DESIGN AND IMPLEMENTATION OF A UNITY-BASED 3D TELESCOPE SIMULATION SYSTEM

Yitian Zhao[1], Quincy Stokes[2]

[1]Portola High School, 1001 Cadence, Irvine, CA 92618
[2]University of California, Irvine, Irvine, CA 92697

ABSTRACT

This project addresses the lack of accessible, interactive astronomy tools by developing a 3D telescope simulator using Unity [1]. The program models over 9,000 stars with apparent magnitudes above 6, accurately representing their positions, magnitudes, and planetary motions. It integrates three main components: a position plotting system, a rotation management system anchored to Polaris, and a data retrieval system that provides detailed information for each celestial object. Key challenges included scaling stars and planets for visibility, managing rotation dynamics, and organizing data pop-ups for clarity. Experiments tested planet size scaling and star overlap, optimizing visibility while maintaining realism. Methodology comparisons revealed that prior works—Python-based sky maps, AR-supported planetarium apps, and Star Walk 2—offered precision or real-time alignment but faced limitations in interactivity, AR support, device dependence, or star database size [2]. Our project improves usability, database completeness, and offline accessibility, providing an engaging, educational platform while future improvements focus on AR integration and sensor-based orientation.

KEYWORDS

Telescope Simulator, Universe, Celestial Sphere, Database, Unity

1. INTRODUCTION

Studies highlight a widespread lack of access to astronomical observation tools: in a WJARR study conducted in Uganda, 89.4% of sampled students reported that they had never observed the night sky using a telescope, and in a survey of public attitudes toward astronomy in Japan, approximately 70.6% of respondents indicated that they did not own common astronomy-related equipment such as binoculars or telescopes. To address this accessibility gap, this project restores the full visual experience of the night sky through a 3D simulation that approximates the performance of a physical telescope [3]. The platform incorporates specific, database-sourced stellar information to ensure credibility and accuracy, aiming to serve as a reliable and helpful tool for astronomy enthusiasts to explore the night sky. By offering a free, interactive environment for learning about optics and astronomy, it broadens access to observational experiences that would otherwise require specialized equipment.

In the first article, Viyaleta Apgar builds an accurate sky map using Python formulas to calculate star positions, magnitudes, and distances, which achieves high precision. However, it doesn't translate the information into a 3D model, making user interaction limited [4]. Our project sacrifices some precision for usability, allowing interactive exploration of constellations. The second article uses GPS and device sensors to align stars with the user's environment. This method provides real-time star positioning but suffers from sensor and GPS inaccuracies and a limited star database [5]. Our project improves on this by integrating a larger star catalog and

David C. Wyld et al. (Eds): CSITEC, NLPI, CMCA, NECOM, DMML, SP, ADCO – 2026
pp. 21-30, 2026. CS & IT - CSCP 2026 DOI: 10.5121/csit.2026.160803

more accurate astronomical data, though it currently lacks AR visualization. The third article combines real-time GPS, sensors, and a star catalog to display celestial objects on user devices. While this enables interactive mapping, it depends on device accuracy and connectivity. Our project addresses these issues by supporting a wider range of devices, implementing more precise algorithms, and allowing offline access. However, AR visualization is not yet supported.

We developed a Unity-based project that models all-stars with an apparent magnitude brighter than 6, incorporating approximately 9,000 stars in detailed 3D form, each embedded with specific and accurate astronomical data [6]. Although it does not fully replace the experience of a physical telescope, the simulation enables users to observe far more than what is visible to the naked eye, helping them understand the positions, magnitudes, and apparent movements of celestial bodies. The system is designed to restore realistic celestial rotation, with Polaris positioned at the center of rotation to mirror real-world sky dynamics. Planets are modeled independently, each with scaled orbital and rotational speeds that approximate actual orbital mechanics. Users can interactively explore detailed information about individual stars and planets, enhancing both engagement and learning. Compared to the high cost and geographic limitations associated with owning a real telescope, this cost-effective and mobile-accessible solution removes financial barriers, making astronomy exploration widely available [7]. As a result, it serves as a practical tool for education and outreach while inspiring future generations of astronomers.

