Space Foundation System: An Approach to Spatial Problems in Games

Daniel Dyrda
Technical University of Munich
Munich, Germany
daniel.dyrda@tum.de

Claudio Belloni
Technical University of Munich
Munich, Germany
claudio.belloni@tum.de

***Abstract*—Current implementation approaches to spatial problems in digital games are not optimal. We identify a significant mismatch between the problem domain and the implementation domain. In this paper, we present the *Space Foundation System*, a technology for game engines based on a graph data structure focused on location-based semantics. We present a high-level algorithm and implementation approach for deriving a graph by subdividing the game's space based on integrity. The vision and possible applications of the framework are explored through the use of a gameplay scenario. The presented approach increases the abstraction level of the implementation domain and allows for the implementation of solutions for spatial problems closer to the problem domain.**

***Index Terms*—Game Engineering, Game Design, Design Engineering, Space, Game Space.**

I. Introduction

According to Salen and Zimmerman [1], a game's space is a fundamental element of the experience of digital games. Similarly to Shell [2], they describe space as a function of *representation* ("how the space is displayed to the player" [1, p. 394]) and *interaction* ("how a player navigates through the space" [1, p. 394].) Space is an essential aspect of games; it is therefore not surprising that many features commonly found in games are fundamentally spatial.

A. Scenario

To understand spatial features and the underlying problem, we present a scenario from a generic, fictional action-adventure game which we reference throughout the text. The scenario is illustrated in Figure 1.

A group of adventurers embarks on a quest: their journey begins with the discovery of a letter, cryptically hinting at the Tower of Ars—a mysterious structure famed for its treasures yet unknown location. Unfamiliar with the tower's whereabouts, the party ventures towards a nearby hamlet, seeking guidance.

Upon their arrival, the atmosphere shifts; tranquil music envelops them as weapons are sheathed, signaling safety. They decide to consult the village elder, who directs them westward, beyond the great tree, through an enchanted forest, where the tower lies in wait. From there, they'll be able to see the tower.

As the adventurers enter the woods, the forest's monstrous guardians reveal themselves. One of the player's companions proposes a dilemma–whether to sneak past the enemies unnoticed or confront them head-on. Emerging victorious from the forest's depths, the adventurers are greeted by the sight of the tower, majestically perched atop a hill.

This visual beacon guides them closer until they stand mere steps away from its entrance. Here the game's atmosphere shifts: the environment darkens under a stormy sky and thunder can be heard in the distance. The tower's name is displayed on screen—Tower of Ars.

But before they can enter, the tower's locked entrance must first be opened. The same mystical creatures from the forest roam nearby - perhaps one of them may drop the key. Through combat, the adventurers secure the key and enter the tower. The final challenge awaits at its summit.

Once they reach the top, the adventurers slowly move into a large room. As they cross the entrance, they hear the roar of a dragon, landing in the room a moment later with a thunderous crash. The player's sword becomes enveloped in flame, signifying newfound abilities and setting the stage for an epic confrontation within one of the game's magical chambers: the red dragon's lair.

B. Spatial Features

Many game features are directly related to location and spatial configuration.

1) Location Intrinsics: Developers often need to specify location-specific characteristics, including aesthetic elements like music, ambient sounds, and post-processing effects, as well as functional aspects such as item placement and environmental factors. In our scenario, the aesthetics change upon arrival at the tower. The creatures' loot table is adapted depending on their location within the game, with the creatures in front of the tower having a chance to drop the tower key.

2) Story: Narrative progression in games is deeply tied to the traversal of space [1], [3]. Narrative events like cutscenes and dialogues are often triggered by the player reaching specific locations, while the question of temporal distance is left open due to the inherent interactivity of the medium. In the scenario, the villager introduces the quest upon the player's arrival in the hamlet, the companion introduces gameplay opportunities after spending some time in the forest, and the dragon's animation is played upon reaching the top of the tower. Besides the distribution of narrative events in

