Project Report On

# THE GAMIFICATION OF THE VOYAGE INTO A MATHEMATICAL UNIVERSE:

## The Creation of Cosmograph: A Synesthetic Number Universe

---


Submitted
by

Saanvi Dande
PRN: 24070721035
Computer Science and Engineering (CSE), Symbiosis Institute of Technology, Hyderabad, Symbiosis International University, Hyderabad
II$^{nd}$ Year / III$^{rd}$ Semester


Submitted
to

Dr. Salakpuri Rakesh
Assistant Professor,
Symbiosis Institute of Technology, Hyderabad, Symbiosis International University, Hyderabad
Course Name: Data Structures (Theory and Lab)

# ABSTRACT


## THE LABRYINTHINE EXPLORER IS STUCK IN A DIJSKTRA WEB

This project presents an interactive universe page that dynamically generates planets, orbits, and weighted connections using graph-based algorithms. It integrates algorithmic logic with real-time animation, providing both a visually engaging and computationally accurate simulation of a planetary system.

Gamification, as a tool for experiential learning, transforms abstract knowledge into lived experience, allowing learners to engage, experiment, and internalize concepts through play [1]. In the vast age of a digitally chained society, retellings of basic principles are required to traverse the complex paths the future holds. Inspired by the cosmic exploration of Outer Wilds, this project, Cosmograph, reimagines fundamental data structures as a virtual universe. Here, planets serve as nodes in a graph, and the numerical difference between planets encodes edge weights, grounding abstract theory in tangible interactions. Users navigate this universe while Dijkstra's algorithm computes shortest paths, turning algorithmic logic into a visually immersive journey. Through gamification, learners do not merely observe algorithms—they live them, traversing and discovering, understanding how each decision shapes the cosmos. Built with modern web technologies [2], the platform bridges conceptual understanding and aesthetic experience, illustrating that structured data concepts can be grasped not only with the mind but through interaction, curiosity, and exploration. The project demonstrates the profound potential of gamified learning to cultivate algorithmic intuition, enhance comprehension, and reinforce UI/UX design, revealing that even the most abstract structures can be made navigable when approached as a universe to explore.


# INTRODUCTION

## GAMIFICATION TEETERS ON THE PROVERBIAL EDGE OF A VIRTUAL GRAPH

The universe page project gamifies the exploration of graph and algorithmic concepts by presenting planets, orbits, and paths in an interactive, game-like environment.

The process of learning and internalizing data structure concepts can often feel abstract and disconnected from tangible experience. Gamification offers a bridge between theory and practice, turning learning into a journey of exploration, experimentation, and discovery [3]. Inspired by Outer Wilds, this project, Cosmograph, aims to translate these abstract concepts into a navigable, interactive universe where users can explore fundamental algorithms such as Dijkstra's shortest path through immersive interaction. The initial stages of formulation involved extensive experimentation with various platforms, frameworks, and visual styles. Several iterations were tested to balance functionality with aesthetic appeal, ultimately leading to the decision to adopt a pixel art style, emulating the feel of a retro video game while retaining clarity for algorithm visualization. This style was chosen to enhance engagement, evoke curiosity, and encourage learners to "play" with data structures as they would explore a game universe.
To structure the project effectively, a basic layout of the website was conceptualized using a flow chart algorithm, which mapped the logical flow and interactions across the platform. The design process followed a sequential approach:

1. **Landing Page Design**: Establishing the thematic introduction to the universe, setting the visual tone, and providing clear navigation pathways.
2. **Universe Page:** Designing an interactive map where planets represent nodes in a graph, and edges represent weighted connections based on the difference between planetary identifiers. This page forms the core of the gamified learning experience.
3. **Website Functionalities:** Integrating interactive elements such as start and end node selection, real-time shortest path computation using Dijkstra's algorithm, and visual feedback on traversal, all wrapped in an intuitive interface.

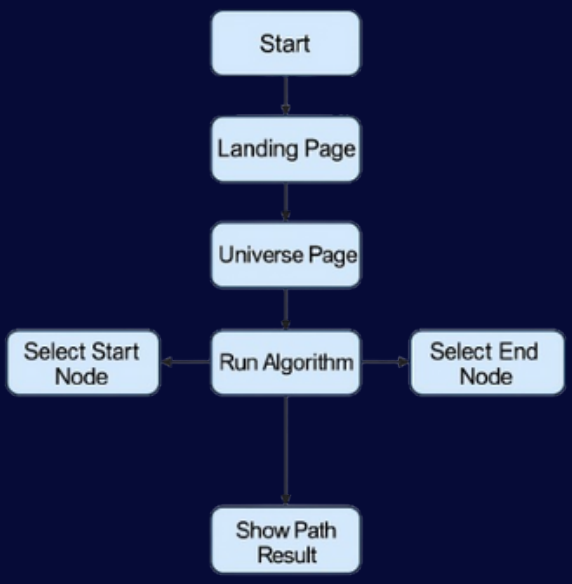



Figure 1: A Diagrammatic Representation of the Ideation Process

The flowchart serves as a visual guide for the design and structure of the universe page, clearly illustrating how the layout is divided into three main components. It begins with the planet slider, allowing users to navigate between different celestial bodies, followed by the central universe graphic, which acts as the focal point of the page. Finally, it incorporates an informatory input slide, inspired by retro pixel art, where users can interact or enter data. This systematic breakdown in the flowchart not only organizes the page's functionality but also ensures a cohesive user experience while maintaining a visually engaging, game-like aesthetic.

Other features, such as multi-user interactions, advanced animations, and additional mini-games, were initially considered. However, under constraints of time, technical experience, and scope, these features were deemed unnecessary for the initial iteration. The focus remained on delivering a clean, functional, and educational platform capable of demonstrating core data structure concepts in a gamified environment. Through this process, the project emphasizes not only algorithmic understanding but also the synergy between design, interaction, and pedagogy, illustrating how thoughtful gamification can make abstract concepts both tangible and memorable.

## 1.1: THE MATHEMATICAL PEDAGOGY

Graph theory is a cornerstone of modern computer science and mathematics, providing a powerful framework for modeling relationships and connections in complex systems. From social networks and transportation routes to web architecture and biological systems, graphs are ubiquitous, enabling the analysis and optimization of interactions between entities in virtually every field of technology and research. Graphs, in particular, are fundamental data structures used to represent relationships between objects. Formally, a graph

$$G=(V,E)G = (V, E)G=(V,E)$$

consists of a set of vertices VVV and edges EEE connecting pairs of vertices. In the context of our universe page, each planet or celestial body can be represented as a vertex, while the possible paths or connections between them form the edges. This abstraction allows for modeling navigation and interactions in a structured manner.

