MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE

S.T.A.R.S. - Stellar Teaching and Astronomical Realism Simulator

LICENSE THESIS

Graduate: **Octavian-Mihai Matei**

Supervisor: **Șl. dr. ing. Adrian Sabou**

2024

MINISTRY OF EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

DEAN,
Prof. dr. eng. Mihaela DÎNȘOREANU

HEAD OF DEPARTAMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Octavian-Mihai Matei**

S.T.A.R.S. - Stellar Teaching and Astronomical Realism Simulator

1 **Project proposal:** *The project will simulate the behavior of stars using Newtonian-based formulae in an educational setting. The aim is to enhance the visual learning experience of students during Physics classes to experiment and visualize stellar interactions in different scenarios. The students will be able to get real-time information about different statistics of real-world stars using the GAIA database and more.*
2 **Project contents:** *Introduction, Project Objectives, Bibliographic Research, Analysis and Theoretical Foundations, Detailed Design and Implementation, Testing and Validation, User's Manual, Conclusions, Bibliography, Appendices.*
3 **Place of documentation:** Technical University of Cluj-Napoca, Computer Science Department
4 **Consultants:**
5 **Data of issue of the proposal:** November 1, 2023
6 **Data of delivery :** July 10, 2024

Graduate: 

Supervisor: _____

Contents

Chapter 1. Introduction - Project Context

The world's usage of electronic devices has steadily increased since the creation of the first personal computer and now, more rapidly, since the introduction of the smartphone. As a consequence, the adoption of the internet has skyrocketed in the last few years, and as of April 2024, a total of 5.44 billion people, which accounts for around 67.1% of the Earth's population, have internet access, as reported by the article [1] by Kepios, which states that a new "supermajority" was established for the first time in human history on internet usage. As a consequence, this enables the majority of the population to have access to scientific discoveries and, at the same time, to educational applications and websites, which would further advance the development of humankind.

As the processing power of computers has increased exponentially, according to Moore's Law, first established in 1965 and later revised and discussed in the article [2] by Gordon E Moore, the potential for educational applications has also been explored. In addition, another article [3] investigates the use of digital and simulative learning technologies in higher medical education, by using analysis and synthesis methods, as it demonstrates potential solutions to digital medical education problems. Content analysis is crucial for understanding basic information materials and this study identifies the interpretation of digital and simulative technologies, their role in modern medical education, and the main stages of simulative training. In conclusion, the authors found compelling evidence for combining these technologies to improve the educational process.

Moreover, in the educational and scientific side of the internet, the field of astronomical visualizations has seen substantial advancements in recent years, driven by the increasing availability of real astronomical data and powerful computational tools provided by reputable institutions in the field like NASA, the National Aeronautics and Space Administration, and ESA, the European Space Agency. In addition, these improvements are accelerated by the open-source data that those organizations release to a wider public, such as the three releases of data from the GAIA experiment by ESA that enabled non-affiliated scientists all around the world to research a spectacularly sized database, containing detailed astrometric, photometric, and spectroscopic data for approximately 1.8 billion stars in the Milky Way galaxy and beyond. In fact, the total size of Gaia DR3 data of around 1.5 petabytes (PB) and validated by third parties in various articles, such as [4] provides the right environment for scientists and hobbyists to test numerous theories and property values of all those objects released. Furthermore, the scientific potential is huge, as seen in proofs and releases in articles such as [5] or [6].

As a response to the rapid development of these educational astrophysics applications, this project will focus on our thesis project, entitled Stellar Teaching and Astronomical Realism Simulator, shortly S.T.A.R.S., which aims to contribute to this field by developing an application that integrates real astronomical data with interactive visual methods to ease the effort of physics teachers in high-schools in displaying and experimenting with different physics related scenarios in a controlled way. The students would

also highly benefit from this project, as they would be able to quickly visualize all the aspects that the teacher presents during classes.

The S.T.A.R.S. project represents a significant step forward in astronomical simulations, offering a tool that combines accuracy, computational efficiency, and user-friendliness. By addressing the existing gaps in the field, this project has the potential to greatly enhance the educational experience and facilitate a deeper understanding of celestial mechanics. It will be tested by a variety of methods to ensure high-quality data visualization and full engagement from the users, by performing performance-oriented tests, user experience tests provided by high school teachers and students alike and an open beta session on public platforms to test the demand for such an application in the real world.

The main features of the application include the possibility of visualizing star information, such as the size, speed, spectral output, and many more properties, which are fundamental attributes of an educational-focused project. Moreover, the ability of the teachers to modify the information guide "on the fly" to accommodate the student's needs and the curriculum represents a step up to the traditional physics-oriented applications on the market. Another crucial feature is the capability to retrieve data from any source, provided it adheres to a specific format. Currently, this is implemented using GAIA R3 data, but it can be extended to other data sources that follow the same formatting, granting teachers and students unprecedented control over the simulation's input data.

This project will be structured in the following chapters, each with its own objective, and will delve into the details of the development, implementation, and prospects of S.T.A.R.S.:

- **Chapter 2 Project Objectives** - the main objectives and secondary objectives that make up this project
- **Chapter 3 Bibliographic Research** - the research done about educational games to achieve the full implementation of the application
- **Chapter 4 Analysis and Theoretical Foundation** - a critical analysis of the high school curriculum, the algorithms proposed for implementation, and the different frameworks and design patterns analyzed
- **Chapter 5 Detailed Design and Implementation** - will contain the problem statement, the stakeholders, and user descriptions, with their needs. This chapter will continue with a description of the main use cases met in this application and an explanation of module implementations.
- **Chapter 6 Testing and Validation** - functional testing for our application, as well as performance and user experience testing
- **Chapter 7 User's Manual** - a step-by-step guide to install and utilize the application
- **Chapter 8 Conclusions** - the contributions of the application, a critical analysis of the results, as well as future plans for development

Chapter 2. Project Objectives

This project aims to develop an educational video game that uses Newtonian-based equations and properties to help mainly high school students understand certain astronomical concepts. The application offers the user an interactive environment where they can learn about galactic interactions and the properties of stars. Additionally, the application can be used by physics teachers to make their lessons more attractive and easier for students to assimilate. Considering these, we selected the following target users:

- Highschool students in the ninth and twelfth grade
- Teachers who will use the application in their lectures and classes
- Hobbyists and enthusiasts who want to experiment with different scenarios

To reach this goal, we can identify several sub-objectives that would help us achieve our goals:

2.1. Educational Content

To provide a time-worthy educational application, we need to focus on the content that will be displayed on screen, so as to conform to the needs of the 9th and 12th grade students. The main objective is to follow the Romanian Curriculum in finding the right properties to show, which is a must. Another main objective is to provide teachers with the tools to show educational content in an efficient way.

2.2. Accuracy of simulation

The accuracy of simulations is also one of the main objectives that need to be included in this project, as the educational worthiness is proportional to the level of detail in the application. The more detail incorporated into the simulation, the closer it will resemble real-life processes and phenomena, so a high level of fidelity will allow students to gain a more comprehensive understanding of the Newtonian-based physics and properties of the stars being explored through the simulation. In essence, accuracy becomes the cornerstone of the project's educational effectiveness.

2.3. Speed of computations

In an educational video game focused on astronomical concepts, fast computation speed allows for real-time exploration and experimentation, as the users can observe the immediate effects of the equations and laws of nature. We need to prioritize key physics calculations for real-time interactions and focus on computations that directly affect the user experience, such as gravitational forces, object movements, and visual representations. By emphasizing speed alongside accuracy, we can create an educational game that

is both informative and engaging, fostering a deeper understanding of astronomy for your target audience.

2.4. User Experience

Another important objective of the application is to aim to be as captivating as possible to motivate students to use it and learn aspects related to physics. This videogame should also be prioritized by teachers to make lessons as interactive and engaging as possible for students. We will focus on the following two clear sub-objectives:

- Streamlined Controls: We aim to devise controls that are intuitive for manipulating objects and navigating the user interface. This approach ensures seamless navigation, reduces frustration and enhances learning opportunities.
- Visual Feedback: We will incorporate clear visual cues for every user action, such as on-screen prompts, animations, or sound effects. These elements serve to validate their interactions and provide guidance within the environment.

By focusing on these UX aspects, we aim to cultivate a more immersive and user-friendly learning experience. This approach empowers students to interact confidently with the content, facilitating a deeper understanding of astronomical concepts.

2.5. Detailed representation of data

To ensure a comprehensive learning experience, we'll prioritize a detailed representation of star properties and data within the application, as this goes beyond just basic information.

- Data accessibility and exploration: We'll allow students to explore detailed data associated with various star types and allow them to dive into a star's mass, luminosity, temperature, or spectral classification.
- Contextualization of data: Providing context for stellar data is crucial. We'll explain how different properties interact and influence a star's behavior, from the relationship between temperature and color or spectrum and luminosity. Understanding these connections builds a strong foundation for astronomical knowledge.

By prioritizing a rich, interactive and in-depth representation of stellar data, we empower students to go beyond memorization and truly grasp the complexity and beauty of stars within the vast universe.

2.6. User-defined information

This project aims to go beyond pre-defined scenarios by offering students the ability to personalize their learning experience through user-defined information, which would include the following objectives:

- Customizable star properties: Students and teachers can define the initial positions, velocities, and accelerations of stars within the simulation, which would allow them to explore specific scenarios they're curious about or recreate real-world astronomical events. Moreover, the teachers would have the ability to model specific scenarios to show in a hands-on approach the theoretical properties explained during the courses

- Custom input for the information panels: This empowers teachers and enthusiasts to easily contribute additional content, expanding the knowledge base within the game or writing specific data entries that benefit the student for that specific course. This also fosters a collaborative learning environment.

2.7. Conclusion

In conclusion, this project outlines the creation of an educational video game aimed at engaging high school students in grades 9-12 in a variety of astronomical topics. By combining Newtonian physics with stellar characteristics, the application offers an interactive environment in which users may learn about galactic interactions and the fascinating universe of stars. This teaching tool not only benefits the pupils but also provides the physics professors with the capacity to convey the material in an engaging and participatory manner. Moreover, by achieving these goals, the teachers would be able to personalize the contents of these simulations according to the needs of the classes, by exploring different scenarios.

By meeting these goals, this project has the potential to redefine the way high school students learn about astronomy and Newtonian physics, from a memorization-based course to an interactive and fascinating study of the cosmos.

Chapter 3. Bibliographic Research

3.1. Education through games

3.1.1. Educational Games

From the beginning of our species, people played games, either to distract them or just for fun, one thing is sure: games are as old as time and continue to play an essential part in our lives. One of the first pieces of writing that mention playing is in 345 B.C. in "Plato Laws 1 & 2" [7] translated and commented by Susan Sauvé Meyer, where Plato describes playing as an essential development part of a child. There, he describes the usage of replica tools and equipment that would help a boy grow into his future occupation and become a good grown man. Furthermore, in his works, he also relates to the obligation of caretakers to tend to such playful activities, as riding with the children, playing war games with them etc.

Additionally, Elliot Avedon and Brian Sutton-Smith's pioneering examination of educational games over time, "The Study of Games" [8], focuses on games invented before the 1970s. The authors analyze how playing games affects people's cognitive, psychological, and cultural development as well as how society standards are shaped by gaming. By studying games from ancient civilizations until the mid-twentieth century, the authors highlight the origins and history of educational games. This historical context allows us to better comprehend how games continue to be useful teaching tools since it shows how flexible they may be to promote learning in a variety of cultural and chronological contexts. One fascinating example they discuss is how the ancient Greeks and Romans used games like Ludus latrunculorum and Astragaloi to teach strategic thinking and mathematical skills, as these games were not just for entertainment, they were actually designed to teach important knowledge and skills to the pupils. Moreover, in medieval Europe, games such as Chess were employed by educational institutions to teach military tactics, moral values, and social etiquette.

3.1.2. Educational Video Games

As soon as desk-sized main-frame computers were invented, it was clear that these machines could help students learn by practicing in simulated environments. The first examples of this trend can be seen with the release of "The Sumerian Game"[1] released in the 1960s for the IBM 7090 time-shared mainframe computer, as seen in Figure 3.1. In this text-based game, the main goal is to teach the students how to manage resources in the context of ancient Sumer, as they are required to tell the game how much grain will be used as food, seed for planting, and storage, as a way to teach economic principles to them.

[1]For more information, see `https://en.wikipedia.org/wiki/The_Sumerian_Game`.

Richard L. Wing's 1966 study, "Two Computer-Based Economics Games for Sixth Graders," [9] further exemplifies this approach, by developing interactive games that simulate economic scenarios. Wing aimed to engage young learners in a dynamic educational process, as these games allowed students to experiment with economic decision-making in a risk-free environment so deepening their understanding of complex concepts through experiential learning. For example, one of Wing's games has tasked students with managing a virtual marketplace, where they had to set prices for goods, manage inventory, and respond to variations in supply and demand. This hands-on approach helps to illustrate the principles of economics tangibly, making abstract concepts more accessible and memorable for young learners. Building on this foundation, in 2003, the article "Harness-

Figure 3.1: Student using the IBM 7090 to play the game

ing the Power of Games in Education" [10] by Kurt Squire and Henry Jenkins explores the modern potential of video games in educational settings. The authors argue that video games create immersive, engaging, and interactive experiences that foster learning, critical thinking, and collaboration. Their work underscores how games, evolving from the traditional board and physical games discussed by Avedon and Sutton-Smith, have adapted to incorporate advanced technology, providing customizable and situated learning experiences. This progression highlights the enduring relevance and adaptability of games as educational tools, demonstrating their capacity to transform traditional learning methods and motivate students through engaging, experiential learning opportunities.

The video game industry's revenue surge from 1970 to 2022, as depicted in the graph Figure 3.2 from the article in Visual Capitalist entitled "50 Years of Video Game Industry Revenues, by Platform" [11] by Pallavi Rao, isn't just about financial triumph. It's a testament to a booming gaming desire and potential demand. With advancements in technology, mobile accessibility, and dropping price points, as games are captivating as of 2022, a wider demographic than ever before, so, as a consequence, this creates a fertile ground for educational games. Imagine a scenario where millions who are already comfortable with the interactive format – from casual gamers to hardcore enthusiasts – are presented with well-crafted educational titles. These games can seamlessly integrate learning into the existing gaming experience, transforming memorization into a thrilling challenge or historical events into an immersive adventure. It's not about replacing playtime with studying, but rather merging the two, by leveraging the existing appeal of games and the vast player base, educational games have the potential to revolutionize

learning. This convergence of entertainment and education holds immense potential, capitalizing on the ever-expanding gaming audience to cultivate a generation of enthusiastic learners.

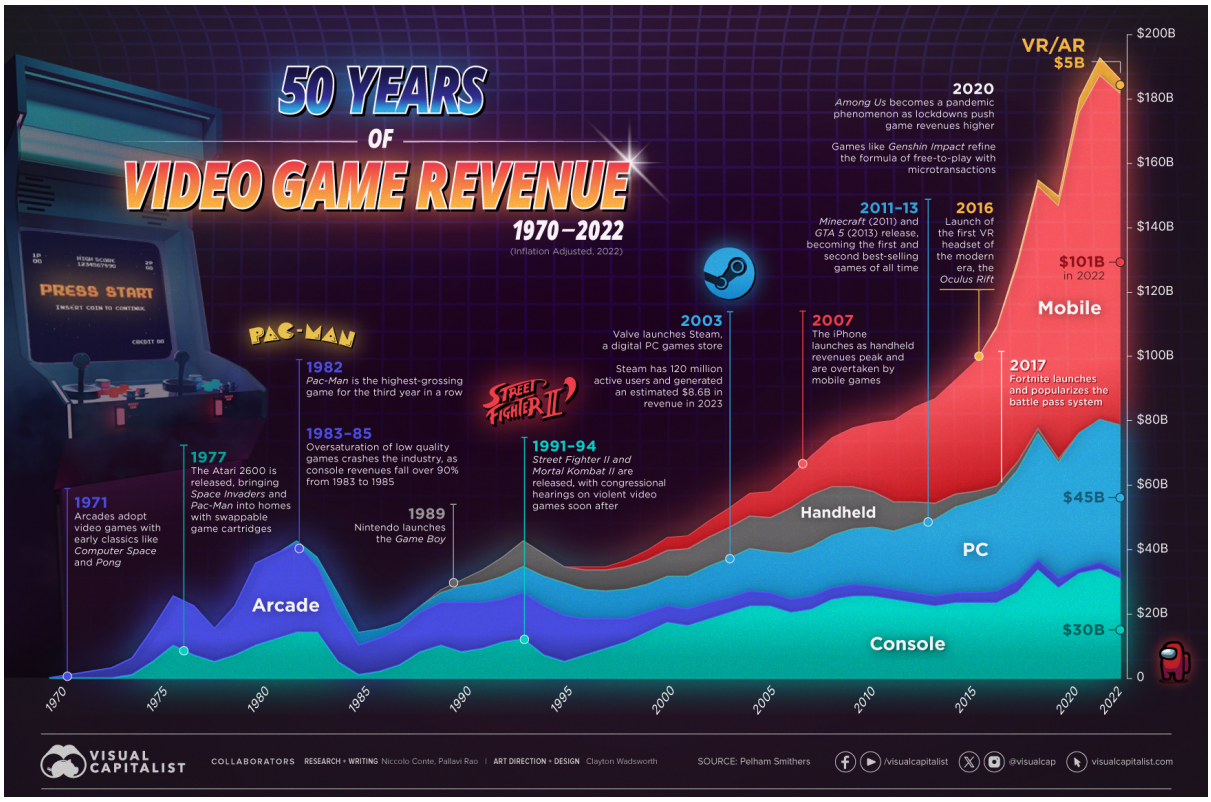

Figure 3.2: Total revenue in video games 1970-2022

Creating educational games that reward users for their progress fosters a constructive learning environment, motivating students to learn new concepts and enhancing their understanding of a certain subject and their cognitive development. Additionally, these games can improve memorization and socialization skills by incorporating elements of play and positive reinforcement, making users more likely to engage with educational material and retain information, as this approach encourages critical thinking, problem-solving, and teamwork while promoting interaction with others in a collaborative setting and ultimately supporting a well-rounded educational experience.

Educational games can take a variety of forms, such as those used in military institutions, as is the case of The U.S. Army Corps of Engineers Safety Trainer 360[12] which is a virtual reality construction site game that the Huntsville Center Safety Office and the Army Game Studio developed. The game allows users to navigate a computer-simulated construction site while correcting potential hazards, as seen in Figure 3.3. Currently, the game is being used for the Occupational Safety and Health Administration 10-Hour Training Course, which is required for all employees in a quality assurance role and is recommended for any employee visiting contractor construction, maintenance and service sites.

Touch Surgery Simulations[2], a mobile app featuring surgical simulations, is another impressive example of educational games that portray life-or-death situations. The users can access 200 simulations from more than 17 different specialties. The efficiency of using

[2]More information about Touch Surgery Simulations `https://www.touchsurgery.com/simulations`

Figure 3.3: Safety Trainer 360 interface

such an application was analyzed recently in an article entitled "Touch Surgery: Analysis and Assessment of Validity of a Hand Surgery Simulation App" [13] by Tulipan et al. in the specific scenario of a carpal tunnel release procedures. Participants with varying levels of surgical experience, from residents to medical students, were evaluated on their performance within the application's simulation module. As they found out, all groups improved their scores with repeated attempts, suggesting the app can enhance surgical knowledge and procedural steps. Unsurprisingly, the experts outperformed the other groups, highlighting the existing skill gap, but interestingly, novices found the app more helpful and were more likely to use it again, suggesting its potential as a learning tool for those early in their surgical training journey. Overall, the study supports the validity and usefulness of Touch Surgery as a training aid for hand surgery, with its ability to improve performance and user confidence, particularly for those with less experience, the app offers a valuable addition to surgical education.

Assassin's Creed: Odyssey is a game created by one of the biggest gaming studios in the world and the Discovery Tour, AC:ODT for short, feature was developed in collaboration with historians as a way to promote history in classrooms innovatively and interactively. A recent study by Darrell and de Rooy [14] explored the potential of it as a tool for history education in middle school classrooms and their findings suggest that AC:ODT can be a valuable asset for engaging students and promoting a deeper understanding of ancient Greece. The article highlights the power of this game's immersive environment where the students are transported to a meticulously recreated ancient Greece, surpassing the limitations of traditional textbooks and static images. This in-game world and game features, as the study suggests, can lead to a more engaging and impactful learning experience, because AC:ODT caters to a variety of learning styles, from visual learners who benefit from the immersive environment, as seen in Figure 3.4, to auditory learners who can access narrated tours. This aligns with the game's open-ended nature, which allows teachers to tailor learning experiences to individual needs, as suggested by the research. Students who struggle with traditional methods can find a new path to engage with history, while advanced learners can delve deeper into specific topics. It's clear that Assassin's Creed: Odyssey Discovery Tour offers a unique and powerful approach to historical education, by leveraging the game's immersive environment, interactive elements, and vast historical content, educators can create dynamic learning

experiences that spark curiosity, critical thinking, and a lifelong love for history. This innovative tool allows students to step beyond the confines of the classroom and truly walk in the footsteps of the past.

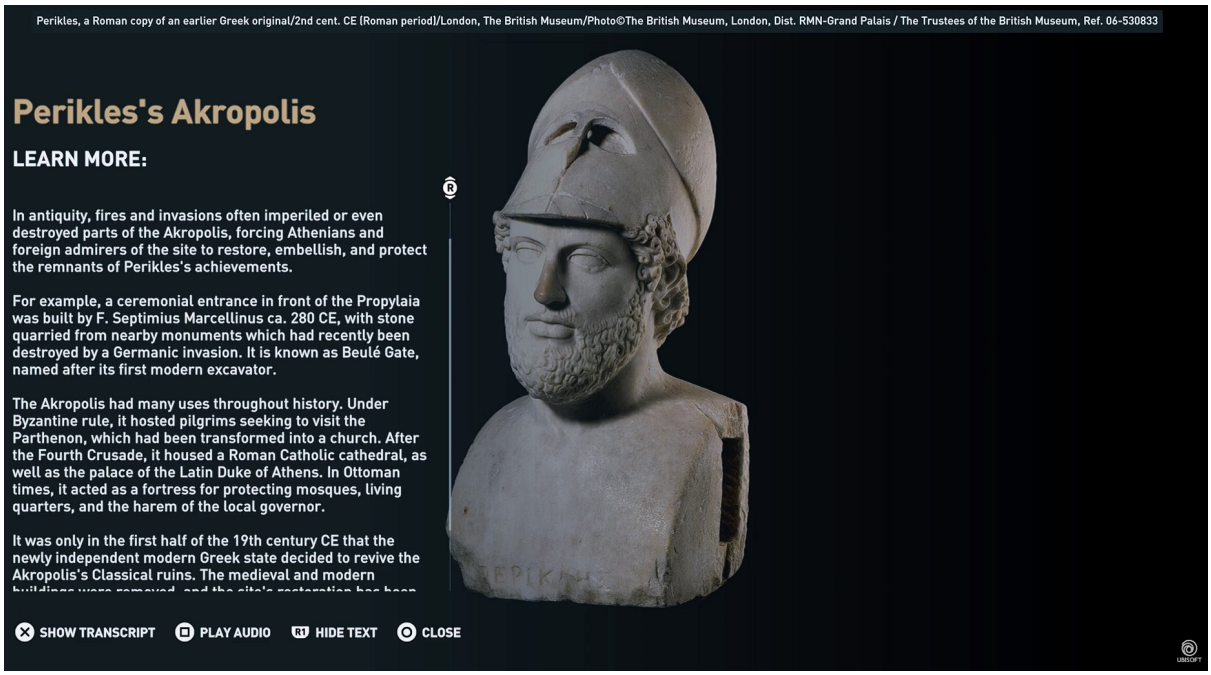

Figure 3.4: Pericle's Akropolis visual interaction UI

3.2. Physics-Oriented Educational Games

3.2.1. Physicus

A video-game called "Physicus" was created to educate high school students fundamental physics principles, related to electricity, magnetism, light and color. Its usefulness was examined in a recent study titled "Game-Based Learning of Physics Content: The Effectiveness of a Physics Game for Learning Basic Physics Concepts" [15]. This game creates an immersive virtual laboratory, which is a departure from standard classroom approaches. Students participate actively in this activity by moving objects around the game world to explore with basic physics concepts like force and gravity. In contrast to simulations, the game runs in real time, letting students see the results of their decisions right away. Compared to textbook graphics, this promotes a deeper comprehension of cause-and-effect linkages.

However, "Physicus" goes beyond dry mechanics. The developers cleverly weave physics principles into a captivating storyline, transforming the learning experience into an engaging adventure. This narrative structure not only fosters student interest and motivation but also reinforces the practical applications of physics in real-world scenarios. This fusion of entertainment and education creates a powerful learning tool, as evidenced by the aforementioned study. Students who interacted with the game demonstrated a significantly improved grasp of physics concepts compared to those using traditional methods. This highlights the potential of well-designed educational games like "Physics:

Save the World!" to revolutionize science education by making it interactive, engaging, and ultimately more effective.

3.2.2. Astronomy Lab on PC: Relativity, Lunar Landing, Space Flight, and Interstellar Travelling

Astronomy Lab on PC[3] is a VR space educational game created by experts from California Institute of Technology and the Department of Physics at Oxford University. It integrates physical science formulas into visual effects, spread over 4 distinct sub-games called scenarios in-game, where the users can explore the lunar or solar system or even travel the galaxy at sub-luminous speeds. These sub-games are as follows:

1 **Traveling in the lunar system** - in this scenario, the users will be in a simulated satellite, where they can observe Earth from above or even simulate an Apollo landing mission.
2 **Wanderers in the solar system** - in this case, the users will explore the entire solar system, where they can observe and learn about most objects around our star, as seen in Figure 3.5.
3 **Travel around the world in a hot air balloon** - the application has the the ability to present to the users a great deal of attractions on Earth using a simulated hot air balloon. This gives the users a close and hands-on approach to learning about castles, volcanic eruptions, ruins, etc.
4 **Traveling at 80% speed of light** - this scenario teaches users about the relativistic effects, such as time dilation, Doppler shift, etc, in an interactive and intuition-based interaction with one of the most hard-to-understand environments.