In Section 4, two experiments were conducted to evaluate the functionality and user experience of the telescope simulator. The first experiment tested the scaling of planets and stars to ensure visual accuracy and clarity. Initially, overlapping occurred when stars were uniform in size, but by randomizing star sizes between 2.5 and 5 units, overlaps were eliminated, balancing realism and visibility. This highlighted the importance of proper scaling to maintain both educational accuracy and a coherent visual simulation. The second experiment gathered user feedback through a survey posted on itch.io, assessing overall satisfaction, clarity and usability, likelihood of recommendation, and educational value. Results were exceptionally high, with median scores of 10 for satisfaction, usability, and engagement, and a mean of 9 for educational usefulness. These outcomes indicate that the app's accessibility, interactivity, and ability to simulate the night sky for users who have never observed it significantly influenced positive responses, demonstrating both the effectiveness and educational impact of the project.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Improving Celestial Visualization Readability

When stars were plotted from the database, it was found that the stars with the correct scale are hard to see since they are too small compared to the entire celestial sphere, and the celestial objects in the solar system, especially the sun, take up too much area on the screen. To increase the readability, we could artificially increase the brightness or the size of the star while decreasing the size of the sun. While this also makes some stars in the star cluster overlap each other. We could also zoom in on the initial field of view.

2.2. Anchoring Celestial Rotation to Polaris for Realistic Motion

When we are trying to add rotation, we set the Earth as the center at first, but this created issues, such as all the stars and planets having their rotation synchronous. We could reset the center of the rotation to Polaris. By anchoring the rotation to Polaris, we can create a realistic celestial

sphere effect, allowing the stars to rotate naturally across the sky. And we can also scale the movement speeds of each planet either up or down according to their orbital periods, ensuring that their motions reflect real astronomical behavior rather than simply following the stars' rotation. We could also separate rotation scripts for planets and stars, giving each category of celestial body distinct dynamics.

2.3. Improving Star Data UI Organization and Readability

After adding information about the stars and displaying their data in a small pop-up box, all the words get crowded into a huge mess, making the data hard to read. To solve this, we could organize the information neatly, such as putting it in columns, with the labels and the data at different ends, or we could add a filter system, allowing the user to sort the information they want.

3. SOLUTION

The telescope simulator is built with a modular architecture containing three components: Position plotting system, rotation management system, and data retrieval system, which work together to form a complete, interactive 3D astronomy simulation that accurately represents celestial motion and information.

1. Position plotting system: Forms the foundation of the simulation. It is responsible for rendering all celestial objects, including stars, planets, and the Sun, at their correct spatial locations based on astronomical coordinates. These parameters are converted into 3D coordinates, which determine the exact position of every object within the Unity environment [8]. To maintain accuracy and readability, the plotting system applies scaling algorithms to adjust objects' visual size and brightness. Since real stars vary greatly in distance and luminosity, the true scale would make most of them nearly invisible. Therefore, controlled adjustments are made to ensure that every celestial body remains visible while preserving relative proportions.

2. Rotation management system: The Rotation Management System controls the dynamic behavior of the celestial environment. It simulates the natural rotation of the night sky by anchoring the celestial sphere's axis to Polaris, the North Star. This design choice mirrors real-world celestial motion, where Polaris remains nearly stationary while other stars appear to revolve around it. Planets within the simulation are programmed with independent rotation and revolution parameters that correspond to their actual orbital characteristics. Each planet's movement is scaled proportionally to its real-world orbital period, ensuring scientifically accurate motion. The Rotation Management System is essential for creating the illusion of a living sky, one that evolves naturally over time, just as it does in reality [9].

3. Data retrieval system: The information system manages the interactive and informational aspects of the simulator. When a user selects or hovers over a celestial object, this system retrieves the corresponding scientific data and displays it in a structured, easy-to-read interface. Each object is linked to the HYG database record containing its physical and observational properties, including its name, distance, magnitude, spectral classification, and orbital parameters.