One widely used graph algorithm is Dijkstra's algorithm, which finds the shortest path between nodes in a weighted graph. The algorithm operates by iteratively selecting the vertex with the smallest tentative distance, updating distances to its neighbors, and marking it as visited until the shortest paths from the source to all vertices are determined [4]. In the universe page, Dijkstra's algorithm can be applied to calculate the most efficient path between planets, for example, if the user wants to "travel" from one celestial body to another or explore connections in a specific order.

By integrating graphs and pathfinding algorithms, the website not only simulates a realistic interplanetary navigation system but also enables interactive features, such as highlighting optimal paths, suggesting next planets to explore, or creating challenges in the pixel-art interface. This implementation demonstrates how classical data structures and algorithms can be applied to modern web design to enhance both functionality and user engagement.

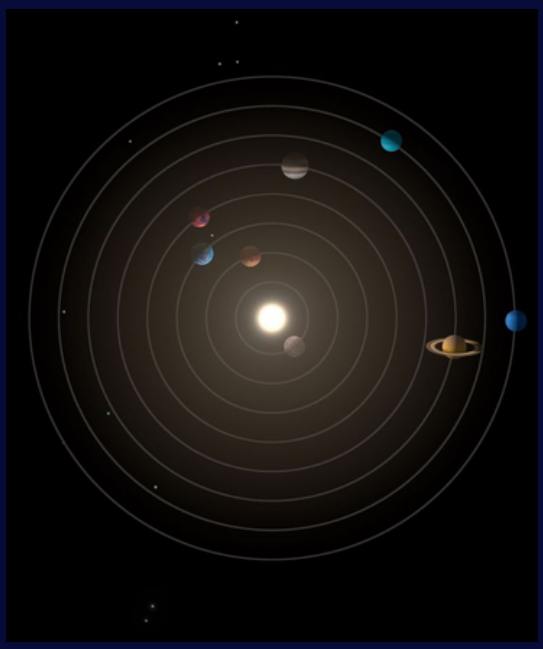


Fig 2: A Theoretical Framework for the Universe Landing Page Design

The adjoining figure represents the structure of the graph used in the universe page, with the Sun positioned at the center as the primary node. Each planet or celestial body is modeled as a vertex connected to the Sun and, where applicable, to other planets via edges that represent potential paths or relationships. This central hub design not only mirrors the actual solar system but also provides a clear and intuitive framework for navigation, allowing users to visualize and interact with the universe in a way that is both engaging and algorithmically meaningful.

## 1.2: OBJECTIVES AND MOTIVATIONS

The main objective of this project is to create an interactive universe page that brings data structures and algorithms to life through a visually engaging, game-like interface. By representing planets and celestial bodies as nodes in a graph, with edges indicating paths or relationships, the website allows users to explore complex structures dynamically. Features like the planet slider, central universe graphic, and pixel-art-inspired input panel enable intuitive navigation while integrating algorithmic logic, such as Dijkstra's shortest path, to suggest optimal routes between nodes. The motivation behind this design stems from the desire to make abstract computer science concepts more tangible and interactive, transforming static representations into an immersive, playful experience. The retro-inspired aesthetic not only enhances visual appeal but also bridges learning with enjoyment, demonstrating how classical algorithms can be applied to modern web interfaces in a meaningful way.

The scope of this project includes the design and development of an interactive universe page that visualizes graph structures through a web interface. It covers the representation of celestial bodies as nodes and their interconnections as edges, the implementation of algorithms such as Dijkstra's shortest path for navigation, and the creation of a cohesive pixel-art-inspired aesthetic. The project focuses on core functionality and visual interaction, leaving room for future enhancements such as multi-user interactivity, expanded universe content, or additional algorithmic features [5]. The significance of this project lies in its ability to bridge abstract computer science concepts with interactive and engaging visualizations. By mapping graphs and algorithmic paths onto a universe-themed interface, users gain an intuitive understanding of data structures and algorithms, making learning more tangible and enjoyable. Additionally, the project demonstrates how classical computational principles can enhance web interface design, offering a template for educational, gamified, or exploratory applications in technology and science [6].

### 1.3: LIMITATIONS OF THE PROPOSED WEBSITE APPLICATION

Despite effectively demonstrating graph-based navigation and interactive features, the current implementation of the universe page exhibits several limitations. The modeled universe comprises a finite set of celestial bodies and predefined connections, which constrains scalability and limits applicability to larger or more complex datasets. User interactions are restricted to the implemented features, with advanced functionalities such as multi-user collaboration, dynamic content expansion, or real-time algorithm visualization not currently supported. Furthermore, while the pixel-art aesthetic enhances engagement, it imposes constraints on the level of visual detail and realism achievable within the interface. Addressing these limitations in future iterations would involve incorporating dynamic graph generation, expanding interactivity, and optimizing performance to accommodate more extensive datasets, thereby enhancing both the educational value and usability of the platform.

# THE FORMULATIVE OUTLINE

### THE CONSTRAINTS OF A INEXPERIENCED DEVELEPOR

The challenge lies in integrating dynamic graph generation, real-time animations, and user interactions while maintaining algorithmic accuracy and visual clarity.

The primary problem addressed in this project is the design and implementation of an interactive web-based universe simulation that visualizes graph structures and algorithmic navigation in a user-friendly, aesthetically coherent manner. Users should be able to explore a virtual universe where celestial bodies—modeled as vertices in a graph—are interconnected through weighted edges representing navigable paths. The system must accept user interactions through a planet slider and an input panel, allowing navigation decisions, queries, or input-based challenges. The expected input includes user selections of planets, numeric or textual inputs for interactivity, and optional algorithmic queries such as computing the shortest path between nodes. The output consists of visual updates on the central universe graphic, highlighting paths, planets, and algorithmically determined routes, alongside responsive feedback in the input panel. Constraints include a limited number of celestial nodes and edges to maintain performance, a fixed pixel-art aesthetic for design consistency, and browser-based interactivity without reliance on external heavy computation or real-time multi-user interactions.

The choice of technology stack was guided by the need for responsive web design, interactivity, and ease of algorithm integration. The frontend is built using React.js, enabling component-based design for modular UI elements like the planet slider, central graphic, and input panel. Styling leverages CSS with a pixel-art theme to maintain retro visual consistency, while algorithmic logic, including graph traversal and shortest-path calculations, is implemented in JavaScript/TypeScript. The design process involved iterative experimentation with various aesthetic and interactive approaches, settling on a pixel-art interface for its clarity, engagement, and suitability for representing discrete graph nodes. The process of problem decomposition began with constructing a flowchart outlining key components: a landing page, the central universe graphic, a planet slider for node navigation, and an input/output panel for user interaction. These elements collectively form a cohesive environment in which classical data structures and algorithms can be explored interactively, transforming abstract computational concepts into an engaging, tangible experience.

