Designing with Player Paths: Presenting a Game Design Framework

Martin Schacherbauer
Technical University of Munich
Munich, Germany
martin.schacherbauer@tum.de

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

***Abstract*—Player paths are a central design paradigm game designers use for video games. However, keeping track of all designed player paths during development is difficult. This paper proposes a framework that supports developers who use player paths as a central design paradigm. To do so, we look at the definitions of common terms regarding player paths and analyze current workflows used in the video game industry by analyzing presentations at game development conferences. By doing so, we can ensure that our proposed framework uses knowledge from both practitioners and scholars to bridge the gap between them. The framework is based on the usage of graphs and is entirely independent of the functionality of the graph in the video game, ensuring maximal flexibility for the developers. The framework uses features available in game engines, making it possible to use in any development environment. Last, we also demonstrate the tool's functionality via a prototype of the framework in Unity.**

***Index Terms*—Editor tool, framework, game design, game development, game engineering, player paths, space, video games**

I. Introduction

One of the main differences between traditional media—like movies and music—and video games stems from the interaction between the medium and the user [1]. A movie watcher can not influence the movie's ending. The linearity allows the movie creator to directly influence the movie watcher's experience. With video games, the player's actions are vital to how the game proceeds and the player's experience. If a player does not give any input when playing a game, the game might not progress or end with the player losing. The player's experience shaped by the game and the player's past experiences [2] influences these actions.

The non-linearity of games is represented as a space of possibility for the player [3]. This possibility space is what game designers can design through rules and structures [4], and the player performs actions and takes a path through it. However, it is the goal of game designers to create an experience for the player—in this relationship, the game is just a secondary artifact [3]. This problem is known as the second-order design problem [4]. As the possibility space allows for multiple outcomes depending on the player's input on a given game state [5], it is even more important for game designers to focus on the experience they want to create. Following Totten's perspective that rewards and narratives are spatially embedded [3], we argue that choice is spatially embedded in the game's environment. We also see that many video games support different paths for players to approach the same goal—the game's possibility space reflects those paths. For instance, if the objective is to retrieve an item from a guarded spot, the player might be able to take the direct route protected by enemies, resulting in combat. In contrast, another player might try to find hidden paths and reach the item undetected.

Based on the interaction from the player, game developers need an overview of their planned player paths and how they support them in the game itself. This paper aims to propose a framework that directly supports a design philosophy around planned paths. We want to contribute to the field of game design by showing what tools and methods the game development industry uses and propose how game developers can use game engines' features to design with player paths as a central design philosophy.

To do this, we

- investigate the meaning of *choice*, *possibility space*, and *player paths* in game development,
- give an overview of which methods and tools the video games industry currently uses, and
- propose how designers can use graphs and player paths in additional ways during the development and design process.

We chose these points to investigate and understand the tools currently used and allow for a comparison between them. Furthermore, we can draw informed conclusions on why game developers chose them in the first place which will allow us to propose extensions of existing methods to enable a more player path-centric design philosophy. The paper focuses on literature work on level design, choice in video games, and different player playstyles. We use talks from level designers in the industry as they give a more detailed insight into the processes in game development companies. By incorporating perspectives from academia and industry in our proposed framework, we want to bridge the gap between academia and industry, as both sides could benefit from each other [6].

II. Terminology

Before analyzing ways of developing video games with player paths in mind, looking at terms relevant to the topic is necessary.

This is the accepted author manuscript of the following publication: M. Schacherbauer and D. Dyrda, "Designing with Player Paths: Presenting a Game Design Framework," IEEE Conference on Games, 2025. DOI: doi.org/10.1109/CoG64752.2025.11114252. The published version is available at IEEE Xplore: ieeexplore.ieee.org/document/11114252.

A. Choice

Salen and Zimmerman [4] introduce two levels of choices: *micro* and *macro*. They describe the difference in the following way: "The micro level represents the small, moment-to-moment choices a player is confronted with during a game. The macro level of choice represents the way these micro-choices join together like a chain to form a larger trajectory of experience" [4, p. 61]. It is interesting to note that they use experience in connection to the choices on the macro level. Furthermore, they also introduce the term meaningful play, where they mention an additional property of choice and play—meaningfulness [4]. Considering choices on both the micro and macro level is not feasible for a game designer, as that would include all button presses, whenever the player picks something up, or when the player uses the settings menu. Instead, choices on the macro level, which are meaningful and intentionally designed, must be considered and planned around.

It also has to be noted that an action alone is not enough to determine whether the choice made by the player is meaningful or not—the context around the action also has to be considered. For example, suppose the player stands in front of a quest item and interacts with it. In that case, this action progresses their current mission, while interacting with a simple collectible on the ground might have a limited impact on that quest's progress. With this, we decide to focus on the macro-level choices to be considered for the rest of the paper, unless stated explicitly otherwise, due to the practicality and importance of those choices in the game.

B. Space of Possibility

Salen and Zimmerman [4] introduce the space of possibility after quoting Sudnow's *Pilgrim in the Microworld* [7]. They argue that Sudnow's description of his actions while watching TV gives an important insight for game developers: game developers have to look at the game from the player's perspective, not just by watching someone else play the game. This ties into their definition of the space of possibility. They argue that the space of possibility encompasses all actions a player can perform in the game system in the future. They conclude their definition of the space of possibility by stating that this space can not be designed directly by the game developer as game developers can design the meaning of actions but cannot predict the meaning the player draws from that action.

