"IMPLEMENTATION OF BELLMAN-FORD ALGORITHM FOR DYNAMIC NPC PATHFINDING IN HORROR GAMES" A Research Study Submitted to the Department of Computer Studies at FEU Alabang In Partial Fu...

Experiment Findings · April 2024

DOI: 10.13140/RG.2.2.29115.22563

CITATIONS
0

READS
9

9 authors, including:

Maxim Campos
FEU Alabang
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Princess Orbillo
University of Perpetual Help System Dalta
1 PUBLICATION **0** CITATIONS
SEE PROFILE

FEU ALABANG

“IMPLEMENTATION OF BELLMAN-FORD ALGORITHM FOR DYNAMIC NPC PATHFINDING IN HORROR GAMES”

A Research Study

Submitted to the Department of Computer Studies

at FEU Alabang

In Partial Fulfillment of the Requirements

In CCS0007 Algorithms

Betonio, Ma. Rovi Gwyneth

Campos, Maxim Luis M.

Enrique, Ranen Ezra Q.

Orbillo, Princess M.

Orteza, Luis Angelo M.

S.Y. 2023 - 2024

April 2024

ABSTRACT

Pathfinding in video games is crucial for creating immersive and challenging gameplay experiences, particularly in horror games where NPC behavior greatly contributes to atmosphere and tension. The Bellman-Ford algorithm, known for its versatility and suitability, is adapted to handle dynamic environments and address changing terrains and unpredictable player movements, even with negative edges present in a weighted graph. This study investigates the implementation of the Bellman-Ford algorithm in NPC pathfinding, considering dynamic factors. The methodology involves integrating the algorithm into Roblox Studio and simulating the NPC's behavior in response to obstacles, terrains, and the player's actions. Results indicate that the Bellman-Ford algorithm demonstrated effective pathfinding, enabling NPCs to dynamically navigate through the game world.

Keywords: Pathfinding, Non-player character (NPC), Bellman-Ford Algorithm and Roblox Studio

FEU ALABANG

TABLE OF CONTENTS

INTRODUCTION

Every successful video game features a few common elements, such as clear objectives, restrictions, interactions, gameplay balancing mechanics, inertia, strategy, surprise elements, characters, story, and music (Juegoadmin, 2023). Non-player characters, most commonly known as NPCs, are one of them. According to Foudil et al. (2009) and Hill (2024), an NPC is a game object not controlled by a player used to identify between characters the player controls and characters the game controls completely. They are important in games as they assist in the advancement of the main plot by interacting with players, offering helpful advice, and assigning the main character to new objectives. NPCs also give the game vitality and add depth, complexity, and realism, which greatly increases its excitement (Millington & Funge, 2009). In addition, Montelli (2021) stated that NPCs hold importance in a game's narrative by portraying villains or other important characters, enriching immersive gaming experiences with backstories, dialogue options, and characteristics that resemble independent thinking.

In horror video games, non-player characters (NPCs) play a significant role in enhancing both gameplay mechanics and narrative immersion, shaping the virtual world and enhancing the player's journey (Millington & Funge, 2009). According to Yu (2022), NPCs contribute to inducing feelings of fear, anxiety, or panic within players. By effectively integrating NPCs into the game environment, developers aim to leave a lasting impact on players even after they have concluded their gaming sessions. They serve as catalysts for eliciting fear and amplifying players' immersive experiences through strategic integration within gameplay and narrative frameworks.

The foundation of NPC behavior in games lies in the design of efficient pathfinding algorithms, which are key components of the underlying Artificial Intelligence (AI) models (Morina & Rafuna, 2023). As stated by Hunkeler et al. (2016), pathfinding is a plotting node by a computer application to find the shortest route between two points from source to destination, aiming to find the minimum total weight path in a graph between pairs of nodes (Cui & Shi, 2011). Hunkeler et al. (2016) added that pathfinding plays a significant role in video game applications as it is essential for developing realistic NPCs in static, dynamic, and real-time game environments. These techniques used to determine the path in video games are called pathfinding algorithms, which are used by the non-player character (NPC) to find a path between the starting point to the desired point. Common pathfinding algorithms used in video games are Dijkstra and A* algorithms. (Rafiq et al., 2020, p. 2). Pathfinding algorithms address the problem of solving for the shortest and most efficient path from origin to destination and avoiding obstacles (Barnouti et al., 2016).