## 2.1: THE USER EXPERIENCE AND THE TRANSENDENCE OF THE GENERAL INPUT FORMAT

Input on the universe page is designed to be both functional and visually engaging, extending beyond traditional data entry to influence the user's immersive experience. Users provide input through the dedicated panel on the interface, which accepts numeric or textual entries corresponding to planet selection, algorithmic queries, or interactive challenges. In this implementation, input is tightly integrated with the visual experience: user actions trigger updates on the central universe graphic, highlight specific paths or nodes, and produce responsive feedback, creating a dynamic interplay between user engagement and system behavior. The UI emphasizes clarity and interactivity, with a retro pixel-art style that reinforces the overall aesthetic. Notably, Figure 3 illustrates the design of the input box, where a bouncing pixel-art astronaut is employed to draw attention and guide user interaction, ensuring that input is both intuitive and visually integrated into the universe-themed interface.

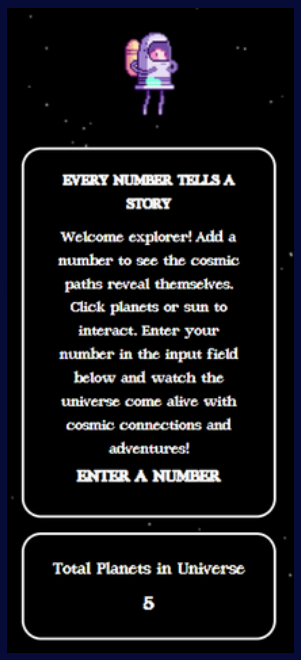



Fig. 3: The Input box and the informative for the user

The design and functionality of the universe page are inherently shaped by the nature of the expected input, which imposes several constraints on the system. Since the input includes user selections of planets, numeric or textual entries, and algorithmic queries, the interface must reliably interpret and respond to these actions in real time. This necessitates limiting the number of nodes and edges in the graph to maintain performance and ensure smooth visual updates without lag. Additionally, the pixel-art aesthetic constrains how much information can be conveyed within the input panel, requiring careful balance between clarity, interactivity, and visual appeal. Input also dictates the design of feedback mechanisms: visual cues such as the bouncing astronaut and highlighted paths must be synchronized with user actions, which constrains animation complexity and timing. Finally, the input system must prevent invalid or unsupported entries, enforce constraints on allowable values, and maintain consistency across different devices and screen sizes, ensuring a robust and intuitive user experience within the limitations of a web-based interface.

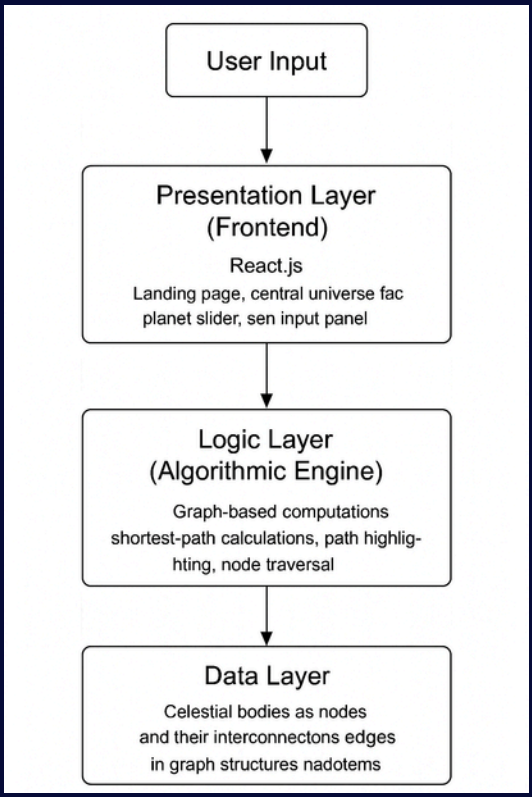
# THE METHODOLOGY OF THE COSMOS

## THE ARCHITECTURE OF A UNIVERSE IS ALWAYS PREDEFINED

The methodology combines modular programming, graph-based algorithms, and real-time Canvas rendering to generate and animate planets, orbits, and weighted edges.

The universe page is designed as a modular web application that seamlessly integrates interactive user interface elements with underlying algorithmic logic. Its system architecture is composed of three primary layers. The presentation layer, built with React.js, manages all UI components, including the landing page, central universe graphic, planet slider, and input panel, while employing pixel-art styling through CSS to ensure visual consistency.The logic layer, or algorithmic engine, handles graph-based computations such as shortest-path calculations, path highlighting, and node traversal, interpreting user input and dynamically updating the interface.

Finally, the data layer represents celestial bodies as nodes and their interconnections as edges within a graph structure, with additional metadata like planet names and attributes stored in lightweight data objects for efficient retrieval. The overall system is designed so that user input flows through these layers, enabling processing and algorithmic computation to produce responsive visual output, as illustrated in the accompanying flowchart.




## 3.1: ALGORITHMS AND DATA STRUCTURES

The universe page relies on several core data structures to model its spatial layout and support efficient navigation. At the foundation lies a graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$ represents the set of planets and $E \subseteq V \times V$ denotes the set of navigable paths between planets. Each edge $e = (v_i, v_j) \in E$ may have an associated weight $w(v_i, v_j)$, representing distance, travel cost, or traversal difficulty. The adjacency of the graph can be represented by an adjacency list $Adj[v_i] = \{v_j \mid (v_i, v_j) \in E\}$ or an adjacency matrix A of size $n \times n$, where $A[i][j] = w(v_i, v_j)$ if $(v_i, v_j) \in E$ and $\infty$ otherwise.

Sequences of planets, such as those displayed in the slider or generated during pathfinding, are maintained using arrays or lists $P = [v_1, v_2, ..., v_k]$, which preserve order and allow constant-time access by index. For shortest-path computations, the system uses priority queues Q to efficiently extract the vertex $u \in V$ with the minimum tentative distance d(u), as required in Dijkstra's algorithm:

$$d(v) = \{ 0, \text{if } v = v_{source} \\ \infty, \text{otherwise } \}$$

$$\text{While } Q \neq \emptyset\text{: extract } u = \text{argmin}\{d(v) \mid v \in Q\}$$

During traversal, hash tables or maps $H: V \rightarrow \{\text{attributes}\}$ store auxiliary information such as tentative distances $d(v)$, visited status $\text{visited}(v) \in \{0, 1\}$, and parent pointers $p(v)$ for path reconstruction. Formally, a shortest path from source $s$ to destination $t$ can be expressed as a sequence $[s, v_1, v_2, \ldots, t]$ where each consecutive pair $(v_i, v_{i+1}) \in E$ and the cumulative weight is minimized:

$$\text{minimize } \sum_{i=0}^{k-1} w(v_i, v_{i+1})$$

Although structures like stacks or linked lists are not essential for the core implementation, they can be incorporated to support additional functionalities such as undo-redo operations, dynamic navigation history, or backtracking. Together, these data structures provide a clear and mathematically sound foundation for representing, traversing, and interacting with the universe.