In a general sense, whenever a human acts, they do so after forming a goal—and forming a goal is making a choice [8]. Taking an action in a game inherently requires the player to have made a choice based on the current game state. Therefore, the space of possibility can also be described by and contains all the choices a player can make given any game state.

This paper adopts the first definition of the space of possibility based on actions. However, the following approaches will not be able to take all the actions of the player at any time into account because that is simply not feasible. Instead, we argue that while it is important to look at the complete space of possibility for designing video games, reducing the actions in the space of possibility to the actions that have a designed and meaningful outcome covers the most important cases that have to be looked at by game developers. This approach is suitable for visualization during development because it reduces the number of possible actions while considering those with a meaningful outcome in the game.

C. Player Paths

While the space of possibility describes all possible actions, a player path contains the actions the player took to progress in the game. While the possibility space is non-linear, the player path is the linear sequence of events a player takes. Often, these actions are embedded in the game's virtual world, as space of possibility also suggests.

One definition from level designer Tommy Norberg [9] revolves around the level designer's intention for a path. He calls the path intended by the game's designer to be taken by most players *main path*. To give the player agency, a level can include multiple *secondary paths*, allowing for multiple approaches to the same goal. He also mentions parts of paths, such as *desired paths* that players traverse to cut corners, *detours*, or *exploration paths* for players who want to explore the game's world. Considering the player's perception of paths, the *perceived path* is the path players think they should take. He embeds his model for paths in the game's space. All these paths need to be carefully designed by game designers.

Per our definition of player paths based on the player's actions and choices, they also reflect different playstyle preferences. It follows that player motivation and goal analysis can also be an approach to define player paths. A vast array of past studies puts players into different categories according to their goals and behaviors in video games with video game-specific distinct groups [10]–[12], which can also include aspects from the psychological field like BrainHex [13]. Another approach comes from Kahila *et al.* [14], who group players based on their metagaming activities, i.e., activities beyond what they do while playing the video game [4]. In a meta-synthesis by Hamari and Tuunanen [15], they conclude their analysis of 14 works on player types in video games by proposing five main player motivations—*achievement*, *exploration*, *sociability*, *domination*, and *immersion*.

In the scope of this paper, we refer to player paths as designed ways to complete an objective, similar to Norberg's definition [9]. We do not look at the micro level of a player path, which includes the concrete 2D or 3D movement through a level, but instead at the macro level, which includes designed elements from the game developer. We propose this more zoomed-out view due to feasibility during game development and the diminishing returns of inspecting player paths that only differ in their movement. Instead, our work gives game developers tools and methods to visualize how players can interact within the designed game space and objectives. Still, the motivations found by the aforementioned works should play an important role in designing player paths.

This is the accepted author manuscript of the following publication: M. Schacherbauer and D. Dyrda, "Designing with Player Paths: Presenting a Game Design Framework," IEEE Conference on Games, 2025. DOI: doi.org/10.1109/CoG64752.2025.11114252. The published version is available at IEEE Xplore: ieeexplore.ieee.org/document/11114252.

III. Problem Statement

Instead of designing levels with a specific type of player in mind, designers use the concept of choice [16]. As a result of those choices, which are game-specific and rely on the available actions to a player, the game designers can assign those actions to a specific playstyle, such as a stealth or confrontational approach, tying into the experience [16]. This is represented in the player paths. Designing a level becomes more complex, as levels might support multiple player paths, and game designers need to be aware of the relationships between different player paths instead of tracking one path. Keeping track of those player paths can become difficult for game developers and designers without the required knowledge and tools [17]. Therefore, this paper aims to make player paths a more attractive design paradigm by presenting solutions already used in the video game industry to this problem and proposing a framework that assists designers while planning a level to support multiple player paths embedded in the game's space simultaneously.

IV. Related Work

The previous sections provide an academic overview of the topic. To assist in bridging the gap between academia and industry [6], for this section, we analyzed various talks at video game conferences. We read interviews in media outlets focusing on video games to understand how game designers think and discuss spatial choice and player paths. We did this to gather information on recent developments in the video game industry. Awareness of recent developments is especially important because the video game industry changes rapidly, and our proposed framework should accommodate the current industry practice. We try to merge concepts from both parties to create an approach that both practitioners and researchers can use.

A. Path Visualization

In a talk at *devcom* 2023, level designer Pears [18] talks about a game he worked at while highlighting one concrete example of how he designed one game's level. He mentions a major challenge for level designers when supporting multiple player paths: leading all the players to the same objective. This is especially important because the objective is the point where the game's story progresses. Designers need tools that also show the points of key actions and choices of a level with an adequate symbol. During his presentation, he shows multiple pictures of the level, similar to Fig. 1, with paths drawn as lines and symbols for objectives. We cannot determine whether the game engine directly shows these lines and symbols during development or whether they are just added for the presentation. Tools showing planned paths for different playstyles might benefit the workflow as it allows designers to illustrate the planned paths intuitively [17]. Furthermore, actively marking points where paths converge or separate allows designers to design more consciously around such points to better grasp how different players progress in a level, even if they switch between playstyles.

B. Pacing Graphs

Level designer Ellis [19] describes one way to plan a level based on its gameplay beats and pacing. Gameplay beats refer to "the main events and activities that the player will undertake and how these are spaced out and timed in your level" [19, Ch. Gameplay Beats]. Pacing represents intensity throughout the game based on the player's planned experience. Interestingly, he mentions different playstyles and how they affect the pacing of a level. While he does not propose having different pacing graphs for different player paths, he instead talks about a way to keep a level's tempo similar independent of the player's actions if the designer wants it that way.