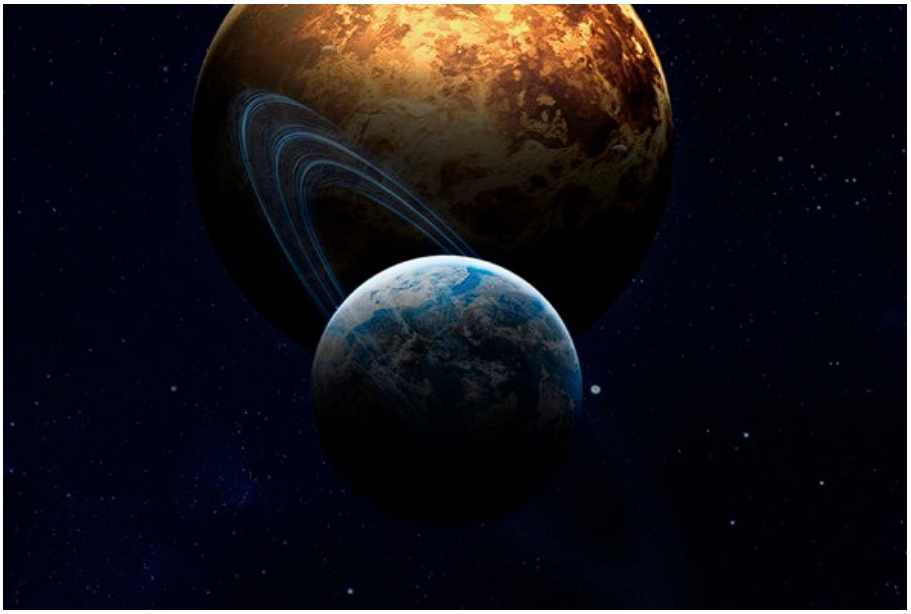

Figure 3.5: Screenshot from Scenario 2

[3]More information about Astronomy Lab on PC, `https://store.steampowered.com/app/1405980/Astronomy_Lab_on_PC_Relativity_Lunar_Landing_Space_Flight_and_Interstellar_Travelling`.

3.2.3. Cosmonium

Cosmonium[4] is a free, open-source program that allows users to explore the universe in a virtual 3D environment. Unlike some astronomy software that relies on static images, Cosmonium offers an interactive experience. Users can navigate the solar system, visiting planets, moons, and even venturing out to explore distant galaxies. As they travel, they can zoom in on celestial bodies to get a closer look or zoom out to see the vastness of space. Moreover, with just a click, you can access detailed descriptions, physical proper-

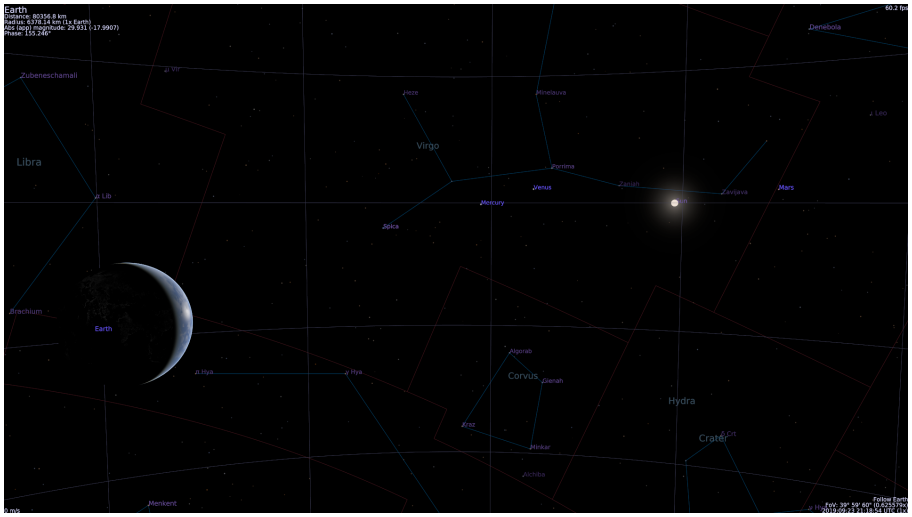

Figure 3.6: Names and constellations

ties, and interesting facts, such as the names and positions of constellations in the Milky Way galaxy as seen in Figure 3.6, which promotes a deeper understanding of the universe. This accessibility of knowledge enhances the user experience, making exploration more engaging and meaningful. Cosmonium is a useful resource for our research because of its user-friendly design, which captivates people in astronomical learning. Through an analysis of Cosmonium's effective integration of exploration and information acquisition, we gained important insights into the design principles of educational games centered around galactic simulations, which have the potential to excite and encourage scientific students.

3.2.4. In Collaboration with NASA

Kerbal Space program

Since its introduction in 2015, the highly regarded space flight simulation game "Kerbal Space Program"[5] (KSP) has fascinated players and space enthusiasts alike. With Squad's space program builder and manager, rocket designer and launcher, and realistic physics simulation, KSP lets users explore planets and manage their own space program in the fictional star system of Kerbol. Space fans and even some real-world engineers have been drawn to KSP because of its realistic yet simple orbital physics, which has garnered the game's devoted fan base. The impact of the game goes beyond simple enjoyment. In 2016, a NASA-funded initiative took advantage of KSP's widespread appeal to involve

[4]More information about Cosmonium, `https://github.com/cosmonium/cosmonium`.
[5]Steam page for KSP, `https://store.steampowered.com/app/220200/Kerbal_Space_Program/`.

players in a virtual asteroid-sampling endeavor, as seen in Figure 3.7, as reported in Karl B. Hille's article "Gamers Tackle Virtual Asteroid-Sampling Mission" [16], this project demonstrates the potential of games like KSP to not only entertain but also to serve as valuable training and simulation tools. My project took direct inspiration from KSP, by

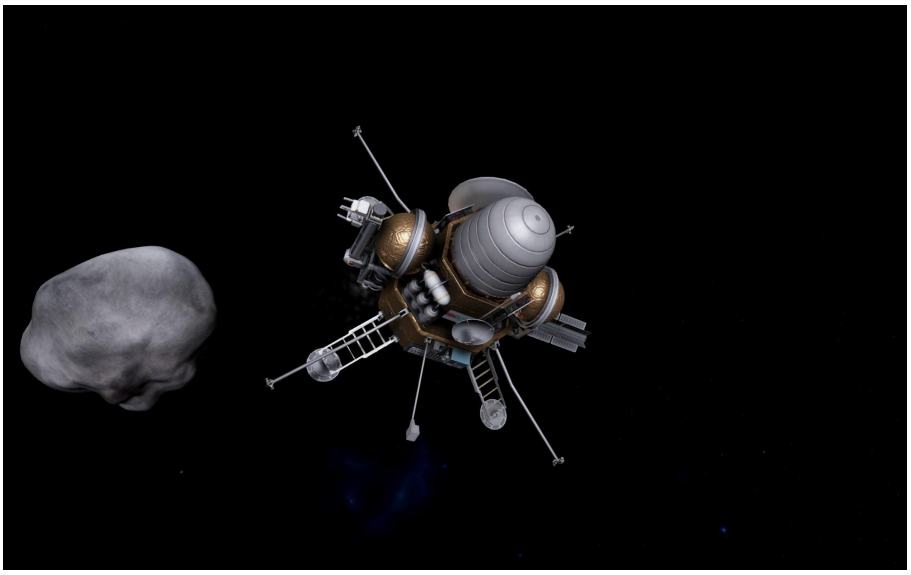

Figure 3.7: Asteroid-sampling mission[6]

following in this tradition of leveraging interactive and educational gaming to simulate complex space-related mechanics. By drawing from KSP's rich simulation environment and NASA's successful public engagement strategies, my project aims to provide a similar hands-on learning experience.

Moonbase Alpha

NASA's Moonbase Alpha[7], released in 2011, goes beyond entertainment value and serves as a one-of-a-kind educational tool in the classroom. The game's storyline immerses students in a realistic simulation of problem-solving within the context of space exploration as they work together to fix a major system failure at a lunar base. Students learn critical thinking and collaborative skills while navigating the lunar environment and using tools, which are crucial for future scientists and engineers. In addition, the game gradually integrates real-world difficulties encountered by lunar colonists, as seen in Figure 3.8, where one of the missions is to repair a pipe connecting two different lunar modules. This promotes awareness of the difficulties of space travel and the significance of resource management in such a setting. Because of its captivating gameplay and science-based setting, Moonbase Alpha is a great resource for instructors who want to encourage students to pursue careers in STEM.

[6]Realistic design of a Probe from user u/Invaderchaos on Reddit: `https://www.reddit.com/r/KerbalSpaceProgram/comments/i62jfw/asteroid_sample_return_mission_bluedog_design/`

[7]More information about Moonbase Alpha, `https://spinoff.nasa.gov/Spinoff2011/cg_4.html`.

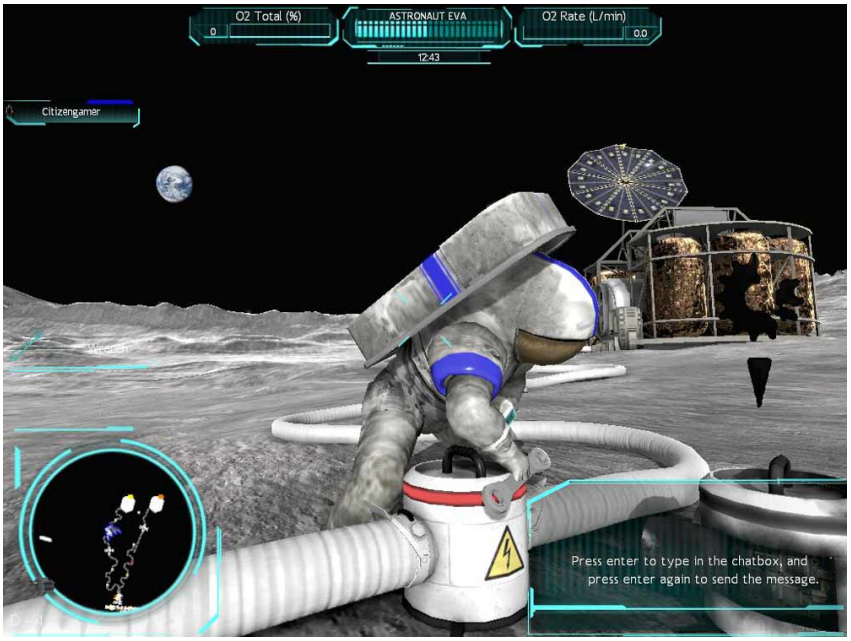

Figure 3.8: Pipe repairing mission

3.3. Conclusions

In conclusion, a lot of studies argue that education through games is one of the oldest forms of learning, from Plato's example of war games to the medieval times when chess was used as a simulation of military strategies, to today when the abstract physical properties of stars can be simulated in a user-friendly way. The hands-on method was approached by every important institution and with the clear rise in videogames, this trend doesn't seem to slow down any time soon.

As for the current implementations of the main idea of the thesis, a lot of these games take a simplified implementation of physical phenomena or focus on different things than our project. They serve as proof that there is demand in the area of physics for these games, as well as starting points to describe and design our own interpretation of Newtonian-based physics simulation game.

Chapter 4. Analysis and Theoretical Foundation

4.1. High School curriculum

The application having an educational focus would need to follow parts of the Romanian Physics Curriculum for high school students. In the analysis of the official documents, there were identified two possible main groups of students. The first group is the 9th grade, whose educational programme focuses on Newtonian physics [17] and the other group is the 12th grade, whose curriculum focuses on celestial body motions and spectral radiation[18].

Since the subject matter must satisfy all of these learning objectives, it is imperative to choose a topic that can smoothly combine the more complex ideas of celestial mechanics with the fundamentals of Newtonian physics. Because orbital mechanics inevitably applies Newton's principles to describe the motion of planets, moons, and other celestial bodies, it presents a special chance to connect the curriculum for both classes. This branch of physics naturally develops into the study of more complicated phenomena involving gravitational interactions and body movements in space, in addition to offering an interesting and useful framework for the concepts of Newtonian mechanics.

Furthermore, adding stellar statistics to the simulation can improve the 12th grader's educational experience even more. Understanding the broader background of astronomical objects' fundamental statistics, such as the relationship between mass and inner composition or the strong bond between stellar classification and their spectral output, students may investigate with a visual aid by including these facts in the simulation.

So a simulation for orbital mechanics seems the best solution to combine the two syllabuses. Firstly, the simulation would run on Newtonian physics which would provide the learning framework for the 9th graders, while the more detailed stellar statistics and their motions would benefit the 12th graders.

4.2. Hardware Acceleration

To compute vast amounts of data, we need to think about what kind of processing unit would be the best for that job. By using a normal personal computer as the desired form factor for our application, we have two main competitors, the central processing unit (CPU) and the graphics processing unit (GPU). Each of these processing units offers distinct advantages and disadvantages depending on the nature of the computational workload.

Based on the type of computing workload, each of these processing units has unique benefits and drawbacks. Table 4.1 is a comprehensive visual tool that we created to help you better understand the relative advantages and disadvantages of CPUs and GPUs for our particular application. We can decide if a CPU or GPU would be the best option

for effectively processing our data by looking at this table, which lists important design decisions for the creation of such hardware.

Table 4.1: CPU vs GPU

	CPU	Graphics processing unit (GPU)
Function	Generalized component that handles main processing functions	Specialized component that was designed for parallel computing
Processing	Designed for serial instruction processing	Designed for parallel instruction processing
Design	Fewer, more powerful cores	More, less powerful cores
Best suited for	General purpose computing	High-performance computing

After careful consideration of our application's requirements and the specific strengths of these processing units, we have decided to choose the GPU for our data processing needs. The next step is to determine the best software framework to leverage the GPU's capabilities effectively, as there are two primary frameworks for GPU programming: CUDA and OpenCL. Both frameworks are designed to facilitate parallel computing on GPUs, but they have distinct characteristics and advantages which will be explored in the following chapters.

4.2.1. CUDA

CUDA (Compute Unified Device Architecture) is a programming model created by Nvidia for general-purpose computing on GPUs, it allows programmers to leverage the parallel processing power of GPUs for tasks beyond just graphics processing. As we can see in the processing flow diagram from Figure 4.1, the data is fetched from the system memory, stored into the GPU's memory and to and from there all the memory manipulation and processing are done on specialized units which can approach tens of thousands of CUDA cores.

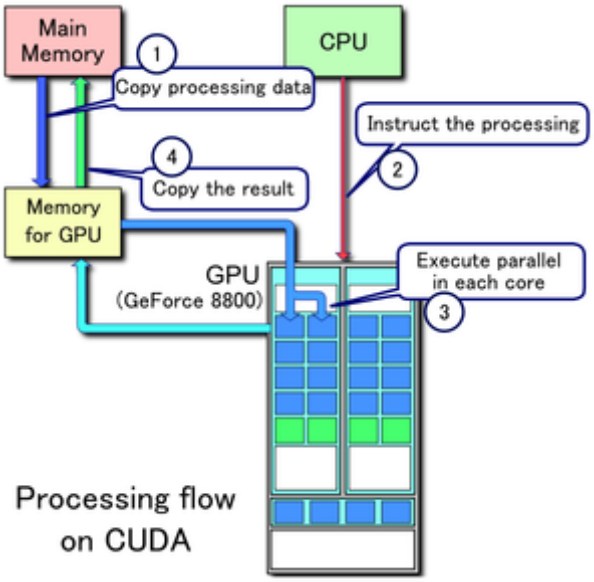

Figure 4.1: CUDA Processing Flow[1]

[1]Cuda Processing Flow on `https://en.wikipedia.org/wiki/CUDA`

Pros:

1 **Performance Optimization** - CUDA is highly optimized for NVIDIA GPUs, allowing for maximum performance and efficiency, as it provides direct access to the GPU's parallel processing power, enabling fine-tuned optimizations that can lead to significant speedups in computations.
2 **Developer Support and Community** - NVIDIA provides comprehensive documentation, tutorials, and a large number of resources for developers and a very strong community of CUDA developers, making it easier to find support.
3 **Extensive Libraries and Tools** - CUDA offers a rich set of libraries (e.g., cuBLAS, cuFFT, cuDNN) that cover a wide range of applications, from linear algebra to deep learning.

Cons:

1 **Hardware Dependency** - CUDA is proprietary to NVIDIA GPUs, meaning it cannot be used with GPUs from other vendors like AMD or Intel. This limits hardware choices and can be a significant drawback for environments with diverse hardware.
2 **License Restrictions** - Use of CUDA may involve licensing terms and conditions set by NVIDIA

Conclusion:

To sum up, CUDA is a robust and effective framework for GPU programming, primarily on NVIDIA hardware, with an extensive set of tools, libraries, and community support that make it an excellent option for applications requiring high speed. However, as our project will be installed on a range of unique platforms, it is vital to take into account the dependence on the proprietary framework by a private firm as well as potential portability concerns on AMD or Intel GPUs.

4.2.2. OpenCL

OpenCL (Open Computing Language) is an open standard supported by multiple hardware vendors, including AMD, Intel, and NVIDIA, as well as by various CPUs, GPUs, FPGAs, and other accelerators. This broad compatibility allows for greater flexibility and portability of applications across different hardware platforms. The main operating flow is similar to CUDA's flow, with some variations of the actual implementation of the drivers and the main architecture, which can be better seen in Figure 4.2. Let's analyze the pros and cons of this framework:

Pros:

1 **Open Source** - Open-source implementations of OpenCL, such as those provided by various vendors ensure transparency and accessibility, as well as the possibility of personal development or tweaking of the framework for one's needs.
2 **Vendor-Neutral** - It provides a unified programming model for heterogeneous computing, thus being able to run on diverse devices from different manufacturers without being tied to a specific vendor's ecosystem.

[2]OpenCL on `http://thebeardsage.com/opencl-architecture-and-program/`

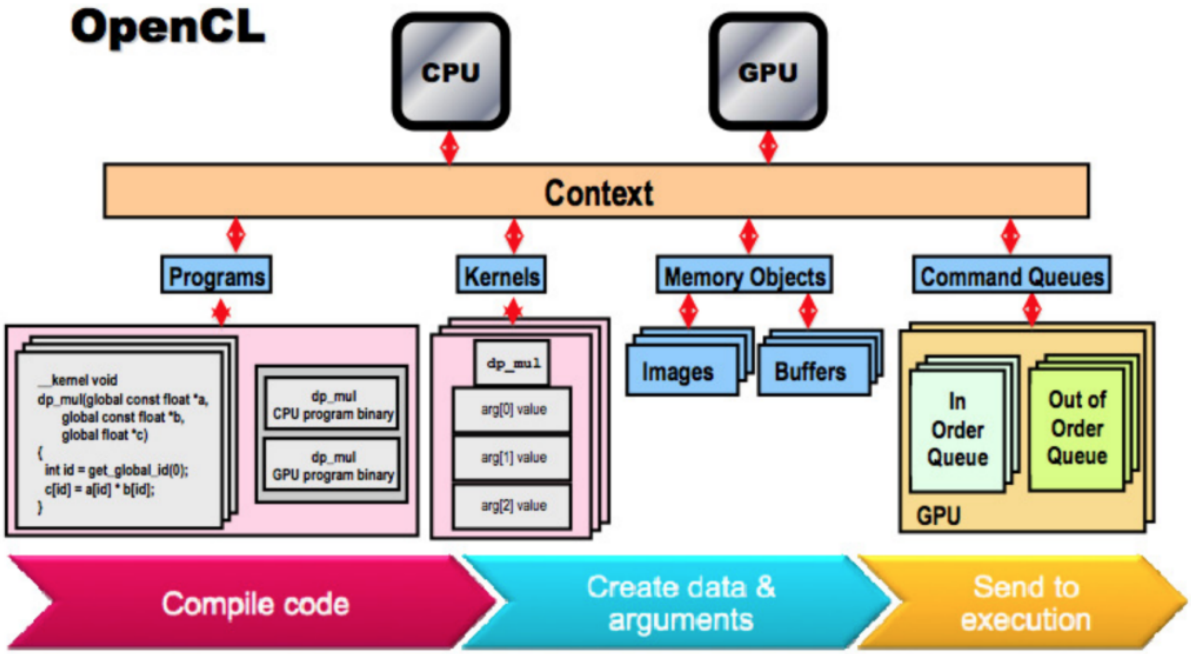

Figure 4.2: OpenCL Architecture[2]

3 **Heterogeneous Programming Model** - The framework allows us to program for various processing units within a system, thus taking advantage of the strengths of CPUs, GPUs, FPGAs etc for different tasks within a single application.

Cons:

1 **Less Mature Ecosystem** - While OpenCL is a mature standard, vendor-specific solutions, like CUDA, have more mature tools and high-level libraries available due to their focus on a single platform.
2 **Performance** - Due to its broader target and focus on portability, OpenCL might not always achieve the absolute best performance compared to vendor-specific solutions like CUDA[19]
3 **Community and Documentation** - Although OpenCL has a growing community, it is not as large or as active as the CUDA community

Conclusion:

All things considered, OpenCL provides portability and flexibility, which makes it an important tool for developers who have to build code that works across several hardware platforms. This is highly beneficial for heterogeneous computing environments and cross-platform apps. However, OpenCL tries to serve a wide range of devices, it cannot be as closely tuned for the unique capabilities of any one platform as a solution created for a single hardware type. Additionally, the lack of direct support from the GPU manufacturer means that fixes and upgrades may not always follow the introduction of new technology. All of these factors come at a performance cost.

4.2.3. Which framework to use?

We took into account a host of factors, including hardware compatibility, portability, speed, and flexibility while choosing between CUDA and OpenCL for our application. Even though CUDA's direct relationship with Nvidia GPUs allows for better performance, portability suffers as a result. Therefore, we have opted to adopt *OpenCL* as its compatibility with a wide range of hardware and its open-source implementations' transparency meet our needs and allow our program to run on a variety of platforms.

4.3. N-Body Simulation

It is clear that considering the amount of objects that need to be simulated, there should be a streamlined method to compute the attraction of planets and their movements. For this purpose, the two algorithms were analyzed side by side to decide what to use in the final product.

4.3.1. Naive method

The main idea of this method is very straightforward, by directly computing the forces of attraction using Newton's law of gravitation for every pair of stellar objects we get the approximate values that we need. The main disadvantage of this method is that it is computationally expensive, leading to a complexity of $O(N^2)$.

Steps in the Naive Method

1 **Initialization** - Take N objects and check if their path has been altered by the simulation from the previous step by computing the *Euclidean* distance between their current position and the position expected.
2 **Placement** - Place the K filtered objects in the simulation space with initial positions, masses, and velocities.
3 **Force Calculation** - For each game object i in the K sized set, calculate the gravitational force exerted by every other game object j in the initial N sized set:

$$\mathbf{F}_{ij} = G\frac{m_i m_j}{|\mathbf{r}_{ij}|^3}\mathbf{r}_{ij}$$

where:
- G is the gravitational constant.
- m_i and m_j are the masses of objects i and j.
- $\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$ is the vector from object i to object j.
- $|\mathbf{r}_{ij}|$ is the distance between objects i and j.

4 **Summing Forces** - Sum all the pairwise forces acting on each object i:

$$\mathbf{F}_i = \sum_{j \neq i} \mathbf{F}_{ij}$$

5 **Updating Positions and Velocities** - Use the computed forces to update the positions and velocities of the game objects using numerical integration methods.
6 **Repeating** - Repeat the force calculation and updating steps for each time step of the simulation.

Computational Complexity

- **Time Complexity** - $O(N^2)$. For each of the N stellar objects, the force calculation involves $N - 1$ other stellar objects.
- **Space Complexity** - $O(N)$ for storing the positions, velocities, and forces of the stellar objects.

4.3.2. Fast Multipole Method

In galactic simulations, FMM approximates the gravitational influence of distant groups of stellar objects using multipole expansions, allowing for efficient and accurate force calculations.

4.3.3. Steps in the Fast Multipole Method

1 **Tree Construction**
   - Hierarchically divide the simulation space into cells using an octree structure.
   - Each cell contains a subset of objects and is recursively subdivided until each cell has at most one object or a small number of stellar objects.

2 **Multipole Expansion**
   - For each cell, compute a multipole expansion to approximate the gravitational potential of all stellar objects within the cell as seen from far away.
   - This involves calculating coefficients that describe the combined effect of the stellar objects in terms of their total mass, center of mass, and higher-order moments.

3 **Multipole to Multipole Translation**
   - Translate and combine multipole expansions from smaller cells to form the multipole expansion for their parent cell.
   - This represents the influence of many game objects with a smaller number of coefficients.

4 **Multipole to Local Translation**
   - For each cell, translate the multipole expansion of distant cells into a local expansion.
   - The local expansion describes the potential within the cell due to distant stellar objects.

5 **Local to Local Translation**
   - Translate local expansions from parent cells to their children, refining the local potential within each cell.

6 **Direct Calculation**
   - Calculate the gravitational forces directly for nearby stars within a cell or neighboring cells.

7 **Force Evaluation**
   - Determine the gravitational forces on each object by summing contributions from both the local expansions and the direct calculations for nearby stellar objects.

Computational Complexity

- **Time Complexity** - Approximately $O(N)$ or $O(N \log N)$, depending on the implementation and the specifics of the problem.
- **Space Complexity** - $O(N)$ for storing the positions, velocities, forces, and expansion coefficients.

4.3.4. Analysis on Fast Multipole Method versus Naive Method

Both algorithms were implemented in a basic and simple way in Python to run simulations to determine, in a controlled way, the best method to be further taken into account to be implemented. There were 213 snapshots of performance for 10000 game objects with randomized initial positions, velocities and accelerations, which would be approximate to the scope of the final project. So the results of these tests are shown in Table 4.2 and contrary to theoretical computations for time complexity, the naive method (List processing time) was more efficient, by having a median value of 27 versus the 94 from the Fast Multipole Method. This difference can also be seen in the mean values for the algorithms, where the naive method has lower bounds for the delay, but the stability of the FMM method cannot be contested by looking at Figure 4.3.