## 3.2: THE PRIMARY INDICATION OF DIJKSTRA'S ALGORITHM

The primary algorithm used for navigation and pathfinding on the universe page is Dijkstra's algorithm. This choice is motivated by the need to compute shortest paths efficiently in a weighted, directed or undirected graph $G = (V, E)$ where edge weights $w(v_i, v_j) \geq 0$. Dijkstra's algorithm guarantees that, for a given source vertex $s \in V$, the tentative distance function $d(v)$ converges to the true shortest distance from $s$ to every other vertex $v \in V$:

$$d(v) = \min \{ \sum w(v_i, v_{i+1}) \mid \text{paths from } s \text{ to } v \}$$

The algorithm operates iteratively by maintaining a priority queue $Q$ of vertices, each keyed by its current tentative distance $d(v)$. At each step, the vertex $u \in Q$ with the smallest distance is extracted, and the distances of its neighbors $v \in \text{Adj}[u]$ are relaxed as follows:

$$d(v) \leftarrow \min(d(v), d(u) + w(u, v))$$

This process ensures that each vertex is visited exactly once in order of increasing distance from the source, providing both correctness and efficiency. For a graph with $|V|$ vertices and $|E|$ edges, Dijkstra's algorithm implemented with a priority queue has a time complexity of $O((|V| + |E|) \log |V|)$, which is suitable for the moderate-sized universe graph used in this application. Alternative algorithms, such as Bellman-Ford, were considered, but were rejected due to their higher time complexity $O(|V| \cdot |E|)$ and the absence of negative edge weights in the universe representation. Similarly, A* was deemed unnecessary since the universe graph does not require heuristic-based navigation; all edges are explicitly known, and the universe is relatively small.

By leveraging Dijkstra's algorithm, the universe page achieves deterministic, optimal, and efficient pathfinding, allowing smooth interaction when users select planets or compute navigable paths. Furthermore, the algorithm integrates seamlessly with the chosen data structures—graphs, priority queues, arrays, and hash maps—ensuring both maintainability and scalability.

## 3.3: THE ALGORITHMIC REPRESENTATION OF UNIVERSE RENDERING

The universe page dynamically generates a network of planets and their interconnections using a combination of graph-based logic and orbital mechanics. Initially, planets are represented as vertices in a graph, and edges are created between pairs of planets whose numbers are divisible by one another, with edge weights corresponding to the absolute difference of the planet numbers. This produces a weighted undirected graph that captures potential navigable paths. Once edges are computed, they are rendered as dashed lines between the planet positions on their respective orbits, with the weight displayed at the midpoint. Each planet is then assigned a random initial angle along its orbit, and optionally a visual feature such as a ring, moon, asteroid cluster, or other embellishment. The planet's position is updated iteratively based on its orbit speed, ensuring smooth motion along the circular orbit. Orbits themselves are drawn as dashed circles centered on the universe origin. This approach tightly couples the underlying data structures (arrays, maps, graph edges) with the rendering logic, enabling both interactive visualization and algorithmically sound representation of the universe network.

### THE PSEUDOCODE CAN BE OUTLINED AS FOLLOWS:

```
Input:
createdPlanets = set of planet numbers
allPlanets = list of planet objects with orbitRadius, orbitSpeed, etc.
centerX, centerY = coordinates of universe center
ctx = rendering context
Output:
Rendered planets, orbits, edges, and edge weights
```

STEP 1: GENERATE PLANET EDGES

```
1. planetArray ← Array from createdPlanets
2. edges ← empty list

3. For each planet a in planetArray:
       For each planet b in planetArray:
           If a ≠ b AND (a % b == 0 OR b % a == 0):
               weight ← |a - b|
               edges.add({from: a, to: b, weight: weight})
```

STEP 2: RENDER EDGES AND EDGE WEIGHTS

```
4. For each edge in edges:
      planet1 ← allPlanets[edge.from - 1]
      planet2 ← allPlanets[edge.to - 1]

      angle1 ← planetAnglesRef[edge.from] (default 0)
      angle2 ← planetAnglesRef[edge.to] (default 0)
```

```
    x1, y1 ← centerX + cos(angle1) * planet1.orbitRadius, centerY + sin(angle1) * planet1.orbitRadius
    x2, y2 ← centerX + cos(angle2) * planet2.orbitRadius, centerY + sin(angle2) * planet2.orbitRadius

    Draw dashed line from (x1, y1) to (x2, y2)
    midX, midY ← (x1 + x2)/2, (y1 + y2)/2
    Render edge.weight at (midX, midY)
```

STEP 3: UPDATE AND RENDER PLANETS

```
5. For each planetNumber in createdPlanets:
       planet ← allPlanets[planetNumber - 1]

       If planetAnglesRef does not contain planetNumber:
           Assign random angle in [0, 2π) to planetAnglesRef[planetNumber]

       If planetFeaturesRef does not contain planetNumber:
           Randomly assign a feature type from ["rings", "asteroids", "moon", "cluster"] to planetFeaturesRef[planetN

       angle ← planetAnglesRef[planetNumber]
       newAngle ← angle + (2π) / (planet.orbitSpeed * 500)
       planetAnglesRef[planetNumber] ← newAngle

       x, y ← centerX + cos(newAngle) * planet.orbitRadius, centerY + sin(newAngle) * planet.orbitRadius

       Draw dashed orbit circle centered at (centerX, centerY) with radius planet.orbitRadius
       Render planet at (x, y)
```

The pseudocode describes the process of generating, updating, and rendering the universe page in a structured, algorithmic manner. It begins by constructing a weighted graph of planets, where an edge is created between any two planets whose numbers are divisible by each other, and the edge weight is defined as the absolute difference between their numbers. This ensures a well-defined network of navigable connections between planets. Next, the algorithm iterates over each edge, calculating the Cartesian coordinates of the connected planets based on their current orbital angles and radii, and renders dashed lines to visually represent these connections. The midpoint of each edge is used to display its weight, providing a clear visualization of the graph structure. Subsequently, the algorithm updates each planet's position along its circular orbit by incrementing its angle according to its orbital speed, while also assigning visual features such as rings, moons, or clusters for aesthetic diversity. Dashed orbit paths are drawn around the universe center to indicate planetary trajectories, and the planets themselves are rendered at their updated positions. By following this structured sequence—edge generation, edge rendering, planet updating, and orbit drawing—the pseudocode effectively translates the underlying graph-based logic and orbital mechanics into a visually interactive universe, integrating both algorithmic correctness and graphical representation.