We can apply this concept to designing levels with multiple paths. This approach might be beneficial in terms of seeing how different types of players might experience the level. Different players' experiences can be mapped by creating versions of this timeline graph based on the players' actions during the level, as demonstrated by the tool PaceMaker [20]. For instance, a player with a primary focus on achievement might spend more time carefully investigating every room to find hidden secrets [13]. The specific pacing graph can specify the beats in more detail by setting up pacing graphs for different player paths. This results in a more accurate prediction of the perceived intensity.

C. Quest Systems

Game developer Szczepanski [21] discusses his experience with quest systems in Horizon Zero Dawn [22]. He mentions two kinds of systems—strict systems with a centralized system managing all quests and their interactions and relaxed systems where the quests directly interact with each other. The main finding for our paper's framework is that tool developers for quests often define specific keywords for the quest designers. These restrict what a quest can do. He also mentions that graphs are suitable for such quest systems, and edges connecting nodes can be more meaningful than simple cause-and-effect relationships.

Game programmer Iwański [23] supports our finding that graphs represent quest systems in video games. The project he worked on, *Cinematic Feel* [24], uses predefined nodes for both their quest and scene graphs. Both these systems control the game's output. Scene graphs have a bigger scope and can contain quest graphs. Additionally, quest systems follow an event-based approach, while scene systems follow a tick-based approach. One key takeaway from this talk was the simulator view, where game developers can test specific sequences of graph nodes starting with a selected node. However, these tests require a manual setup and can not cover all player cases due to the complexity of some of the scene or quest graphs.

D. Playing with Certain Playstyle Preferences

Vandenberghe [25] proposes the 5 Domains of Play model that allows game developers to play video games like another type of player. The model closely derives from the Five-Factor Model, which describes people's personalities based on their motivations [26]. As the model could not find a specific

This is the accepted author manuscript of the following publication: M. Schacherbauer and D. Dyrda, "Designing with Player Paths: Presenting a Game Design Framework," IEEE Conference on Games, 2025. DOI: doi.org/10.1109/CoG64752.2025.11114252. The published version is available at IEEE Xplore: ieeexplore.ieee.org/document/11114252.

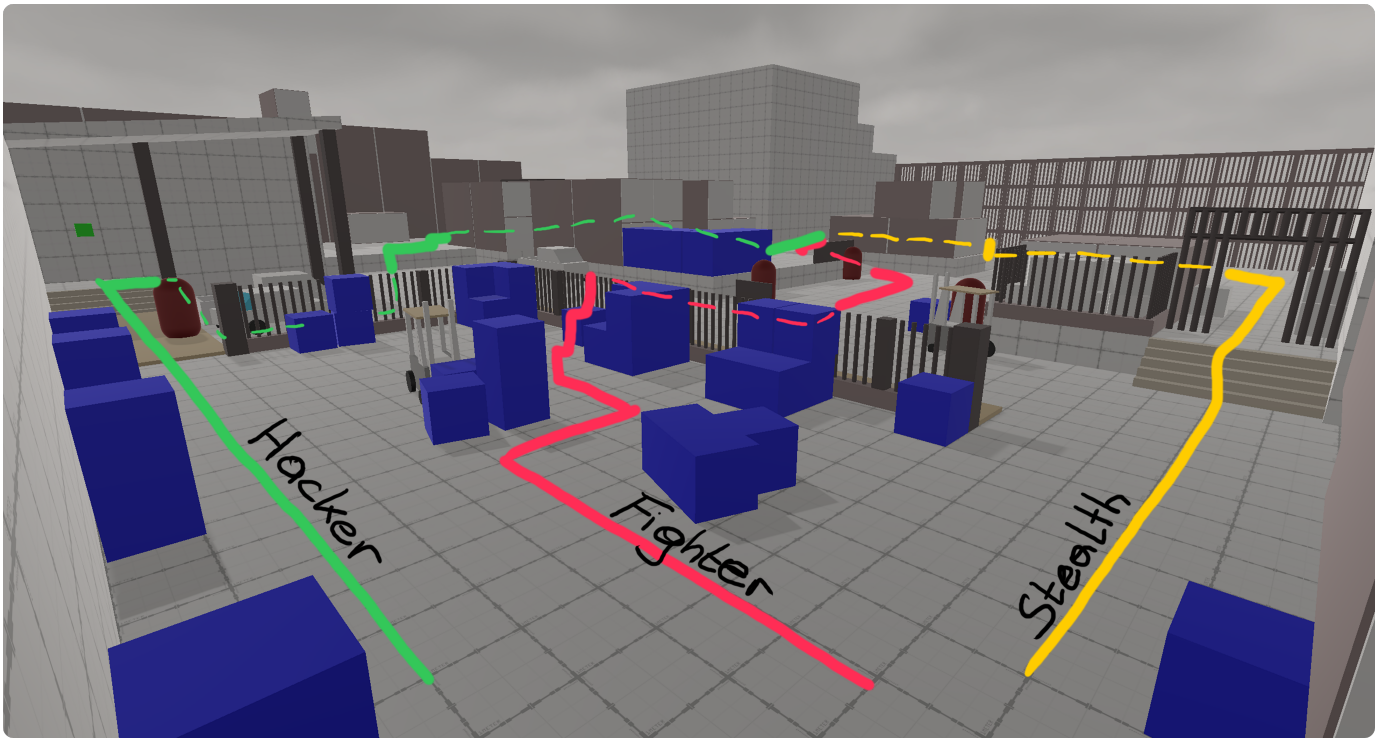

Fig. 1. Blockout of level Ebunike with drawn-in paths. Inspired by Pears [16]

characteristic in gaming for one of the five factors [25], his model only shows four domains represented on a 2D graph with two characteristics on each axis. He proposes that by playing video games while taking on the role of a specific type of player, game developers can learn what kind of effects other video games, or also their own game, have on a player that tends to some domain, e.g., skilled play.