Table 4.2: FMM vs Naive Statistics

Statistical Method	Fast Multipole Method	Naive
Count	213	213
Mean delay [ms]	95.38	69.77
Median delay [ms]	94	27
Minimum delay [ms]	90	26

The main reason why the naive method, although having $O(N^2)$ theoretical complexity, is more efficient in my particular case is the movement of the main camera, aka the object from which the octree is spanned, so the whole tree needs to be rebalanced at almost every update tick in order to update the position and precision of each child tree. By looking at the density distribution of these two algorithms in Figure 4.4, we can see the two behaviors in action, where FMM has a constant processing time, in order to update the octree at every step, whereas the Naive method has two areas of tendencies, depending on the size of data that needs to be computed, either computes the whole N sized set or the smaller and better to compute K sized set from the filtering function.

In conclusion, contrary to theoretical computations, the best method to use is the naive one, as the number of game objects is still under the point of exhaustive processing delays for such simulations.

4.4. Physics Problems

In order for the objects in the simulation to properly orbit the galactic center, we would need to compute their mass by doing computations regarding the centripetal force, as well as the Universal Law of Attraction for the Newtonian-based physics that are integral to our application.

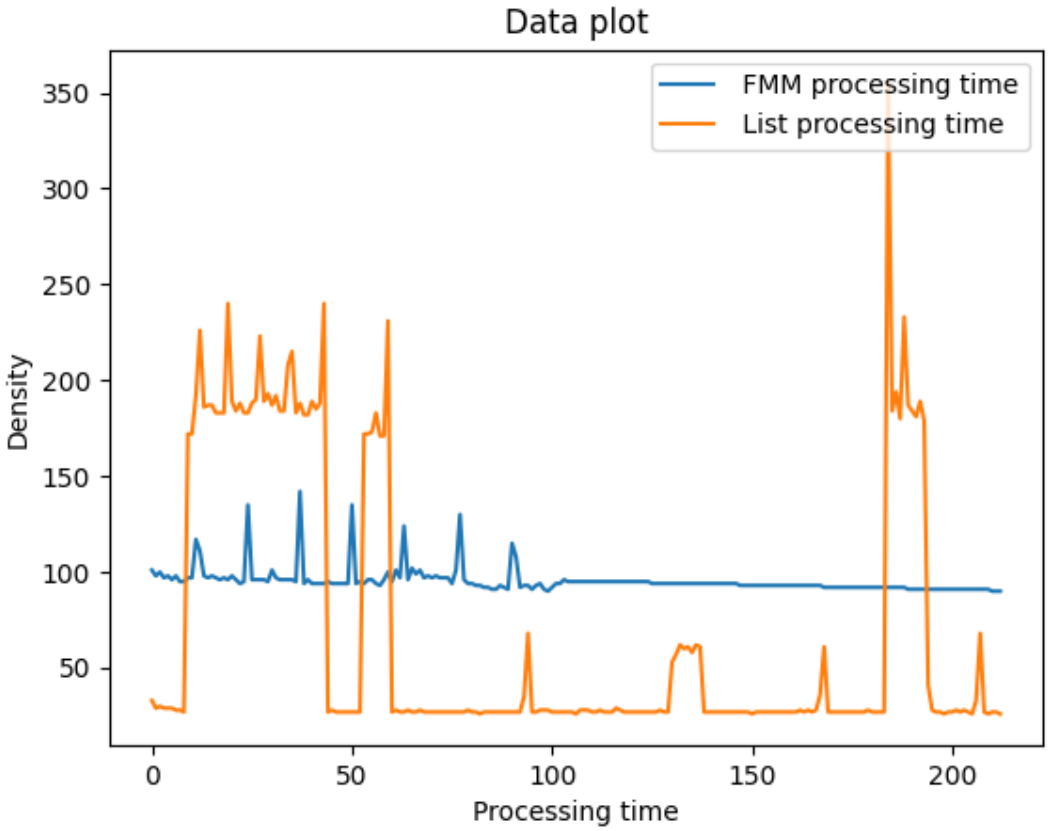

Figure 4.3: Data distribution of Fast Multipole method vs naive method

The centripetal force required to keep an object, in this case, the Sun in circular motion is given by:

$$F_c = \frac{m_{\text{sun}} v^2}{r} \tag{4.1}$$

Using the Universal Law of Attraction[20] between the Sun and Sagittarius A*:

$$F_{\text{gravity}} = \frac{G m_{\text{sun}} m_{\text{Sag A*}}}{r^2} \tag{4.2}$$

We know that the force of gravity in both equations 4.1 and 4.2 is equal, so we get:

$$m_{\text{sun}} \frac{v^2}{r} = \frac{G m_{\text{sun}} m_{\text{Sag A*}}}{r^2} \tag{4.3}$$

Solving for $m_{\text{Sag A*}}$:

$$m_{\text{Sag A*}} = \frac{v^2 r}{G} \tag{4.4}$$

We know the orbital velocity of the Sun around Sagittarius A* is approximately $2,3 \times 10^5$ m/s, as discussed in the article The Circular Velocity Curve of the Milky Way from 5 to 25 kpc[21], and the distance between the Sun and Sagitarrius A* is about 2.55×10^{20} meters, the mass of the Sun is 1.988×10^{30} kg. Plugging in these values in

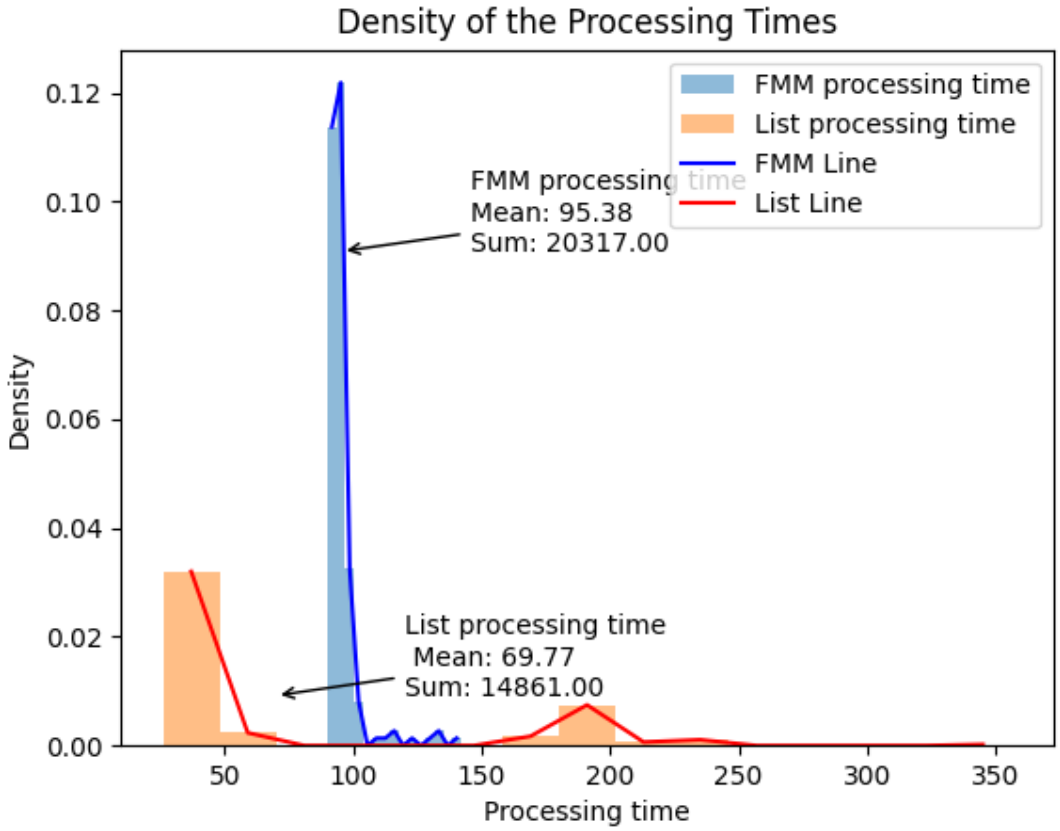

Figure 4.4: Density of Fast Multipole Method vs naive method

the equation 4.4, we can find the mass of Sagittarius A* required to keep the Sun in its orbit.

$$m_{\text{Sag A*}} = \frac{(2.3 \times 10^5 \text{ m/s})^2 (2.55 \times 10^{20} \text{ m})}{6.674 \times 10^{-11} \text{ m}^3 \text{ kg}^{-1} \text{ s}^{-2}} \tag{4.5}$$

$$m_{\text{Sag A*}} = \frac{(2.3 \times 10^5 \text{ m/s})^2 (2.55 \times 10^{20} \text{ m})}{6.674 \times 10^{-11}} \tag{4.6}$$

$$m_{\text{Sag A*}} = \frac{52900000000 \times 2.55 \times 10^{20}}{6.674 \times 10^{-11}} \tag{4.7}$$

$$m_{\text{Sag A*}} = \frac{1.34895 \times 10^{31}}{6.674 \times 10^{-11}} \tag{4.8}$$

$$m_{\text{Sag A*}} \approx 2.021 \times 10^{41} \text{ kg} \tag{4.9}$$

In conclusion, Sagittarius A* would need to be approximately 2.021×10^{41} kg in the simulation, as resulting from equation 4.9, which means it is about 2.3655×10^4 times more massive than it was measured by third parties. This difference in mass is explained firstly by the usage of Newtonian-based equations, which provide a simpler and more rough view of celestial dynamics, as well as by not considering the total mass of the galactic center, just the mass of supermassive black hole in the center of it, which,

as reviewed in the article by V. V. Bobylev and A. T. Bajkova[22], is within the general consensus for the galactic center. For simplification reasons, we will use this computed value to act as the center of the simulation.

4.5. Unity3d

4.5.1. Component

Unity3D is a renowned and powerful game engine that uses a component-based architecture, making it a popular choice for game development. This architecture allows the users and developers to create complex and dynamic environments by attaching various properties to game objects called components[23]. Each component provides a specific functionality, which simplifies the process of building and maintaining behaviors within the game. Here are some key components in Unity3D, explained in detail:

- **Transform Component** is essential in Unity3D since it determines a game object's location, rotation, and scale. Unity contains a Transform component for each game object, which controls the item's orientation and placement in the world.
- **Mesh Renderer** is responsible for rendering the mesh of a game object. It uses materials to build the object's apparent look based on the mesh data supplied by the Mesh Filter.
- **Script Components** script components enable developers to provide game objects with unique behaviors. The game object can be controlled by these scripts in a variety of ways, from basic motions to intricate AI behaviors.
- **Camera Component** is essential for rendering the scene from a particular viewpoint, that of a main camera or a set of cameras. For example, a set of cameras can be used to render the main view and a minimap.

4.5.2. MonoBehaviour

All the classes natively to Unity use MonoBehaviour as their parent class. The MonoBehaviour [24] instance provides a framework that allows attaching a script to each object in the scene. It enables the programmer to work with coroutines or manipulate messages received from various events in the scene. The main functions that allow event management are:

- **Awake** is called when the script instance is loaded. It is used especially for initializing any variable or game state before loading the scene.
- **Start** is called every time the object is instantiated or when the scene is loaded. This function is generally used to instantiate the attributes of an object
- **Update** is called every frame of execution. This function is the most used in Unity, allowing monitoring of the state of the object in the scene. The function becomes inactive when the object is disabled. This method also has other implementations as follows:
  - **LateUpdate** is called after all Update functions have been called[25]
  - **FixedUpdate** is called every fixed frame-rate frame used for physics simulations[26]
- **OnDestroy** is called every time the object is destroyed
- **OnCollisionEnter** is called every time a collision is detected
- **OnTriggerEnter** is called every time a collision is detected

4.6. Design Patterns Used

4.6.1. Singleton

The singleton design pattern in Unity3d is a way to create a class that has only one instance throughout your entire game, as seen in Figure 4.5. This single instance can be accessed from anywhere in the code using the class name itself followed by the property *Instance*. The pros and cons of using this approach are:

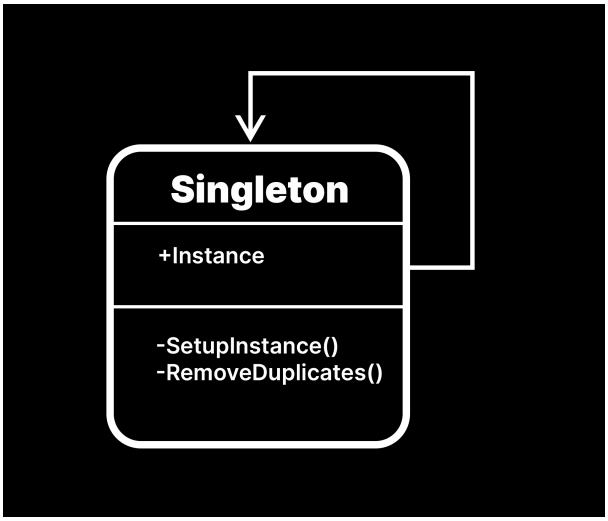

Figure 4.5: Singleton UML[3]

Pros:

- **Global Access** - they are a convenient way to access shared data or functionality from anywhere in your code
- **Enforced Uniqueness** - since there's only one instance, you can be sure that any changes you make to the data in the singleton will be reflected everywhere
- **Simple Solution** - singletons can be a quick and easy way to share data or functionality

Cons:

- **Tight Coupling** - Singletons can make the code more tightly coupled, meaning different parts of the code become reliant on the singleton existing and functioning correctly
- **Hidden Dependencies** - it can be easy to forget about them and accidentally create dependencies on them in other parts of the code

Conclusion

While singletons can be a convenient tool, it's important to be aware of their drawbacks before using them extensively, so only in a couple of situations would be permitted

[3]Singleton on `https://blog.unity.com/games/level-up-your-code-with-game-programming-patterns`

their usage, as it would be an implementation of Unity Main Thread Dispacher for transforming parallel computations into sequential ones in the case of objects inheriting the MonoBehavior class.

4.6.2. Component

Traditional game development approaches often suffer from sprawling codebases characterized by lengthy, convoluted methods. This reliance on monolithic "God Objects," exemplified by Singleton Game Managers, creates a breeding ground for bugs that seem to proliferate with each new feature.

Fortunately, the GameObject-Component Pattern, the cornerstone of Unity's design philosophy, offers a solution. This pattern advocates for the decomposition of these monolithic structures into smaller, modular components. These components can be effortlessly attached, modified, or detached from GameObjects without requiring new code. The resulting codebase fosters greater flexibility and maintainability.[27]

4.6.3. Model View Controller

The MVC design pattern is the go-to pattern in visual projects and, as a consequence, using it in Unity is the logical extension of this principle, as the main goal of Unity is to deal with data and present it in a user-friendly manner, the abstract representation of this pattern being shown in Figure 4.6. As a consequence, on the official tutorials and blogs created by the Unity Development team [28], they recommend using MVC or MVP as a way to structure the project

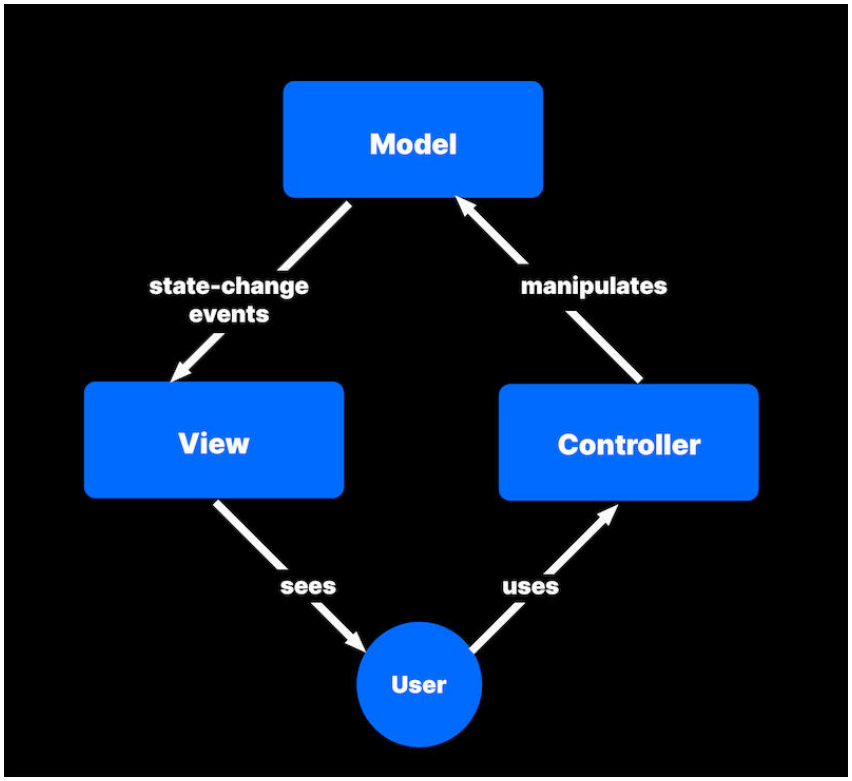

Figure 4.6: MVC Controller[4]

[4]MVC on `https://unity.com/how-to/build-modular-codebase-mvc-and-mvp-programming-patterns`

Pros:

- **Separation of Concerns** - it promotes clean code by separating the data, presentation, and user interaction, thus it makes the code easier to understand, maintain, and modify.
- **Reusability** - since the model and view are independent, they can potentially be reused across different parts of the project.

Cons:

- **Complexity** - MVC can introduce some overhead, as setting up and managing the different components can add complexity.

Conclusion

MVC is a powerful pattern for structuring code in Unity, especially for larger or more complex projects, as it promotes clean, maintainable, and testable code.

4.6.4. Observer

The main idea of this pattern is centered around two types of objects: the subject-object is the one that raises the event, and the observer-object responds to that stimuli[29]. The abstract view of such a case can be seen in Figure 4.7.

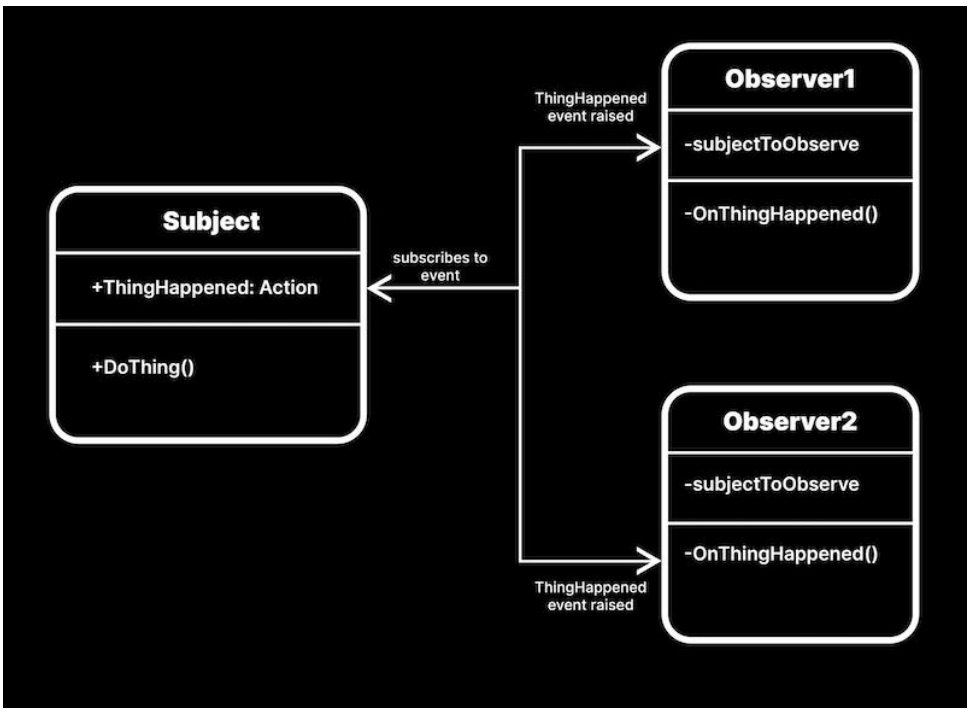

Figure 4.7: Observer Abstract[5]

[5]Observer on `https://unity.com/how-to/create-modular-and-maintainable-code-observer-pattern`

Pros:

- **Decoupling** - Subjects don't know specific observers, promoting flexibility.
- **Maintainability** - modifications or additions have less impact on other parts of the system.
- **Reusability** - observers can be used in various contexts without changes.
- **Readability** - dependencies are clearly defined, enhancing code clarity.

Cons:

- **Performance overhead** - notifying all observers whenever the subject changes can have a performance impact, particularly if there are many observers or frequent updates, as the most probable communication frequency would be 1 message/1 frame.
- **Debugging challenges** - debugging issues can become more complex due to the indirect communication between objects, as tracing the flow of information through the subject-observer network can be time-consuming.

Conclusion

Observer is a powerful pattern for structuring code in Unity, as it offers valuable benefits for managing dynamic communication in games, as it can leverage the accessibility problems between button presses and their actions or collision detection and their responses.

Chapter 5. Detailed Design and Implementation

5.1. Positioning

5.1.1. Problem Statement

Schools have limited resources when it comes to the visualization of the theoretical properties or problems. This poses a difficulty for a lot of students to understand what they are computing fully. This can lead to problems along the way, as they are just memorizing and not understanding the Newtonian-based equations and properties. Another issue is the lack of software to visualize basic systems for hobbyists quickly. Table 5.1 shows the description of the problem statement.

Table 5.1: Problem statement

The problem of	• Accurate simulation and visualization of galaxy-wide • Newtonian-based systems in an educational context
Affects	Teachers, Students, Hobbyists
The impact of which is	• Inaccurate physical systems • Difficult visualization of data • Educational purposes for this type of problem
A successful solution would be	• An immersive application where teachers can show simple or complex Newtonian-based systems • Students can visualize the interaction between stellar bodies • Hobbyists can interact and visualize the data that they want

5.1.2. Product Position Statement

Schools are encountering a huge problem when visualizing the educational curriculum, especially in the Physics classes. Although some other games or applications provide in some manner simulation capabilities, there is a lack of additional educational

information that those applications do not provide, such as spectrograms, real-life stars etc. Table 5.2 shows the description of the product position statement.

Table 5.2: Product position statement

For	Teachers, Students, Hobbyists
Who	Need an easy to use Newtonian-based system
The S.T.A.R.S product	Is a standalone application that allows the users to procedurally generated star systems, clusters of stars or galaxies for visual aid or educational purposes
That	Helps to quickly visualize and test equations
Unlike	Cosmomium[1]
Our product	More scientific oriented and easier to use

5.2. Stakeholder and User Descriptions

5.2.1. Stakeholder summary

The main stakeholders that have been identified for this application are shown in the table 5.3. The main stakeholders that were identified are teachers, students, and hobbyists, as this application is first and foremost an educational piece of software.

Table 5.3: Stakeholder summary

Name	Description	Responsibilities
Thesis Coordinator	University employee	Monitors the software progress and correctness
Teacher	Employee in an educational institution	Use educational programs for students in the Physics classes
Student	Enrolled in an educational institution	Attends classes and learns about physics
Hobbyist	A plug-&-play system that would allow fast, but a reliable generation of star systems and/or galaxy systems for game development, educational purposes etc	They ensure that the system is reliable, well-tested and can provide fast, but accurate results.

5.2.2. User summary

From the stakeholder summary, we can identify a series of users that are shown in Table 5.4. These were identified as the main types of people that will use the educational software developed.

[2]Available at `https://github.com/cosmonium/cosmonium`

Table 5.4: Users summary

Name	Description	Responsibilities	Stakeholder
Student	Enrolled in an educational institution	Attends classes and learns about physics	Student
Physics Professor	Employee in an educational institution	Use educational programs for students in the Physics classes	Teacher
Game Developer	Technical person who would exporter use the system for fact checking	Shall use the data provided by the system to create game worlds	Hobbyist

5.2.3. User Environment

1 `Users`
The Module will be operated by only one user at a time

2 `Time limits`
The time limit should be the duration of the laboratory that the students attend.

3 `Collaboration`
Several people can be involved in the tasks that could be performed through this application module, i.e. A render is generated by the game developer, fixed by the professor and displayed as educational material to the students.

4 `Infrastructure`
The OS Platforms on the user side are various versions of Windows with the latest service packs. No network is necessary by default by the application, because no direct collaboration is expected from the users.

5 `Third Party integration`
Several integrations will be available:

- **Game engines** - Unity3D
- **Versioning** - GitHub
- **Marketplace** - Itch.io

5.2.4. Summary of Key Stakeholder or User Needs

Priority levels are as described on a scale of 0-3, where 0 is Most Urgent and 3 is Optional, as we can see in Table 5.5

Table 5.5: Summary of key needs

Need	Priority	Concerns	Current Solution	Proposed Solution
Accuracy of Generated Data	0	Teacher, Student	A lot of other applications focus on simplification of the physics to serve a niche portion of the users	The physics coded in the application should reflect as closely as possible the real world
Usability	0	All	The current implementations do not offer a clean and easy way of interacting with stellar objects. In the educational system, the only visualization method is the school manual	A user-friendly application that enables the users to instinctively know how to use the software
Generation Repeatability	1	Teacher, Student	Because of the chaos theory, explained in this article [30] by Mardling, a lot of scenarios cannot be repeated, even though under the hood the same computations are made	The system is predictable
Hyperfine-tunning	2	Teacher	There is no other application that allows the users to tune their scenarios	Ability to access more advanced options to hyper-tune the generation process.
Low PC components usage	3	All	A lot of the applications on the market require powerful computers or in the case of scientific usage, even supercomputers	The application uses few PC resources in the simulation part

5.2.5. Alternatives and Competition

Cosmonium[3] is a 3D astronomy and space exploration program that lets users navigate the solar system, exploring all the planets and their moons. It also allows visits to nearby stars, revealing the true scale of our galaxy and the Universe. However, our application surpasses Cosmonium by offering superior scientific accuracy and a strong educational focus, making it an exceptional tool for learning and discovery.