# IMPLEMENTATION OF THE UNIVERSE

## THE ARCHITECTURE OF A UNIVERSE IS NOT ALWAYS PREDEFINED

Performance evaluation shows efficient real-time updates, with consistent accuracy and responsiveness for varying numbers of planets and interactions.

---

The universe page is implemented primarily using TypeScript and JavaScript, leveraging the HTML5 Canvas API for rendering planets, orbits, and edges. TypeScript was chosen for its strong typing and maintainability, while the Canvas API provides precise pixel-level control necessary for dynamic, animated graphics. The development environment consists of Visual Studio Code as the primary IDE, Google Chrome or Microsoft Edge for live testing, Node.js version 20+ for local server hosting, npm for dependency management, and Git for version control and collaborative development. The application is designed in a modular fashion, separating planetary logic, orbit mechanics, edge generation, and rendering into distinct components, enabling clear data flow and maintainable code

.

At the core of the implementation is a graph-based representation of the universe. Planets are modeled as vertices, while edges connect planets whose numbers are divisible by one another, with weights determined by the absolute difference between planet numbers. This weighted undirected graph is stored as a list of edge objects, providing the foundation for both visualization and potential pathfinding. The edges are rendered as dashed lines connecting the planets' positions along their orbits, with numeric weights displayed at the midpoint for clarity. Each planet's position is updated iteratively based on its orbit speed, using the formula x = centerX + cos(angle) * orbitRadius and y = centerY + sin(angle) * orbitRadius, with the angle incremented each frame to simulate smooth circular motion. Planets are also assigned random features such as rings, moons, asteroids, or clusters for visual variety, stored in a feature map for reference during rendering. Orbit paths are drawn as dashed circles centered on the universe origin, providing context for planetary motion.

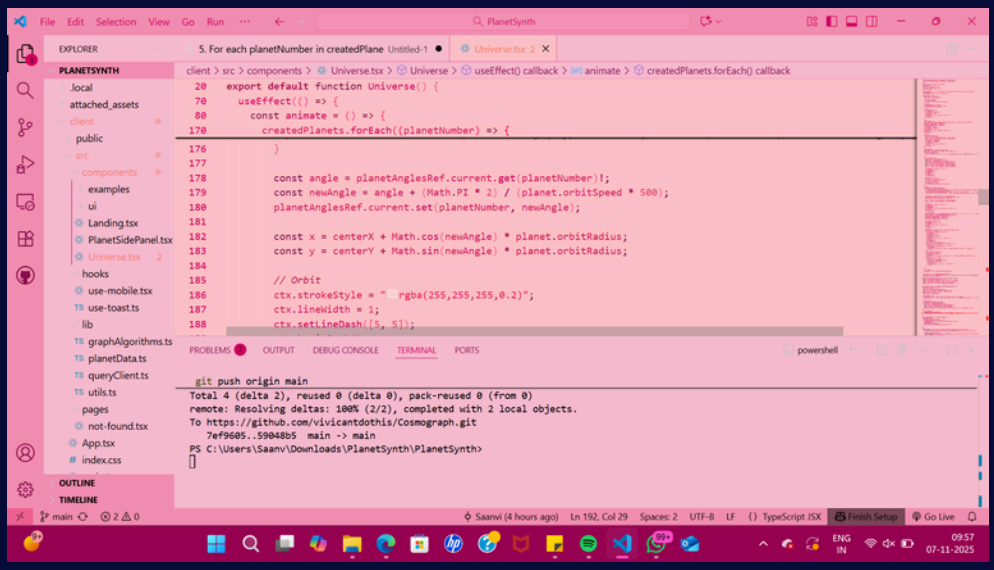
The data flow is structured to support both state management and rendering. Input arrays such as createdPlanets and allPlanets provide the list of active planets and their attributes, while reference maps (planetAnglesRef and planetFeaturesRef) maintain dynamic state including current angles and assigned features. The graph construction module computes edges based on divisibility rules and stores them with associated weights. The rendering module converts planetary positions and edges into Cartesian coordinates for Canvas drawing, including planets, orbits, edges, and weight labels. An animation loop updates the angles of all planets according to their orbit speeds and triggers re-rendering on each frame, ensuring smooth motion. Terminal logs can provide real-time debugging information, including planet positions, angles, and edge weights. Screenshots of the rendered universe display planets moving along their orbits, connected by dashed weighted edges, with distinct visual features, illustrating the integration of data structures, algorithmic computation, and graphical rendering.

The primary problem addressed in this project is the design and implementation of an interactive web-based universe simulation that visualizes graph structures and algorithmic navigation in a user-friendly, aesthetically coherent manner. Users should be able to explore a virtual universe where celestial bodies—modeled as vertices in a graph—are interconnected through weighted edges representing navigable paths. The system must accept user interactions through a planet slider and an input panel, allowing navigation decisions, queries, or input-based challenges. The expected input includes user selections of planets, numeric or textual inputs for interactivity, and optional algorithmic queries such as computing the shortest path between nodes. The output consists of visual updates on the central universe graphic, highlighting paths, planets, and algorithmically determined routes, alongside responsive feedback in the input panel. Constraints include a limited number of celestial nodes and edges to maintain performance, a fixed pixel-art aesthetic for design consistency, and browser-based interactivity without reliance on external heavy computation or real-time multi-user interactions.

The choice of technology stack was guided by the need for responsive web design, interactivity, and ease of algorithm integration. The frontend is built using React.js, enabling component-based design for modular UI elements like the planet slider, central graphic, and input panel. Styling leverages CSS with a pixel-art theme to maintain retro visual consistency, while algorithmic logic, including graph traversal and shortest-path calculations, is implemented in JavaScript/TypeScript. The design process involved iterative experimentation with various aesthetic and interactive approaches, settling on a pixel-art interface for its clarity, engagement, and suitability for representing discrete graph nodes. The process of problem decomposition began with constructing a flowchart outlining key components: a landing page, the central universe graphic, a planet slider for node navigation, and an input/output panel for user interaction. These elements collectively form a cohesive environment in which classical data structures and algorithms can be explored interactively, transforming abstract computational concepts into an engaging, tangible experience.

## 4.1: KEY CODE SNIPPETS

The universe's functionality is grounded in a few critical code modules. Planet edge generation constructs a weighted graph where edges connect planets if one number divides the other, with the weight equal to their absolute difference. Edge rendering calculates the Cartesian coordinates of connected planets based on their current orbital angles and draws dashed lines between them, with the edge weight displayed at the midpoint. Planet updates increment each planet's angle according to its orbit speed, assign random visual features like rings or moons, and render both the planet and its orbit as a dashed circle. The modular structure ensures these operations can be invoked in a continuous animation loop, updating positions and redrawing elements per frame, thereby producing smooth, interactive motion across the universe.