Such an approach can raise awareness among game developers regarding the different motivations of players. When analyzing a game with a special focus on a specific playstyle preference, game designers can more accurately describe the experience and also adjust the experience if wanted. Especially for indie developers, who often struggle to get sufficient playtesters [27], analyzing games based on different motivations might reduce players' issues with the game.

V. Framework

One concept we found repeatedly in the aforementioned talks and works was the concept of graphs as an abstraction. This abstraction layer allows game developers to work in parallel—such as designing a level, while another developer works on fundamental mechanics needed for the level. Our framework aims to use graphs as a tool to model choice in space and resulting player paths to help during the process of designing levels. As another side effect, the graph acts as a common language between multiple disciplines—such as developing, designing, and managing. This, in turn, reduces potential miscommunication as different teams use different communication styles [18].

The workflow we want to support with the tool is as follows:

1) Identify choices to incorporate in the level.
2) Represent these choices in the editor.
3) Connect these choices with paths.
4) Visualize the path in the editor and evaluate it.

We use graphs as this abstraction layer between the functionality in the video game and the design process during development. Especially of interest are graphs that model choices on the macro level in the game and link them to the game space. Tool developers can use edges and vertices (nodes) to visualize the usage and functionality of those graphs during development. While graphs are used multiple times, such as unlock requirements, skill trees, and many more, our primary focus is player paths and how they can be visualized for the designer or developer during development. This section focuses on how game engines provide all the features required to support our framework. Section VI shows a proof of concept of the proposed framework.

A. Graph

As one of the core concepts revolves around graphs, the game engine has to support a view for graph-like structures. This view is available in most game engines due to their versatile usage. For instance, graphs often visualize shaders. In the case of a high-level game engine, which focuses on giving game developers with little experience the option to make a game, graphs often visualize quest dependencies in the game engine. By saying this, we also have to acknowledge that the framework requires some programming knowledge. However,

This is the accepted author manuscript of the following publication: M. Schacherbauer and D. Dyrda, "Designing with Player Paths: Presenting a Game Design Framework," IEEE Conference on Games, 2025. DOI: doi.org/10.1109/CoG64752.2025.11114252. The published version is available at IEEE Xplore: ieeexplore.ieee.org/document/11114252.

we still want to show that even with game engines that do not focus on programming, implementing our framework is still possible. Given this view, the developers can visualize and build their graph system, provided that they have an implementation for the nodes and edges.

B. Nodes

The nodes' definitions must fulfill the game's requirements, but we propose to include at least a position that is related to the node—linking nodes to space—and some way to identify which nodes belong to which player path. The framework achieves the first requirement by returning a vector of the position of the node's objective. This position can refer to the central position of all enemies the player has to neutralize or the position of an item the player must collect. For the second requirement, a node can be assigned to one or multiple player paths by defining an enumeration flag, with each bit referring to a specific player path. By allowing the assignment of multiple paths to a node, we ensure complete flexibility for the designer.

C. Edges

We have to note that for this framework, edges only connect two nodes with a *cause-and-effect* meaning. It is possible to give edges other meanings, such as *if node A is completed, node B is unavailable to the player*, making the graph and the system more expressive. However, the framework should act as a base with functionalities that can be used in various video games. Therefore, we decided against putting these functionalities in the scope of this framework.

D. Visualization in the Editor View

Inspired by how Pears' [16] screenshot shows the level with the paths drawn, see Fig. 1, our proposed framework shows player paths according to the given graph by drawing lines in different colors into the editor view. The visualization tool has a graph containing nodes and edges with the proposed values in the previous paragraphs as input. As we want the visualization only in the editor view, we propose to use game engine features that only show in the editor view. Because game developers often require information in their editor view that is not supposed to show in the game, most game engines offer such features and, therefore, a good option to display that information.

In order to visualize paths, the visualization tool finds all pairs of connected nodes that share at least one path type. For each pair, the visualization tool draws a line between the two positions, with the color depending on the player path type. Different colors allow the user to distinguish multiple paths in the editor view. Visualizing a pair of nodes that shares more than one player type needs to be addressed, as otherwise, one path might draw right over the other. Therefore, the visualization tool must allow users to select which player paths they want to show in the editor, also allowing people with colorblindness to use the tool. Other approaches are possible, e.g., adding small offsets to the position or using transparent colors.

VI. Implementation in a Game Engine

In this section, we will show a prototype of the proposed tool, as discussed in section V. We developed the tool in Unity [28]. We chose Unity because of the authors' familiarity with the engine and the programming language C#. Unity is also among the industry's two most used game engines [29].

Beginning with the graphs, we use the open-source tool *xNode*. xNode extends Unity with an additional xNode-view, basic tools, and class definitions for graphs and nodes. Users can use their custom-defined graphs and nodes in the xNode-view. These nodes show predefined entry and exit points, and the user can draw edges between the nodes using the entry and exit points, as seen in Fig.3. Furthermore, it allows users to create a custom editor for their defined nodes, which makes xNode suitable for more advanced graphs.

