

Modeling Player Navigation: A Critical Review of Techniques and Applications

Qizhou Deng

1. Introduction:

Human-Computer Interaction (HCI) is a multidisciplinary field that investigates how humans interact with computers and other digital systems. The main goal of HCI is to design, develop, and evaluate user interfaces and systems to make them more user-friendly, efficient, and accessible [1]. Game User Research (GUR) is a subfield of HCI that specifically focuses on understanding and improving the player experience in video games. GUR researchers conduct studies to understand player behavior, preferences, and expectations, as well as to identify usability issues in game interfaces and mechanics [1]. The development of engaging and user-friendly video games requires a deep understanding of player behavior and preferences. Human-Computer Interaction (HCI) and Game User Research (GUR) are essential disciplines that focus on improving player experience through the application of user-centered design principles and methodologies. A prevalent method utilized in Game User Research is user testing (also known as playtesting), where researchers examine and evaluate the actions of users engaging with game systems to confirm that the actual player experience aligns with the developer's objectives. Moreover, Modeling player navigation is a crucial aspect in user testing, as it directly impacts the evaluating a game's usability, accessibility, and overall player experience. Modeling and predicting player movement patterns enables game designers to create levels that are engaging, immersive, and provide a satisfying challenge to the player. Modeling player navigation typically involves creating a system that simulates how a player would move through a game environment. This can include movement on foot, as well as movement on vehicles or other modes of transportation [2].

The motivation for studying modeling player navigation stems from the importance of providing enjoyable game experiences for players. Designing levels that cater to various playstyles is essential in attracting and retaining players. Moreover, by improving the efficiency and effectiveness of game testing and evaluation processes, developers can deliver better experiences and more successful products. Finally, by comprehending and modeling how players navigate through game worlds, developers can optimize level design, improve gameplay mechanics, and ultimately create more immersive and enjoyable experiences.

The principal aim of this scholarly essay is to conduct a thorough examination of the literature concerning modeling player navigation by assessing various methodologies, strategies, and applications. This essay seeks to offer a holistic understanding of the existing research and advancements in this domain, emphasizing the primary challenges, prospects, and trajectories for both game developers and researchers. By meticulously evaluating the relevant literature,

the essay intends to furnish valuable insights and direction for those involved in enhancing player navigation and, ultimately, refining the overall gaming experience. By delving into the literature on these specific aspects, the essay will contribute to a comprehensive comprehension of the current research and development status in player navigation modeling, accentuating the crucial challenges, opportunities, and forthcoming prospects for game developers and researchers.

2. visual analysis to player navigation

Introduction:

Collecting and analyzing various types of behavioral data during playtesting sessions is crucial for modeling player navigation processes. Effective detection and gathering of player information are essential components of this process. Visual analysis tools can significantly contribute to understanding player navigation patterns and informing game design decisions.

In this paper [3], a visualization tool is presented. The tool is designed to display the changes that occur in three-dimensional (3D) virtual environments. It is utilized to analyze and illustrate data collected during virtual world events and simulations. The resulting visuals are beneficial in facilitating social navigation within 3D virtual worlds, assessing, and improving virtual world design, and providing a means to study the growth of virtual world communities. A different tool, mentioned in [4] has been created for the visual analysis of navigation patterns involving moving entities, such as users, virtual characters, or vehicles in 3D virtual environments (VEs). This tool, known as VU-Flow, offers a range of interactive visualizations that emphasize noteworthy navigation behaviors of individual or grouped moving entities, whether they are navigating the VE collectively or independently. These visualizations aid in enhancing the design of VEs and examining the navigation behavior of players. A method has been proposed to automatically detect landmarks from players' navigation he weighted entropy of the distribution of visiting players [5]. The framework in [6] includes a simple visualization tool for displaying the simulated data generated by the AI agents. This enables developers to quickly assess the agents' navigation patterns, identify problem areas within the game world, and make informed decisions on necessary design adjustments.

Analysis tools and techniques, such as those presented in the mentioned studies, can significantly contribute to understanding player navigation patterns in virtual environments. These tools enable researchers and game designers to extract meaningful insights from player behavior data, which can be applied to optimize game design, enhance player engagement, and tailor the gaming experience to different player types. Integrating these visual analysis techniques with the proposed player clustering algorithm and detection method can potentially provide a more comprehensive view of player navigation processes.