[3]Available at `https://github.com/cosmonium/cosmonium`

5.3. Product Features

- `FEAT 01: Simple User Interaction (Usability)`
  The user interface system should be able to support simple user interaction such as rotation, zoom, or translation of the generated object or cluster of objects
- `FEAT 02: Stellar information (Usability)`
  Each star should provide a set of useful information about itself, for example, its position, velocity, spectral class according to the Harvard Class [31], in particular the relationship between the class of the star and their physical properties, as explained in the book [32] by Russell and Henry Norris
- `FEAT 03: Simulation of objects (Generation repeatability)`
  The performance of the application regarding the simulation times and graphic fidelity should be measurable.
- `FEAT 04: Newtonian accurate generation (Accuracy of generated data)`
  The stakeholders require a system as close as possible to the real world, so the underlying physical approximations and mathematical equations will use the best possible available algorithms and data structures to generate and retrieve the data inside the system.
- `FEAT 05: Sharing of inputs (Usability)`
  The application should provide a way to export the current scenario in JSON form. This feature would be especially useful for the teacher and student stakeholders who would be able to exchange files for educational or grading purposes.
- `FEAT 06: Change of physical properties (Visualization hyperfine-tuning, Usability)`
  To create the generation, the system will provide the user with a series of parameters to choose from, to create the desired simulation. These can range from simple size, temperature or density of the star to the overall gravitational constant.
- `FEAT 07: Internal Wiki (Usability)`
  The users would have access to an internal Wikipedia page that can be adjusted from a JSON file in the application's folder. This way, the teacher stakeholder can modify the Wiki with information that is suitable for their class curriculum.

5.4. Other Product Requirements

1 `High Visual Application`
  The application's UI should be very suggestive, i.e., the different stars should be colored with the respective star's color, the path of the stars should be easily identifiable and the data provided should be clearly written.
2 `Usability`
  The UI of the application should easily guide the user's attention and path in order to provide a good User experience.
3 `Maintainability`
  The application should be easy to maintain with patches and regular updates.
4 `Performance`
  The performance of the application regarding the simulation times and graphic fidelity should be measurable.
5 `Extensibility`
  The application should be able to accommodate more functionality as the stake-

holder's expectations grow.

6 `Testability`
The application should provide ways to test its accuracy and features

7 `Documentation`
A User Guide should be developed in both .pdf and .chm formats.

5.5. Use Case Specifications

Now that the requirements and features were identified in the previous pages, we can build a general Use Case diagram, Figure 5.1, to identify the use cases specific to our application. Following this diagram and the features presented, we can build the next use cases: Use a specific size of the GAIA data, Check star details and Check Wiki page for a specific information

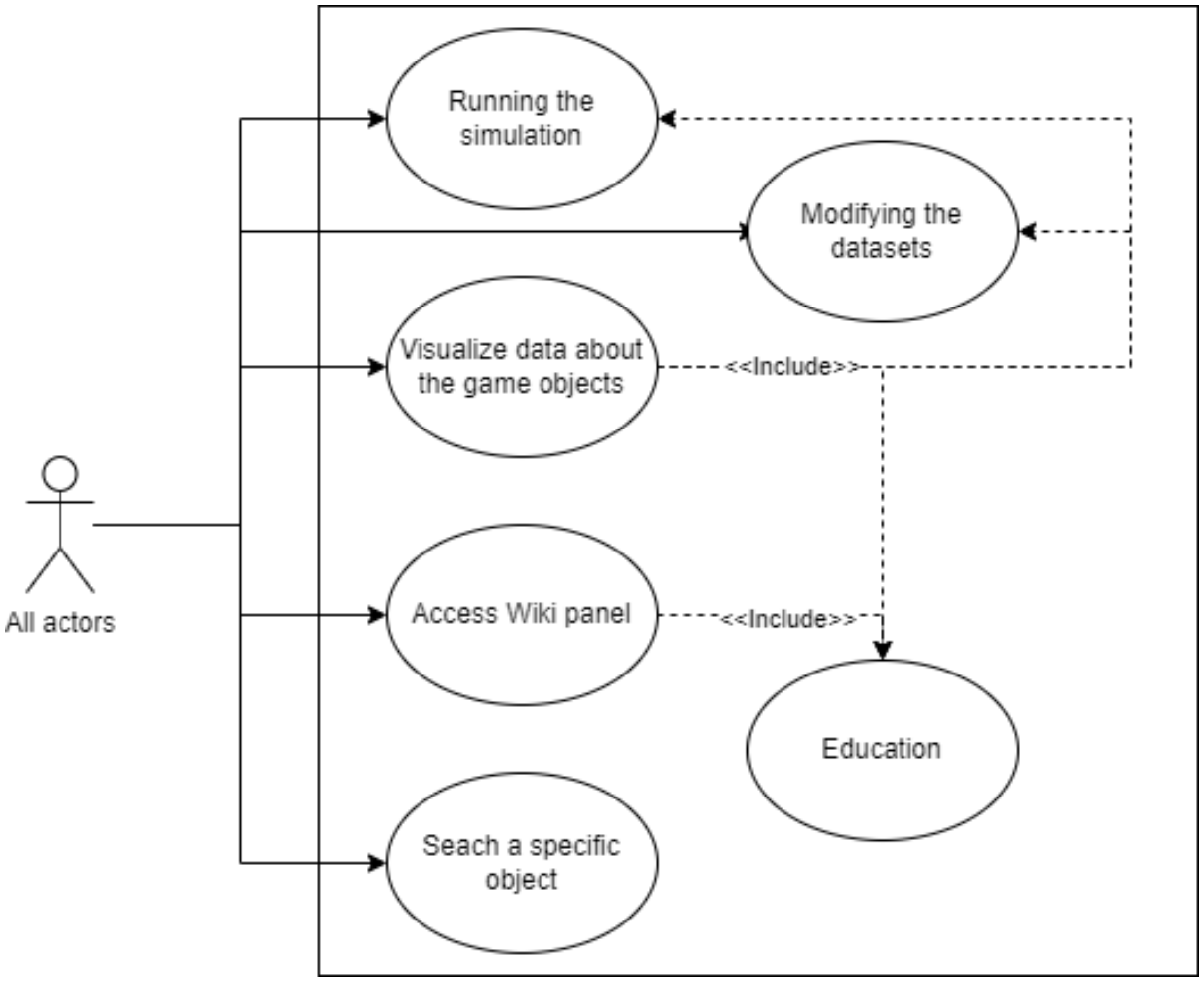

Figure 5.1: General use case diagram

5.5.1. Use a specific size of the GAIA data

1 `Brief Description`
The user will utilize the Python program to generate a new set of data for the simulation. Once the data fetching is done, the Unity application will have to fetch the new data set and utilize it.

2 `Primary Actor`
The primary actor is the teacher.

3 `Stakeholders and Interests`
The teacher is interested in modifying the data sets to be able to utilize them in the context of the class.
Students are interested in acquiring knowledge from different scenarios to understand the subject better.

4 `Flow Diagram`
Figure 5.2 shows the flow of events for this use case.

5 `Basic Flow`
`Use-Case Starts`
5.1 The teacher opens the Python application
5.2 The teacher modifies the size of the dataset
5.3 The Python application fetches the data from ESA servers
5.4 The Python application saves the new dataset
5.5 The teacher opens a new scenario in the main application
5.6 The main application loads the dataset
5.7 The main application computes data using the dataset
5.8 The teacher can visualize the new scenario
`Use-Case Ends`

6 `Alternative Flows`
6.1 `The file has an output path different from StreamingAssets`
This can occur in step 5d
`Alternative Flow Starts`
6.1.1 The user will locate the output path of the file
6.1.2 The user will locate the path of StreamingAssets
6.1.3 The user will move the file to StreamingAssets
`Alternative Flow Ends`
The flow returns to step 5e

6.2 `The Python application encounters an error because of networking connection`

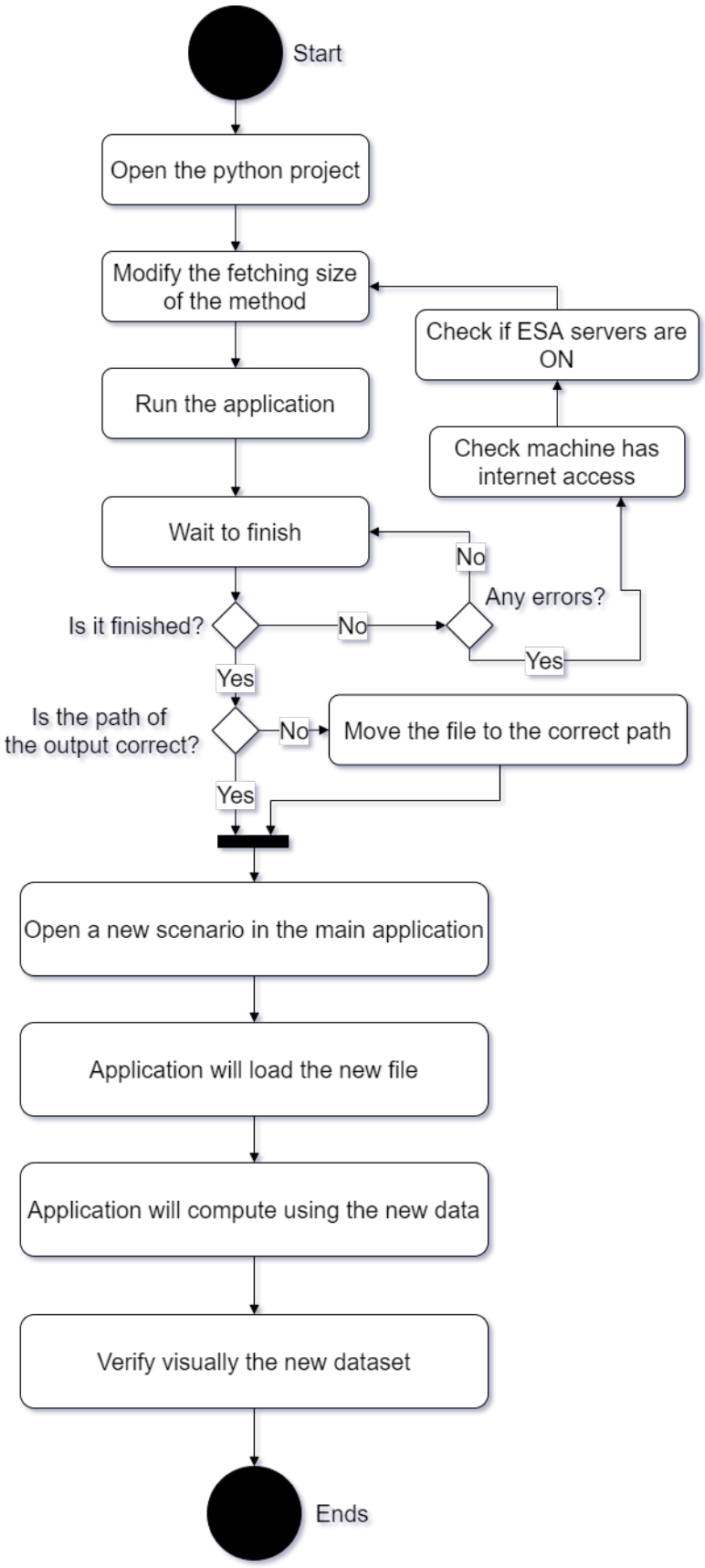

Figure 5.2: Flow chart for GAIA fetching use case

This can occur in step 5c
`Alternative Flow Starts`
6.2.1 The system will inform the user of the issue
6.2.2 The user will have to make sure the machine is connected to the internet
6.2.3 The user will have to make sure ESA servers can be accessed
6.2.4 The user will have to modify the fetching size
`Alternative Flow Ends`
The flow returns to step 5b

7 `Preconditions`
7.1 The machine is connected to the internet
7.2 User has write access to the paths
7.3 ESA server is online and accessible

8 `Postconditions`
NONE

5.5.2. Check star details

1 `Brief Description`
The user will be able to access individual stars in order to see additional information about them.

2 `Primary Actor`
The primary actors are the hobbyist, the teacher, and the student.

3 `Stakeholders and Interests`
The teacher is interested in showing the class different stars and their properties.
The student is interested in learning the relations between the different properties of the stars.
The hobbyist is interested in visualizing the data of the star that they created.

4 `Flow Diagram`
Figure 5.3 shows the flow of events for this use case.

5 `Basic Flow`
`Use-Case Starts`
5.1 The user opens the main application
5.2 The user opens the scenario
5.3 The system loads the dataset
5.4 The system computes data using the dataset
5.5 The user clicks on a star
5.6 The user clicks on more details
5.7 The user can visualize the detailed data regarding the star
5.8 The user closes the panel

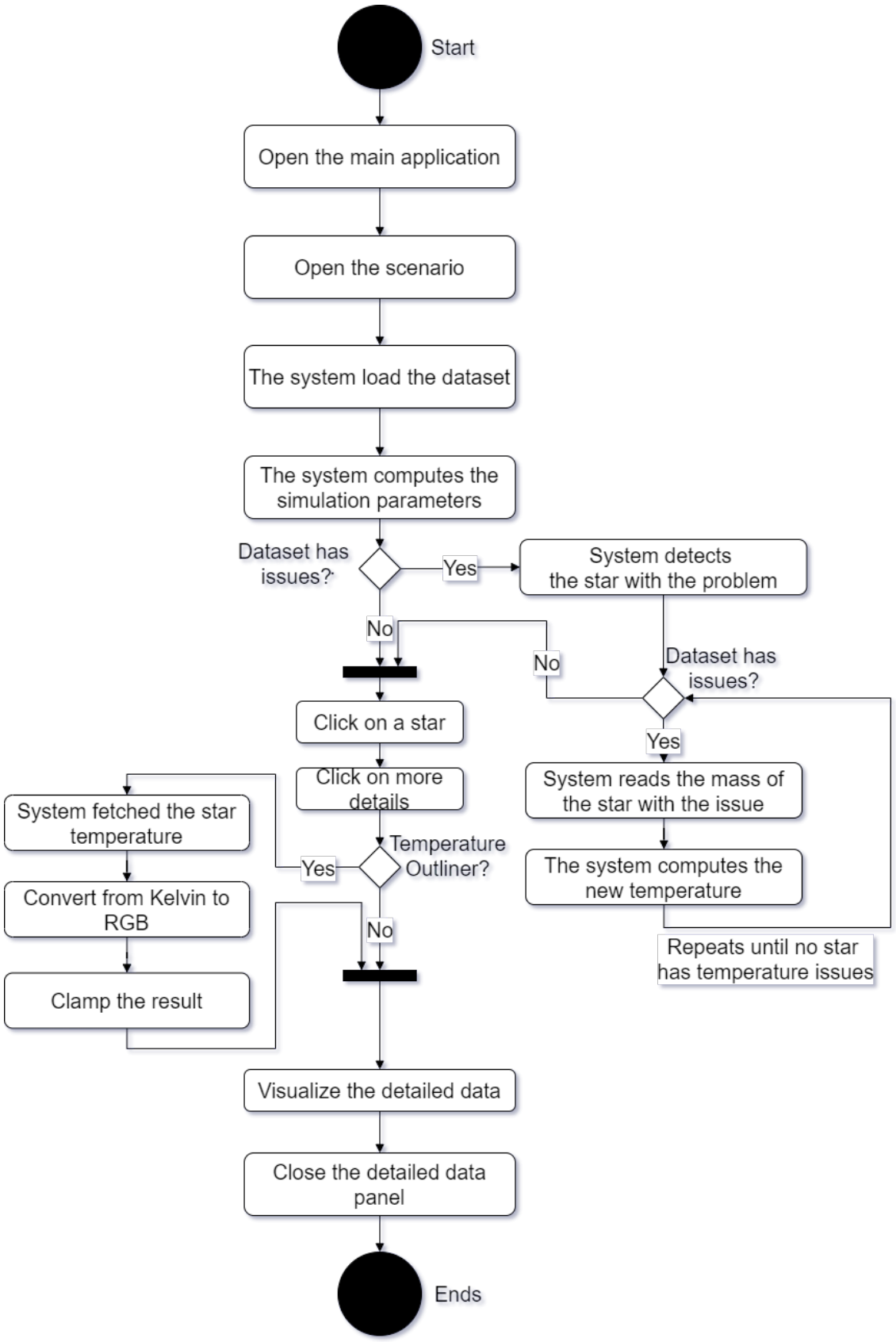

Figure 5.3: Flow chart for checking star details use case

`Use-Case Ends`

6 `Alternative Flows`

6.1 `The star doesn't have a recorded temperature`

This can occur in step 5d

`Alternative Flow Starts`

6.1.1 The system will detect the absence of the temperature

6.1.2 The system will read the mass of the star

6.1.3 The system will compute a new temperature according to this article [33]

`Alternative Flow Ends`

The flow returns to step 5d

6.2 `The star is an outline in the dataset regarding the temperature`

This can occur in step 5g

`Alternative Flow Starts`

6.2.1 The system will take the star's temperature

6.2.2 The system will convert the star's temperature from Kelvin to RGB

6.2.3 The system will clamp the results in the interval [0, 255]

`Alternative Flow Ends`

The flow returns to step 5g

7 `Preconditions`

7.1 There is a scenario stored in StreamingAssets

7.2 User has read and write access to the paths

8 `Postconditions`

NONE

5.5.3. Check Wiki page for a specific information

1 `Brief Description`

In case of questions about the terminology used or specific questions, the user can refer to the built-in wiki page

2 `Primary Actor`

The primary actors are the teacher and the student.

3 `Stakeholders and Interests`

The teacher is interested in writing and maintaining information that relates to the subject.

The student is interested in learning by reading additional information.

4 `Flow Diagram`

Figure 5.4 shows the flow of events for this use case.

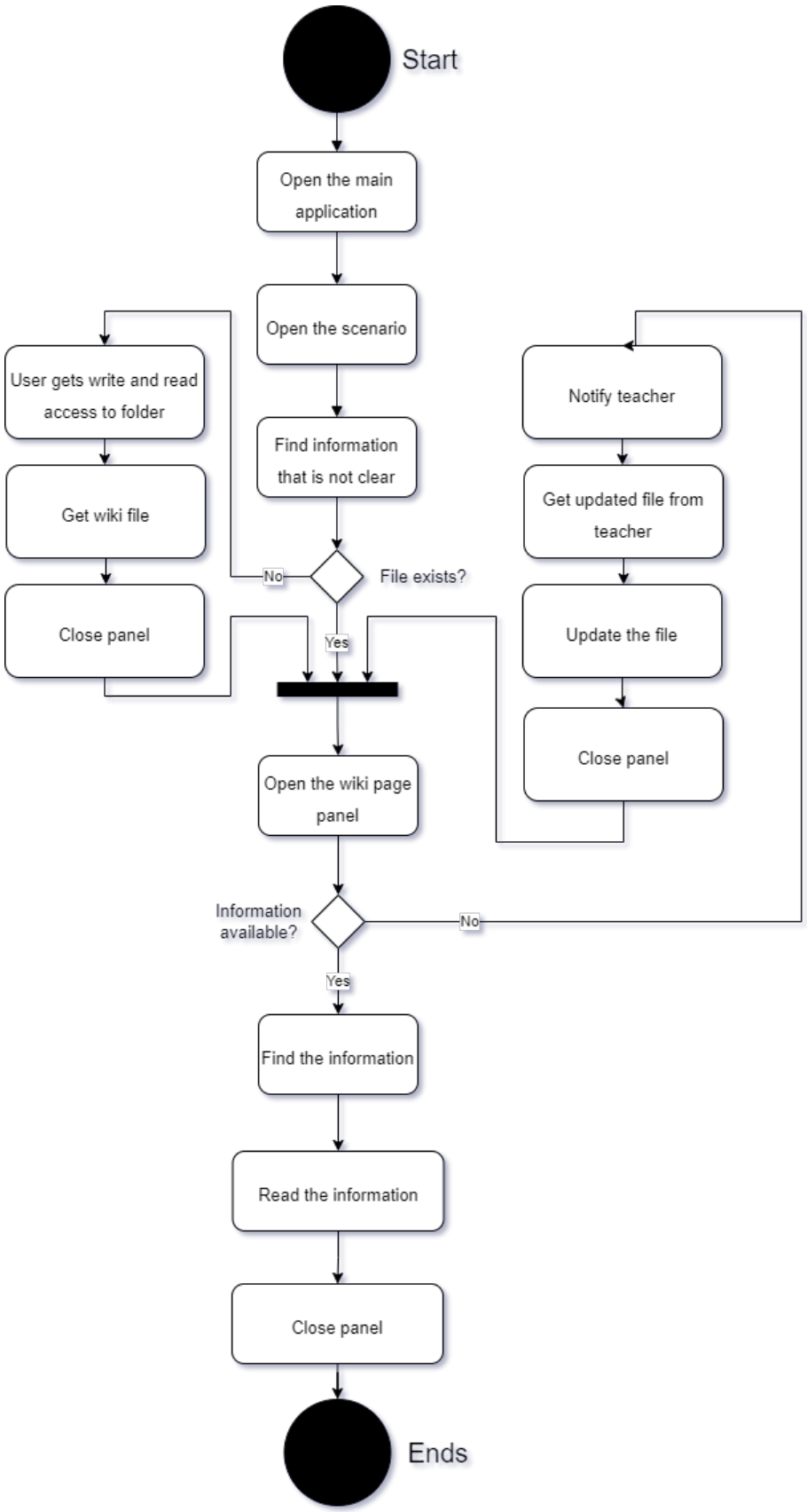

Figure 5.4: Flow chart for wiki page use case

5 `Basic Flow`
`Use-Case Starts`
5.1 The student opens the main application
5.2 The student opens the scenario
5.3 The student finds information that is not clear
5.4 The student opens the wiki page panel
5.5 The student finds the information that interests them
5.6 The student reads about the specific concept
5.7 The student closes the panel
5.8 The user closes the panel
`Use-Case Ends`

6 `Alternative Flows`
6.1 `The wiki file is empty or not accessible by the system`
This can occur in step 5d
`Alternative Flow Starts`
6.1.1 The student will ask the teacher for write and read access to StreamingAssets
6.1.2 The student will get the correct wiki file in that location
6.1.3 The student will close the wiki panel
`Alternative Flow Ends`
The flow returns to step 5d

6.2 `The information searched doesn't exist`
This can occur in step 5e
`Alternative Flow Starts`
6.2.1 The student will notify the teacher about this
6.2.2 The user with access to the file will update the file
6.2.3 The user will update the existing file
6.2.4 The student will close the wiki panel
`Alternative Flow Ends`
The flow returns to step 5d

7 `Preconditions`
7.1 There is the wiki file stored in StreamingAssets
7.2 User has read and write access to the paths

8 `Postconditions`
NONE

5.6. Detailed Implementation

5.6.1. Project Structure

The project will have the following structure, as presented in Figure 5.5. The project is split between the Unity3d project containing the actual implementation of the educational application and a module with Python scripts used to access the GAIA API

and write the results to be later used by the main application.

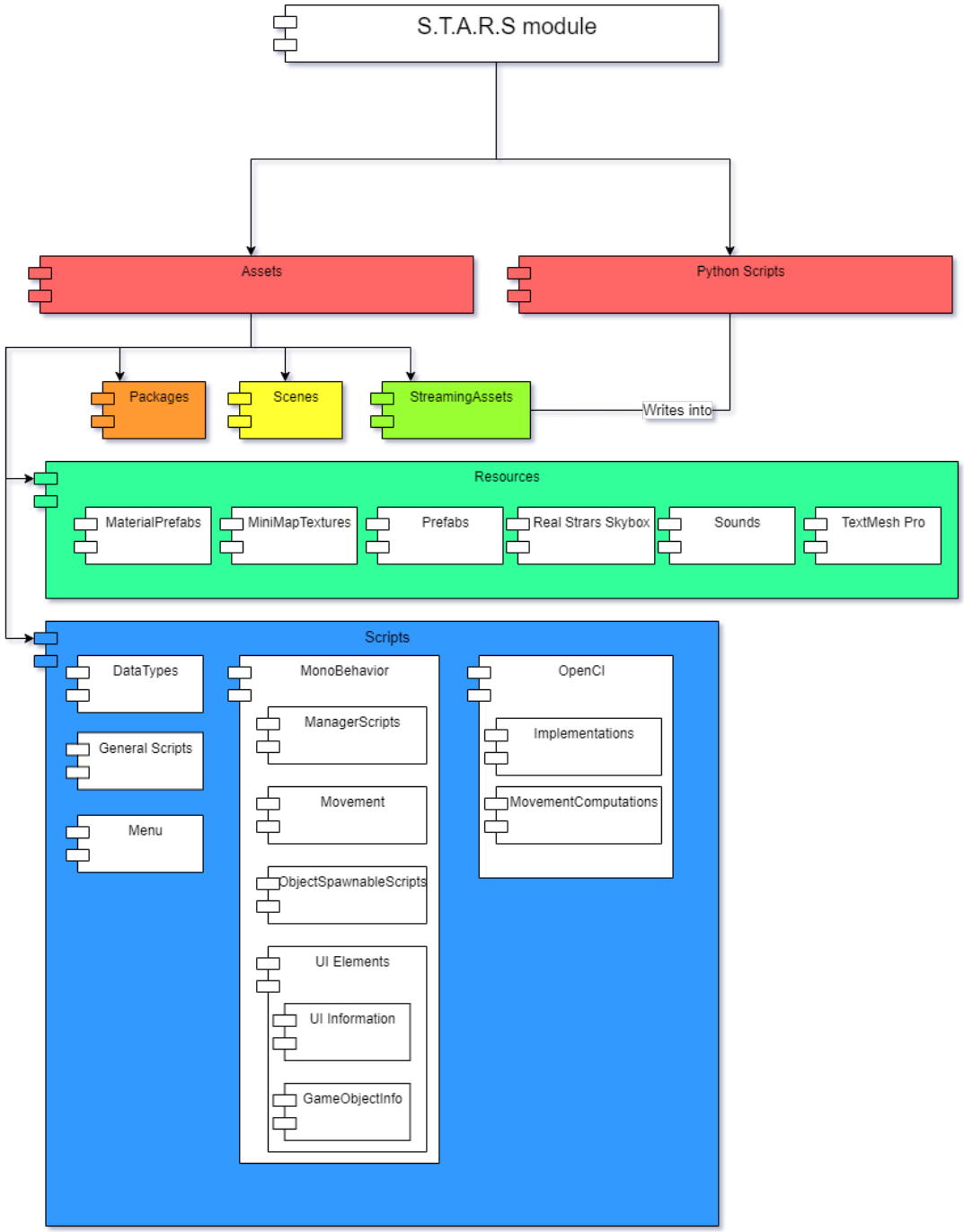

Figure 5.5: General structure of the project

5.6.2. Object Pooling

One of the main problems with the initial design using the naive implementation of the simulation was the number of active objects in the memory.