Using the base class *NodeGraph* provided by the xNode-package, we derive a new class *LevelNodeGraph*, which will contain all our graph nodes. With this implementation, users can create a graph in Unity with all kinds of nodes. To access any values in the scene, we create an additional class, *SceneLevelNodeGraph*, which derives from the xNode class *SceneGraph<LevelNodeGraph>*. The derivation allows us to attach that class as a script to a game object in a scene and, therefore, reference other in-scene objects.

Two additional parts are required before implementing the nodes. First, we define an enumeration type, *PlayerPathType*, with the *Flag* attribute. By accordingly using the defined values in the enumeration as bit-flags (1, 2, 4, 8, and so on), Unity automatically lets users assign each type defined in the enumeration separately to a variable of the enumeration type. In our implementation, we define an interface *IObjective*, representing an objective in the game. All our objectives implement this interface, as seen in Fig. 2. The only method required is *GetCenterPosition()*, which returns the objective's central position. The position should follow the objective's goal, such as the center point of an area the player has to enter or the middle point of all enemies that need to be defeated.

Regarding the nodes, we implement a base node containing all the functionality in order to fulfill the requirements described in section V. The class *LevelObjectiveNode* derives from the xNode class *Node*. This allows designers to define the node's entries and exits and visualize these in the xNode-view while using the functionality provided by the xNode-package. Furthermore, we implement a few additional types of nodes with different objectives to show the tool's possible expressiveness. They derive from *LevelObjectiveNode*. For the baseline functionality, we define variables for both the input and output of the node. By adding the *Input* and *Output* tags to each of the variables, respectively, xNode visualizes both in the graph view, see Fig.3. Additionally, edges between different nodes can be drawn by clicking on one of the ports. The class has a member of type *PlayerPathType*, representing the node's planned player path(s). The node also provides a function, *GetObjective()*, that deriving classes must implement. The method returns the *IObjective* associated with the node.

This is the accepted author manuscript of the following publication: M. Schacherbauer and D. Dyrda, "Designing with Player Paths: Presenting a Game Design Framework," IEEE Conference on Games, 2025. DOI: doi.org/10.1109/CoG64752.2025.11114252. The published version is available at IEEE Xplore: ieeexplore.ieee.org/document/11114252.

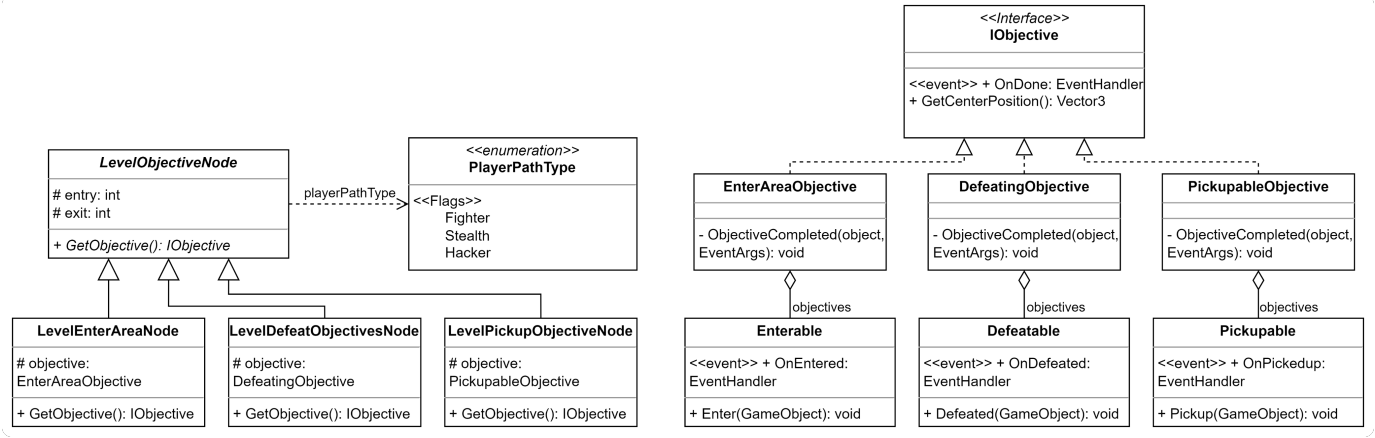

Fig. 2. UML diagram containing the node setup (left) and the objective setup (right).