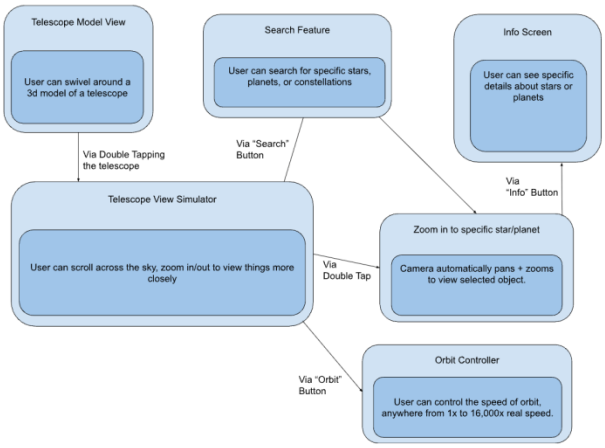


Figure 1. Overview of the solution

Data retrieval system: Provides users with scientific information about celestial objects, implemented using Unity UI and C# data-binding scripts through accessing astronomical databases to display details. This component relies on structured data mapping, ensuring accurate, interactive, and organized information presentation within the simulation.

Figure 2. Interactive 3D Telescope Simulation Interface Showing Stellar Rendering

```
void Start()
{
    UnityEngine.Random.InitState(100);

    TextAsset json = Resources.Load<TextAsset>(
        "stars_mag6p5_85000_unity");
    if (json == null)
    {
        Debug.Log("File could not be loaded.");
        return;
    }
    StarCatalog catalog = JsonUtility.FromJson<StarCatalog>(json.text);

    // Spawn stars + build parents
    foreach (StarJson s in catalog.stars)
    {
        Transform parent = GetOrCreateConstellationParent(s.con);
        GameObject currStar = Instantiate(starPrefabs[UnityEngine.Random.Range(0, starPrefabs.Count)], s.Position, Quaternion.identity, parent);
        currStar.name = s.name;
        Star star = currStar.GetComponent<Star>();
        star.InitializeData(s);
    }


    BuildConstellationCentroids();

    cb.BuildConstellations();
}
```

Figure 3. StarCreator Initialization Workflow for Stellar Object Generation

This is the start method of our StarCreator script. In this method, we load the pre-computed csv file of star information and then loop through each object to turn them into real game objects. It also sorts them into their correct constellations.

First, we attempt to load the file from the Resources folder, if it doesn't exist, we return early and log into an error. Then, we cast our json text object into a custom StarCatalog class, which is essentially just a list. After that, we loop through the list, and for each star:
First, we get its proper constellation parent (GetorCreateConstellationParent). This function returns a parent transform to sort the star. After that, we instantiate a new random star prefab, set its name to the star's name, and initialize its star component. This loads its particles, size, rotation, and other functions. When all the stars have been initialized, we call BuildConstellationCentroids, which loops through each constellation, creating a new empty game object at the physical centerpoint. This is later used for searching for specific constellations.

Position plotting system: The Position Plotting System is responsible for rendering celestial objects at accurate coordinates within the 3D environment [10]. Implemented using Unity's transformation and vector systems in C#, it converts astronomical data into precise spatial points. This component relies on coordinate transformation algorithms to ensure scientifically accurate star and planet placement.