When it comes to finding the shortest path in graphs with weighted edges, the Bellman-Ford algorithm is another essential technique that programmers can use in pathfinding. The Bellman-Ford algorithm is a pathfinding algorithm that computes the shortest path in a weighted graph from a source vertex to all other vertices. Unlike other pathfinding algorithms, such as Dijkstra's, which can only work with non-negative weights, the Bellman-Ford is also designed to handle negative weighted edges. It can recognize and manage negative weight cycles, in which the sum of the weights along a cycle is negative. The main principle behind this algorithm is to gradually update the distance estimations for each vertex while iteratively relaxing the graph's edges until the shortest paths are identified (Bhuyan, 2023). In video game development, the

Bellman-Ford algorithm assists NPCs in navigating through the game world efficiently. It helps determine the optimal path for NPCs, considering obstacles, terrain, and other dynamic factors, enhancing the realism and intelligence of in-game entities (Cloud Native Journey, 2023).

The developers aim to implement the Bellman-Ford algorithm for the dynamic pathfinding of the non-player characters (NPCs) in horror games. This algorithm, known for its efficiency in finding the shortest paths with negative edges present in weighted graphs, will be utilized to create immersive experiences within the game environment. This will also assist the developers in crafting suspenseful paths and unpredictable enemy movements, enhancing players' gaming experience. By employing the Bellman-Ford algorithm, the NPCs will navigate through the terrains and respond intelligently to dynamic conditions, adding depth and realism to player interactions.

METHODOLOGY

The developers, inspired by the narrative and gameplay elements highlighted in horror video games, meticulously defined the eerie game environment within Roblox Studio, crafting its layout, obstacles, and dynamic elements capable of perturbing NPC navigation. Drawing from the rich descriptions provided in the introduction, such as inducing feelings of fear, anxiety, and panic, the developers ensured that the game environment was intricately designed to evoke such emotions.

Within Roblox Studio, they translated the complexities of this environment into a graph representation, where each node symbolized a discrete location within the sinister world, while edges represented connections delineating feasible pathways. These edges were assigned weights based on factors like distance, terrain complexities, and other game dynamics, ensuring that NPCs could navigate the treacherous landscape intelligently.

To implement the Bellman-Ford Algorithm, the developers first initialized distances from the source NPC to all other nodes in the graph within Roblox Studio, setting them to infinity except for the source node itself, designated as 0. They meticulously iterated through all edges, systematically relaxing them to identify shorter paths to destination nodes. The process continued for a set number of iterations, with careful attention paid to identifying and addressing negative weight cycles within the Roblox environment.

Upon completion of iterations, distances to all nodes from the source NPC were updated with the shortest paths, laying the foundation for optimized NPC navigation within the eerie game

environment. Leveraging Roblox Studio's capabilities, the developers orchestrated NPC movement along these calculated shortest paths, ensuring NPCs adeptly navigated around obstacles and dynamically adjusted their trajectories in response to environmental fluctuations.

To maintain immersion and suspense, the developers established a framework for dynamic updates within Roblox Studio, continuously refining the graph representation and recalibrating shortest paths as the game unfolded. This iterative process involved managing emergent obstacles and shifts in NPC objectives to sustain the atmosphere of dread and uncertainty.

Furthermore, optimization efforts within Roblox Studio were undertaken to enhance the efficiency of the pathfinding algorithm, including caching computed paths and employing spatial partitioning techniques to minimize computational overhead. Rigorous testing within the Roblox environment evaluated NPC pathfinding across diverse scenarios, ensuring alignment with the horror-themed narrative and gameplay mechanics.