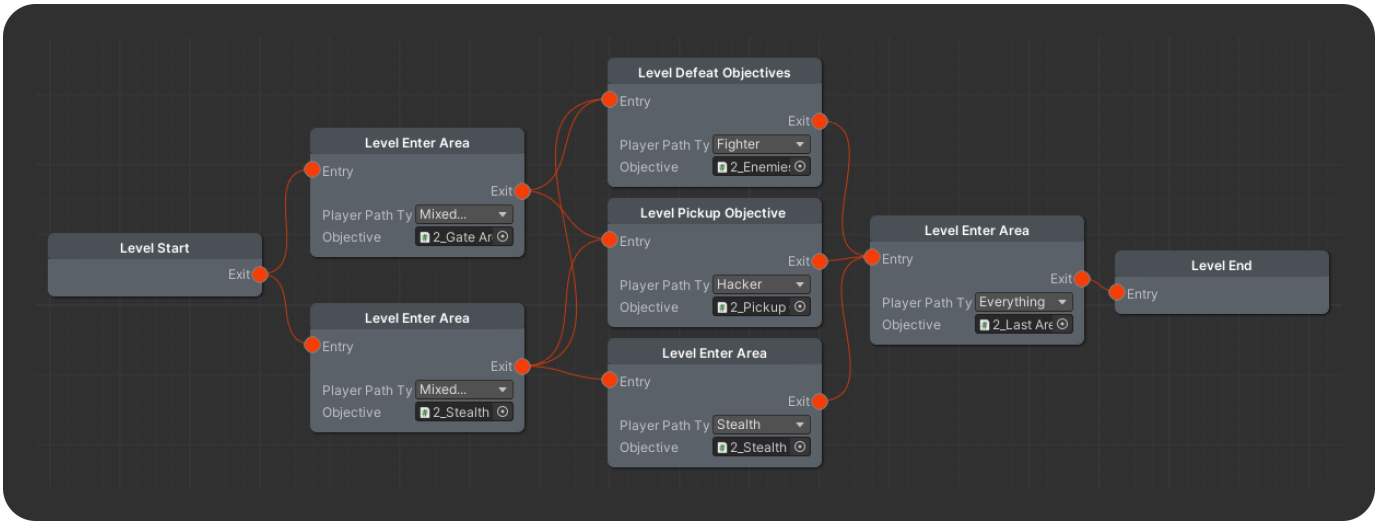

Fig. 3. Example graph for a level realized with xNode and Unity.

Last, we implement the class responsible for visualizing the planned player paths in the editor, *LevelGraphVisualizer*. It visualizes a *SceneLevelNodeGraph*. Additionally, it keeps a list of each player path containing the nodes attributed to it in a dictionary, with each player path type as its key. We opted to update those lists the first time the graph is assigned and allow the user to manually trigger that update whenever they want. We draw the lines in *OnDrawGizmos()*. The method iterates through all the entries in the dictionary's lists, sets a color according to the *PlayerPathType*, and then iterates through all the nodes in the saved list. For each node, we check all connected nodes. The visualization tool draws a line between the two node's objectives if they are of type *LevelObjectiveNode* and also in the list (which is equivalent to checking whether the same player path type is set). Fig. 4 shows a demonstration of the tool, with 4.c) being the realization of the proposed functionality in Fig. 1. The project and code are available to interested scholars and practitioners on request.

VII. Evaluation & Validation

A. Alignment with Existing Approaches

The framework's goal was to use common parts of game engines in order to support many game engines. We build on standard game development practices, such as graphs, editor-only graphics, and the game space for our framework. Furthermore, we realize concepts of choice and pacing diagrams by visualizing them in the game space via our framework. Our framework complies with and builds on existing approaches.

B. Practical Implementation

As shown in section VI, the framework can be implemented in Unity. The implementation requires knowledge of the chosen game engine and object-oriented programming. Given these circumstances, the implementation of the framework is feasible, especially for tool developers.

C. Common Language

As seen in Fig. 3, the framework's basis on graphs allows for the tool developer to give game designers, even without

This is the accepted author manuscript of the following publication: M. Schacherbauer and D. Dyrda, "Designing with Player Paths: Presenting a Game Design Framework," IEEE Conference on Games, 2025. DOI: doi.org/10.1109/CoG64752.2025.11114252. The published version is available at IEEE Xplore: ieeexplore.ieee.org/document/11114252.

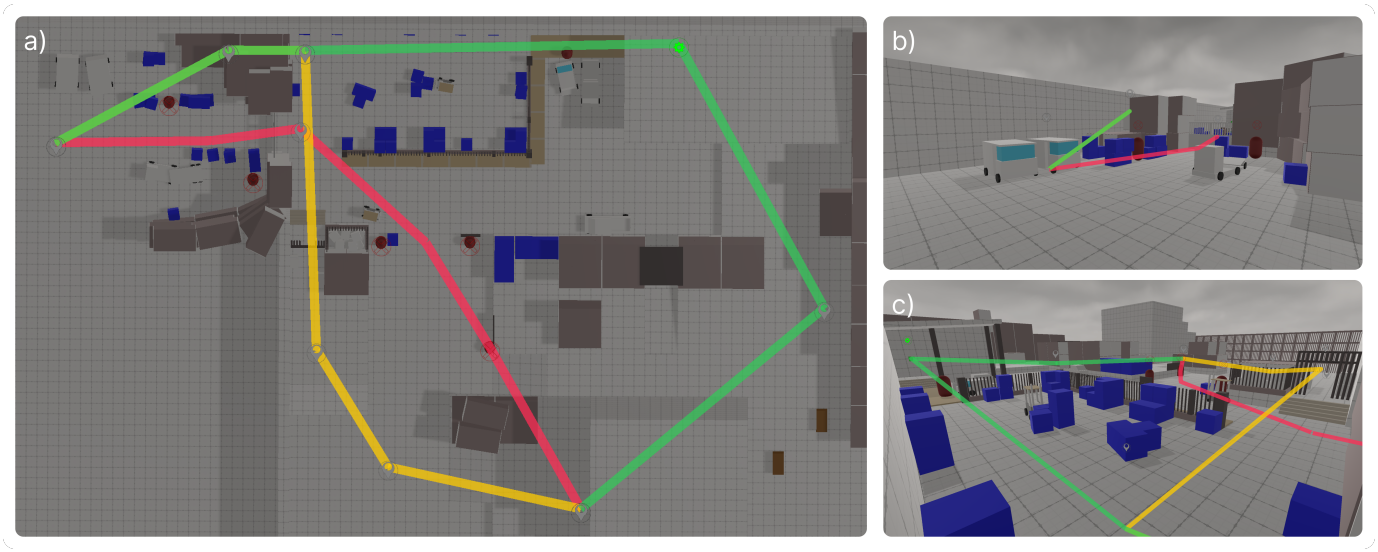

Fig. 4. Blockout of Level Ebunike, based on [18], with player paths visualization. (a) is a top-down representation of the entire level, (b) a player view of the first part of the level, and (c) a player view of the second part of the level. Made in Unity.