979-8-3503-5067-8/24/$31.00 ©2024 IEEE

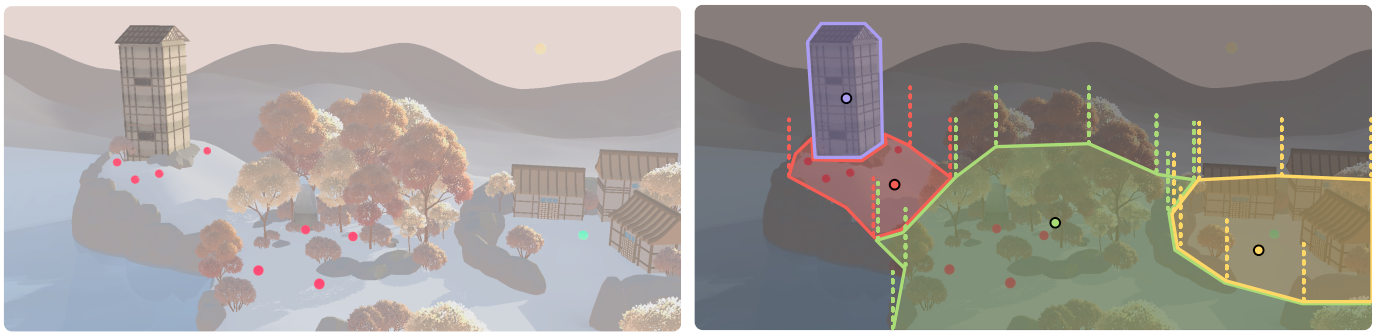

Fig. 1. The scenario's environment (enemies as red spheres). The right side shows the environment as subdivided by integrity of the space.

space, their content also often depends on spatial qualities. Quests often require navigating to objectives defined by spatial landmarks. The villager introduces the quest and explains the path to the tower by referencing the landmarks found along the way.

3) Rules: Many rules in games are specified in relation to space. Certain mechanics, such as saving the game progress, initiating combat, or using certain skills, may be restricted to specific locations. In our scenario, combat cannot be initiated in the village's safe zone, and special abilities can be used in designated areas like boss arenas. Rules are often defined regarding space. Boss behavior, like initiating the fight or resetting aggro, depends on the player's location. Entering new locations can trigger in-game effects, such as experience gains, achievement unlocks or displaying the location's name.

C. Problems

This paper addresses the central issue of how these features are implemented in modern game engines. It is primarily concerned with how game engines handle information about the location of agents in the game's space. All of the aforementioned spatial features rely on a more abstract concept than exact positions which we will call *location*. Their problem domain revolves around determining the location an agent is in, detecting location changes, and being able to understand routes between two locations in the game's space.

Currently, a naïve solution to derive location-related information from a position using common game engine features often involves using the engine's physics system to approximate locations and detect location changes of agents. Developers might place a set of primitive shapes as trigger collider volumes to approximate the location of interest and execute specific behavior upon collisions with the player or other agents. This process takes time and can introduce errors if not created or updated carefully. The process results from game engines providing features that primarily reside in the implementation domain (usually concerning the game's fundamental handling of space, such as coordinate space, vectors, and quaternions), while the problems we attempt to address require a higher level of abstraction.

It should be noted that game engines also address this issue by taking similar, though more limited, steps. They include built-in solutions like Unity's *LookAt()* function, which provides an interface to manage the rotation and spatial configuration of two objects without delving into the complexities of position vectors, Euler angles, or quaternions.

D. Goal

The goal of this paper is to present a technology for game engines that addresses spatial problems. This technology aims to reduce implementation complexity and to better align solution approaches with the domain of spatial features.

II. Vision

We propose a game engine subsystem dedicated to solving spatial problems: the *Space Foundation System* (SFS), an independent component of the *Gameplay Foundation* (see model in [4, p.39]). The SFS enables every object in the game to determine its current location and detect location changes in realtime during runtime.

A. Location

Game rules are often directly linked to location. By *location*, we refer to the term commonly used in everyday language, indicating where something exists or where an event occurs as perceived by humans according to its intended use or inherent characteristics. In this context, we define a location as a specified subset of elements within the game's space that possesses the quality of *insideness*, meaning it can be perceived by the player as being *inside* it (e.g., *in the hamlet* or *in the dragon's chamber*). Thus, our concept of location is a discretization of space: a location is a distinct element enclosed by delimiting spatial features that partition the space into disjoint inside and outside subspaces.

It follows that the dimension of a location in an n-dimensional space also has n dimensions. As [4] notes, the most prevalent numbers of dimensions in digital games are two and three, typically represented in game engines through Cartesian coordinates. A three-dimensional location is defined as a volume, often depicted as a mesh, whereas a two-dimensional location is represented as an area, such as a polygon. Consequently, locations in one-dimensional spaces are described by line segments, and zero-dimensional spaces are defined using sets to represent collections of elements. Figure 2 illustrates this definition of a location.

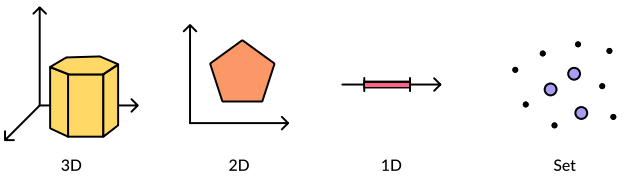

Fig. 2. A visualization of location in 3D, 2D, 1D, and as a set.

From the concept of insideness, we can also derive the relations of *IsNeighbor*, which we define as the potential for immediate change from being inside one entity to being inside another, and semantic paths between locations, which we define as a sequence of changes of insideness in the sense of *IsNeighbor*. This location-based perspective allows us to move from a purely geometric representation of the scene to a semantic specification of the game's space through the concept of location.