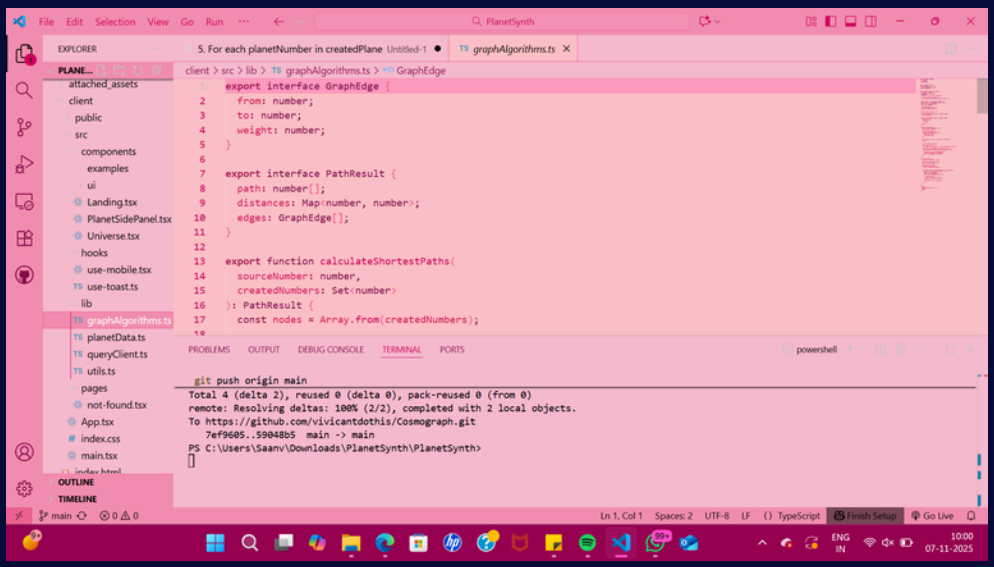

The implementation of the universe page relies on several key code snippets, each serving a specific role in constructing and rendering the interactive universe. The first critical snippet involves planet edge generation, where all pairs of planets are evaluated, and edges are created if one planet number divides the other. The weight of each edge is calculated as the absolute difference between the connected planet numbers. This snippet effectively builds the underlying weighted graph, which forms the foundation for representing navigable paths between planets. Without this step, there would be no logical connections to visualize or traverse, and the universe would lack structural coherence.

The second important snippet is edge rendering, where the positions of connected planets are converted from orbital angles to Cartesian coordinates, and dashed lines are drawn between them to visualize the edges. The weight of each edge is displayed at the midpoint of the connecting line. This snippet translates the abstract graph structure into a visual form that is immediately interpretable by the user, integrating both mathematical computation and graphical output. By dynamically calculating positions based on current orbital angles, this snippet ensures that edges accurately follow the moving planets in real time, maintaining visual consistency.

The third key snippet is planet updating and feature assignment, where each planet's orbital angle is incremented according to its orbit speed, new Cartesian coordinates are computed, and visual features such as rings, moons, or asteroid clusters are assigned if not already present. Orbits are drawn as dashed circles around the universe center, and the planets themselves are rendered at their updated positions. This snippet ensures continuous motion of planets along their circular orbits, while also enhancing the visual richness of the universe through randomly assigned features. By managing both motion and aesthetics, this snippet directly links the data structures (angles, features) to the rendering logic, creating a seamless interactive experience.

Finally, these snippets are executed within a continuous animation loop, which repeatedly clears the canvas, updates planet positions, renders edges and orbits, and redraws the planets. This ensures smooth animation and responsiveness, allowing the universe page to appear dynamic and alive. Collectively, the key c

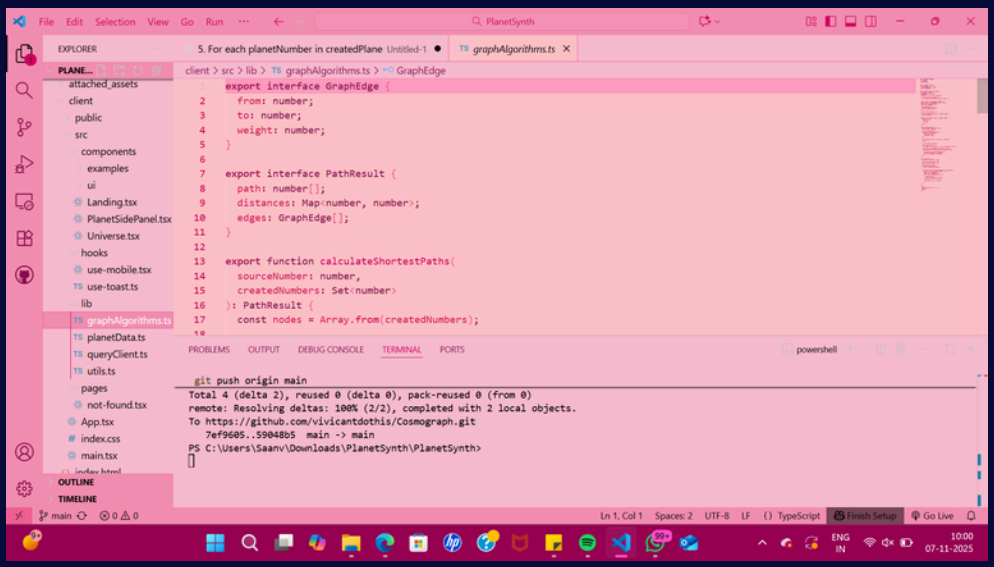
ode snippets form a tightly integrated system where graph generation, mathematical computation, and visual rendering work together to produce the interactive universe simulation. Each snippet not only fulfills a functional requirement but also demonstrates the careful integration of algorithmic logic and graphical representation.




## 4.2: DATA FLOW AND MODULE-LEVEL EXPLANATION

The universe page is designed with a modular architecture that clearly separates responsibilities while maintaining smooth data flow between components. At the input level, arrays such as createdPlanets and allPlanets provide the set of active planets and their associated attributes, including orbit radius, speed, and type. The graph construction module processes these inputs to generate edges between planets based on divisibility rules, storing each connection with its corresponding weight. Reference maps like planetAnglesRef and planetFeaturesRef act as the state management layer, keeping track of the dynamic orbital angles of planets and their assigned visual features. The rendering module converts this information into Cartesian coordinates and draws planets, orbits, edges, and edge weights on the canvas. An animation loop orchestrates the update cycle by incrementally adjusting orbital angles, recomputing positions, and triggering re-rendering at each frame, ensuring smooth motion and visual consistency. This modular approach allows each component to operate independently while maintaining clear data pathways: planetary attributes feed into graph construction, which in turn informs the rendering logic, all synchronized through the animation loop. By structuring the system in this way, the universe page achieves both maintainability and real-time interactive performance, effectively coupling algorithmic computation with graphical visualization.