programming knowledge, the opportunity to create and use these graphs, as the tool developer can give the nodes intuitive names that hint at the purpose and functionality of the node. The suitability of graphs as a common language is further supported by level designers [21], [23] and scholars [17].

D. Potential Applications

As the framework relies on modeling choice in the possibility space, our framework is applicable for games that aim to carefully design around choices and multiple player paths. Games that prominently feature player paths, for instance, by giving the player exclusive skills, benefit most from the framework, as it helps game designers visualize different paths and see how players with specific skills might complete the level. However, these paths do not need to be tied to specific skills, as player motivation can lead to different approaches of players as well [25]. Therefore, our framework can benefit any game that incorporates choice in the game space. For instance, sandbox games that position unique special structures in the game space can use the framework to visualize the different paths through these structures based on player motivations.

VIII. Discussion

A. Challenges & Limitations

1) Explicit Choice Modeling: Our framework proposes the explicit modeling of choice in video games. While explicit modeling has advantages, such as being able to more accurately design for a specific experience [19], it restricts the game designer as it forces them to explicitly model choice. Furthermore, explicit choice also means restricting choice, as it is impossible to model an infinite number of outcomes. However, as the complexity introduced by choice already leads to game developers introducing the illusion of choice to reduce the game's overall complexity [30], this might not be a negative aspect. Still, it has to be mentioned that the framework relies on explicit choice modeling.

2) Additional Workload: Our framework requires a graph as an input with nodes representing some choice embedded in space. However, a game designed around having choice embedded in space does not mean that it has a graph representation for that choice. Furthermore, no matter if the game project already has an implementation for the graph, the framework's requirements, while minor, pose an additional workload for the game developers. However, as graphs are already used by game developers [19], [21], [23], such graphs are often already available.

3) Simplicity of Framework: The framework's base functionality is limited. This limitation is even more present in the implementation, as, for instance, walls and floors do not influence the path drawing. The feature complexity is limited. While this means that custom-domain usages require more implementation work, this is also a benefit of the implementation. The tool's simplicity leads to less bloated engine components while supporting the established design approach.

B. Opportunities & Implications

1) Building on the Framework: The framework is lightweight and extensible, as it does not require any specific knowledge about the choices except for the position and an assigned player path. Implementing the position query can also be handled by other approaches, such as the Space Foundation System [31]. The framework can be used as a baseline for more works, like path coverage or simulation of specific paths during development, as already shown by [23]. Furthermore, the framework allows for more debug information to be displayed based on the player paths, such as the predicted time to complete the path, the number of skill checks required to complete the path, or the number of users that took a

This is the accepted author manuscript of the following publication: M. Schacherbauer and D. Dyrda, "Designing with Player Paths: Presenting a Game Design Framework," IEEE Conference on Games, 2025. DOI: doi.org/10.1109/CoG64752.2025.11114252. The published version is available at IEEE Xplore: ieeexplore.ieee.org/document/11114252.

specific path after a playtest session. Similar gameplay analysis approaches are described in [32] and [33].

2) Spatially Embedded Choice in More Games: Not every game is designed around player paths and has choice embedded in space. For instance, choices in games where dialogues represent the player's only choice are not necessarily tied to the game space—but the resulting reaction could be. Similarly, completing a level in a real-time strategy game campaign might benefit from giving players the option of attacking the highly protected castles from multiple points with different types of units. The main goal of the framework was to propose and support a specific workflow, and even though this kind of workflow is most commonly used in adventure games, we think that our framework can inspire and aid game developers of other genres to think about spatially embedded choice.

3) Using Graphs for Evaluation: The graph-based approach allows game developers to apply well-known graph algorithms. These can be used to find the shortest path for a specific player path, check whether a player can reach certain nodes given a player path, or find out whether the path includes cycles. Game designers can generate more information about their layout and player paths, which helps them to refine or fix problems related to a player path.

IX. Conclusion

This paper addresses choice, possibility space, and player paths in video game development. We argue that choice is often represented in space in video games, i.e., the possibility space, and found that game designers commonly use graphs to visualize choice. Our framework proposes to combine choice in space and graphs to represent the possibility space through player paths. Nodes in the graph represent choices that can be mapped to specified positions in the game space. Our framework uses these spatially embedded choices and visualizes different player paths based on carefully designed linear sequences of choices to the game designer. The framework allows designers to work more consciously with player paths in their development process.

References

[1] J. Schell, *The Art of Game Design: A book of lenses*, 3rd ed. CRC Press, 2019.

[2] L. Ermi and F. Mäyrä, "Fundamental components of the gameplay experience: Analysing immersion," in *Proceedings of DiGRA 2005 Conference: Changing Views: Worlds in Play*. Tampere: DiGRA, 2005.