B. Transform

In game engines, we typically specify an object's position, rotation, and scale using a *Transform* component. According to the Unity Scripting API [5], every object in a scene has a *Transform* component, with the engine storing the position as a *Vector3*. This means that regardless of an object's dimensions, it can be associated with a position in space. Godot follows a similar approach with *Transform2D* and *Transform3D*, represented respectively by 2×3 and 3×3 matrices [6]. Our vision is to enhance the concept of transform components (or an analogous interface in other game engines) with the *location API*, ensuring that every object in a scene has a transform that grants access to its location. Consequently, each object can be linked to a specific location in space. Thus, the engine offers functionality to directly query the player's location through hypothetical calls such as *player.transform.location*, returning the location object with the location identifier. This property is designed to be reactive, allowing to subscribe to notifications about changes at runtime.

C. Features of the Space Foundation System

The SFS provides the following features via a static class:

1) ***IsInsideOf(transform, location)****:* Returns *true* if *transform* is in location *location*.

2) ***IsNeighbor(location1, location2)****:* Returns *true* if *location1* and *location2* are neighbors of each other. We define neighborhood as sharing a delimiter.

3) ***HasRelation(location1, location2, relation)****:* Returns *true* if there exists an edge *(location1, location2)* in *relation*.

4) ***GetShortestPath(location1, location2, optional relationMask)****:* Returns a path connecting *location1* with *location2*. The relations to be considered for the computation can be restricted via *relationMask*. The weight for the calculation can be defined within the SFS. The resulting sequence is not a concrete path, as a navigation mesh feature could retrieve it. Instead, it is a sequence of locations that provides information about the route via data associated with the locations.

5) ***Subscription****:* Allows subscribing to a location to be notified when an agent enters or leaves the location at runtime.

Further features can be derived from this modest set via extensions of the SFS or custom systems, such as linking data to locations and querying data via location identifiers.

D. Challenges

We identify the following core challenges to reach our goal:

- C1 **Space Topology Model:** Creating a framework for the modeling of locations and relations.
- C2 **Identification of Locations:** Subdividing space by the integrity of space.
- C3 **Identification of Relations:** Determining relations within the resulting discrete set of locations.
- C4 **Location System:** Providing a general game engine component for working with the resulting data structure.

III. Related Work

We are unaware of a generalized interface for game engines that implements the concept of location based on the integrity of space. We dedicate this section to research that provide meaningful approaches to abstracting a game's space to locations.

A. Level Design

Existing research on level design provides many concepts applicable to their design, understanding, and, albeit indirectly, abstraction of space.

Totten [7] proposes various categorizations of space based on their characteristics. While this does not present a solution, these categories can be useful for the abstraction of space.

In the article on *Molecule Design*, Azar [8] advocates for a topological approach to level design by proposing a graph-based approach to the design of games' spaces which emphasizes their topological features. While our work shares this vision, molecule design is, as the name indicates, a *design* tool and needs a more rigorous underlying structure to support a formal framework, especially concerning the representation of spaces through graphs.

In their thesis, Kelder [9] aims towards formalized (procedural) game space creation with an emphasis on game design patterns. To achieve this goal, they present a data structure for representing spaces called the *functional game space model*. The author describes it as "a model for architectural game-space in space driven-games to enable reasoning on an analytical design methodology" [9, p. 100]. While this model satisfies our need for a formal representation of space, it lacks the potential to meaningfully address problems outside of the domain of procedural content generation.

B. Game Analysis

Many approaches from game research emphasize the *analysis* of spaces' ludic qualities. Maram et al. [10] investigate the measurement of player experience through player strategies and decision patterns based on spatial abstraction. They describe a spatial abstraction process based on domain knowledge and provide two examples. However, the paper lacks a

comprehensive procedure for implementing this abstraction in practice, particularly within a game engine.

C. Scene Graph

Another approach currently present in most game engines is the scene graph, a hierarchical data structure containing all elements in the scene, and thus a potential candidate for spatial solutions. While its merits should not be understated, the permanent conflict between its role in the render pipeline and its role as a more design-focused tool for representing a game's space makes the need for a dedicated framework more apparent [11], [12]. We argue that this conflict is the result of scene graphs, as made available by most game engines, presenting themselves as "a combined structure for communicating, rendering, execution, and modeling" [11, p. 57], or at least not stating otherwise. Currently, naïve approaches to the scene graph that focus exclusively on modeling and spatial arrangements can affect the game's performance due to the scene graph's role in the rendering pipeline. With this in mind, developing a data structure dedicated to design and implementation independently of execution is reasonable.

D. Architecture & Urban Planning

Attempts to formalize and discretize natural space are not novel in architecture and urban planning.

The development of patterns in the highly influential *A Pattern Language* [13] has undoubtedly inspired the same approach to level design that is found, among many others, in [14], [15] and [16].