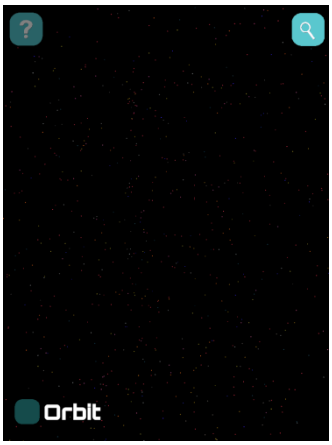


Figure 4. Planetary and Stellar Visualization within the Simulation Environment

```
# 1  Load and magnitude-filter ------------------------------------------
df = pd.read_csv(CSV)
df = df[df['mag'] <= MAG_LIM].copy()          # ~ 5 k rows

# 2  Robust star name ---------------------------------------------------
df['name'] = df['proper'].fillna('').str.strip()
mask = df['name'] == ''
df.loc[mask & df['hip'].notna(), 'name'] = (
    'HIP' + df.loc[mask & df['hip'].notna(), 'hip'].astype(int).astype(str)
)
mask = df['name'] == ''
df.loc[mask, 'name'] = (
    'ID' + df.loc[mask, 'id'].astype(int).astype(str)
)
df = df.drop_duplicates(subset='name').reset_index(drop=True)

# 3  Build direction vectors --------------------------------------------
xyz  = df[['x', 'y', 'z']].to_numpy()
dirs = xyz / np.linalg.norm(xyz, axis=1)[:, None]

# 4  Rotate so Polaris → +Z ---------------------------------------------
polaris_idx = df['name'].str.lower() == 'polaris'
if not polaris_idx.any():
    raise RuntimeError("Polaris not found!")

v = dirs[polaris_idx][0]                  # current Polaris dir
k = np.array([0, 0, 1])                   # target axis (+Z)

cross = np.cross(v, k)
sin0  = np.linalg.norm(cross)
cos0  = np.dot(v, k)
```

Figure 5. Python-Based Stellar Data Preprocessing and Coordinate Transformation Code

This is part of our python file that loads in our downloaded hsv file of star information from the HYG database. It filters stars that are too dim, transforms their world positions into usable vector3 values, and sets them on the proper rotation.

First, we load in the CSV file into “df” and immediately filter out rows with a magnitude of 6.5 or lower. This is their apparent brightness and can be thought of as the stars we can visibly see with our own eyes from earth. After that, we set the name property to remove any whitespace and spaces, unifying the names of the stars. The most important part in this script is step 3, Building the Direction Vectors [11]. This converts their xyz positions into a numpy array, which we can theme use to compute the norm of a vector. We take this number, and later multiply it by our desired radius, which places all-stars at the same physical distance from our centerpoint, while maintaining their proper direction (vector).
Rotation management system: Controls the motion of celestial objects, simulating the Earth’s rotation and planetary orbits. Implemented in Unity with C# transform rotations; it anchors movement around Polaris to mirror real sky behavior. This component uses rotational matrices and time-based updates to ensure smooth, accurate celestial motion.

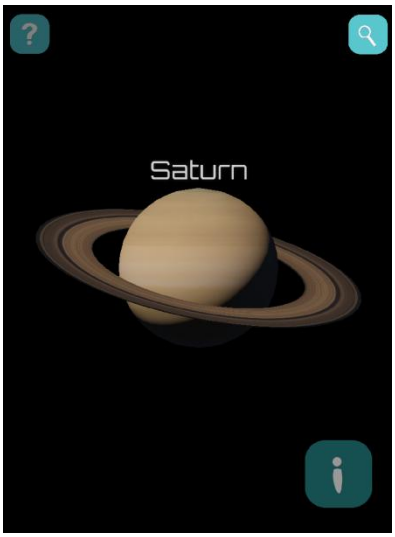


Figure 6. Dynamic Celestial Rotation Anchored to Polaris in the 3D Environment

```
void Update()
{
    if (canOrbit)
    {
        // Apply scaled orbit
        if (PlanetParent != null)
            PlanetParent.Rotate(
                Vector3.up,
                _orbitDegPerSec * _timeScale * Time.deltaTime,
                Space.Self
            );

        // Apply scaled spin
        transform.Rotate(
            Vector3.up,
            _spinDegPerSec * _timeScale * Time.deltaTime,
            Space.Self
        );
    }
}
```

Figure 7. Planetary Orbit and Rotation Control Implementation in Unity

The above code is the Update function of our planet class. We use a series of nested gameObjects to ensure the sun rotates properly, with planets that orbit around it, that also rotate independently.

In the code above, first, we only actually apply any rotation if the canOrbit flag is set to true. This lets us toggle the orbiting on and off, which serves as an additional feature to the user. After passing that check, we ensure that our PlanetParent isn't null. The PlanetParent is the parent object of this planet, which is what actually controls the orbit around the sun. We apply a rotation to that object, at some amount of orbit degrees per second, giving us a realistic planet rotation, which can be uniquely changed for each planet. After that, we apply the actual planet's rotation. We do this in the same way, by simply applying a rotation to the planet itself, based on some spin degrees per second.