Two types of tests need to be performed, one before the object pooling and another one after the implementation of the pooling. The two types of tests followed the same methodology, which is explained below:

1 Set the simulation speed to a coefficient C
2 For a number of N frames, do:
  2.1 Record the frames per second while standing still
  2.2 Record the frames per second while moving the camera to unloaded chunks
  2.3 Record the frames per second while moving the camera back and forth, thus panning to previously loaded chunks

For both of these experiments, the system was left to stabilize and then the number of frames, N, was set to 15 and the speeds were set gradually in the set [0, 1, 5]. Following this procedure, the results, as can be seen in Table 5.6, show a net improvement of the delay in milliseconds, from an average of 220 ms delay before the object pooling implementation versus an average delay of 150.5 ms delay afterward, so a net decrease of *70.081* milliseconds, or *31.76%*, in the delay.

Table 5.6: Object Pooling tests results

Time Coefficient	**Frame Tested**	**Before Object Pooling**			**After Object Pooling**		
		Standing still [ms]	*Moving camera to unloaded chunks [ms]*	*Moving camera to and from previously seen parts [ms]*	*Standing still [ms]*	*Moving camera to unloaded chunks [ms]*	*Moving camera to and from previously seen parts [ms]*
0	1	49	163	153	20	73	50
	2	30	138	141	24	85	43
	3	36	156	147	28	69	40
	4	31	142	152	31	87	40
	5	40	168	167	24	69	41
	6	43	131	121	21	74	47
	7	43	158	159	27	70	48
	8	42	167	163	23	76	49
	9	34	132	122	29	86	48
	10	46	151	147	27	69	44
	11	34	158	164	30	68	50
	12	31	168	174	23	90	47
	13	47	130	131	29	60	42
	14	46	163	157	21	86	41
	15	43	166	163	23	74	44
1	1	232	306	313	188	197	193
	2	234	258	252	181	210	202
	3	230	252	247	203	202	187
	4	241	280	286	214	210	209
	5	233	252	247	181	200	218
	6	210	254	262	200	214	217
	7	207	347	338	214	185	191
	8	229	329	333	182	188	214
	9	214	308	304	213	215	197
	10	237	282	273	198	189	220
	11	231	267	274	197	214	216
	12	242	322	320	188	212	206
	13	204	320	317	198	196	197
	14	230	348	338	195	217	212
	15	227	275	280	212	216	210
5	1	220	313	321	195	219	204
	2	203	287	287	181	202	183
	3	250	331	330	180	213	198
	4	245	274	264	192	185	212
	5	227	275	265	206	196	208
	6	215	257	247	207	220	188
	7	206	345	342	199	205	187
	8	200	348	339	216	208	188
	9	240	335	339	215	214	208
	10	239	280	283	217	210	198
	11	244	326	319	186	212	186
	12	204	336	336	182	215	190
	13	230	256	259	216	213	194
	14	201	291	285	219	181	194
	15	207	322	331	195	194	186

5.6.3. GAIA data fetching

Code section 5.1 exemplifies the call to the GAIA API. The output conditions were selected following the GAIA DATA RELEASE 3 documentation published by the Gaia Collaboration members [34]. and their meaning is the following:

1 parallax_over_error condition ensures that only objects with sufficiently precise parallax measurements relative to their measurement errors are included, indicating high-quality distance estimations.

2 radial_velocity ensures that only objects with recorded radial velocities, which are crucial for understanding their motion in space
3 Conditions for flux over error filter out objects with relatively low signal-to-noise ratios in their photometry measurements, indicating higher confidence in their observed magnitudes across different wavelength bands

```python
def get_gaia_data_top_n(self, n: int):
    query = f"""
                SELECT TOP {n}
                designation,
                teff_gspphot,
                phot_g_mean_mag,
                ra,
                dec,
                parallax,
                pmra,
                pmdec,
                radial_velocity

                FROM gaiadr3.gaia_source

                WHERE parallax_over_error > 10
                AND radial_velocity IS NOT NULL
                AND phot_g_mean_flux_over_error > 50
                AND phot_bp_mean_flux_over_error > 20
                AND phot_rp_mean_flux_over_error > 20
            """
    job = Gaia.launch_job(query)
    result = job.get_results()
    data = result.to_pandas()
    data.columns = map(str.lower, data.columns)
    return data
```

Listing 5.1: Python Gaia fetching

Code section 5.2 exemplifies the computation used to transform from the Parallax and G-band magnitude measured to a value expressed in solar masses. This algorithm was derived from the methods and considerations discussed in the Articles [35, 36] for GAIA R1 and The Article "Gaia Data Release 2 - Using Gaia parallaxes" [37] for Gaia R2, as the methods are the same, as the equipment was calibrated in the same manner and their meaning didn't change.

```python
def estimate_mass(self, g_mag, parallax):
    distance_pc = 1000 / parallax

    # Estimate absolute magnitude in G-band
    M_G = g_mag - 5 * np.log10(distance_pc / 10)
    L_Lsun = 10 ** ((4.74 - M_G) / 2.5)  # Simplified luminosity
   estimation
    M_Msun = L_Lsun ** (1 / 3.5)

    return M_Msun
```

Listing 5.2: Python Gaia mass computation

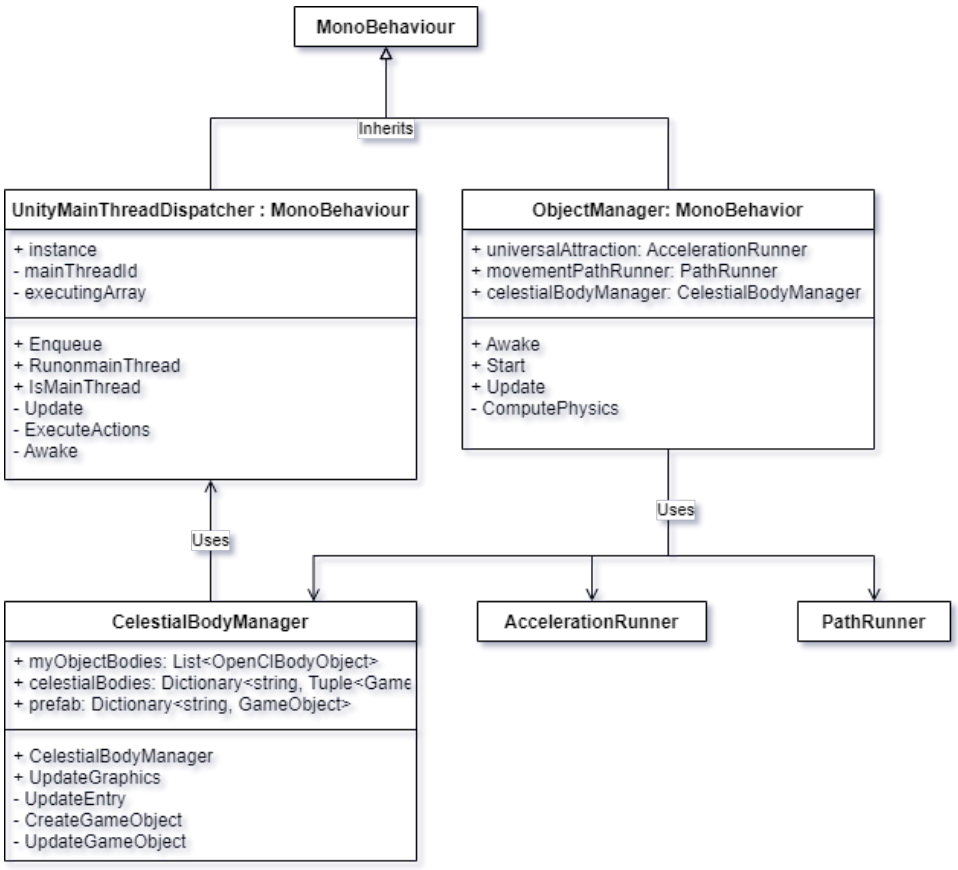

Figure 5.6: ObjectManager UML

5.6.4. GameObjects Manager

As we have to manage thousands upon thousands of game objects, the solution illustrated in Figure 5.6 has been implemented. The main idea of these classes is to fetch, create and update necessary objects to be ready to be visualized. The main classes in this configuration are:

- **UnityMainThreadDispacher** - Unity utilizes only a singular thread for all the operations on the objects inheriting the MonoBehavior class. This leads to some inefficiencies in the processing of such classes and it disables the ability to multithreaded them. Because of these constraints, the class UnityMainThreadDispacher was created to transform multithreading into single threading when operating on MonoBehaviour children. The instance fetches the main thread in Unity by fetching and storing *Thread.CurrentThread.ManagedThreadId* and this is used to locate and execute **Actions** on that thread.
- **CelestialBodyManager** - has the role of storing all the game objects that will be utilized in the simulation. The main steps of this class are the following:
  1 On instancing, it fetches the game objects from the Gaia csv
  2 On *UpdateGraphics*, it creates parallel threads for updating the game objects
  3 Each thread updates the current entry with the distances relative to the respective time dilation
  4 Enqueues an action from UnityMainThreadDispacher to be able to access the Monobehavior data of each game object and to be able to create a new object

in case it doesn't exist in the scene or hide it if it is outside the camera frustum The call of the UnityMainThreadDispacher is mandatory, as the update method modifies MonoBehavior objects attached to the main game objects in a parallel manner, so only the non-MonoBehavior accesses to the memory can be done. On the other hand, the assignments or modifications of MonoBehavour components, for example:

$$obj.GetComponentInChildren < DirectionArrowDraw > ().direction$$

$$obj.GetComponentInChildren < PathDraw > ()$$

$$obj.GetComponent < Body > ().mass$$

Require the main Unity3d thread to be the one doing all the accessing and computations.

- **ObjectManager** - has the role to:
  - create the OpenCL classes instances
  - manage the instance of CelestialBodyManager class
  - initiate the update of CelestialBodyManager class

  All these roles of the ObjectManager class can be visualized better by following the code at the Listing 5.3

```
1  void Update()
2  {
3  if (Mathf.Abs(timeDilationValue) > Mathf.Epsilon)
4  {
5      celestialBodyManager.myObjectBodies = computePhysics(
       celestialBodyManager.myObjectBodies);
6  }
7  else
8  {
9      celestialBodyManager.myObjectBodies = computePath(
       celestialBodyManager.myObjectBodies);
10 }
11
12 celestialBodyManager.UpdateGraphics(Camera.main, timeDilationValue);
13 }
```

Listing 5.3: C# code for GameObject Manager

5.6.5. OpenCL computations

General computation design

For the implementation of the path and acceleration computations, we use as a basis the design from Figure 5.7. The design is split into multiple main classes that have the following meanings:

- OpenCL kernel functions. These functions are running on the GPU, taking full advantage of the parallel computations that these devices were created for
- A OpenCL interface implementation that uses the OpenCL API published by the library **Silk.NET.OpenCL**[4]. This class implements methods related to memory management and kernel arguments.

[4]Available at `https://github.com/dotnet/Silk.NET`

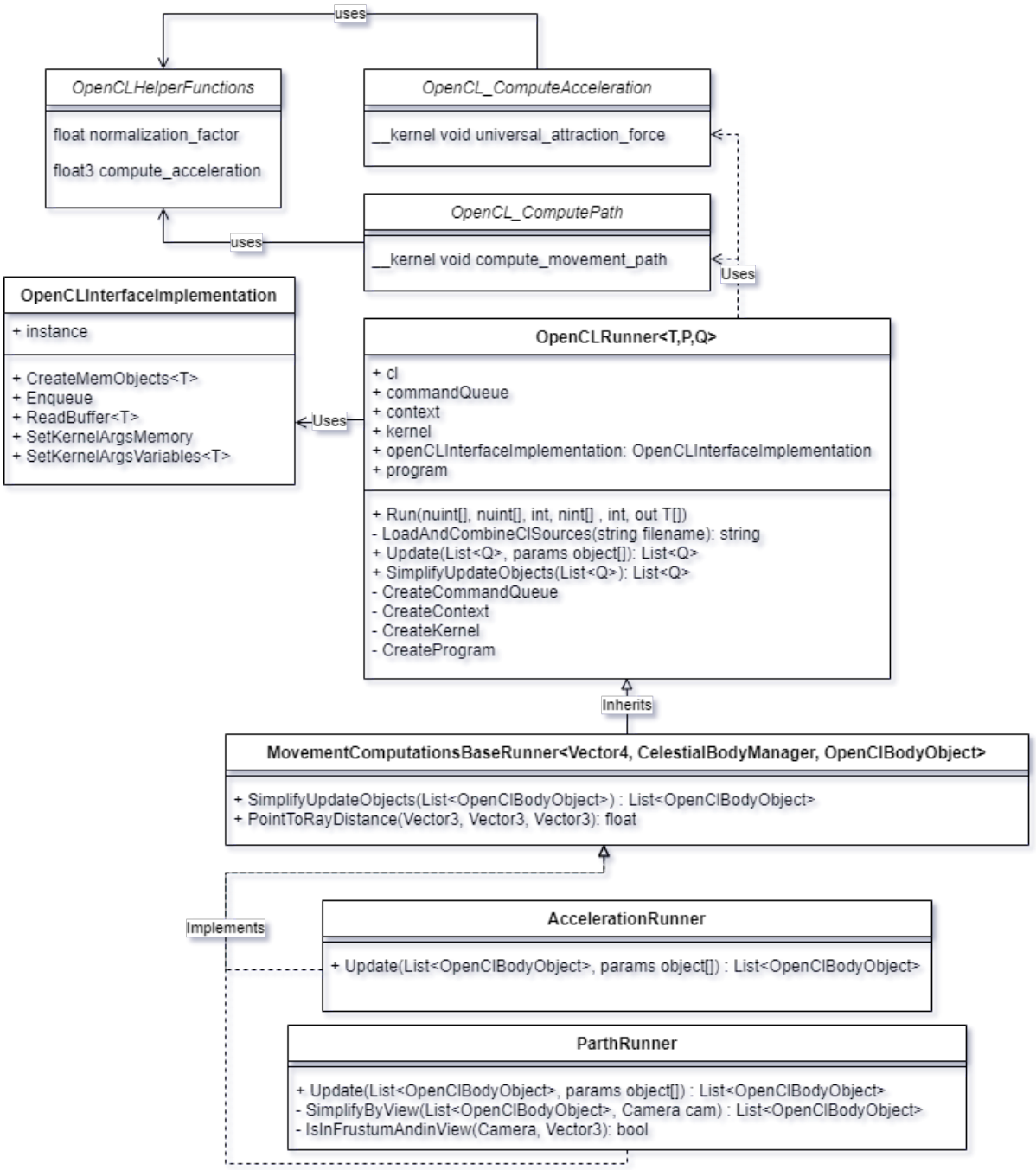

Figure 5.7: OpenCL architecture UML

- The abstract class OpenCLRunner that combines all the necessary kernel files into a single one to be used. It also creates another layer of abstraction over *OpenCLInterfaceImplementation*, in order to create a more *plug-and-play* methodology over the OpenCL interface.
- MovementComputationsBaseRunner is used as the base runner for the path and acceleration classes by creating a concrete implementation of the generic *OpenCLRunner* class. Movement base runner full implementation is available in Appendix A.1
- AccelerationRunner implements the *MovementComputationsBaseRunner* class. In the *Update* method, the flow is specific to parameters and return values of the

acceleration kernel function. Acceleration runner full implementation is available in Appendix A.2

- PathRunner implements the *MovementComputationsBaseRunner* class. In the *Update* method, the flow is specific to parameters and return values of the path kernel function. Path runner full implementation is available in Appendix A.3

After implementation, using 10000 stars and the supermassive black hole in the center of our galaxy, Sagittarius A*, as computational example, we select one random star to check the visual feedback of the first burst of updates, as exemplified in Figure 5.8, we can clearly see the acceleration arrow of the star, as well as the path that the star will take as computed with 10 steps, meaning a movement evolution of 10 million years.

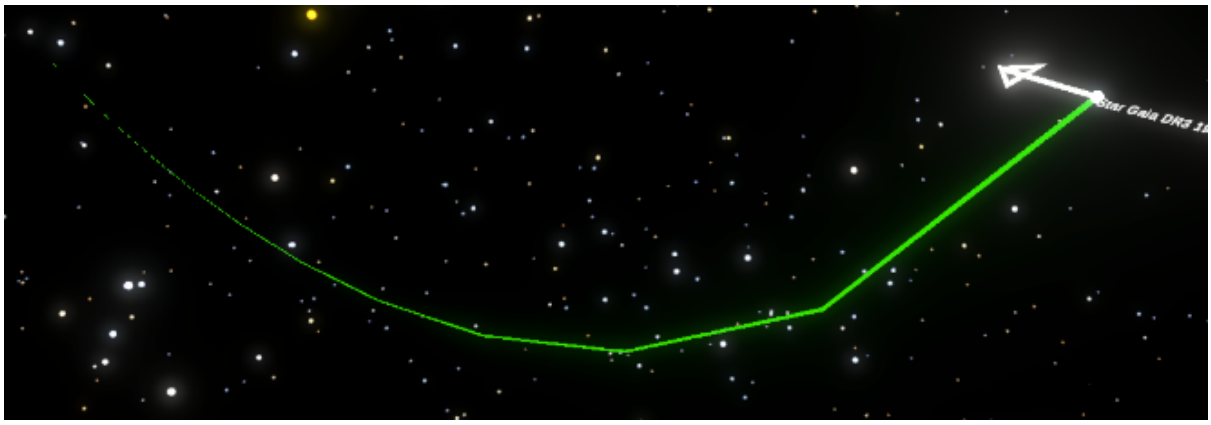

Figure 5.8: Computed path and acceleration for a star

OpenCL helper methods

As seen in the Listing 5.4, the system first needs to compute the force of all the other game objects using Newton's law of universal gravitation, then using that, it calculates the result of Newton's second law of motion, all the equations explained in his work, Principia mathematica[20].

```
float3 compute_acceleration(int current_index,
                            float3 position, float mass,
                            const __global float* globalData,
                            int length, int offset){

    float3 totalForce = (float3)(0.0f, 0.0f, 0.0f);
    for (int j = 0; j < length; j++) {
        if (current_index != j) {
            float3 position_other =
                (float3)(globalData[j*offset],
                         globalData[j*offset + 1],
                         globalData[j*offset + 2]);

            float mass_other = globalData[j*offset+3];
            float3 diff = position_other - position;
            float distSquared = dot(diff, diff) + EPSILON;

            float forceMagnitude = G * mass * mass_other / distSquared;

            totalForce += normalize(diff) * forceMagnitude;
```

```
        }
    }
    return totalForce / mass * normalization_factor(); // kpc/(Myr^2)
}
```

Listing 5.4: OpenCL helper method

Acceleration computation

The OpenCL kernel function for the acceleration computation uses the helper methods shown in Subsection 5.6.5 to compute for each index the current acceleration. As we can see in Listing 5.5, for each step, the system computes the result of the second law of motion of Newton[20], utilizing the mass and position of the current object relative to all the other objects.

```
__kernel void universal_attraction_force(__global float3* acceleration,
                                         __global float* target,
                                         __global float* globalData,
                                         int length, int offset)
{
    int c_index = get_global_id(0);

    if (c_index < length) {
        float3 position = (float3)(target[c_index * offset],
                                   target[c_index * offset + 1],
                                   target[c_index * offset + 2]);
        float mass = target[c_index * offset + 3];

        // kpc/(Myr^2)
        acceleration[c_index] = compute_acceleration(c_index, position,
                                                     mass, globalData,
                                                     length, offset);
    }
}
```

Listing 5.5: OpenCL Compute Universal Attraction Force

Path computation

The OpenCL kernel for the path computation computes for each index, representing each game object, the path for the next n steps. As seen in the Listing 5.6, this kernel function fetches the acceleration, velocity, and position of each object and uses the equation for the displacement of an object undergoing constant acceleration to compute the path of the object for the next n steps, where each time step is considered to be 1 million years.

```
__kernel void compute_movement_path(__global float3* new_position,
                                    __global float* target,
                                    __global float* globalData,
                                    int length, int offset, int steps)
{
    int c_index = get_global_id(0);

    if (c_index < length) {
        float3 position = (float3)(target[c_index * offset],
                                   target[c_index * offset + 1],
```

```
                                    target[c_index * offset + 2]);
        float mass = target[c_index * offset + 3];
        float3 velocity = (float3)(target[c_index * offset + 4],
                                    target[c_index * offset + 5],
                                    target[c_index * offset + 6]);
        float3 acceleration = (float3)(target[c_index * offset + 7],
                                       target[c_index * offset + 8],
                                       target[c_index * offset + 9]);
        // Store the new position back into the global memory
        new_position[c_index * steps + 0] = position;
        float dt = 0.1f;
        // Time in Myr, assuming each step is 1 Myr
        for(int step = 1; step < steps; step++){
            velocity += acceleration * dt;
            position += (velocity * dt) + (0.5f*acceleration*dt*dt);

            // Store the new position back into the global memory
            new_position[c_index * steps + step] = position;
        }
    }
}
```

Listing 5.6: OpenCL Compute Movement Path

Results

Following the implementation of these elements, we can test to see if the computations of the acceleration and path are done on the GPU by querying the TaskManager, Performance panel on the operating system. As we can see in Figure 5.9, the GPU is at around 90% load, meaning that the majority of CUDA cores are computing in parallel the information regarding the states of the game objects.

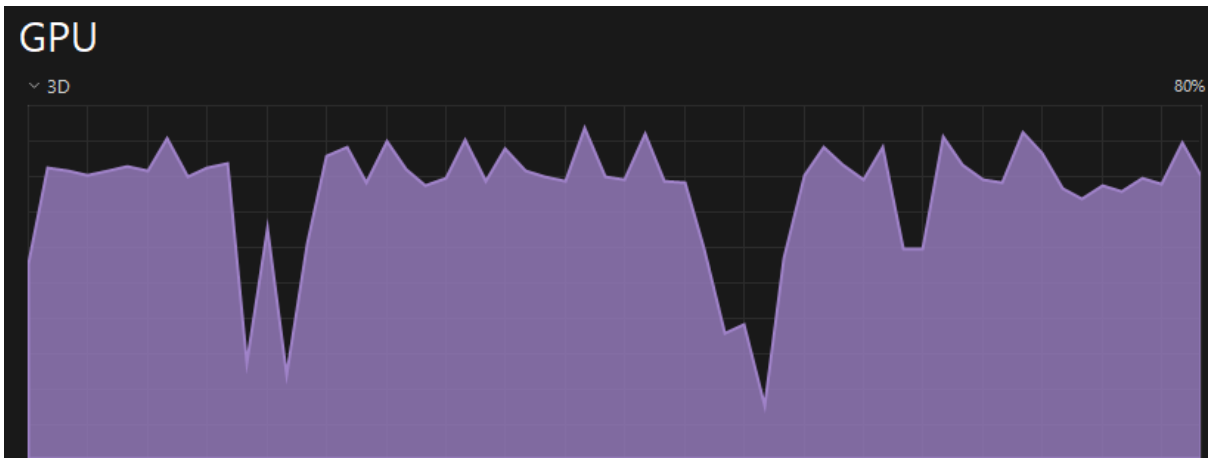

Figure 5.9: GPU usage

5.6.6. Star Search

This module is made up of two interconnecting modules that allow for searching of gameobjects and linking the TextMeshPro links to actual objects as follows:

- **OnPointerClick** class that inherits both MonoBehaviour, to be able to attach it to a gameobject, as well as IPointerClickHandler, which its implementation responds to pointer click events. For full implementation, please refer to Appendix A.6, but the main steps of this class are as follows:
  1 Check if there is any left button event in the application
  2 Get the *TextMeshProUGUI*
  3 Find the index of the intersection between the text and the mouse position called linkIndex
  4 Find the stellar gameobject with that linkIndex in the *myObjectBodies* from the objectManager and call it itemData
  5 If itemData is not null, then compute the normalized direction of the vector between the camera and the item
  6 Translate the position of the main camera to the position of that object, minus a positional offset
  7 Rotate the camera to look at the itemData
- **SearchEngine** class inherits MonoBehaviour in order to implement the OnInputValueChanged that will be used in the GUI editor of Unity on the search input field. As seen in Listing 5.7, once the input field has changed its state and size is greater or equal to 1, the system will search for the objects that contain that value in their name and create a *StringBuilder* with all the objects found.

The most important part of these classes is the *link* from the implementation of SearchEngine, which will be used by the class OnPointerClick to identify what object that name refers to. A name-based solution was chosen, because of the lower RAM allocation and management.

```
public class SearchEngine : MonoBehaviour
{
    // various class methods, fields and properties
    public void OnInputValueChanged()
    {
        List<OpenClBodyObject> objects =
            objectManager
                .celestialBodyManager
                .myObjectBodies
                .AsParallel()
                .Where(obj => obj.name
                                .ToLower()
                                .Contains(searchField.text.ToLower()))
                .ToList();
        StringBuilder sb = new StringBuilder();
        // build the text with
        // "- <link=\"" + obj.name + "\">" + obj.name + "</link>"
        scrollViewText.text = sb.ToString();
    }
}
```

Listing 5.7: C# code for Star Search

5.6.7. Star Details

Class Implementation

The class DetailedStarInformation takes a target star and computes a variety of statistics and information to be displayed to the user, these include the following statistics:

1 Inner and Outer boundaries for the habitable zones relative to the Sun
2 Position in kiloparsecs relative to the galactic center
3 Mass of the star
4 Velocity in kiloparsec per million years
5 Acceleration in kiloparsecs per million years squared
6 Harvard Spectral Class
7 Spectral bands of the star
8 Temperature, Color and Relative Luminosity

Of all of these, the most important calculations are done for the last statistic which will be explained in more detail in the next part

Temperature, Color and Relative Luminosity

The relationship between temperature and color is well-known and highly used, especially when it comes to lights and stars. As the article written by Tanner Helland [38] explains in great detail the mathematical reasoning behind it, they published an addendum in January 2022 leading to Kiryl Ambrazheichyk[5] c# implementation of the system. We modified the algorithm, as Kiryl's algorithm displays white for all the stars in our simulation, which would be the color that the eyes see, but in the educational context, we would need the spectral color of the star.