Code snippets form a tightly integrated system where graph generation, mathematical computation, and visual rendering work together to produce the interactive universe simulation. Each snippet not only fulfills a functional requirement but also demonstrates the careful integration of algorithmic logic and graphical representation.




## 4.2: DATA FLOW AND MODULE-LEVEL EXPLANATION

The universe page is designed with a modular architecture that clearly separates responsibilities while maintaining smooth data flow between components. At the input level, arrays such as createdPlanets and allPlanets provide the set of active planets and their associated attributes, including orbit radius, speed, and type. The graph construction module processes these inputs to generate edges between planets based on divisibility rules, storing each connection with its corresponding weight. Reference maps like planetAnglesRef and planetFeaturesRef act as the state management layer, keeping track of the dynamic orbital angles of planets and their assigned visual features. The rendering module converts this information into Cartesian coordinates and draws planets, orbits, edges, and edge weights on the canvas. An animation loop orchestrates the update cycle by incrementally adjusting orbital angles, recomputing positions, and triggering re-rendering at each frame, ensuring smooth motion and visual consistency. This modular approach allows each component to operate independently while maintaining clear data pathways: planetary attributes feed into graph construction, which in turn informs the rendering logic, all synchronized through the animation loop. By structuring the system in this way, the universe page achieves both maintainability and real-time interactive performance, effectively coupling algorithmic computation with graphical visualization.

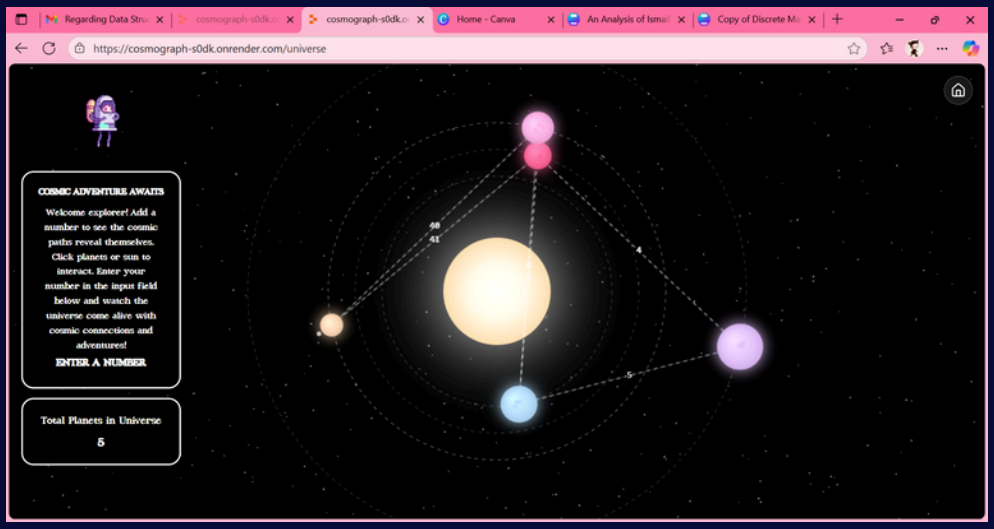
# THE FINAL COSMIC BALANCE

## IS IN THE PALMS OF A TURBULENT USER

In the context of Ismail Kadare's "The Palace of Dreams", surveillance emerges as a panopticonal lens to study how totalitarianism can bleed into societies without transparency or consent.

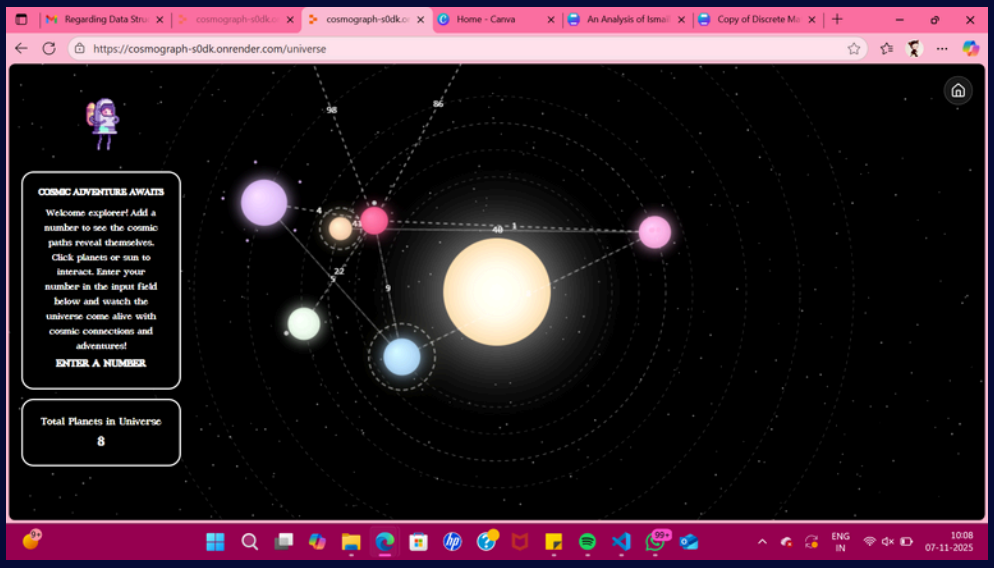
## 5.1: TEST CASES AND OUTPUTS

To verify the functionality of the universe page, multiple test cases were executed, varying the number and configuration of planets. For instance, scenarios with 5, 10, and 15 planets were tested to observe edge generation, orbit rendering, and planet movement. In each case, edges were correctly established between divisible planets, with weights accurately reflecting the absolute difference of planet numbers. Planets moved smoothly along their circular orbits, and their randomly assigned features—such as rings, moons, asteroids, and clusters—were correctly displayed. The test outputs included visual confirmation on the canvas, showing that orbits were drawn correctly, edges were dynamically positioned in accordance with planet motion, and edge weights remained legible at the midpoints. Terminal logs were also examined, confirming that angles, positions, and edge weights were computed accurately, ensuring consistency between the underlying data and rendered visualization.