4. EXPERIMENT

4.1. Experiment 1

A possible blind spot is the size scaling of the planets. Since the planets are zooming in on purpose for visibility, there is a risk that their proportions may appear unrealistic compared to other celestial objects. Ensuring this balance is important to maintain both educational accuracy and a visually coherent simulation.

To test the planet size scaling, Planetary size scaling was evaluated through controlled visual comparison within the simulation environment I will check if they look proportionate to each other while still being visible on screen. Because the planets are intentionally zoomed in for visibility, I will iteratively adjust scaling factors to balance realism with clarity. The experiment is set up this way to ensure that users can both observe the planets clearly and understand their relative sizes accurately, finding a balance that optimizes visual accessibility for educational purposes.

During the experiment, I observed that Asterope appeared to overlap with HIP 17588 in the simulation. This layering occurs because both stars are plotted closely together in the 3D celestial sphere, and their apparent size on-screen can cause visual overlaps.

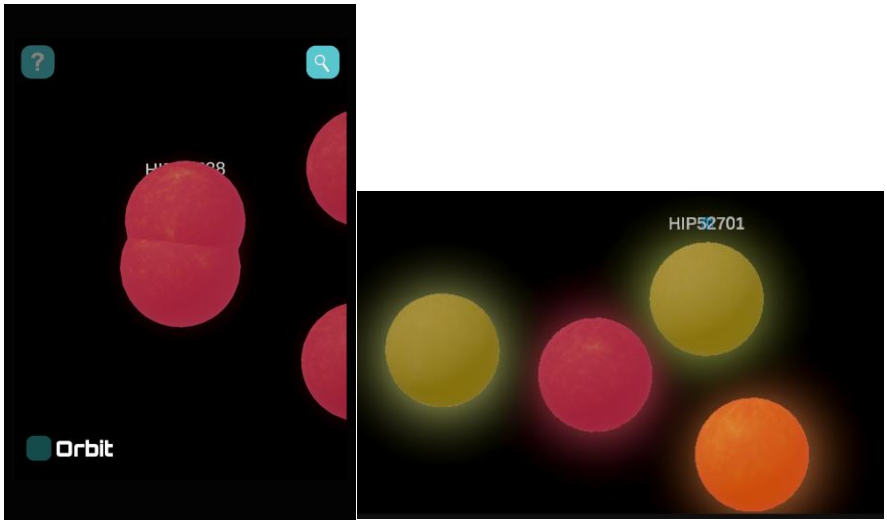


Figure 8. Impact of Star Size Randomization on Overlap Frequency in Stellar Clusters

Under the control, when all of the stars are the same size, we find that 5 out of 10 of the stars in a cluster appear overlapping. We then apply the randomization between size (2.5 to 10), and we find that only 2 out of 10 stars in a cluster appear overlapping. We tried to reduce the maximum

size to 5, and no overlapping appears in the cluster. The problem of overlapping gets solved when the size of the stars is randomly between 2.5 and 5. If we set the size between 5 to 10 units, there are 7 out of 10 stars overlapping in clusters. The data with the randomization between 2.5 and 5 meet the expectation since it successfully solved all overlapping scenarios. It turns out this way through reducing the size of all of the stars, which reduces the probability of stars that are close together overlapping on each other.