With all of this knowledge, a custom shader was created using the visual shading tool in Unity3d for material, as seen in Figure 5.10, and exposing certain parameters to code, for example, the temperature of the star, density of solar storms etc we get the simulation texture.

Results

Following the implementation described above also for the star detailed star panel was impossible in the current design and implementation, so a more simplified approach using some ideas of Kiryl's implementation was possible, exemplified in Listing 5.8, where we can observe the result in Figure 5.11. The main idea remains, but the computations are modified to accommodate the color shift that was applied to the stars, as otherwise, all the stars would be white or too bright to display in RGB mode.

```
Color TemperatureToRGB(double temperature)
{
    temperature -= 3000;
    double red, green, blue;

    if (temperature <= 3600)
    {
        red = 255;
        green = temperature / 100 - 2;
        blue = 0;
```

[5]Available at https://gist.github.com/ibober/6b5a6e1dea888c01c0af175e71b15fa4

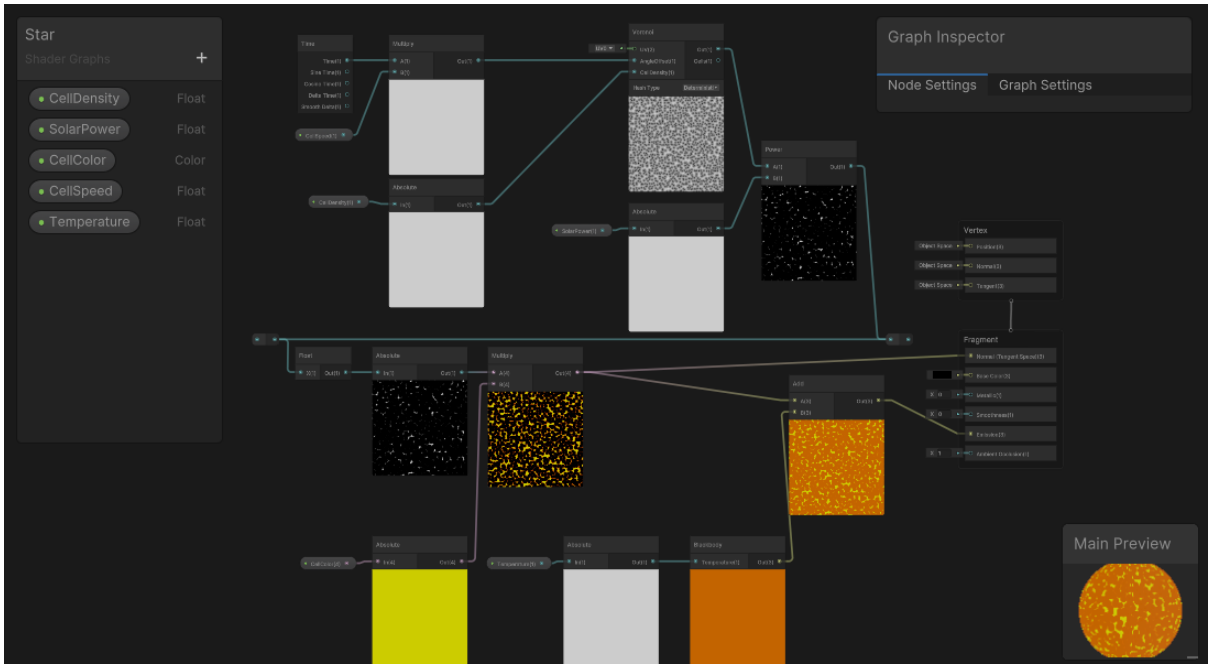

Figure 5.10: Shading tool Unity3D

```
    }
    else
    {
        red = temperature / 100 - 55;
        green = 255 * Math.Pow((temperature / 1000 - 0.66), -0.133);
        blue = 255;
    }

    int r = (int)Math.Clamp(red, 0, 255);
    int g = (int)Math.Clamp(green, 0, 255);
    int b = (int)Math.Clamp(blue, 0, 255);

    return new Color(r / 255f, g / 255f, b / 255f);
}
```

Listing 5.8: C# code for Star Temperature

5.6.8. Map Renderer

This class inherits MonoBehaviour, so it is mandatory to have the following classes:

- **Start** - where the camera is initialized and the min and max bounds
- **Update** - in the form of **LateUpdate** where minimum and maximum bounds are updated, where the rendering of the map is done using the RenderMap method and the rendering of the frustum using RenderCameraFrustrum.

For the full code of the rendering, please refer to Appendix A.5. Below we will present the main steps of RenderMap and RenderCameraFrustrum.

- **RenderMap** - is used to render the map in the following manner:
  1 The graphics library is cleaned
  2 Activate the render target to be the map texture
  3 OpenGL pushes the matrix stack
  4 OpenGL sets a matrix for pixel-correct rendering

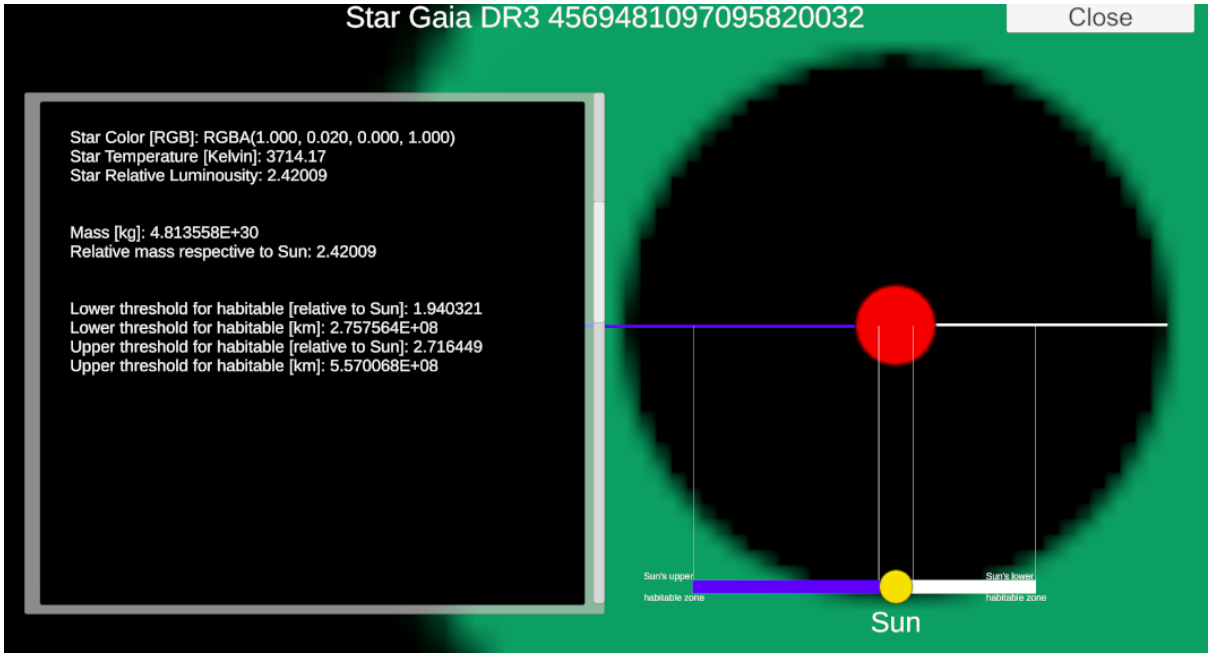

Figure 5.11: Temperature clamping for the star details

5 The system normalizes the star positions on the screen
6 The system creates white rectangles where the positions of the stars are
7 The system draws on the texture of the rectangles
8 OpenGL pops the matrix stack

- **RenderCameraFrustrum** - is used to render the camera frustum in the following manner:
  1 The graphics library is cleaned
  2 Activate the render target to be the frustum texture
  3 The system computes the normalized positions of the frustum corners and loads them in an array
  4 OpenGL pushes the matrix stack
  5 OpenGL sets a matrix for pixel-correct rendering
  6 The system draws OpenGL Vertex3 lines between the 4 corners
  7 The system draws OpenGL Vertex3 lines between the 4 corners and the normalized camera position
  8 OpenGL pops the matrix stack

In doing so, the results of these 2 methods are combined and displayed in the main UI elements for the user to see the position and orientation of the camera as seen in Figure 5.12

5.6.9. Wiki Fetching

The Wiki page of the project is aimed at creating an internal space in the application for reference to the concepts displayed within it and the concepts that a teacher might include. By designing a way to include dynamic change of Wikipedia, the current implementation which includes the class wiki page was done, where for the full code, please refer to Appendix A.7.

The main methods of this implementation will be described below:

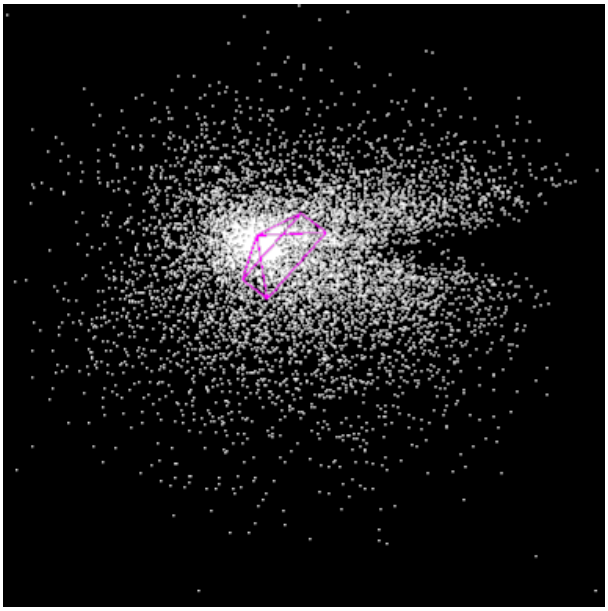

Figure 5.12: Camera frustum and map

- **UpdateWikiData** has the objective of creating all the buttons related to categories in the wiki page based on the contents of the StreamingAssets and has the following steps displayed in Listing 5.9. The Listener on click acts as an observer for that specific event and will call SelectCategory in case of a click on a category.
- **SelectCategory** aims to create all the buttons related to the titles of the pages of the clicked category button. The main idea of the implementation remains the same as in the UpdateWikiData method, but the method that each listener points to relates to the title and description of the information panel that will be created upon clicking.

```
public class WikiPage : MonoBehaviour
{
    // various class methods, fields and properties

    void Start(){dataFetching = DataFetching.Instance; UpdateWikiData()
    ;}
    private void UpdateWikiData(){
        wikiPageObjLists = dataFetching.WikiFetching("wiki_data.json");

        for (int i = 0; i < wikiPageObjLists.Count; i++)
        {
            string category = wikiPageObjLists[i].Category;

            GameObject btn = Instantiate(buttonTemplate);
            btn.GetComponentInChildren<TMP_Text>().text = category;
            btn.transform.SetParent(categoryContent.transform);
            btn.GetComponent<Button>().onClick.AddListener(() =>
    SelectCategory(category));
        }
    }
}
```

Listing 5.9: C# code for Wiki Fetching

Chapter 6. Testing and Validation

6.1. Functional Testing

Functional testing was used in the project to test how the application responds when the user performs an action that changes the state of the system. The testing was carried out to detect the errors in the project that appeared during the development of the application. This type of testing is a form of white-box testing, as it involves verifying the software's functionality through the examination of the internal code structure during the development of the project.

6.1.1. GAIA Data Fetching

Throughout the project, rigorous testing of the GAIA API was required. The availability of the data was extremely important. As we can see in Figure 6.1, the temperature of the stars is not always available, as for only 10 entries, only 8 have available temperatures.

	mass	temperature	pos_x	pos_y	pos_z	velocity_x	velocity_y	velocity_z
count	10.000000	8.000000	10.000000	10.000000	10.000000	10.000000	10.000000	10.000000
mean	0.712118	5665.122070	111.637410	140.552643	0.499659	-398.831910	-9667.380701	-4695.339456
std	0.433715	1626.180908	1003.490138	380.971915	168.047220	18742.540653	22992.026302	19266.328619
min	0.245107	3487.891357	-1242.319397	-325.243438	-138.080117	-44825.975783	-50317.483701	-49157.410968
25%	0.360647	3562.507080	-281.229159	-110.431415	-115.281578	-1951.749392	-22884.290671	-7012.136270
50%	0.624444	4827.235352	-133.171918	37.100566	-36.010914	2335.073474	-10615.068793	1547.860252
75%	1.001186	6333.186035	645.574231	375.448836	42.983001	6464.798196	867.300605	6407.439833
max	1.527396	7557.097656	2271.105747	903.926569	431.236152	29731.809096	31756.693670	15581.481016

Figure 6.1: Gaia Input with 10 entries

For validating the velocities, we can use Newton's law of universal gravitation [20], Equation 6.1, and the formula for centripetal acceleration, Equation 6.2.

$$F = G\frac{m * m_{\text{center}}}{r^2} \tag{6.1}$$

$$a = (v^2)/r \tag{6.2}$$

$$v = \sqrt{\frac{G(m + m_{\text{center}})}{r}} \tag{6.3}$$

Where:

$$F \text{ is the gravitational force between the two objects,} \tag{6.4}$$
$$G \text{ is the gravitational constant,} \tag{6.5}$$
$$m \text{ is the mass of the star,} \tag{6.6}$$
$$m_{\text{center}} \text{ is the mass of the galactic center, } 4.31 \times 10^6\, M_\odot, \tag{6.7}$$
$$M_\odot \text{ is one solar mass,} \tag{6.8}$$
$$r \text{ is the distance between the centers of mass of the two objects.} \tag{6.9}$$

From Equation 6.1 and Equation 6.2, we get the formula for the speed of the object relative to the galactic center, Equation 6.3, which can be transformed easily in a 3D vectorial problem. Solving for each velocity, we get the expected velocities in a margin of 1% of the accepted error percentage.

6.1.2. GitHub Workflow Integration

Once a new feature is implemented or in progress, the developer pushes those modifications on an auxiliary branch of the GitHub repository, where the workflow dotnet-format, shown in Figure 6.2, is run to check for style and formatting correctness.

Figure 6.2: DOTNet-format

Figure 6.3: Build and deploy

After the feature is successfully implemented and the branch is merged to the main branch of the repository, a new workflow is run. This workflow, illustrated in Figure 6.3, compiles the project, creates the necessary files and executable and uploads the contents in an archive to the project's page on itch.io website[1], illustrated in the Figure 6.4

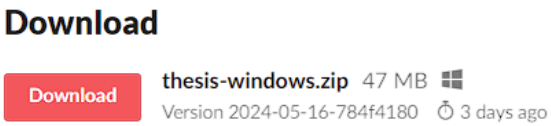

Figure 6.4: Final project hosted on ITCH.IO

6.2. Performance Testing

For performance testing, Unity provides a tool called Profiler[2]. This tool creates charts for the usage of CPU, GPU and Memory, with additional information per every method called during a scenario or time frame.

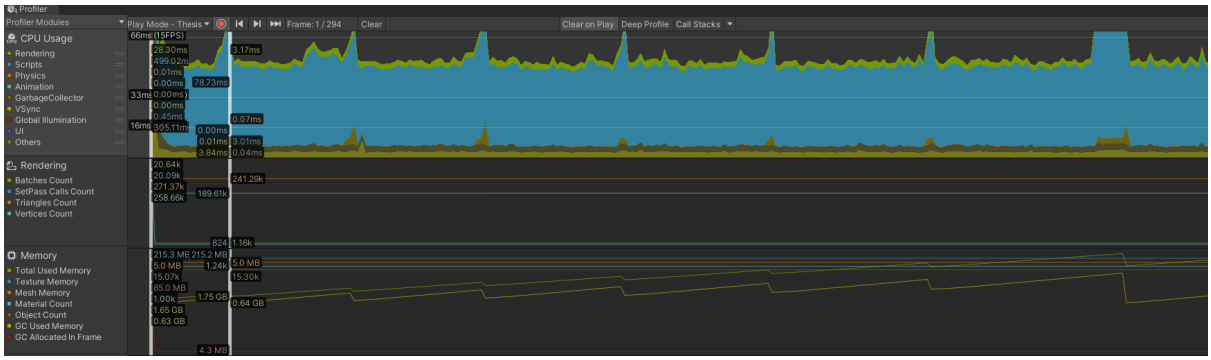

Figure 6.5: Unity Profiler usage

The following scenarios were tested:

- The first vertical line of the Figure 6.5 represents the initial read and allocation of memory for all the stars in the scenario, as well as the initial OpenCL computation of the system. As we can see, the CPU takes around 66ms to process all the data and the memory allocation is around 1.65GB.
- The second vertical line of the Figure 6.5 represents the OpenCL update of the objects that are located in the frustum of the camera, which decreases the memory usage to 0.64GB. This approximate value can be seen throughout the normal update of the system, because of the usage of optimization procedures and conditions explained in chapter 5

As seen in Figure 6.5 and also discussed in chapter 5, the processing time of the CPU once stable has a mean value of 69.77ms.

[1]Available at `https://tavisit.itch.io/thesis`
[2]Available at `https://docs.unity3d.com/Manual/Profiler.html`

6.3. User Testing

The user testing was done in two ways: to test the educational value of this project as well as the real-world impression of such a project. This type of testing is a form of black-box testing, as it involves verifying the software's functionality without examining the internal code structure.

6.3.1. Educational

The Unity3D application was distributed to 3 high schools in Craiova and given to a sample of students to test the simulation game and complete a Google form about their experience. The main objective of this endeavor was to test the efficiency and applicability of such applications in real classroom environments.

The questions were split into two categories, the first part is related to the current way of teaching astrophysics, which is intuitive and the second part is a review of the thesis application.

Do we need such an app?

To test if the high school curriculum would allow for such an application and if the desire for such an application would be feasible from the point of view of the students and the teachers, we introduced a form to be completed upon user testing.

The questions are as follows:

1 "How intuitive were the classes?" on a scale of 1 to 10
2 "How good the visualization of the problems was?" on a scale of 1 to 10
3 "Would you use an application to visualize these concepts?" with "Yes", "No" and "Maybe" answers

The outcome of the last question can be seen in Figure 6.6 where it shows the demand of an application like ours in the educational market is very high. Moreover, the results of these questions are shown in Figure 6.7, which leads to the conclusion that the curriculum is heavy and the intuitive understanding of the concepts presented is not adequate to the standard that the majority of the students would like it to be.

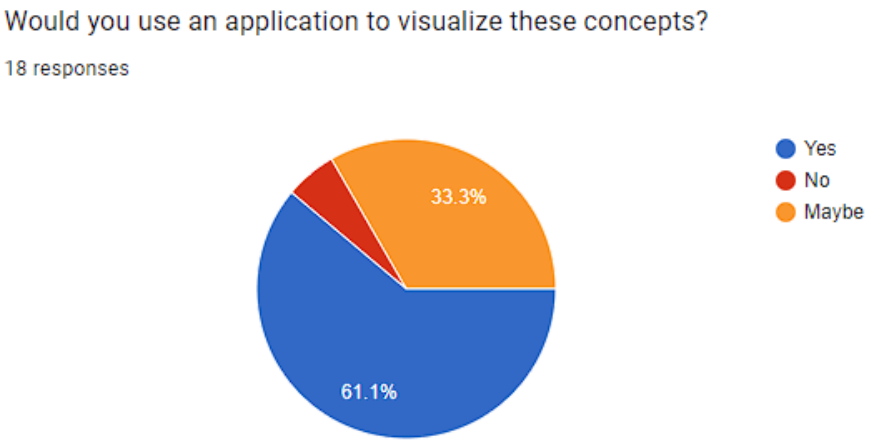

Figure 6.6: Is there a market for educational applications?

Figure 6.7: General questions results

Statistics on the S.T.A.R.S. application

To test the current implementation of the application, after the students were able to properly test it, they were asked a couple of questions on the features and usability of the S.T.A.R.S. application.

The questions are as follows:

1 "How intuitive was the application?" on a scale of 1 to 10
2 "The application helps me to understand better the subject?" on a scale of 1 to 10
3 "The ability to modify the scenarios helped me" on a scale of 1 to 10
4 "The ability to modify the wiki page helped me" on a scale of 1 to 10
5 "The ability to search for a specific star helped me" on a scale of 1 to 10
6 "The ability to check the specific statistics of stars helped me to understand their properties" on a scale of 1 to 10
7 "Would you use the application in the future?" with "Yes" or "No"

The application can be deemed successful based on the responses to these questions, which are displayed in Figure 6.8. The total score for each respondent is 60 points and the minimum satisfactory result is 30 points, but as computed from the form results, the average points, per respondent is 46.78 and the average per question is 7.79. Most students thought the application would be very helpful in their astrophysics classes since it would

make it easier for them to understand what the teacher was saying. This remark highlights how well the program works to improve learning by giving complicated astrophysical ideas interesting and understandable visual aids.

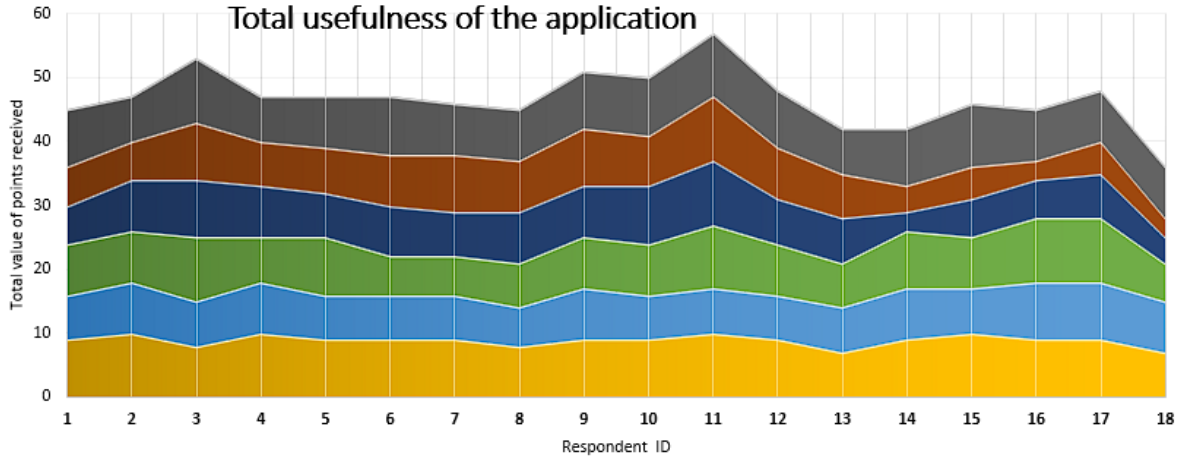

Figure 6.8: Questions about the application results

Teachers can help students make the connection between abstract theoretical notions and practical comprehension by incorporating this application into the classroom. It has long been known that visual aids are an excellent teaching tool, and our application improves this strategy. Moreover, the ability to view models and simulations of celestial bodies occurring in real time gives students a more intuitive and engaging learning experience. As seen in Figure 6.9, most questioners will use this application in the future.

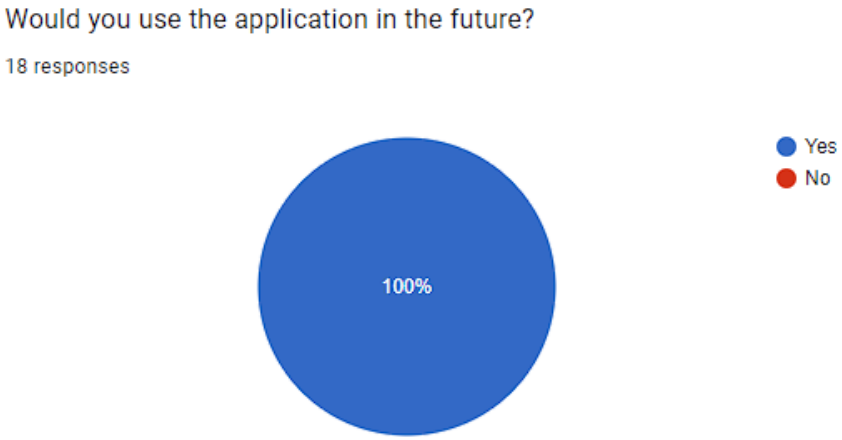

Figure 6.9: Would the users use this application in the future?

6.3.2. Itch.io statistics

Additionally, I am making my application available on a public website, itch.io[3], to increase its accessibility and encourage greater participation and collaboration both within and outside the educational community. By offering the application freely, I aim to

[3]Available at `https://tavisit.itch.io/thesis`

contribute to the pool of educational tools and welcome user feedback that could inspire new ideas and advancements. This approach ensures that more individuals, including students, educators, and researchers, can benefit from the application, leading to wider dissemination of knowledge and resources.

By making the application accessible to a broader audience, I hope to foster an environment of open collaboration where users can share their experiences and suggestions. This interaction is crucial for the continuous improvement of the application, as it allows for real-world testing and input from a diverse group of users. Such feedback is invaluable for identifying potential enhancements and new features that can make the application even more effective and user-friendly.

The information presented in Figure 6.10 shows a somewhat steady number of views and downloads, suggesting that the application is fulfilling its intended reach. This highlights the application's potential to impact research and education while also validating its usefulness significantly. Increased user interaction and exposure are good signs of the application's effectiveness and ability to connect with a broad audience seeking cutting-edge teaching resources.