Lynch [17] focuses on cities' *legibility* and proposes that individuals develop mental maps of urban areas, which consist of five spatial elements: paths, edges, districts, nodes, and landmarks. Further pursuits of these concepts reach, among many other fields, into level design (one example is [7].)

Spatial signatures are a recent attempt at tessellating (effectively discretizing) urban spaces that makes use of "two perspectives of how space can be understood and organized" [18, p.5]. These perspectives focus respectively on *delimiters*, which "delimit the landscape and partition it into smaller, fully enclosed portions" and *anchors*, "a discrete set of relevant features" [18, p.5].

Space syntax analysis deals with a formal methodology for analyzing (real) spaces through spatial abstractions. Graphs are used to represent and analyze space analogously to the goal of this paper. The discretization of a space into a set of nodes occurs by subdividing any given concave space into the least possible number of convex sub-spaces. The graph's edge relation represents accessibility between nodes [19], [20].

IV. Solution Approach

While the concepts discussed above provide valuable contributions towards a solution, the space signatures suggested by [18] stand out for their potential. The comprehensive space tessellation outlined in their work offers a promising method for formalizing game spaces through partitioning. Our proposed solution synthesizes various elements from Section III, positioning the space signatures from [18] as the central point of reference for our approach. To address the challenges identified in our vision, we start with a general overview and propose a graph-based data structure to model the topology of space (C1). After this, we introduce the general concept of identifying locations by subdividing space via discretization based on the integrity of space (C2) and identify the spatial relations between the resulting locations (e.g., *IsNeighbor*) (C3). Finally, we present the concrete algorithm for our approach. Lastly, we show how this can be incorporated into a game engine component that provides functionality based on the resulting data structure (C4). For the rest of the paper, we will mainly focus on continuous three-dimensional space as commonly found in modern games. Solutions for 2D, 1D, and 0D can be applied accordingly.

V. The Location Graph

The cornerstone of our proposed methodology is the conceptualization of a *location graph*, which serves as the foundational data structure for encoding spatial relationships. This graph, designed for both human and machine readability, facilitates the serialization and visualization of spatial topology, underpinning subsequent algorithmic applications.

Nodes within this graph correspond to a comprehensive partitioning of the game's space, achieved through a process informed by the principle of *spatial integrity*, which will be elaborated upon in the following sections. Each spatial segment is interpreted as a location and encapsulated as a graph node. Semantic data about the location are attributed accordingly. Spatial connections between nodes are established based on their proximal and relational adjacency, offering a structured representation of the game space's topology.

Formally, the graph is defined by a set of subspace elements S and a collection of binary relationships R_i, with a foundational minimal relationship R_0 signifying direct neighborhood through a shared delimiter. Additionally, application-specific relationships may be introduced to address the unique requirements of the game's mechanics and rules. The formal structure is designed in a way that allows future enhancements, such as hierarchical abstractions within the graph, to support more complex spatial interpretations.

VI. Discretizing Space With Delimiters & Anchors

Our approach relies on the discretization of continuous space into individual portions based on the integrity of space. The concept of wholeness in spatial perception, known as integrity of space or *genius loci*, denotes the perceived completeness of a space relative to its surroundings, as highlighted by [7]. This notion is fundamentally interpretative, suggesting that spaces are evaluated for their holistic integration within a larger spatial context. To mitigate the subjective variability inherent to this interpretation, one might consider quantifying integrity based on degrees of spatial closure. Such a quality is pivotal to our earlier discussions on location, aiming to ensure that each distinct *whole* space is accurately represented by a dedicated node within the resultant graph structure.

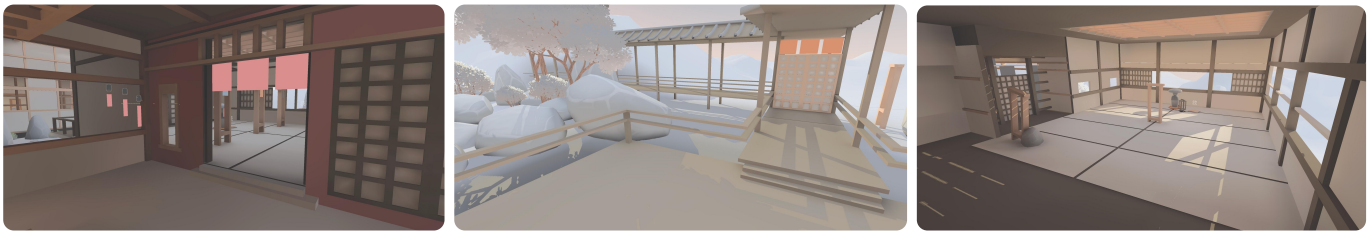

Fig. 3. Examples of different delimiters, such as fences, stairs, floor tiles and walls.