In conclusion, Visual analysis techniques and tools, such as those mentioned in the literature, play a significant role in understanding player navigation in online games. Researchers can gain valuable insights into player navigation patterns and make better decisions about game design and player experience optimization by combining these tools with the proposed player

clustering algorithm and detection method.

3. Pathfinding algorithms applied in different game scenarios

Introduction:

Pathfinding algorithms are essential components in modeling player navigation in video games, particularly when dealing with dynamic and complex environments. These algorithms are responsible for finding the shortest or most efficient route between two points while considering obstacles and other constraints. In the context of player navigation, pathfinding algorithms help create more realistic and immersive experiences by ensuring that non-player characters (NPCs) exhibit intelligent and believable movement behaviors. In game development, a navigation mesh is a common data structure used to represent the playable area of a game level in a way that is suitable for pathfinding. A navigation mesh is essentially a collection of interconnected polygons or triangles that describe the walkable surfaces of a level, such as floors, walls, and platforms. The mesh can be generated manually or automatically, and can be optimized for specific types of movement, such as walking or flying. In this topic we will focus on evaluating some of the pathfinding algorithms used to modeling player navigation and provide some critical reviews to determine which one is more suitable to model player navigation.

A* Algorithm: The A* algorithm is one of the most popular and widely used pathfinding algorithms in game development. It combines the strengths of both Dijkstra's algorithm and Greedy Best-First Search, utilizing a heuristic function to estimate the remaining cost to reach the goal, thus guiding the search in a more efficient manner. While the A* algorithm is known for its efficiency and optimality, it may encounter performance issues in large or highly dynamic environments, and its reliance on a suitable heuristic function can make it sensitive to changes in the game world. Additionally, the A* algorithm may not perform well in situations where the path cost is not directly proportional to the distance between nodes.

Dijkstra's Algorithm: Dijkstra's algorithm is another well-known pathfinding algorithm, often used as a basis for comparison with other algorithms. It focuses on finding the shortest path between two points by iteratively selecting the node with the smallest known distance from the starting point and updating the distances of its neighbors. Despite its simplicity and guaranteed optimality, Dijkstra's algorithm has certain limitations when applied to player navigation. Its primary drawback is that it can be slow, especially in large environments or when searching for a path over a long distance, as it explores all possible paths before settling on the shortest one. Furthermore, Dijkstra's algorithm does not incorporate any heuristic information to guide its search, which can result in a large number of nodes being explored before finding the optimal path. This paper [7] introduces the Dynamic Pathfinding Algorithm (DPA) works by continuously updating the navigation mesh and considering both fixed and moving obstacles while determining the optimal path. This enables NPCs to make real-time decisions and adapt to changes in the game environment, such as other vehicles or track elements that may obstruct their path. It combined Algorithm A* and Dynamic Pathfinding

Algorithm based on the advantages in [8]. Further, they conducted experiments on a racing game to verify the efficiency of different algorithms. Experimental results demonstrate that NPCs using a combination of the Dynamic Pathfinding Algorithm and the A* algorithm perform better than those relying solely on the A* algorithm. This is because the DPA allows NPCs to react to dynamic obstacles more effectively, enhancing their overall performance on the track. However, the obstacle position and the shape of the track significantly impact the performance of the DPA, as these factors influence the complexity of the navigation problem.