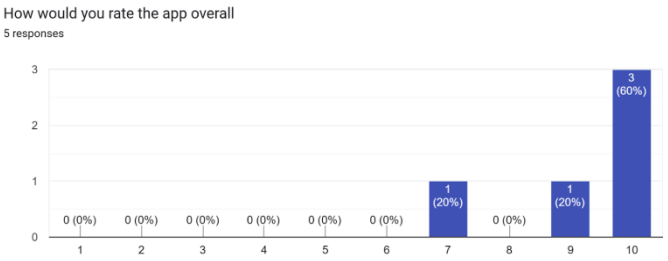
4.2. Experiment 2

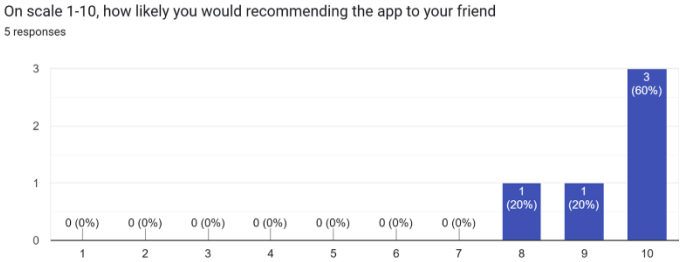
It helps evaluate how effectively the project meets the user's needs and what features can cause confusion. It provides insights from another perspective that are hard to identify by testing alone, including usability, clarity, engagement, and learning outcomes.

Survey results experiment goes here: Rate of the app overall, 9.2 out of 10, clarity and usability 4.4 out of 5, how likely you would recommend the app to your friend 9.4 out of 10, helpful on educational purposes 9 out of 10.

I posted the game on itch.io and asked for participants with a mix of ages and astronomy experience to try the game and fill in the post-task survey form to collect their opinions. I chose to ask a general measure of the user's satisfaction to see the app's overall quality. The clarity and usability assess how well users are able to navigate and understand the interface. Likelihood to recommend the app measures users' engagement and the app's potential popularity in the long term. Finally, the educational helpfulness rating measures whether the app fulfills its learning goals.

Provide the data from your experiment, as a graph. You can either paste in a Google Sheets graph, or submit a single image.





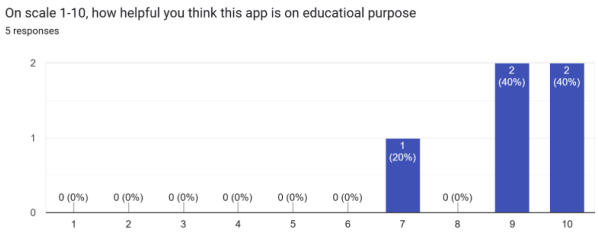


Figure 9. User Evaluation Results of the Telescope Simulator (Satisfaction, Usability, Engagement, and Educational Value)

The median scores for overall satisfaction, usability, and engagement are all 10. The mean score for educational purposes is 9. The overall rating of the app is 9.2 out of 10. Clarity and usability are rated at 8.8 out of 10. The likelihood of recommending the app to a friend is 9.4 out of 10, and it is helpful for educational purposes at 9 out of 10. Engagement receives the highest score. It is surprising that the overscores are higher than my expectations, while I thought it would be challenging for the users to interact with a 3D environment. The result turned out high because the app provides a platform for some of the users that never observed the night sky before. The biggest effect on the result is likely the app's accessibility and interactive design.

5. RELATED WORK

In the scholarly article "I Made a Sky Map in Python. Here's How." by Viyaleta Apgar, the author uses Python to build an extremely accurate sky map [12]. She uses formulas to calculate real positions corresponding to the physical locations, with accurate magnitude and star distance. Our project lacks accuracy since the scale is distorted. This solution doesn't help much because it is hard to apply the process to a 3D model or maintain accuracy during the interaction.

In the scholarly article "Real-World Oriented Smartphone AR Supported Learning System Based on Planetarium Contents for Seasonal Constellation Observation," the system solves our problem by applying GPS and sensors, aligning them with the stars corresponding to the user's environment [13]. However, this app has limitations, such as the GPS and sensor being sometimes inaccurate, and there are relatively few stars. Our app currently does not support reality visualization technology, limiting users from directly overlaying constellations on the live sky. Despite this, our project improves by integrating more accurate astronomical data with a more extensive database.

In the scholarly article "The Cost & Features for Developing a Popular Software like Star Walk 2 The Night Sky Map", the solution works by combining real-time GPS data, device sensors, and a star catalog to map celestial objects onto the user's screen [15]. However, it has similar limitations: GPS and sensor inaccuracies and dependence on specific devices, which limit their educational depth. Our project currently does not support AR visualization, but improves this by implementing more precise algorithms, supporting a wider range of devices, and enabling offline use.