According to the insights from III and mainly [18], we can derive a space's topology based on closure by identifying elements that separate or aggregate a given space and using them to derive a space's integrity. The extensive use of architectural concepts in level design suggests that this assumption is valid in the context of games' spaces and how players perceive them. To reduce subjectivity as much as possible, we propose a detailed algorithm to determine wholeness concerning the organization of a space, based on the two fundamental spatial elements used in [18]: *delimiters* and *anchors*. In contrast to their application in [18], we aim to apply both concepts simultaneously to derive one meaningful partition of the space, which will form the node set of the location graph.

A. Delimiters

1) Delimiting Features: The concept of *delimiter* is based on delimiting features "...conceptualised as a line that acts as a boundary, dividing space into what falls within each of its sides" [18, p. 5]. This concept is closely related to *edges*, as described by Lynch [17]. In the context of urban spaces, delimiters "...include the road and street networks, but also others such as railways or rivers" [18, p. 5]. In contrast, in (digital) architecture, delimiters usually take the form of vertical planes or other spatial features that break spatial or visual continuity. Ching [21] describes the role of vertical forms as such: "Vertical forms have a greater presence in our visual field than horizontal planes, and are therefore more instrumental in defining a discrete volume of space and providing a sense of enclosure and privacy for those within it. In addition, they separate one space from another and establish a common boundary between the interior and exterior environments" [21, p.134]. This brief characterization is easily brought into the context of a game's spaces, which heavily rely on (visible or invisible) vertical planes meant to restrict player movement and organize their spatial perception.

2) Specifying Delimiters: Regarding concrete in-engine implementations, delimiters are polygons or segments extending in a vertical direction. One possible approach is the implementation of delimiters as typical game objects that may be used to freely specify a delimiter's location in the editor by insertion into the scene. As an object, it may be specified as a child of an existing object in the context of the scene graph to keep it attached to the parent's transform when rearranging the scene. Otherwise, the two extremities may be attached to existing geometry independently for additional flexibility. In addition to this explicit procedure, an implicit approach is also possible. Since borders are primarily conceived as insurmountable limits, they inevitably correlate to the presence of a mesh or collider component. This enables the engine to automatically detect and analyze these delimiting elements.

3) Delimiter Data: In addition to their presence, delimiters also carry data that identifies their delimiting characteristics, essentially detailing the nature of the relationship between adjacent spaces. This information specifies how spaces on either side of a delimiter are interconnected. Independently of any supplementary data, a delimiter inherently establishes a fundamental relationship of adjacency, or minimal relation, between the two subspaces it separates. The potential for additional, more complex relationships is significant. For instance, these relationships may indicate the feasibility of traversal between spaces for players or agents. A wall, acting as a delimiter, precludes passage, whereas a change in terrain texture from grass to rock permits movement. Similarly, a transition from shore to water may allow traversal under specific conditions, such as the agent's capability to swim or dive. Examples are shown in Figure 3.

B. Anchors

1) Anchor Concept: The concept of *anchors* describes "...elements that do not partition space per se, but act as origins to which the rest can be attached" [18, p.5]. Anchors are complementary to delimiters and act as central elements within a space. Similarly to the concept of districts and landmarks presented by Lynch [17], they represent the spatial elements that evoke a sense of integrity in their (immediate) surroundings. They divide space between the part that belongs to the anchor and the space beyond it. It follows that anchors possess a *subspace of influence* based on the size of the space they dominate. This principle follows the human tendency to organize space around landmarks and other reference objects.

2) Specifying Anchors: Anchors can be formally defined as points representing the center of the anchor elements and a maximal distance that models their subspace of influence. As such, anchors are implemented as components attached to objects in the context of a transform. The presence of the component marks a feature as an anchor. Further details of the anchor element are specified on this component. Anchor points are specified by placing reference points in the scene or identifying spatial features as anchors. In practice, elements that have meaning to the player are good candidates for

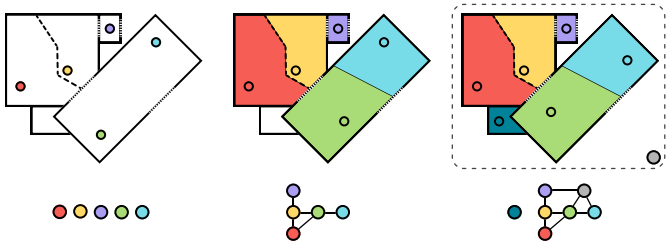

Fig. 4. A 2D example illustrating the steps of the proposed algorithm. The upper graphics show a 2D environment with anchors displayed as colored circles with their corresponding subspace colored accordingly, delimiters displayed as dotted lines (elements such as differences in floor textures), and solid lines (for non-traversable elements like walls.) The lower graphics show a set or graph representing the location graph.

anchors. Objects with a function, such as specific components attached to a game object, could be identified as anchors automatically. In principle, anchor elements don't need to be points themselves and may instead be meshes, volumes, or any other kind of structure as long as it fulfills the semantic role of an anchor. For now, we will focus on anchors represented as points to avoid edge cases of overlapping anchors, or anchors and delimiters. In these cases, we can use the common transform component as the anchor's central point.