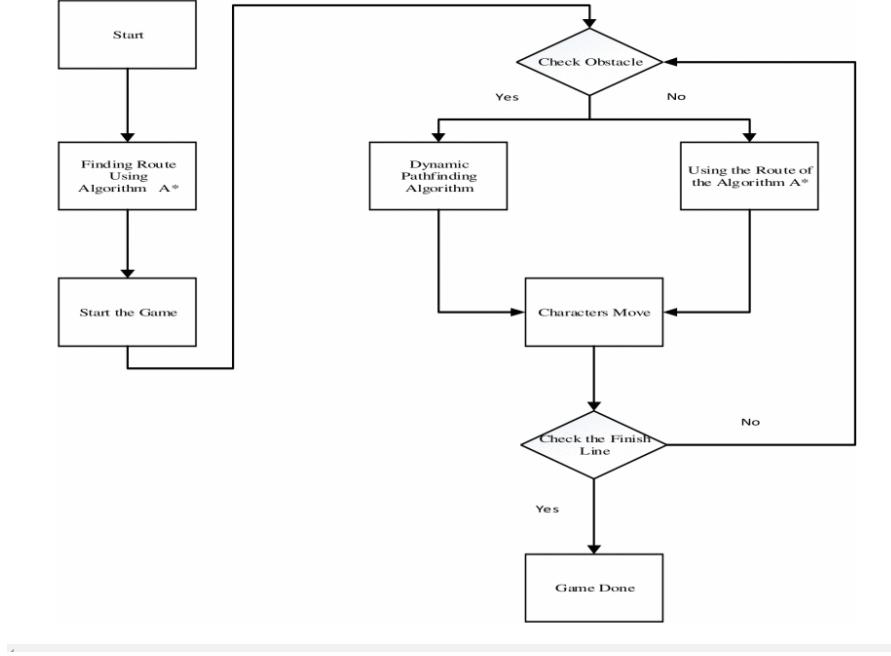


Figure 1.

Combined processing scheme algorithm A * and dynamic pathfinding algorithm

A probabilistic pathfinding algorithm is presented in this article [10], which is a novel approach to efficiently analyze 2D spaces and find certain paths. The Rapidly Exploring Random Tree (RRT) is a pathfinding algorithm used to efficiently explore a state space. Its random behavior enables the tool to represent a wide range of player behaviors [10]. The general RRT algorithm is based on five primary components, which can be encapsulated into functions or procedures:

1. Initialization: Begin with an initial state (root node) and create an empty tree data structure to store the nodes.
2. Sampling: Generate a random sample state in the configuration space (usually with a uniform distribution). This state acts as the target point for the current iteration.
3. Nearest Neighbor: Find the nearest node in the existing tree to the sampled state, using a distance metric suitable for the problem domain.
4. Steer: Create a new node by extending the tree from the nearest node toward the sampled state, within a pre-defined maximum distance (or step size). This step may involve calculating control inputs or solving simple kinematic constraints.
5. Collision Check: Verify if the new node and its connecting path to the nearest node

- are collision-free, i.e., they do not intersect with any obstacles.
6. Tree Expansion: If the new node is collision-free, add it to the tree, connecting it to the nearest node.
 7. Goal Check: Check if the newly added node is within a specified distance from the goal state. If it is, the algorithm may terminate, and the path can be reconstructed by backtracking from the goal to the root node.

Algorithm 1 Path Finding

```

1: procedure COMPPATHS( $x_{init}, \chi_{goal}, \chi_{free}, N, M$ )
2: Initialize( $\mathbf{T}, x_{init}, \Sigma$ )
3:   while  $i \leq M$  do
4:     while  $j \leq N$  do
5:        $x_{rand} \leftarrow Sample(\chi_{free})$ 
6:        $x_{near} \leftarrow KNN(x_{rand}, \mathbf{T})$ 
7:        $[x_{new}, \sigma] \leftarrow Steer(x_{near}, x_{rand})$ 
8:       if CollisionFree( $\sigma, \chi_{free}$ ) then
9:          $\mathbf{T.edges} \leftarrow \mathbf{T.edges} \cup \sigma$ 
10:         $\mathbf{T.vertices} \leftarrow \mathbf{T.vertices} \cup x_{new}$ 
11:        if  $\sigma.last \in \chi_{goal}$  then
12:           $\Sigma \leftarrow \sigma$ 
13:          break
14:        end if
15:      end if
16:    end while
17:  end while
18: end procedure

```

Fig.2 implementation follows the algorithm 1 pseudo-code

Experimental results show that the algorithm can effectively improve the simulation of the player's stealth movement in RPG and FPS games.

Another path finding approach has been proposed in [8], which use the damage function to calculate the optimal path in first person shooting games (FPS). The damage function quantifies the flux of damage at every point in space throughout the level, with the assumption that an optimal path exists through this damage field that minimizes player damage. The damage function can be created for various use cases and adjusted according to gameplay requirements and level design constraints.