6. CONCLUSIONS

One of the limitations of our project includes the lack of AR functionality, which prevents users from overlaying constellations directly onto the live sky. We could implement the argumentative reality by replacing the current star and plant object, taking into account gyroscope positions in

real time. Another issue is star size accuracy. Currently, the star may not appear in the correct relative size, affecting the reality of the sky map [16]. We could scale the stars icons based on their apparent magnitude from the database, so instead of randomizing the data, brighter stars appear larger and dimmer stars appear smaller. Our project does not utilize gyroscopic sensors, which limit precise orientations and makes it harder to align the sky map with the user's real-world view. We could use the device's gyroscope to track the orientation and rotation.

In conclusion, our project has done a great job of providing users with accurate star positions and constellation guidance. Limitations exist, such as a lack of AR integration that could enhance users' understanding. Future improvements focus on making the experience more interactive for learners.

REFERENCES

[1] Thorne, Ben, et al. "The Python Sky Model: software for simulating the Galactic microwave sky." Monthly Notices of the Royal Astronomical Society 469.3 (2017): 2821-2833.
[2] Tian, Ke, et al. "Real-world oriented smartphone AR supported learning system based on planetarium contents for seasonal constellation observation." Applied Sciences 9.17 (2019): 3508.
[3] Debray, Bibek, and L. D. Mago. "India's Trade Prospects with the CIS." Foreign Trade Review 27.1 (1992): 13-34.
[4] Ferrari, Enzo, Pablo Herrero Teijón, and Camilo Ruiz. "Unlocking the cosmos: evaluating the efficacy of augmented reality in secondary education astronomy instruction." Journal of New Approaches in Educational Research 13.1 (2024): 8.
[5] Beltozar-Clemente, Saul, et al. "Augmented reality mobile application to improve the astronomy teaching-learning process." Advances in Mobile Learning Educational Research 2.2 (2022): 464-474.
[6] Say, Serkan, and Volkan Pan. "The Effect of Instruction with Augmented Reality Astronomy Cards on 7th Grade Students' Attitudes towards Astronomy and Academic Achievement." Online Submission (2017).
[7] Leonardi, Laura, Laura Daricello, and Livia Giacomini. "Learning astronomy through Augmented Reality: EduINAF resources to enhance students' motivation and understanding." European Planetary Science Congress. 2021.
[8] de Moraes Rossetto, Anubis G., et al. "An analysis of the use of augmented reality and virtual reality as educational resources." Computer Applications in Engineering Education 31.6 (2023): 1761-1775.
[9] Malchenko, Svitlana L. "From smartphones to stargazing: the impact of mobile-enhanced learning on astronomy education." Science Education Quarterly 1.1 (2024): 1-7.
[10] Beltozar-Clemente, Saul, et al. "Augmented reality mobile application to improve the astronomy teaching-learning process." Advances in Mobile Learning Educational Research 2.2 (2022): 464-474.
[11] Coşkun, Mehmet, and Yasemin Koç. "The effect of augmented reality and mobile application supported instruction related to different variables in 7th grade science lesson." Psycho-Educational Research Reviews 10.2 (2021): 298-313.
[12] Schüssler, Fabian, et al. "Astro-COLIBRI: Empowering Citizen Scientists in Time Domain Astronomy." arXiv preprint arXiv:2407.10821 (2024).
[13] Palacios, A., et al. "POLLUX: a database of synthetic stellar spectra." Astronomy & Astrophysics 516 (2010): A13.
[14] Guiotto Nai Fovino, Lucrezia, et al. "Evaluating the effectiveness of sonification in science education using Edukoi." Personal and Ubiquitous Computing 28.5 (2024): 693-711.
[15] Maleke, B., D. Paseru, and R. Padang. "Learning application of astronomy based augmented reality using android platform." IOP Conference Series: Materials Science and Engineering. Vol. 306. No. 1. IOP Publishing, 2018.
[16] Malchenko, Svitlana L. "From smartphones to stargazing: the impact of mobile-enhanced learning on astronomy education." Science Education Quarterly 1.1 (2024): 1-7.