C. Location Graph Creation

The following describes the procedure for deriving a location graph from a space using delimiters and anchors. The procedure is illustrated in Figure 4. An example in the context of our scenario is illustrated in Figure 1.

Anchors are represented by nodes. Afterward, the volume of each node is expanded until it encounters delimiters. The algorithm comprises the following steps:

1) Preparation: The specification of all delimiters and anchors, as presented in the latter half of VI-A and VI-B, corresponds to the preparation step, as a complete list thereof is required by the following algorithm.

2) Anchor Step: Each anchor in the scene is added as a node to the location graph. The origin of this anchor is stored for further processing.

3) Expansion Step: Anchor points propagate their influence within their immediate surroundings through a uniform flooding process, defining their respective subspaces of influence. This follows the methodologies employed in forest fire spread or flood fill algorithms. The propagation is constrained by encountered delimiters, the maximal distance to the anchor's origin, or the influence zones of other anchors. Encountering a delimiter halts the spread, leading to the storage of the anchor's identifier in a list of the delimiter or a general list in case the halt was due to an overlap with another anchor's influence. This mechanism ensures that, by the conclusion of this phase, every spatial point within the radius of an anchor is attributed to the nearest anchor, respecting delimiter boundaries. The ultimate goal is to assign each space segment to a unique anchor, facilitating clear and unambiguous spatial partitioning. The resulting volumes are processed into a format representing the subspace, such as a mesh in the case of a 3D space. They are stored with a link to the space identifier to allow further access.

4) Resolving Underspecification: The initial expansion step may not guarantee complete space assignment due to the potential presence of spatial points not assigned to any anchor, leading to underspecification. This issue is particularly evident in spaces fully enclosed by delimiters without an internal anchor (see VI-C-b) or spaces not within the influence radius of any anchor. To address this challenge, we differentiate between two cases: fully enclosed spaces and the rest. For fully enclosed spaces, we introduce a *nameless identifier* for each underspecified space, ensuring its inclusion in the spatial model as a distinct node with the associated subspace. This approach effectively resolves the partitioning dilemma by adequately representing enclosed, anchor-less spaces. For the second case, any space not encompassed by delimited boundaries is aggregated under a universal *void* node. This categorization serves a dual purpose: it guarantees that the API returns a valid location for any query, thereby formally eliminating underspecification concerns. In the future, the machine will be able to derive the semantics of underspecified spaces based on their surroundings, avoiding the need for nameless identifiers in favor of automatically generated, specified identifiers.

5) Optimization: The results of the subspaces are stored for further access. To allow for fast access when queried for insideness (point-volume intersection calculations for 3D spaces), we create an optimized search structure for the collection of subspace data. This is especially important for 2D and 3D. Any approach used for physics optimization, such as k-d trees or quadtrees, is conceivable (compare [4]).

6) Spatial Relations: At this point, we have computed a partition of the game's space, which acts as a node set for the location graph. We now focus on defining the spatial relations that interconnect these partitions, thus forming the graph's edge set. This involves a detailed examination of the delimiters that define each partition, a process achieved by iterating through delimiter-associated node lists, including a general list for non-specific relations. The analysis aims to identify which partitions are adjacent, based on shared delimiters, and to assign appropriate relational attributes to the edges of the location graph. A foundational relation, R_0, denotes simple adjacency between nodes, defined by a shared delimiter or shared influence border. Beyond this, the system incorporates additional sets of relations corresponding to specific spatial interactions allowed by the delimiters. For instance, a wall implies only an adjacency relation (R_0), whereas a door that can be traversed introduces both adjacency and a traversable relation, populating both R_0 and $R_{Traversable}$ in the graph.

D. The Space Foundation System API

As described in our vision, we propose a static class SFS implementing the features presented in II-C and extend the established transform component of game engines with a location variable as introduced in II-B. Internally, the SFS

maps positions to the corresponding location identifier via point-subspace intersection tests with all subspaces. It allows computations on the location graph, such as finding the shortest path between two locations. Here, common graph-oriented approaches and algorithms such as *A** are conceivable and up to the concrete implementation.

VII. The Space Foundation System

A. The System in Practice

We can now address the space-related problems presented in our scenario I-A within the context of our proposed system.

1) Aesthetics: In the case of music and ambient tracks playing upon entering a specific location, audio files can be assigned to a location, specifying which audio track is played as long as the player remains in this space. When a player enters a location that specifies a track, the current track is interrupted in favor of the new one. The same approach can be taken to many other features. A system subscribes to the location changes of the player and derives data corresponding to the current location. In the case of our scenario, the locations of the village and the tower can be endued with the respective audio files or post-processing specifications.