1. The total number damage received from all direction at time t and location x :

$$total(\vec{x}, t) = \int_S damage(\vec{x}, \vec{v}, t) dv$$

2. The damage along the path, or in taking the path, is the integral over the function $total(\vec{x}(t), t)$, which can be expressed relative to time as:

$$Damage_{path}(t_1, t_2) = \int_{t_1}^{t_2} total(\vec{x}(t), t) dt$$

Using the damage function, we can compute the optimal path through the level, minimizing the damage a player would take. This path can be determined through various optimization algorithms, such as Dijkstra's algorithm, A* search, or other pathfinding techniques. However, the chosen algorithm should balance accuracy and computational

complexity to maintain smooth gameplay.

In conclusion, due to the successful growth of the gaming industry, various game modes indicate that pathfinding algorithms, as a crucial part of modeling player navigation, need to consider the player's perspective and accommodate different gameplay styles to complete the requirements of developers. This section mainly introduces the pathfinding algorithms used in several popular game genres, such as the combination of the Dynamic Pathfinding Algorithm and the A* algorithm in racing games, the RRT pathfinding algorithm in RPG games, and the combination of damage function and A* pathfinding algorithm in FPS games. These examples demonstrate how to evaluate and apply relevant pathfinding algorithms based on game content in modeling player navigation, which aligns with the research objectives of Game User Research (GUR) in understanding player behavior and improving game quality.

4. AI driven tools for modeling player navigation

Introduction:

AI research has consistently maintained a strong connection with games, which serve as a crucial foundation for the advancement of innovative algorithms. Various applications of AI in GUR have been identified, aiming to lessen the human and resource demands associated with different aspects of the GUR process. These efforts can be generally divided into two categories: aiding data analysis and facilitating simulation-driven testing. In game evaluation, simulation-driven testing is examined to assess the playability of game content (i.e., the feasibility of high-level play). Agent-based techniques are employed in design tools to evaluate playability in both human-generated and procedurally generated content [9]. AI-driven approaches are also being explored in the context of usability and user experience evaluations.

A significant body of research has been conducted on AI-driven approaches in games user research and automated testing. These studies have explored the opportunities and challenges associated with using AI-driven tools for user research, as well as the potential benefits of incorporating AI in the game development process. Topics covered in the literature include AI-driven tools for modeling player modeling, and AI-based evaluation methods for game design. PathOS is a tool for predicting player navigation in digital games by simulating virtual agents that model the trajectory of human players. The tool allows game designers and researchers to observe and collect data on how agents navigate a game's world in real-time or post-simulation visualization. PathOS is designed to help answer basic questions about how players will navigate a game's world, such as whether players will follow the intended path through a level or miss out on content placed in certain areas [2]. Tremblay et al., developed a similar modeling tool which is using probabilistic pathfinding methods to effectively analyze 2D space and find stealth paths in FPS and RPG games [10]. The design is integrated directly into the Unity 3D game development framework, as is PathOS, allowing interactive and highly dynamic exploration of how different virtual spaces and enemy configurations affect the stealth movement potential of the player or NPC [10].

Automated testing tools, such as PathOS, have the potential to significantly impact the game development process. By automating aspects of user testing, these tools can save time and resources while providing valuable insights into player navigation and behavior. However, it is important to recognize that automated testing should not replace traditional user testing methods entirely. Instead, AI-driven tools can complement and enhance existing processes, providing developers with a more comprehensive understanding of player navigation and its implications for level design. Additionally, as AI-driven tools continue to evolve, it is essential for researchers and developers to critically assess their effectiveness and ensure they are utilized appropriately in the game development process.

5. Future work

In the section on modeling player navigation algorithms, it can be seen that the pathfinding algorithms used in different games and their own efficiency vary significantly, and future research should be directed toward how to develop new pathfinding algorithms and how to find the most suitable pathfinding algorithm for a particular game mode. We also discovered that for the AI-driven tool for modeling player navigation. However, they have been developed and integrated with the Unity game engine. Still, the tool is only suitable for some games regarding ongoing functionality. A wide variety of games is available and in development, from 2Darcade-style shooters to role-playing games, visual novels, and just about every reasonable middle ground. Future research should be directed towards the development of simulated player navigation tools suitable for a broader range of games, as well as more appropriate pathfinding algorithms.