[3] C. W. Totten, *Architectural approach to level design*, 2nd ed. Boca Raton: CRC Press, Taylor & Francis Group, 2019.

[4] K. Salen and E. Zimmerman, *Rules of play: Game design fundamentals*. MIT press, 2003.

[5] J. Juul, "Introduction to game time," in *First Person: New Media as Story, Performance, and Game*. MIT Press, 2004, pp. 131–142.

[6] J. Greenwood, L. Achterbosch, A. Stranieri, and G. Meredith, "Understanding the gap between academics and game developers: an analysis of gamasutra blogs," in *International Conferences Interfaces and Human Computer Interaction, online. Inderscience Publishers, Kent, Ohio, USA*, 2021, pp. 141–148.

[7] D. Sudnow, *Pilgrim in the Microworld*. Warner Books New York, 1983.

[8] D. A. Norman, *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books, 2013.

[9] T. Norberg, "Player paths," 2021, retrieved March 08, 2025. [Online]. Available: https://x.com/the_Norberg/status/1374469984127045639

[10] R. Bartle, "Hearts, clubs, diamonds, spades: Players who suit muds," 1996.

[11] N. Yee, "Motivations for play in online games," *CyberPsychology & behavior*, vol. 9, no. 6, pp. 772–775, 2006.

[12] V. Klézl and S. Kelly, "Negativists, enthusiasts and others: a typology of players in free-to-play games," *Multimedia Tools and Applications*, vol. 82, no. 5, pp. 7939–7960, 2022.

[13] L. E. Nacke, C. Bateman, and R. L. Mandryk, *BrainHex: Preliminary Results from a Neurobiological Gamer Typology Survey*. Springer Berlin Heidelberg, 2011, pp. 288–293.

[14] J. Kahila, T. Valtonen, S. López-Pernas, M. Saqr, H. Vartiainen, S. Kahila, and M. Tedre, "A typology of metagamers: Identifying player types based on beyond the game activities," *Games and Culture*, 2023.

[15] J. Hamari and J. Tuunanen, "Player types: A meta-synthesis," 2014.

[16] M. Pears, "Cyberpunk 2077," 2017, retrieved March 12, 2025. [Online]. Available: https://www.maxpears.com/2017/02/25/cyberpunk-2077/

[17] G. Wallner and S. Kriglstein, "Visualization-based analysis of gameplay data – a review of literature," *Entertainment Computing*, vol. 4, no. 3, pp. 143–155, 2013.

[18] M. Pears, "Cyberpunk 2077 - Ebunike Level Construction," 2023, Presentation at devcom Developers Conference 2023.

[19] P. Ellis, "Single Player Level Design Pacing and Gameplay Beats - Part 1/3," 2015, retrieved January 10, 2024. [Online]. Available: https://www.worldofleveldesign.com/categories/wold-members-tutorials/peteellis/level-design-pacing-gameplay-beats-part1.php

[20] J. Geheeb, D. Dyrda, and S. Geheeb, "Pacemaker: A practical tool for pacing video games," in *2024 IEEE Conference on Games (CoG)*. IEEE, 2024.

[21] L. Szczepanski, "Building Non-Linear Narratives in Horizon: Zero Dawn," 2017, retrieved January 29, 2024. [Online]. Available: https://www.gdcvault.com/play/1024158/Building-Non-linear-Narratives-in

[22] Guerrilla Games, "Horizon zero dawn," [PlayStation 4, PlayStation 5, Windows], 2017.

[23] S. Iwański, "The Pillars of Scene System in Cyberpunk 2077," 2023, devcom Developer Conference 2023.

[24] CD Project, "Cinematic Feel," 2020, retrieved January 30, 2024. [Online]. Available: https://www.cdprojekt.com/en/capital-group/eu-projects/32017-cinematic-feel/

[25] J. VandenBerghe, "Applying the 5 Domains of Play: Acting Like Players," 2013, retrieved January 17, 2024. [Online]. Available: https://www.youtube.com/watch?v=6uX6ye66NK0

[26] J. M. Digman, "Personality structure: Emergence of the five-factor model," *Annual review of psychology*, vol. 41, no. 1, pp. 417–440, 1990.

[27] N. Moosajee and P. Mirza-Babaei, "Games User Research (GUR) for Indie Studios," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2016.

[28] Unity Technologies, "Unity," retrieved February 07, 2025. [Online]. Available: https://unity.com

[29] G. 2025, "Gdc 2025 state of the game industry," 2025, retrieved March 13, 2025. [Online]. Available: https://gdconf.com/news/gdc-2025-state-game-industry-devs-weigh-layoffs-ai-and-more

[30] B. Alborov, "Illusion of choice is better than choice: choices and illusions as narrative mechanics," 2017, retrieved March 13, 2025. [Online]. Available: https://www.gamedeveloper.com/design/illusion-of-choice-is-better-than-choice-choices-and-illusions-as-narrative-mechanics

[31] D. Dyrda and C. Belloni, "Space foundation system: An approach to spatial problems in games," in *2024 IEEE Conference on Games (CoG)*. IEEE, 2024.

[32] 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.

[33] Z. Teng, J. Pfau, S. S. Maram, and M. Seif El-Nasr, "Interactive player journeys: Co-designing a process visualization system to video game analytics," in *Proceedings of the 19th International Conference on the Foundations of Digital Games*, ser. FDG 2024. ACM, 2024.