2) Game Mechanics and Rules: Spatial rules can be specified and implemented using the proposed system. In the case of our scenario, we can specify that the dragon should attack the player upon entering the location directly with *OnEnter(LocationIdentifier)* instead of using collider volumes as triggers. Special effects like the thunderstorm or narrative events like a party dialogue can all be implemented analogously. The same holds for specifying what items can be found in a location. Similarly to the previous section, restrictions or affordances can be specified directly for a location instead of updating parameters upon collision with a trigger volume. For instance, the boss arena can be specified to allow fire magic, while combat mechanics can be restricted in all villages. Concerning respawn mechanics, we see potential in the specification of stable points per location or groups thereof. Respawning after a game over in a location *currentLocation* would then automatically occur in a prespecified point *currentLocation.stableRespawnPoint*. Taking this concept further, such a point in space could also be computed with location-based semantics: it is possible to compute whether that point's location is visible to enemies, already occupied by allies or other conditions.

3) Wayfinding and Quests: In our scenario, a naïve implementation would include a non-player character (NPC) presenting a route to the player, which was manually created to match the route from the NPC's location to the goal. In our scenario, this is the role of the village elder. Our solution makes a different approach feasible: it is possible for NPCs to be aware of locations and, therefore, to query the semantic path (i.e., the sequence of locations) needed to go from one location to another. Such a system allows for the dynamic creation of directions, which can be embedded in the dialogue system and then used by an arbitrary number of NPCs in any location. Since the system allows the specification of location-specific data, restraints can also be put on the query to match the intended experience. For instance, we may specify whether a location is known to the speaker and allow it to be included in the dialogue. Each location's difficulty, geographical data, enemy presence, and more can be included to adapt the system's depth to the user's needs.

4) Concluding the System in Practice: We conclude that common spatial problems, such as the selection presented in the scenario, can be meaningfully addressed with the presented framework. The proposed features largely eliminate the need for developers to implement suboptimal position-based approximations to determine an entity's location. These improvements result from using more abstract semantics that are closer to the problem domain than current practice.

B. Discussion

1) Workload: A central issue is the additional work required to create and maintain the system. It should be noted that this paper does not propose any concrete implementation beyond high-level approaches, making it impossible to accurately estimate the workload for using this system. Nevertheless, we have outlined opportunities for integrating these approaches into existing features to minimize the workload. In the future, we envision a system that is automatically derived from the scene's geometry and functional components, which would further alleviate this problem.

2) Complexity & Performance: A significant issue is that the current system and its implementation do not offer any control over the granularity of the partitioning. It is evident that spaces of modest complexity can result in enormously large node sets, potentially affecting the system's usability. In our earlier scenario set in an open-world game, the entire quest takes place in the same macro space. It is easy to imagine how the scenario's tower would already require a substantial number of nodes if we assume that each room constitutes its own unit. This complexity is further exacerbated when connecting these nodes within the spatial context of the open map. Such complexity raises concerns about performance in terms of both memory and computation, which will need to be addressed in future work.

3) Abstraction: Currently, the system lacks the means to group nodes meaningfully to allow for abstraction. In Section VII-A, we discussed how the aesthetic features of the tower and its immediate surroundings could be specified directly to the respective location. At present, this would mean specifying this information for every node belonging to the tower, which is less than optimal. We argue that supporting hierarchy within the final result should be a future priority. The integrity of space is a hierarchical quality and depends on the granularity with which a space is viewed. The existence of districts, as proposed by Lynch [17], and their established role in level design (as seen in [7]), supports this thesis. For this reason, we propose pursuing a solution approach based on higraphs, as presented in [22]. This point is crucial for the result's usability. The data structure should be usable for both humans

and machines, making its visualization a central feature. The inclusion of hierarchy would improve this aspect similarly to [22].

4) Subjectivity: Since different individuals may have different interpretations of the same spatial elements, the precise specification of delimiters and anchors can be influenced by subjectivity. We attempt to mitigate this through detailed procedures and by keeping human-guided elements at a minimum.

C. Future Work

Our forthcoming efforts will be detailed in an upcoming paper, where we will introduce an initial prototype to demonstrate our methodology's applicability. While this prototype represents an essential step towards validating our approach, extensive development is required to achieve a robust, production-ready implementation. The geometric complexity of modern game spaces makes the procedure, as presented in the expansion step (see Section VI-C3), susceptible to numerous edge cases that future implementations must address. Moving forward, we will prioritize usability, performance, and seamless integration within game engines. Additionally, exploring new features remains an open avenue for further enhancement.

1) Hierarchy: Future efforts should integrate hierarchical structures within the framework to facilitate abstraction, aiming to enhance system usability without compromising intuitiveness. This solution has to set solid, formal axioms for the relations between nodes from different levels as well as the repercussions on the spatial relations present in the graph.

2) Visualization: There's a clear need for an effective visualization to depict subdivisions within the game scene. This is particularly challenging for three-dimensional spaces.

