, radius);

body.setVelocity(velocityX, velocityY, velocityZ);

body.setAnglePosition(angPositionX, angPositionY, angPositionZ);

body.setAnglePeriod(angPeriod);

body.setBright(bright == 1);

body.setTexture(texture);

if(orbitMass == 0.0) {

positionX \*= this.valuesKoefficient;

positionY \*= this.valuesKoefficient;

positionZ \*= this.valuesKoefficient;

velocityX \*= this.valuesKoefficient;

velocityY \*= this.valuesKoefficient;

velocityZ \*= this.valuesKoefficient;

} else {

result = Mathematics.convertKeplerianToCartesian(this.gConst, semiMajorAxis, eccentricity, argumentOfPeriapsis, longitudeOfAscendingNode, inclination, meanAnomaly, orbitMass);

positionX = result[0][0];

positionY = result[0][1];

positionZ = result[0][2];

velocityX = result[1][0];

velocityY = result[1][1];

velocityZ = result[1][2];

positionX += orbitPositionX;

positionY += orbitPositionY;

positionZ += orbitPositionZ;

body.setPosition(positionX, positionY, positionZ);

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

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

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

velocityX += orbitVelocityX;

velocityY += orbitVelocityY;

velocityZ += orbitVelocityZ;

body.setVelocity(velocityX, velocityY, velocityZ);

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

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

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

}

this.bodies.add(body);

if(name.equals("telescope")) {

this.telescope = body;

}

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

}

}

**Метод gForce класу Space для обрахунку вектора сили тяжіння між об’єктами класу Body**

private float[] gForce(Body obj1, Body obj2) {

float mass1 = obj1.getMass();

float mass2 = obj2.getMass();

float[] position1 = obj1.getPosition();

float[] position2 = obj2.getPosition();

float squaredDistance = pow(position1[0] - position2[0], 2) + pow(position1[1] - position2[1], 2) + pow(position1[2] - position2[2], 2);

float[] force = new float[]{position2[0] - position1[0], position2[1] - position1[1], position2[2] - position1[2]};

float scalar = mass1 \* mass2 / squaredDistance \* this.gConst;

float vectorLength = sqrt(force[0] \* force[0] + force[1] \* force[1] + force[2] \* force[2]);

force[0] /= vectorLength;

force[1] /= vectorLength;

force[2] /= vectorLength;

force[0] \*= scalar;

force[1] \*= scalar;

force[2] \*= scalar;

return force;

}

**Метод tick класу Space, у якому виконуються всі математичні обрахунки моделі за тік (такт, найменший проміжок часу)**

public void tick() {

float[] force;

float mass1, mass2;

Iterator<Body> iter = bodies.iterator();

while(iter.hasNext()) {

Body body1 = iter.next();

if(body1.isDeleted()) {

iter.remove();

} else {

for(Body body2 : this.bodies) {

if(body1 == body2) {

continue;

}

if(this.isCollide(body1, body2)) {

this.collide(body1, body2);

} else {

force = this.gForce(body1, body2);

mass1 = body1.getMass();

mass2 = body2.getMass();

body1.accelerate(force[0] / mass1, force[1] / mass1, force[2] / mass1);

body2.accelerate(-force[0] / mass2, -force[1] / mass2, -force[2] / mass2);

}

}

body1.tick();

}

}

}

**Метод draw класу Space, що зображує на екрані космічні тіла за такт**

public void draw() {

float[] position;

float[] orientation;

float[] angle;

ArrayList<Float[]> lightsPositions = new ArrayList<Float[]>();

float[] lightPosition;

PShape pshape;

for(Body body : this.bodies) {

if(body.getBright()) {

lightPosition = body.getPosition();

lightsPositions.add(new Float[]{lightPosition[0], lightPosition[1], lightPosition[2]});

}

}

noStroke();

fill(255);

for(Body body : this.bodies) {

position = body.getPosition();

angle = body.getAnglePosition();

lightFalloff(1.0, 0.0, 0.0);

if(body.getBright()) {

ambientLight(255.0, 255.0, 255.0);

} else {

ambientLight(this.backlight, this.backlight, this.backlight);

for(Float[] innerLightPosition : lightsPositions) {

pointLight(255.0, 255.0, 255.0, innerLightPosition[0], innerLightPosition[1], innerLightPosition[2]);

}

}

translate(position[0], position[1], position[2]);

pshape = createShape(SPHERE, body.getRadius());

pshape.setTexture(body.getTexture());

orientation = body.getOrientation();

pshape.rotateY(angle[1] + Mathematics.Vector.angle(orientation[6], orientation[7], orientation[8], 1.0, 0.0, 0.0));

pshape.rotateX(angle[0] + Mathematics.Vector.angle(orientation[3], orientation[4], orientation[5], 0.0, 1.0, 0.0));

pshape.rotateY(angle[2] + Mathematics.Vector.angle(orientation[6], orientation[7], orientation[8], 1.0, 0.0, 0.0));

shape(pshape, 0, 0);

translate(-position[0], -position[1], -position[2]);

noLights();

}

}

**Метод drawBackground 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;

}

}

}