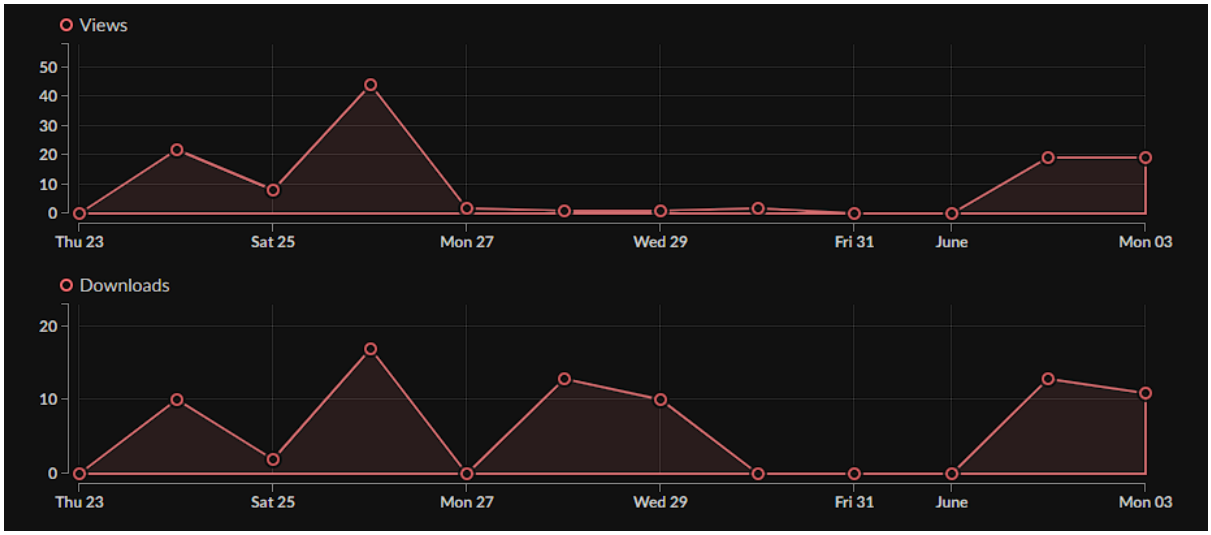

Figure 6.10: Analytics of the application

Chapter 7. User's Manual

7.1. Installation and running Procedure

The Installation Procedure can be split into two main components:

1 Direct Download
2 GitHub Repository
  2.1 Python fetching application
  2.2 Unity3d application

These two methods and their sub-procedures will be explained in the following sub-chapters.

7.1.1. Direct Download

The installation of the PC, using a Windows-based system, is the following:

1 Download the zip file from the ITCH.io page
2 Extract the zip file
3 Go to [current_folder]/build/StandaloneWindows64
4 Select the .exe file, run it and the application should start

7.1.2. GitHub repository

Python fetching application

1 Copy the folder *Python Scripts* to the PC
2 Open the folder in a new *PyCharm* project
3 Run *requirements.txt* to install the required python packages using the command

$$pip\ install\ -r\ /path/to/requirements.txt$$

4 The user should be able to use the Python application now

Unity3d application

1 Clone the repository to the machine,
2 Open Unity Hub and select *Add project from disk* as shown in Figure 7.1,
3 Select the folder where the README.md is,
4 Select the project from the Unity Hub menu,
5 As a first run, the Unity application will create a series of settings and library folders, so it might take a while,
6 Once all the folders are created and loaded, the Unity3d UI will pop up,
7 To create a build, the user should navigate to *File -> Build Settings*
8 This will open the *Build Settings panel* as shown in Figure 7.2

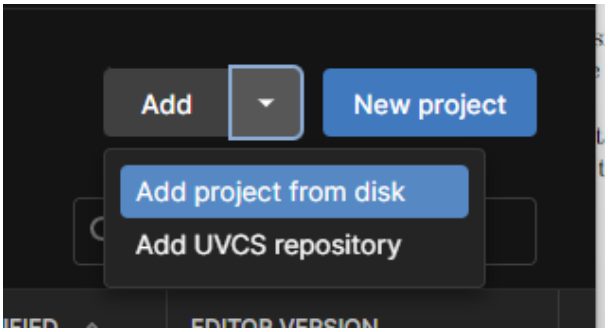

Figure 7.1: Add project From disk

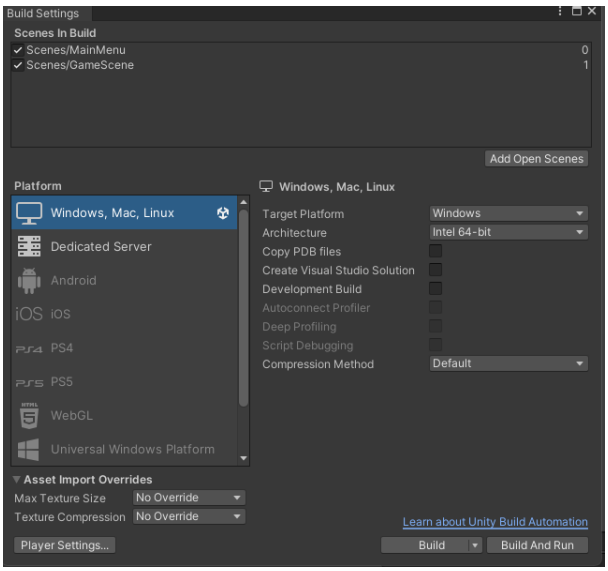

Figure 7.2: Build settings

9 The user should press *Build and Run* button, which will generate and run the application

10 After Unity3d compiles and builds the application, it should start automatically

7.2. Utilization

7.2.1. Modifiable JSONs

By going to the application main folder and going to *Thesis_data/StreamingAssets*, the user will be able to locate the JSONs that can be modified. These are:

1 **galatic_data.json** that represents the stellar data. This file has the formatting presented in Listing 7.1:

```
{
    "Items": [
        {
          "name": [name of the star],
          "mass": [relative to the mass of the sun],
          "temperature": [number in Kelvin],
          "position": {
```

```
            "x": [relative to the galactic center in
    ↪ Kiloparsecs],
            "y": [relative to the galactic center in
    ↪ Kiloparsecs],
            "z": [relative to the galactic center in
    ↪ Kiloparsecs]
        },
        "velocity": {
            "x": [relative to the galactic center in
    ↪ Kiloparsecs/Million years],
            "y": [relative to the galactic center in
    ↪ Kiloparsecs/Million years],
            "z": [relative to the galactic center in
    ↪ Kiloparsecs/Million years]
        }
      }
    ]
}

```

Listing 7.1: Galactic data

2 **wiki_data.json** that represents the wiki data. This file has the formatting presented in Listing 7.2:

```
{
    "Items": [
        {
          "Category": "Category 1",
          "Pages": [
                {
                  "Title": "Concept 1 of Category 1",
                  "Description": "Description 1.1"
                },
                {
                  "Title": "Concept 2 of Category 1",
                  "Description": "Description 1.2"
                }
          ]
        }
    ]
}

```

Listing 7.2: Wiki Page JSON

7.2.2. Actual Game Explanations

Once the application has started, the user will be met with the main menu panel. The user will initialize a new scenario with the button *New Game*.

After the loading screen, the user will be able to see the actual application UI and objects as presented in Figure 7.3. This can be split into several regions that will be explained below in a counterclockwise manner:

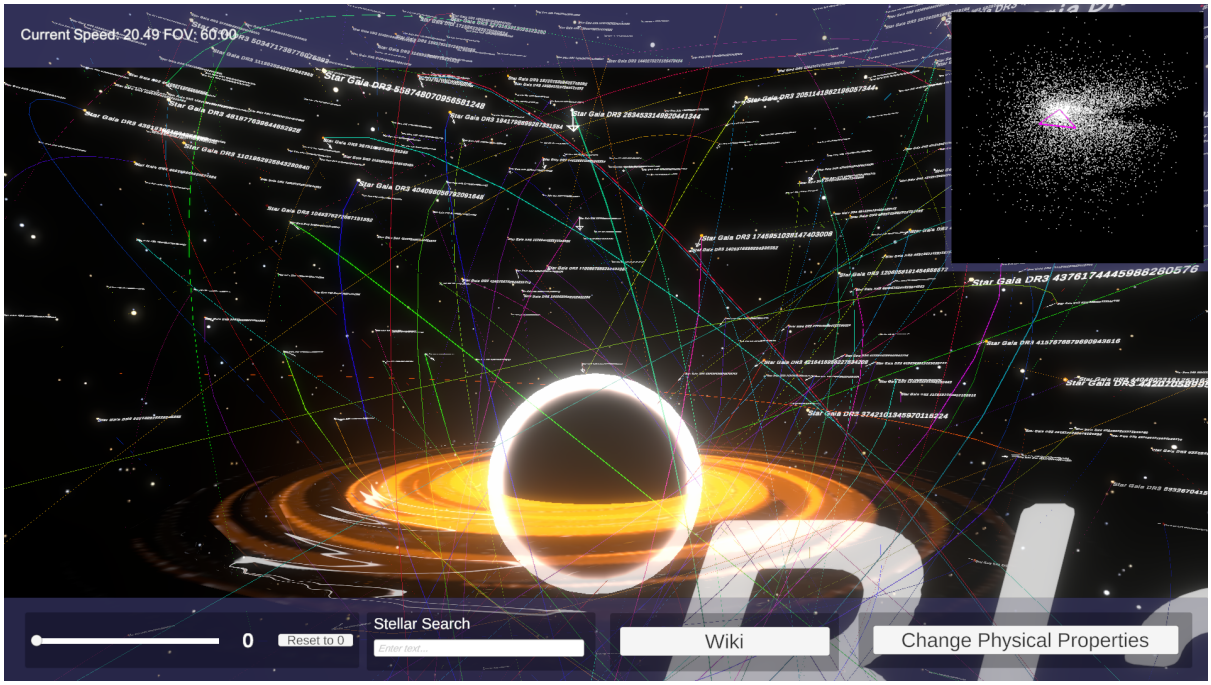

Figure 7.3: In-game image

1 Right top corner represents the scenario map that includes a pink pyramid that represents what the user can see at the moment,
2 Left top corner represents the player speed and FOV that changes using the *SHIFT* key and the *W, A, S, D* keys for movement,
3 Left bottom corner represents the *TIME* speed of the simulation. When the scenario starts or the player moves around, the time speed will be reverted to 0 automatically to not disorient the player,
4 The input field *Stellar Search* enables the user to search for a specific star and go to its location,
5 The button *Wiki* opens the wiki pages of the application,
6 The button *Change Physical Properties* opens the panel to change the physical properties of the simulation, such as the gravitational constant *G*.

By double-left clicking on any star, the camera will move close to its location, to be able to see it properly. By only left-clicking on any star, a panel with simple information will pop up as shown in Figure 7.4. By clicking on *More Information*, a new panel, Figure 7.5, will open up with a lot more information, which contains more information about the star, such as the Star Classification, the relative size, temperature and habitable zones of that particular star. To close this panel and return to the main view, the user should press *Close* button

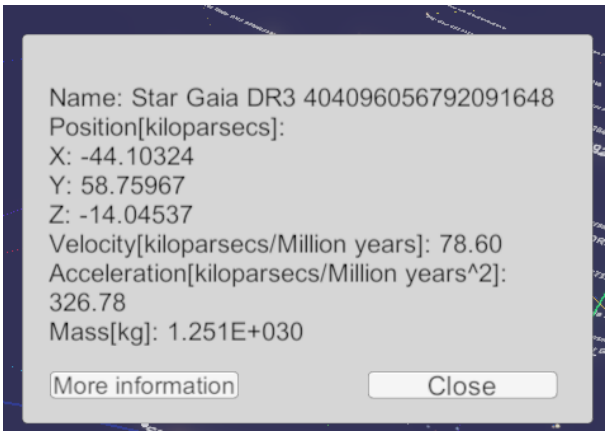

Figure 7.4: Basic information about a star

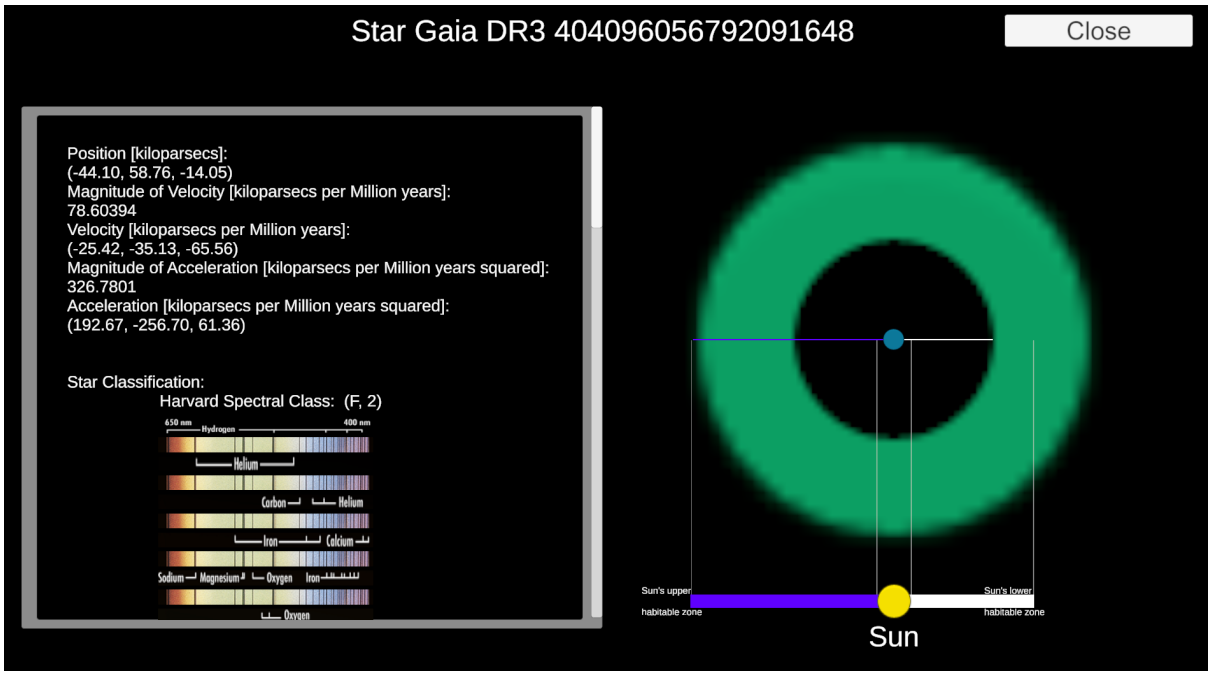

Figure 7.5: More information about a star

Chapter 8. Conclusions

8.1. Contributions

S.T.A.R.S. project made significant contributions to the field of astronomical simulations by developing a robust application that integrates real astronomical data from the GAIA API with the ease of use of visual methods deployed in Unity3d.

The application successfully implemented essential features such as basic planetary movement and detailed information visualization, enhancing the educational and research potential of the tool. The user interface was thoughtfully designed, incorporating elements like a search bar for game objects, an orientation compass, and in-game menus, which collectively improved user interaction and overall experience. These foundational functionalities provided a solid basis for simulating and exploring astronomical phenomena interactively.

Furthermore, the S.T.A.R.S. project further advanced the field of astronomical simulations through its innovative utilization of the Unity3D game engine, the C# programming language, and the OpenCL framework. The integration of the OpenCL framework provided a powerful tool for parallel computing and accelerated processing of complex astronomical calculations. OpenCL's ability to harness the computational power of both CPUs and GPUs enabled the application to simulate intricate astronomical phenomena with unprecedented speed and accuracy.

8.2. Critical Analysis of the results

Following the development of the application, the results are satisfactory. The main objective of the application was achieved, by creating an application that can be used in a classroom by a teacher with students to explain Newtonian-based physics.

Furthermore, an explicit user interface was designed and implemented that guides the user to the most important parts of the application. The wiki page is the perfect example of a user-defined part of the application, where teachers and hobbyists alike can define their own concepts and present them in a easy-to-follow manner.

Moreover, these computations were made on the GPU, thus freeing the CPU from the time-intensive computations needed for these types of simulations. A more in-depth re-factorization would be required to implement the better algorithms discussed in Chapter 4, in case of more objects simulated, the methods currently implemented in OpenCL are more than enough for this stage of the application.

Overall, the basic mechanics of this educational application provide a solid ground for improvements. By addressing the points below, such as expanding support for stellar systems and improving the UI design, the application can become an even more effective software in the arsenal of educators and people curious about space.

8.3. Further Development

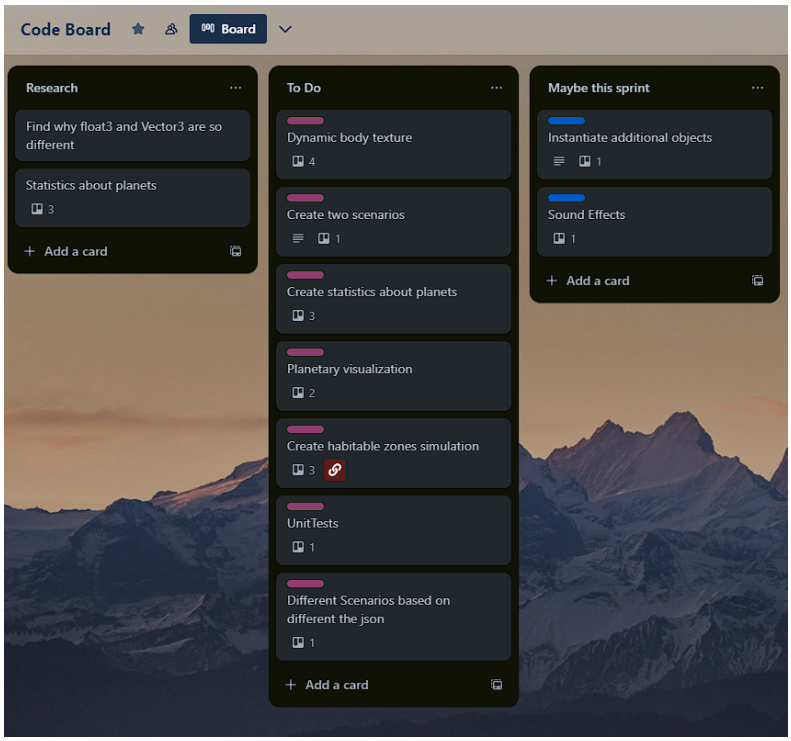

Figure 8.1: To do tasks on Trello

For the near future, the planned tasks are present in the To Do category of the Trello board, as seen in Figure 8.1. The main epics of the tasks are as follows:

- Sound effects for stars, planets and black holes. Even though in space there is no sound, because of the vacuum, the end user would highly benefit from a system like this for auditory cues. In doing so, a user can more easily navigate the space provided by the scenario. Another category of sound effects would be the UI buttons and fields that would give auditory feedback once interacted with.
- The addition of support for planets. From the current implementation, such objects are not feasible, but the introduction of stellar systems support, would be a significant advancement in enriching the educational experience. By incorporating planets into the simulation, students can gain a deeper understanding of celestial mechanics and the dynamics of planetary systems.
- The addition of multiple scenarios. As of now, there is only one scenario available in the system, but in the future, the plan is to add multiple scenarios including, but not limited to:
  - Our star system, to visualize the planets, moons and mechanics that play a crucial role in our existence,
  - A simulation that includes more stars or a different sets of stars for the end user to explore and learn,
  - User defined scenarios by creating custom JSON files that can be read and interpreted by the application. By doing so, the end users could define and refine the mechanics that the application provides and in doing so, transitioning from a normal application to a physics simulator.

Bibliography

[1] S. Kemp, "Digital around the world," https://datareportal.com/global-digital-overview, Kepios, n.d., accessed: May 2024.

[2] G. E. Moore, "Lithography and the future of moore's law," in *Integrated Circuit Metrology, Inspection, and Process Control IX*, vol. 2439. SPIE, 1995, pp. 2–17.

[3] R. Yuriy, S. Huzchenko, N. Lobach, O. Karbovanets, S. Bokova, and L. Isychko, "Modern digital learning and simulation technologies in higher medical education: definitions, innovative potential," *Amazonia Investiga*, vol. 11, no. 60, p. 53–61, Dec. 2022. [Online]. Available: https://amazoniainvestiga.info/index.php/amazonia/article/view/2210

[4] C. Babusiaux, C. Fabricius, S. Khanna, T. Muraveva, C. Reylé, F. Spoto, A. Vallenari, X. Luri, F. Arenou, M. Alvarez *et al.*, "Gaia data release 3-catalogue validation," *Astronomy & Astrophysics*, vol. 674, p. A32, 2023.

[5] R. J. Jennings, D. L. Kaplan, S. Chatterjee, J. M. Cordes, and A. T. Deller, "Binary pulsar distances and velocities from gaia data release 2," *The Astrophysical Journal*, vol. 864, no. 1, p. 26, 2018.

[6] A. Tanikawa, K. Hattori, N. Kawanaka, T. Kinugawa, M. Shikauchi, and D. Tsuna, "Search for a black hole binary in gaia dr3 astrometric binary stars with spectroscopic data," *The Astrophysical Journal*, vol. 946, no. 2, p. 79, 2023.

[7] S. S. Meyer, *Plato: Laws 1 and 2*. OUP Oxford, 2015.

[8] E. M. Avedon and B. Sutton-Smith, "The study of games," *(No Title)*, 1971.

[9] R. L. Wing, "Two computer-based economics games for sixth graders," *American Behavioral Scientist*, vol. 10, no. 3, pp. 31–35, 1966.

[10] K. Squire and H. Jenkins, "Harnessing the power of games in education," *Insight*, vol. 3, no. 1, pp. 5–33, 2003.

[11] P. Rao, N. Conte, and C. Wadsworth. (2023) 50 years of video game industry revenues, by platform. [Online]. Available: https://www.visualcapitalist.com/video-game-industry-revenues-by-platform/

[12] K. Bergeson, "USACE, Army Game Studio collaborate on VR training tool," https://www.army.mil/article/263223/usace_army_game_studio_collaborate_on_vr_training_tool, January 18 2023, accessed: February 2024.

[13] J. Tulipan, A. Miller, A. G. Park, J. T. Labrum IV, and A. M. Ilyas, "Touch surgery: analysis and assessment of validity of a hand surgery simulation "app"," *Hand*, vol. 14, no. 3, pp. 311–316, 2019.

[14] C. Darrell and D. de Rooy, "Assassin's creed: Odyssey discovery tours med genetiskt och genealogiskt historiemedvetande: Virtuella guidade turer i antika grekland som klassrumsbaserad undervisning i högstadiet," 2021.

[15] A. Foster, M. Koehler, and P. Mishra, "Game-based learning of physics content: the effectiveness of a physics game for learning basic physics concepts," in *World conference on educational multimedia, hypermedia and telecommunications Volume: 1*, 01 2006, pp. 2119–2125.

[16] K. B. Hille, "Gamers tackle virtual asteroid-sampling mission," June 2016, accessed: 2024. [Online]. Available: https://www.nasa.gov/solar-system/gamers-tackle-virtual-asteroid-sampling-mission/

[17] Ministry of Education, "PROGRAME ŞCOLARE PENTRU CICLUL SUPERIOR AL LICEULUI, FIZICĂ, CLASA A IX-A," 2013, accessed: 2024. [Online]. Available: https://www.isjcta.ro/wp-content/uploads/2013/06/Fizica_clasa-a-IX-a.pdf

[18] ——, "PROGRAME ŞCOLARE PENTRU CICLUL SUPERIOR AL LICEULUI, FIZICĂ, CLASA A XII-A," 2013, accessed: 2024. [Online]. Available: https://www.isjcta.ro/wp-content/uploads/2013/06/Fizica_programa-F1_F2_clasa-a-XII-a_a-XIII-a.pdf

[19] J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of cuda and opencl," in *2011 International Conference on Parallel Processing*, 2011, pp. 216–225.

[20] I. Newton, *Philosophiae Naturalis Principia Mathematica ("Mathematical Principles of Natural Philosophy").* London: Royal Society, 1687, also published in Cambridge, 1713; London, 1726.

[21] A.-C. Eilers, D. W. Hogg, H.-W. Rix, and M. K. Ness, "The circular velocity curve of the milky way from 5 to 25 kpc," *The Astrophysical Journal*, vol. 871, no. 1, p. 120, jan 2019. [Online]. Available: https://dx.doi.org/10.3847/1538-4357/aaf648

[22] V. Bobylev and A. Bajkova, "Review of current estimates of the galaxy mass," *Publications of the Pulkovo Observatory*, vol. 228, p. 1–20, May 2023. [Online]. Available: http://dx.doi.org/10.31725/0367-7966-2023-228-3

[23] Unity Technologies, "Component," Unity Documentation, 2015, accessed: 2024. [Online]. Available: https://docs.unity3d.com/ScriptReference/Component.html

[24] ——, "MonoBehaviour," Unity Documentation, 2015, accessed: 2024. [Online]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html

[25] ——, "MonoBehaviour.LateUpdate()," Unity Documentation, 2015, accessed: 2024. [Online]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.LateUpdate.html

[26] ——, "MonoBehaviour.FixedUpdate()," Unity Documentation, 2015, accessed: 2024. [Online]. Available: https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html

[27] Simon Nordon, "Unity Architecture: GameObject Component Pattern," Medium, 2024, accessed: 2024. [Online]. Available: https://medium.com/@simon.nordon/unity-architecture-gameobject-component-pattern-34a76a9eacfb

[28] Unity Technologies, "Unity documentation," 2023, accessed: 2024. [Online]. Available: https://unity.com/how-to/build-modular-codebase-mvc-and-mvp-programming-patterns

[29] ——, "Unity documentation," 2023, accessed: 2024. [Online]. Available: https://unity.com/how-to/create-modular-and-maintainable-code-observer-pattern

[30] R. Mardling, "Resonance, chaos and stability in the general three-body problem," *Proceedings of the International Astronomical Union*, vol. 3, no. S246, pp. 199–208, 2007.

[31] H. Brück, "Note on the harvard and potsdam systems of spectral classification," *Monthly Notices of the Royal Astronomical Society*, vol. 105, no. 4, pp. 206–211, 1945.

[32] H. N. Russell, "Relations between the spectra and other characteristics of stars," in *A Source Book in Astronomy and Astrophysics, 1900–1975.* Harvard University Press, 1979, pp. 212–220.

[33] A. W. Mann, G. A. Feiden, E. Gaidos, T. Boyajian, and K. Von Braun, "How to constrain your m dwarf: measuring effective temperature, bolometric luminosity, mass, and radius," *The Astrophysical Journal*, vol. 804, no. 1, p. 64, 2015.