Conclusion

Modeling player navigation systems are a crucial aspect of contemporary game design, and they have become increasingly complex in recent years. The main components of modeling navigation are as follows: modeling player navigation systems, including efficient pathfinding, visualization techniques, and integration with other game systems such as AI-driven tools.

Visualization techniques are essential element of modeling player navigation, as they help designers create accurate navigation meshes that can handle dynamic environments and moving objects. These techniques can be simple or complex, depending on the needs of the game, and can be used to create realistic movement for the player character.

Efficient pathfinding algorithms are critical for modeling player navigation, as they determine the optimal path for the agents to take through the game world. Many different algorithms are used in game development, including A*, Dynamic Pathfinding Algorithm (DPA), and Rapidly Exploring Random Tree (RRT) pathfinding algorithms. These algorithms contribute to handling large game worlds and complex environments, ensuring that agents can move through the game world without encountering significant performance issues or other problems.

AI-driven tools play a crucial role in modeling player navigation, enhancing the game development process, and ensuring a more immersive and engaging experience for players. By utilizing AI techniques such as player modeling, clustering, automated testing, and probabilistic pathfinding, developers can efficiently design and evaluate game levels, predict player behavior, and identify potential issues early in the development cycle.

In summary, this paper critically reviews the techniques and applications used in modeling player navigation. The importance of modeling player navigation in player testing is also illustrated. Through the introduction and evaluation of the important elements of modelling player navigation, some open research questions and directions for future work are inspired.

Reference

- [1] S. Ariyurek, A. Betin-Can and E. Surer, "Automated Video Game Testing Using Synthetic and Humanlike Agents," in IEEE Transactions on Games, vol. 13, no. 1, pp. 50-67, March 2021, doi: 10.1109/TG.2019.2947597.
- [2] S. Stahlke, A. Nova and P. Mirza-Babaei, "Artificial Players in the Design Process: Developing an Automated Testing Tool for Game Level and World Design," in Proceedings of the Annual Symposium on Computer-Human Interaction in Play (CHI PLAY '20), 2020, pp. 267-280, doi: 10.1145/3410404.3414239.
- [3] K. Borner and S. Penumarthy. Social Diffusion Patterns in Three- Dimensional Virtual Worlds, Information Visualization Journal, vol. 2, no. 3, pp. 182-198, 2003.
- [4] L. Ieronutti, R. Ranon and L. Chittaro. VU-Flow: A Visualization Tool for Analyzing Navigation in Virtual Environments, IEEE Transactions on Visualization and Computer Graphics, vol. 12, no. 6, pp. 1475-1485, 2006.
- [5] hawonmas, Ruck & Kurashige, Masayoshi & Chen, Kuan-Ta. (2007). Detection of Landmarks for Clustering of Online-Game Players.. IJVR. 6. 11-16.
- [6] Stahlke, Samantha & Nova, Atiya & Mirza-Babaei, Pejman. (2019). Artificial Playfulness: A Tool for Automated Agent-Based Playtesting. 1-6. 10.1145/3290607.3313039.
- [7] Y. Sazaki, H. Satria and M. Syahroyni, "Comparison of A* and dynamic pathfinding algorithm with dynamic pathfinding algorithm for NPC on car racing game," 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), Lombok, Indonesia, 2017, pp. 1-6, doi: 10.1109/TSSA.2017.8272918.
- [8] Shi, Yinxuan & Crawfis, Roger. (2013). Optimal Cover Placement Against Static Enemy Positions.
- [9] S. Stahlke, A. Nova, and P. Mirza-Babaei, "Artificial Playfulness: A Tool for Automated Agent-Based Playtesting," in Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (CHI EA '19), New York, NY, USA, 2019, pp. 1-6, doi: 10.1145/3290607.3313039.
- [10] Tremblay, J., Andrade Torres, P., Rikovitch, N., & Verbrugge, C. (2013). An Exploration Tool for Predicting Stealthy Behaviour. In Proceedings of AIIDE 2013 (pp. 34-40)