3) Novel Spatial Features: Exploring location-based semantics in game development opens the door to innovative features and broader impacts, particularly in accessibility, enhancing user interfaces with features such as location readers that provide information about the environment or systems that dynamically adjust the difficulty of a location. It also heralds advancements in game AI, making game agents location-aware for improved navigation, strategy, and interaction. Additionally, this approach promises benefits in optimization, analysis, tutorial creation, navigation, and manipulation, especially in immersive virtual reality (VR) experiences.

4) Secondary Scene Graph: Our proposed data structure emerges as a viable complement to traditional scene graphs used in most game engines, addressing the need for a specialized framework that tackles design and implementation challenges from a more conceptual standpoint. As discussed in Section III, moving beyond the rendering-centric role of scene graphs allows for closer alignment with the design and implementation of spatial problems. Our proposed framework satisfies most conditions: it achieves a level of abstraction significantly closer to the problem domain of spatial issues. Unlike scene graphs, whose modeling capabilities result mainly from grouping vertices (see [11]), the graph we propose represents the game's space on a more abstract, topological level, focusing on the semantic interpretation of the geometry rather than single objects and their positions.

VIII. Conclusion

This paper presents the *Space Foundation System*, a technology for game engines aimed at addressing prevalent spatial challenges in digital games. We explored its fundamental principles and a specific algorithm that underlies our solution, along with proposed strategies for its integration into modern game engines.

References

[1] K. Salen and E. Zimmerman, *Rules of play: Game design fundamentals*. MIT Press, 2004.

[2] J. Schell, *The art of game design: A book of lenses*, 3rd ed. CRC Press, 2019.

[3] N. Schrape, "The rhetoric of game space," in *Ludotopia, Spaces, Places and Territories in Computer Games*, E. Aarseth and S. Günzel, Eds. transcript Verlag, 2019.

[4] J. Gregory, *Game engine architecture*, 3rd ed. CRC Press, 2019.

[5] Unity Technologies, "Unity scripting api: Transform," 2024. [Online]. Available: https://docs.unity3d.com/ScriptReference/Transform.html

[6] Godot Engine Contributors, "Godot engine documentation: Transform 2d," 2024. [Online]. Available: https://docs.godotengine.org/en/stable/classes/class_transform2d.html

[7] C. W. Totten, *An architectural approach to level design*, 2nd ed. CRC Press, 2019.

[8] N. Azar, "The metrics of space: Molecule design," 2013. [Online]. Available: https://www.gamedeveloper.com/design/the-metrics-of-space-molecule-design

[9] R. O. D. Kelder, "Towards a framework for analytical game space design," Master's thesis, Universiteit Utrecht, Faculty of Computer Science, n.d.

[10] S. S. Maram, J. Pfau, J. Villareale, Z. Teng, J. Zhu, and M. S. El-Nasr, "Mining player behavior patterns from domain-based spatial abstraction in games," in *2023 IEEE Conference on Games (CoG)*. IEEE, 2023, pp. 1–8.

[11] H. Sowizral, "Scene graphs in the new millennium," in *IEEE Computer Graphics and Applications*, vol. 20, no. 1. IEEE, 2000, pp. 56–57.

[12] T. C. S. Cheah and K.-W. Ng, "A practical implementation of a 3d game engine," in *International Conference on Computer Graphics, Imaging and Visualization (CGIV'05)*, 2005, pp. 351–358.

[13] C. Alexander, S. Ishikawa, and M. Silverstein, *A pattern language: Towns, buildings, construction*. Oxford University Press, 1977.

[14] S. Björk and J. Holopainen, "Games and design patterns," in *The Game Design Reader, A Rules of Play Anthology*, K. Salen and E. Zimmerman, Eds. MIT Press, 2006.

[15] B. Kreimeier, "The case for game design patterns," 2002. [Online]. Available: https://www.gamedeveloper.com/design/the-case-for-game-design-patterns

[16] K. Hullett and J. Whitehead, "Design patterns in fps levels," in *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. Association for Computing Machinery, 2010, pp. 78–85.

[17] K. Lynch, *The image of the city*. MIT Press, 1964.

[18] D. Arribas-Bel and M. Fleischmann, "Spatial signatures - understanding (urban) spaces through form and function," in *Habitat International*, vol. 128, 102641, 2022.

[19] W. Dettlaff, "Space syntax analysis–methodology of understanding the space," in *PhD Interdisciplinary Journal*, vol. 1, 2014, pp. 283–291.

[20] S. Haq, "Where we walk is what we see: Foundational concepts and analytical techniques of space syntax," in *HERD: Health Environments Research & Design Journal*, vol. 12, no. 1, 2019, pp. 11–25.

[21] F. D. K. Ching, *Architecture: Form, space, and order*, 4th ed. John Wiley & Sons, 2015.

[22] D. Harel, "On visual formalisms," in *Communications of the ACM*, vol. 31, no. 5. ACM New York, 1988, pp. 514–530.