[34] Gaia Collaboration, Vallenari, A., Brown, A. G. A., Prusti, T., and d. et al., "Gaia data release 3 - summary of the content and survey properties," *A&A*, vol. 674, p. A1, 2023. [Online]. Available: https://doi.org/10.1051/0004-6361/202243940

[35] O. Malkov, D. Kovaleva, A. Zhukov, and O. Dluzhnevskaya, "Theoretical mass-luminosity relations in gaia g-band," *Astrophysics and Space Science*, vol. 367, no. 4, April 2022. [Online]. Available: http://dx.doi.org/10.1007/s10509-022-04066-1

[36] M. Delbo, J. Gayon-Markt, G. Busso, A. Brown, L. Galluccio, C. Ordenovic, P. Bendjoya, and P. Tanga, "Asteroid spectroscopy with gaia," *planss*, vol. 73, pp. 86–94, 12 2012.

[37] Luri, X., Brown, A. G. A., Sarro, L. M., Arenou, F., Bailer-Jones, C. A. L., Castro-Ginard, A., de Bruijne, J., Prusti, T., Babusiaux, C., and Delgado, H. E., "Gaia data release 2 - using gaia parallaxes," *A&A*, vol. 616, p. A9, 2018. [Online]. Available: https://doi.org/10.1051/0004-6361/201832964

[38] "Convert temperature to rgb algorithm code," https://tannerhelland.com/2012/09/18/convert-temperature-rgb-algorithm-code.html, accessed: May 28, 2024.

Appendix A. Relevant Code sections

A.1. Movement Computations Base Runner

```
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using UnityEngine;

public abstract class MovementComputationsBaseRunner : OpenCLRunner<Vector4, CelestialBodyManager, OpenClBodyObject>
{
    public MovementComputationsBaseRunner(string filePath, string functionName) : base(filePath, functionName)
    {
    }

    protected override List<OpenClBodyObject> SimplifyUpdateObjects(List<OpenClBodyObject> myObjectBodies)
    {
        float deviationThreshold = 1f;

        ConcurrentBag<OpenClBodyObject> pointsToUpdate = new();

        // Parallelize the loop
        Parallel.For(0, myObjectBodies.Count, index =>
        {
            OpenClBodyObject objectToUpdate = myObjectBodies[index];

            if (objectToUpdate.pathPoints == null || objectToUpdate.pathPoints.Count < 2)
            {
                if (objectToUpdate.pathPoints == null)
                {
                    objectToUpdate.pathPoints = new List<Vector3>();
                }
                pointsToUpdate.Add(objectToUpdate);
                return;
            }

            float distanceToPath = PointToRayDistance(objectToUpdate.position, objectToUpdate.pathPoints[0], objectToUpdate.pathPoints[1]);
            float distanceToObjectFromStart = Vector3.Distance(objectToUpdate.position, objectToUpdate.pathPoints[0]);
            float pathLength = Vector3.Distance(objectToUpdate.pathPoints[1], objectToUpdate.pathPoints[0]);
```

```
37
38              if (distanceToPath > deviationThreshold ||
    distanceToObjectFromStart > pathLength)
39              {
40                  pointsToUpdate.Add(objectToUpdate);
41              }
42          });
43
44          return pointsToUpdate.ToList();
45      }
46
47      protected float PointToRayDistance(Vector3 point, Vector3 origin,
    Vector3 target)
48      {
49          var ray = new Ray(origin, target - origin);
50          var cross = Vector3.Cross(ray.direction, point - ray.origin);
51
52          return cross.magnitude;
53      }
54  }
```

Listing A.1: Full Code for Movement Computations Base Runner

A.2. Acceleration Runner

```
1  using System.Collections.Generic;
2  using System.Linq;
3  using System.Threading.Tasks;
4  using Silk.NET.OpenCL;
5  using UnityEngine;
6
7  public class AccelerationRunner : MovementComputationsBaseRunner
8  {
9      public AccelerationRunner(string filePath, string functionName) :
    base(filePath, functionName)
10     {
11     }
12
13     public override List<OpenClBodyObject> Update(List<OpenClBodyObject
    > args, params object[] additionalParameters)
14     {
15         List<OpenClBodyObject> pointsToUpdate = SimplifyUpdateObjects(
    args);
16
17         if (pointsToUpdate == null) return args;
18
19         int argsLength = pointsToUpdate.Count;
20
21         if (argsLength == 0) return args;
22
23         nint[] memObjects = new nint[3];
24         int[] valueObjects = new int[2] { argsLength, pointsToUpdate.
    FirstOrDefault().Flatten().Count() };
25         Vector4[] result = new Vector4[(nuint)argsLength];
26
27         nuint[] globalWorkSize = new nuint[1] { (nuint)argsLength };
28         nuint[] localWorkSize = new nuint[1] { 1 };
29
```

```
30          if (openCLInterfaceImplementation.CreateMemObjects(cl, context,
     memObjects, true, 0, MemFlags.ReadWrite, result)
31              && openCLInterfaceImplementation.CreateMemObjects(cl,
    context, memObjects, false, 1, MemFlags.ReadOnly | MemFlags.
    CopyHostPtr, pointsToUpdate.SelectMany(obj => obj.Flatten()).ToArray
    ())
32              && openCLInterfaceImplementation.CreateMemObjects(cl,
    context, memObjects, false, 2, MemFlags.ReadOnly | MemFlags.
    CopyHostPtr, args.SelectMany(obj => obj.Flatten()).ToArray())
33              && openCLInterfaceImplementation.SetKernelArgsMemory(cl,
    kernel, memObjects, new int[] { 0, 1, 2 })
34              && openCLInterfaceImplementation.SetKernelArgsVariables(cl,
     kernel, valueObjects, new int[] { 3, 4 })
35              && Run(globalWorkSize, localWorkSize, result.Length,
    memObjects, 0, out result))
36          {
37              Parallel.For(0, pointsToUpdate.Count, index =>
38              {
39                  args[args.IndexOf(pointsToUpdate[index])].acceleration
    = result[index];
40              });
41          }
42          return args;
43      }
44  }
```

Listing A.2: Full Code for Acceleration Runner

A.3. Path Runner

```
1  using System.Collections.Generic;
2  using System.Linq;
3  using System.Threading.Tasks;
4  using Silk.NET.OpenCL;
5  using UnityEngine;
6
7  public class PathRunner : MovementComputationsBaseRunner
8  {
9      private Plane[] frustumPlanes;
10     public PathRunner(string filePath, string functionName) : base(
    filePath, functionName)
11     {
12     }
13
14     public override List<OpenClBodyObject> Update(List<OpenClBodyObject
    > args, params object[] additionalParameters)
15     {
16         List<OpenClBodyObject> pointsToUpdate = SimplifyByView(
    SimplifyUpdateObjects(args), (Camera)additionalParameters[1]);
17
18         if (pointsToUpdate == null) return args;
19
20         int argsLength = pointsToUpdate.Count;
21
22         if (argsLength == 0) return args;
23
24         int steps = (int)additionalParameters[0];
25         nint[] memObjects = new nint[3];
```

```
26          int[] intObjects = new int[3] { argsLength, pointsToUpdate.
    FirstOrDefault().Flatten().Count(), steps };
27          Vector4[] result = new Vector4[(nuint)(argsLength * steps)];
28
29          nuint[] globalWorkSize = new nuint[1] { (nuint)argsLength };
30          nuint[] localWorkSize = new nuint[1] { 1 };
31
32          if (openCLInterfaceImplementation.CreateMemObjects(cl, context,
     memObjects, true, 0, MemFlags.ReadWrite, result)
33              && openCLInterfaceImplementation.CreateMemObjects(cl,
    context, memObjects, false, 1, MemFlags.ReadOnly | MemFlags.
    CopyHostPtr, pointsToUpdate.SelectMany(obj => obj.Flatten()).ToArray
    ())
34              && openCLInterfaceImplementation.CreateMemObjects(cl,
    context, memObjects, false, 2, MemFlags.ReadOnly | MemFlags.
    CopyHostPtr, args.SelectMany(obj => obj.Flatten()).ToArray())
35              && openCLInterfaceImplementation.SetKernelArgsMemory(cl,
    kernel, memObjects, new int[] { 0, 1, 2 })
36              && openCLInterfaceImplementation.SetKernelArgsVariables(cl,
     kernel, intObjects, new int[] { 3, 4, 5 })
37              && Run(globalWorkSize, localWorkSize, result.Length,
    memObjects, 0, out result))
38          {
39              Parallel.For(0, pointsToUpdate.Count(), index_openCL =>
40              {
41                  int index_myObjectBodies = args.IndexOf(pointsToUpdate[
    index_openCL]);
42                  if (index_myObjectBodies >= 0)
43                  {
44                      List<Vector3> updatedPathPoints = new();
45                      for (int step = 0; step < steps; step++)
46                      {
47                          updatedPathPoints.Add(result[index_openCL *
    steps + step]);
48                      }
49                      args[index_myObjectBodies].pathPoints.Clear();
50                      args[index_myObjectBodies].pathPoints.AddRange(
    updatedPathPoints);
51                  }
52              });
53          }
54
55          return args;
56      }
57      private List<OpenClBodyObject> SimplifyByView(List<OpenClBodyObject
    > args, Camera cam)
58      {
59          List<OpenClBodyObject> pointsToUpdate = new();
60
61          foreach (var obj in args)
62          {
63              if (IsInFrustumAndInView(cam, obj.position))
64              {
65                  pointsToUpdate.Add(obj);
66              }
67          }
68
69          return pointsToUpdate;
```

```
    }

    private bool IsInFrustumAndInView(Camera camera, Vector3 position)
    {
        frustumPlanes = GeometryUtility.CalculateFrustumPlanes(camera);
        Bounds bounds = new(position, Vector3.zero);
        return GeometryUtility.TestPlanesAABB(frustumPlanes, bounds) &&
    ViewHelper.IsInView(camera, position);
    }
}
```

Listing A.3: Full Code for Path Runner

A.4. Analysis Performance

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler

# Load the dataset
df = pd.read_csv('logs.csv', index_col=0)

# Compute descriptive statistics
descriptive_stats = df.describe()
descriptive_stats.rename(index={'count': 'Count',
                                'mean': 'Mean',
                                'std': 'Standard Deviation',
                                'min': 'Minimum',
                                '25%': '25th Percentile',
                                '50%': 'Median (50th Percentile)',
                                '75%': '75th Percentile',
                                'max': 'Maximum'}, inplace=True)

# Add custom rows for range, variance, skewness, kurtosis, and sum
descriptive_stats.loc['Range'] = descriptive_stats.loc['Maximum'] -
    descriptive_stats.loc['Minimum']
descriptive_stats.loc['Variance (sqrt)'] = df.var().pow(1./2)
descriptive_stats.loc['Skewness'] = df.skew()
descriptive_stats.loc['Kurtosis'] = df.kurt()
descriptive_stats.loc['Sum (sqrt)'] = df.sum().pow(1./2)

fig = plt.figure(figsize=(10, 6), facecolor='w', edgecolor='k')
sns.heatmap(descriptive_stats, annot=True, cbar=False, fmt=".1f")
plt.tight_layout()
plt.savefig('DataFrame.png')

# Print statistics
print("Descriptive Statistics:\n", descriptive_stats)
descriptive_stats.to_csv("./yes.csv")
# Additional stats
mode = df.mode().iloc[0]
iqr = df.quantile(0.75) - df.quantile(0.25)

print("\nMode:")
print("The mode represents the most common value(s) in the dataset.")
```

```
43 print(f"For example, if the mode for 'FMM processing time' is {mode.
       values[0]}, "
44       f"it means that {mode.values[0]} is the most frequently occurring
       value in that column.")
45 print(f"Similarly, if the mode for 'List processing time' is {mode.
       values[1]}, "
46       f"it means that {mode.values[1]} is the most frequently occurring
       value in that column.")
47
48 print("\nInterquartile Range (IQR):")
49 print("The interquartile range (IQR) is a measure of statistical
       dispersion, "
50       "which is computed as the difference between the 75th and 25th
       percentiles.")
51 print("It represents the range of the middle 50% of the data.")
52 print(f"For example, if the IQR for 'FMM processing time' is {iqr.
       values[0]}, "
53       f"it means that the middle 50% of the data lies within a range of
       {iqr.values[0]} units.")
54 print(f"Similarly, if the IQR for 'List processing time' is {iqr.values
       [1]}, "
55       f"it means that the middle 50% of the data lies within a range of
       {iqr.values[1]} units.")
56
57 confidence_level = 0.95
58 degrees_freedom = df[' FMM processing time'].count() - 1
59 sample_mean = df[' FMM processing time'].mean()
60 sample_standard_error = stats.sem(df[' FMM processing time'])
61 confidence_interval_fmm = stats.t.interval(confidence_level,
       degrees_freedom, sample_mean, sample_standard_error)
62
63 degrees_freedom = df[' List processing time'].count() - 1
64 sample_mean = df[' List processing time'].mean()
65 sample_standard_error = stats.sem(df[' FMM processing time'])
66 confidence_interval_list = stats.t.interval(confidence_level,
       degrees_freedom, sample_mean, sample_standard_error)
67
68 print("\nConfidence Interval:")
69 print("A confidence interval is a range of values that likely contains
       the "
70       "true population parameter with a certain level of confidence.")
71 print(f"For example, a 95% confidence interval for the mean "
72       f"'FMM processing time' of ({confidence_interval_fmm[0]:.2f}, {
       confidence_interval_fmm[1]:.2f}) "
73       f"means that we are 95% confident that the true population mean
       lies within this interval.")
74 print(f"Similarly, a 95% confidence interval for the mean "
75       f"'List processing time' of ({confidence_interval_list[0]:.2f}, {
       confidence_interval_list[1]:.2f}) "
76       f"means that we are 95% confident that the true population mean
       lies within this interval.")
77
78 df.plot()
79 plt.legend(loc='upper right')
80 plt.title('Data plot')
81 plt.xlabel('Processing time')
82 plt.ylabel('Density')
83 plt.savefig("./Data.png")
```

```
84
85 plt.figure()
86
87 # Calculate the histogram data for both sets to overlay the line chart
   later
88 hist_data_fmm = np.histogram(df[' FMM processing time'], bins=15,
   density=True)
89 hist_data_list = np.histogram(df[' List processing time'], bins=15,
   density=True)
90
91 # Calculate the bin centers instead of edges
92 bin_edges_fmm = hist_data_fmm[1]
93 bin_centers_fmm = (bin_edges_fmm[:-1] + bin_edges_fmm[1:]) / 2
94
95 bin_edges_list = hist_data_list[1]
96 bin_centers_list = (bin_edges_list[:-1] + bin_edges_list[1:]) / 2
97
98 # Create histograms with density=True to normalize and overlay the line
    chart
99 plt.hist(df[' FMM processing time'], bins=15, alpha=0.5, label='FMM
   processing time', density=True)
100 plt.hist(df[' List processing time'], bins=15, alpha=0.5, label='List
   processing time', density=True)
101
102 # Add line charts
103 plt.plot(bin_centers_fmm, hist_data_fmm[0], label='FMM Line', color='
   blue')
104 plt.plot(bin_centers_list, hist_data_list[0], label='List Line', color=
   'red')
105
106 # Add mean and sum annotations for FMM processing time
107 x_mean = df[' FMM processing time'].mean()
108 y_mean = stats.gaussian_kde(df[' FMM processing time']).pdf(df[' FMM
   processing time']).mean()
109 plt.annotate(
110     f"FMM processing time\nMean: {df[' FMM processing time'].mean():.2f
   }\nSum: {df[' FMM processing time'].sum():.2f}",
111     xy=(x_mean, y_mean),
112     xytext=(x_mean + 50, y_mean),
113     arrowprops=dict(facecolor='black', arrowstyle='->'),
114     fontsize=10)
115
116 # Add mean and sum annotations for List processing time
117 x_mean = df[' List processing time'].mean()
118 y_mean = stats.gaussian_kde(df[' List processing time']).pdf(df[' List
   processing time']).mean()
119 plt.annotate(
120     f"List processing time\n Mean: {df[' List processing time'].mean()
   :.2f}\nSum: {df[' List processing time'].sum():.2f}",
121     xy=(x_mean, y_mean),
122     xytext=(x_mean + 50, y_mean),
123     arrowprops=dict(facecolor='black', arrowstyle='->'),
124     fontsize=10)
125
126 plt.legend(loc='upper right')
127 plt.title('Density of the Processing Times')
128 plt.xlabel('Processing time')
129 plt.ylabel('Density')
```

```
plt.savefig("./Density.png")

plt.figure()

# Calculate the CDF for 'FMM processing time'
cdf_fmm = df[' FMM processing time'].value_counts().sort_index().cumsum()
cdf_fmm = cdf_fmm / cdf_fmm.max()
cdf_list = df[' List processing time'].value_counts().sort_index().cumsum()
cdf_list = cdf_list / cdf_list.max()

# Plot both CDFs
plt.figure(figsize=(10, 5))
plt.plot(cdf_fmm.index, cdf_fmm, label='FMM Processing Time CDF', drawstyle='steps-post')
plt.plot(cdf_list.index, cdf_list, label='List Processing Time CDF', drawstyle='steps-post')

plt.title('Cumulative Distribution Functions (CDFs)')
plt.xlabel('Processing Time')
plt.ylabel('Probability')
plt.legend()
plt.grid(True)
plt.savefig("./CDF.png")
```

Listing A.4: Full Code for Data Analysis Performance

A.5. MapRenderer class

```
using UnityEngine;

public class MapRenderer : MonoBehaviour
{
    public RenderTexture mapTexture;
    public RenderTexture frustrumTexture;
    public ObjectManager objectManager;
    public Material drawMaterial; // A material with a simple shader to draw points
    private Camera cam;

    int interval = 1;
    Vector3 minBounds;
    Vector3 maxBounds;

    void Start()
    {
        cam = Camera.main;

        minBounds = new Vector3();
        maxBounds = new Vector3();
    }

    void LateUpdate()
    {
        Vector3 minBoundsTemp = new Vector3(objectManager.celestialBodyManager.bounds[0], objectManager.celestialBodyManager.bounds[2], objectManager.celestialBodyManager.bounds[4]);
```

```
26          Vector3 maxBoundsTemp = new Vector3(objectManager.
    celestialBodyManager.bounds[1], objectManager.celestialBodyManager.
    bounds[3], objectManager.celestialBodyManager.bounds[5]);
27
28          if (minBoundsTemp != minBounds || maxBoundsTemp != maxBounds)
29          {
30              minBounds = minBoundsTemp;
31              maxBounds = maxBoundsTemp;
32              RenderMap();
33          }
34
35          if (Time.time % interval == 0)
36          {
37              RenderMap();
38          }
39          RenderCameraFrustrum();
40      }
41      public void RenderCameraFrustrum()
42      {
43          RenderTexture.active = frustrumTexture;
44
45          GL.Clear(true, true, Color.clear);
46
47          Graphics.SetRenderTarget(frustrumTexture);
48          Vector3[] frustumCorners = new Vector3[4];
49          cam.CalculateFrustumCorners(new Rect(0, 0, 1, 1), cam.
    farClipPlane, Camera.MonoOrStereoscopicEye.Mono, frustumCorners);
50          for (int i = 0; i < frustumCorners.Length; i++)
51          {
52              frustumCorners[i] = cam.transform.TransformVector(
    frustumCorners[i]) + cam.transform.position;
53          }
54
55          Vector2[] normalizedFrustumCorners = new Vector2[frustumCorners
    .Length];
56          for (int i = 0; i < frustumCorners.Length; i++)
57          {
58              normalizedFrustumCorners[i] = NormalizeToMinimap(
    frustumCorners[i], minBounds, maxBounds);
59          }
60
61          GL.PushMatrix();
62          GL.LoadPixelMatrix(0, frustrumTexture.width, frustrumTexture.
    height, 0);
63
64          GL.Begin(GL.LINES);
65          drawMaterial.SetColor("_Color", Color.magenta);
66          drawMaterial.SetPass(0);
67
68          // Connect the frustum corners
69          for (int i = 0; i < 4; i++)
70          {
71              Vector2 start = normalizedFrustumCorners[i];
72              Vector2 end = normalizedFrustumCorners[(i + 1) % 4];
73              GL.Vertex3(start.x * frustrumTexture.width, start.y *
    frustrumTexture.height, 0);
74              GL.Vertex3(end.x * frustrumTexture.width, end.y *
    frustrumTexture.height, 0);
```

```
75          }
76
77          Vector2 camPosOnMap = NormalizeToMinimap(cam.transform.position
    , minBounds, maxBounds);
78          for (int i = 0; i < 4; i++)
79          {
80              Vector2 start = normalizedFrustumCorners[i];
81              GL.Vertex3(start.x * frustrumTexture.width, start.y *
    frustrumTexture.height, 0);
82              GL.Vertex3(camPosOnMap.x * frustrumTexture.width,
    camPosOnMap.y * frustrumTexture.height, 0);
83          }
84
85          GL.End();
86          GL.PopMatrix();
87          RenderTexture.active = null;
88      }
89
90      public void RenderMap()
91      {
92          RenderTexture.active = mapTexture;
93
94          GL.Clear(true, true, Color.black);
95          drawMaterial.SetColor("_Color", Color.white);
96
97          Graphics.SetRenderTarget(mapTexture);
98          GL.PushMatrix();
99          GL.LoadPixelMatrix(0, mapTexture.width, mapTexture.height, 0);
100         foreach (OpenClBodyObject body in objectManager.
    celestialBodyManager.myObjectBodies)
101         {
102             Vector2 normalizedPosition = NormalizeToMinimap(body.
    position, minBounds, maxBounds);
103
104             Rect rect = new Rect(
105                 normalizedPosition.x * mapTexture.width,
106                 normalizedPosition.y * mapTexture.height,
107                 1,
108                 1
109             );
110
111             Graphics.DrawTexture(rect, Texture2D.whiteTexture,
    drawMaterial);
112         }
113         GL.PopMatrix();
114         RenderTexture.active = null;
115     }
116     private Vector2 NormalizeToMinimap(Vector3 worldPos, Vector3
    minBounds, Vector3 maxBounds)
117     {
118         return new Vector2(
119             (worldPos.x - minBounds.x) / (maxBounds.x - minBounds.x),
120             (worldPos.z - minBounds.z) / (maxBounds.z - minBounds.z)
121         );
122     }
123 }
```

Listing A.5: Full Code for MapRenderer class

A.6. StellarSearchClickable class

```
using TMPro;
using UnityEngine;
using UnityEngine.EventSystems;

public class StellarSearchClickable : MonoBehaviour,
    IPointerClickHandler
{
    [SerializeField] private ObjectManager objectManager;
    public void OnPointerClick(PointerEventData eventData)
    {
        if (eventData.button != PointerEventData.InputButton.Left)
    return;

        var text = GetComponent<TextMeshProUGUI>();
        int linkIndex = TMP_TextUtilities.FindIntersectingLink(text,
    Input.mousePosition, null);
        if (linkIndex == -1) return;

        var linkId = text.textInfo.linkInfo[linkIndex].GetLinkID();
        var itemData = objectManager.celestialBodyManager.
    myObjectBodies.Find(obj => obj.name == linkId);
        if (itemData == null) return;

        Vector3 directionToCamera = (Camera.main.transform.position -
    itemData.position).normalized;
        Camera.main.transform.position = itemData.position +
    directionToCamera * ViewHelper.CalculateOffset(itemData.mass);
        Camera.main.transform.LookAt(itemData.position);
    }
}
```

Listing A.6: Full Code for StellarSearchClickable class

A.7. WikiPage class

```
using System.Collections.Generic;
using System.Linq;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class WikiPage : MonoBehaviour
{
    private List<WikiPageObjList> wikiPageObjLists;

    [SerializeField]
    private Image panelWikiPage;

    [SerializeField]
    private GameObject buttonTemplate;

    [SerializeField]
    private GameObject categoryContent;

    [SerializeField]
```

```
21      private GameObject pagesContent;
22
23      [SerializeField]
24      private TMP_Text informationTitle;
25
26      [SerializeField]
27      private TMP_Text informationBody;
28
29      DataFetching dataFetching;
30
31      // Start is called before the first frame update
32      void Start()
33      {
34          dataFetching = DataFetching.Instance;
35      }
36
37      // Update is called once per frame
38      void Update()
39      {
40
41      }
42
43      public void PanelSetVisibility()
44      {
45          ClearPage(pagesContent.transform);
46          ClearPage(categoryContent.transform);
47          UpdateWikiData();
48          panelWikiPage.gameObject.SetActive(!panelWikiPage.gameObject.
    activeSelf);
49      }
50
51      private void SelectCategory(string category)
52      {
53          WikiPageObjList currentCategory = wikiPageObjLists.Find(x => x.
    Category == category);
54
55          if (currentCategory != null)
56          {
57              ClearPage(pagesContent.transform);
58          }
59          else
60          {
61              return;
62          }
63
64          for (int j = 0; j < currentCategory.Pages.Count; j++)
65          {
66              string title = currentCategory.Pages[j].Title;
67              string description = currentCategory.Pages[j].Description;
68
69              GameObject btn2 = Instantiate(buttonTemplate);
70              btn2.GetComponentInChildren<TMP_Text>().text = title;
71              btn2.transform.SetParent(pagesContent.transform);
72              btn2.GetComponent<Button>().onClick.AddListener(() =>
    SelectPage(title, description));
73          }
74      }
75
```

```
76      private void SelectPage(string title, string description)
77      {
78          informationTitle.text = title;
79          informationBody.text = description;
80      }
81
82      private void ClearPage(Transform t)
83      {
84          var children = t.Cast<Transform>().ToArray();
85
86          foreach (var child in children)
87          {
88              Object.DestroyImmediate(child.gameObject);
89          }
90
91          informationTitle.text = "";
92          informationBody.text = "";
93      }
94
95      private void UpdateWikiData()
96      {
97          wikiPageObjLists = dataFetching.WikiFetching("wiki_data.json");
98
99          for (int i = 0; i < wikiPageObjLists.Count; i++)
100         {
101             string category = wikiPageObjLists[i].Category;
102
103             GameObject btn = Instantiate(buttonTemplate);
104             btn.GetComponentInChildren<TMP_Text>().text = category;
105             btn.transform.SetParent(categoryContent.transform);
106             btn.GetComponent<Button>().onClick.AddListener(() =>
    SelectCategory(category));
107         }
108     }
109 }
```

Listing A.7: Full Code for WikiPage class