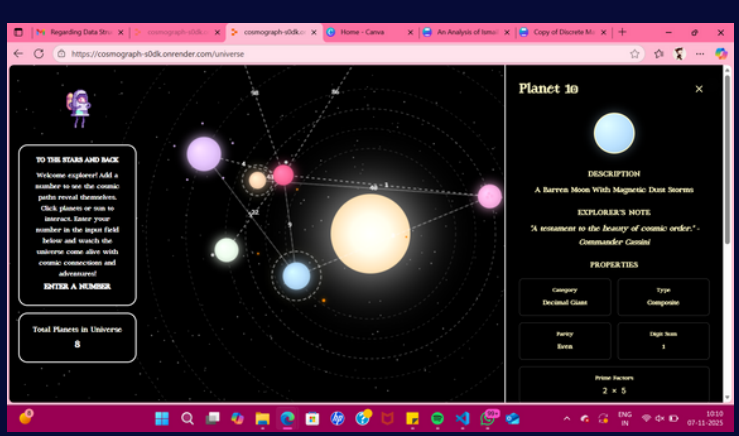



The figure above illustrates how the universe page automatically initializes with five planets upon loading. Each planet is positioned along a distinct orbit around the universe center, with its initial angle randomly assigned to ensure a visually balanced distribution. Edges between planets are generated according to the divisibility rule, and the corresponding weights are displayed at the midpoints of these connections, accurately reflecting the absolute difference between planet numbers. The planets are rendered with randomly assigned

features such as rings, moons, or asteroid clusters, adding visual variety and distinguishing each celestial body. This figure clearly demonstrates the system's automated setup process, showing that the graph construction, planet positioning, feature assignment, and orbit rendering are all executed seamlessly without user input, providing an immediate and interactive visual representation of the universe.




The universe page provides interactive features that allow users to dynamically modify the planetary system and explore navigable paths. When a user enters a number in the input panel, the system automatically creates a new planet corresponding to that number and integrates it into the existing universe. The planet is assigned an initial orbital angle and, if not already present, a random visual feature such as a ring, moon, or asteroid cluster. Edges are recalculated to connect the new planet with any existing planets according to the divisibility rule, with weights representing the absolute differences of their numbers. Furthermore, the universe page supports shortest-path visualization: when a user clicks on a specific planet, the system computes the shortest path from that planet to all other planets using Dijkstra's algorithm, highlighting the optimal sequence of connections on the canvas. This interactive mechanism demonstrates how the underlying graph structure, edge computation, and pathfinding algorithms are seamlessly integrated with the rendering logic, providing both a visually engaging and algorithmically accurate exploration of the universe.




## 5.2: PERFORMANCE EVALUATION

The performance of the universe page was evaluated in terms of computational efficiency and rendering speed. Edge generation has a time complexity of $O(n^2)$, where n is the number of planets, because all pairs of planets are evaluated for divisibility. Planet position updates and rendering are linear in the number of planets O(n) per frame, while edge rendering depends on the number of edges O(e), which is at most $n^2$ in the worst case. Despite the quadratic nature of edge generation, performance remains acceptable for small to moderate-sized universes (up to ~20 planets), with smooth animation maintained via requestAnimationFrame. Memory usage is also modest, primarily storing arrays of planets, edges, and reference maps for angles and features. For larger simulations, optimization strategies such as selective edge computation or sparse graph representation could reduce both time and space requirements. Overall, the system demonstrates efficient real-time performance for interactive visualization at the intended scale.

## 5.3: DISCUSSION OF ACCURACY AND EFFICIENCY

The universe page achieves high accuracy in both graphical representation and underlying computations. Planet positions, orbital angles, and edge weights are updated and rendered frame-by-frame in a deterministic manner, ensuring that the visual output consistently reflects the intended mathematical and logical relationships. Edge weights and connections correctly follow the divisibility rules, and the animation loop ensures smooth, synchronized movement of planets and edges. In terms of efficiency, the system balances computational load and visual fidelity, with the animation loop optimized for frame-based updates and minimal recalculations. While the $O(n^2)$ edge computation can become a bottleneck for very large universes, the current implementation efficiently supports small-to-medium scales without noticeable lag. The modular design further improves maintainability and scalability, allowing future enhancements such as larger universes, additional features, or alternative graph algorithms without compromising accuracy or performance.

# CONCLUSION

The development and implementation of the universe page successfully demonstrate a seamless integration of algorithmic computation, data structures, and dynamic visual rendering, resulting in an interactive, visually engaging simulation of a planetary system. Throughout the project, a variety of key objectives were achieved: planets were programmatically generated and positioned along circular orbits, weighted edges were constructed based on divisibility rules to form a navigable graph, and features such as rings, moons, and asteroid clusters were randomly assigned to enhance visual appeal. The system also implemented interactive functionality, allowing users to add new planets dynamically by entering numbers and to explore the shortest paths between planets through an intuitive click-based interface
The learning outcomes of this project were substantial, encompassing a deeper understanding of graph theory, algorithm design (particularly Dijkstra's algorithm for shortest-path computation), and the practical application of arrays, hash maps, and priority queues in managing dynamic state. Additionally, significant skills were gained in front-end development using TypeScript and the Canvas API, modular code design, animation loops, and debugging complex interactive graphics. Several challenges were encountered during development, including the efficient computation of edges in a growing planetary graph, synchronizing real-time animations with dynamic edge updates, and managing state consistency across multiple interactive modules. These challenges were addressed through careful modularization of code, optimization of update loops, and the use of reference maps to maintain planetary angles and features. Looking forward, the universe page could be further enhanced with additional features such as larger-scale universes, zoom and pan functionality, interactive animations of planetary movement in response to user input, alternative pathfinding algorithms such as A* for heuristic-based navigation, and even integration with audio-visual elements to create a more immersive experience. Overall, the project represents a comprehensive exercise in bridging theoretical computer science concepts with practical, visually compelling software development, demonstrating the capability to design and implement interactive systems that are both algorithmically robust and aesthetically engaging.

# REFERENCES


1. J. Banfield and B. Wilkerson, “Increasing Student Intrinsic Motivation and Self-Efficacy Through Gamification Pedagogy,” Contemporary Issues in Education Research (CIER), vol. 7, no. 4, pp. 291-298, Sept. 2014.
2. A tutorial/resource on the algorithm: “Dijkstra’s Algorithm in Data Structures,” and the use of Node.js and React frameworks.
3. R. Smiderle, S. J. Rigo, L. B. Marques, J. A. P. de Miranda Coelho & P. A. Jaques, “The impact of gamification on students’ learning, engagement and behaviour based on their personality traits,” Smart Learning Environments, vol. 7, Art. no. 3, 2020.
4. Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1, 269–271.
5. The Impact of Using Gamification in Learning Computer Science for Students in University by Risna Oktaviati & Amril Jaharadak (2018). International Journal of Engineering and Technology, 7(4.11), 121-125.
6. Gamification in Computer Science Courses: A Literature Review by Fezile Özdamli & Dlgash Faran Yazdeen (2021). Near East University Online Journal of Education, 4(2), 90-106.