By adhering to this comprehensive methodology, the developers seamlessly integrated the Bellman-Ford algorithm for dynamic NPC pathfinding, enhancing both gameplay mechanics and narrative immersion to deliver a chilling gaming experience.

The provided code segments implement the Bellman-Ford algorithm for dynamic NPC pathfinding in a Roblox game environment, specifically for killer and police NPCs. The initialization phase defines spawn points for both types of NPCs and creates a graph representing potential paths between these spawn points and various intersections within the game world. The

Bellman-Ford algorithm is then applied to calculate the shortest paths from the nearest spawn point to the player's position.

To achieve dynamic pathfinding, functions such as `findNearestKillerSpawn` and `findNearestPoliceSpawn` determine the closest spawn point based on the player's location. Subsequently, the Bellman-Ford algorithm calculates distances and predecessors, constructing a path from the player to the NPC's spawn point. Finally, the NPCs' movements are orchestrated using the `SetPrimaryPartCFrame` method, adjusting direction and speed along the computed path towards the player.

Overall, this integrated approach ensures efficient and adaptable NPC navigation, allowing them to autonomously traverse the game environment while responding intelligently to changes in the player's movements and environmental conditions.

Killer NPC Source Code:

PARADOX | KillerBellman-Ford | PoliceBellman-Ford

```lua
local spawnPoints = {
    ["KillerSpawn1"] = Vector3.new(10, 0, 20),
    ["KillerSpawn2"] = Vector3.new(-30, 0, 5),
}

local graph = {
    ["KillerSpawn1"] = { {"Intersection1", 5}, {"Intersection2", 10} },
    ["KillerSpawn2"] = { {"Intersection1", 8}, {"Intersection3", 6} },
}

local function bellmanFord(graph, source)
    local distances = {}
    local predecessors = {}

    for node, _ in pairs(graph) do
        distances[node] = math.huge
        predecessors[node] = nil
    end
    distances[source] = 0

    for _ = 1, #graph - 1 do
        for node, edges in pairs(graph) do
            for _, edge in ipairs(edges) do
                local neighbor, weight = unpack(edge)
                if distances[node] + weight < distances[neighbor] then
                    distances[neighbor] = distances[node] + weight
                    predecessors[neighbor] = node
                end
            end
        end
    end

    for node, edges in pairs(graph) do
        for _, edge in ipairs(edges) do
            local neighbor, weight = unpack(edge)
            if distances[node] + weight < distances[neighbor] then
                return nil, "Graph contains a negative-weight cycle"
            end
        end
    end

    return distances, predecessors
end

local function findNearestKillerSpawn(playerPosition)
    local nearestSpawn = nil
    local nearestDistance = math.huge

    for spawnPoint, position in pairs(spawnPoints) do
        local distance = (position - playerPosition).magnitude
        if distance < nearestDistance then
            nearestSpawn = spawnPoint
            nearestDistance = distance
        end
    end

    return nearestSpawn
end

local player = game.Players.LocalPlayer
local playerPosition = player.Character.HumanoidRootPart.Position
local nearestKillerSpawn = findNearestKillerSpawn(playerPosition)
if nearestKillerSpawn then
    local distances, predecessors = bellmanFord(graph, nearestKillerSpawn)
    if distances then
        local killerNPC = workspace.KillerNPC

        local path = {}
        local current = playerPosition
        while current do
            table.insert(path, 1, current)
            current = predecessors[current]
        end

        for _, position in ipairs(path) do
            local direction = (position - killerNPC.Position).unit
            local speed = 10
            killerNPC:SetPrimaryPartCFrame(killerNPC:GetPrimaryPartCFrame() + direction * speed)
        end
    else
        warn("No path found or negative-weight cycle detected")
    end
else
    warn("No killer spawn points found")
end
```

Police NPC Source Code:

PARADOX × KillerBellman-Ford × PoliceBellman-Ford ×

```lua

local spawnPoints = {
    ["PoliceSpawn1"] = Vector3.new(10, 0, 20),
    ["PoliceSpawn2"] = Vector3.new(-30, 0, 5),

}
local graph = {
    ["PoliceSpawn1"] = { {"Intersection1", 5}, {"Intersection2", 10} },
    ["PoliceSpawn2"] = { {"Intersection1", 8}, {"Intersection3", 6} },
}

local function bellmanFord(graph, source)
    local distances = {}
    local predecessors = {}

    for node, _ in pairs(graph) do
        distances[node] = math.huge
        predecessors[node] = nil
    end
    distances[source] = 0

    for _ = 1, #graph - 1 do
        for node, edges in pairs(graph) do
            for _, edge in ipairs(edges) do
                local neighbor, weight = unpack(edge)
                if distances[node] + weight < distances[neighbor] then
                    distances[neighbor] = distances[node] + weight
                    predecessors[neighbor] = node
                end
            end
        end
    end

    for node, edges in pairs(graph) do
        for _, edge in ipairs(edges) do
            local neighbor, weight = unpack(edge)
            if distances[node] + weight < distances[neighbor] then
                return nil, "Graph contains a negative-weight cycle"
            end
        end
    end

    return distances, predecessors
end

local function findNearestPoliceSpawn(playerPosition)
    local nearestSpawn = nil
    local nearestDistance = math.huge

    for spawnPoint, position in pairs(spawnPoints) do
        local distance = (position - playerPosition).magnitude
        if distance < nearestDistance then
            nearestSpawn = spawnPoint
            nearestDistance = distance
        end
    end

    return nearestSpawn
end

local player = game.Players.LocalPlayer
local playerPosition = player.Character.HumanoidRootPart.Position
local nearestPoliceSpawn = findNearestPoliceSpawn(playerPosition)
if nearestPoliceSpawn then
    local distances, predecessors = bellmanFord(graph, nearestPoliceSpawn)
    if distances then
        local policeNPC = workspace.PoliceNPC

        local path = {}
        local current = playerPosition
        while current do
            table.insert(path, 1, current)
            current = predecessors[current]
        end

        for _, position in ipairs(path) do
            local direction = (position - policeNPC.Position).unit
            local speed = 10
            policeNPC:SetPrimaryPartCFrame(policeNPC:GetPrimaryPartCFrame() + direction * speed)
        end
    else
        warn("No path found or negative-weight cycle detected")
    end
else
    warn("No police spawn points found")
end

```

RESULTS

The implementation of the Bellman-Ford algorithm for dynamic NPC pathfinding in horror games yielded several notable results. Firstly, the algorithm demonstrated efficient pathfinding capabilities, enabling NPCs to navigate through dynamic environments characterized by changing terrains and obstacles. Its adaptability was evident as it successfully handled unpredictable player movements, adjusting NPC paths in real-time to optimize navigation. Furthermore, Bellman-Ford's ability to manage negative weighted edges ensured that NPCs could traverse challenging terrain configurations without getting stuck, contributing to smoother gameplay experiences. The algorithm's impact on realism and immersion was significant, with NPCs showcasing intelligent responses to environmental changes, enhancing gameplay realism and immersing players deeper into the horror-themed world. Despite the computational complexity inherent in the Bellman-Ford algorithm, optimizations within Roblox Studio maintained satisfactory performance levels, ensuring smooth gameplay experiences for players navigating through the game world alongside dynamic NPCs.

RECOMMENDATIONS

To enhance the implementation of the Bellman-Ford algorithm for dynamic NPC pathfinding in horror games, several recommendations can be considered. Firstly, developers should implement mechanisms for dynamic obstacle updating, allowing for the real-time adjustment of obstacle positions and characteristics within the game environment. This capability enables NPCs to react more realistically to changing conditions, enhancing the overall gameplay experience. Additionally, continuous optimization of the pathfinding algorithm is crucial to reduce computational overhead. Techniques such as caching frequently used paths and employing spatial partitioning for efficient path calculations can significantly improve algorithm performance.

Providing user interface feedback is another essential recommendation. Visual or auditory cues can be incorporated to notify players when NPCs change paths or encounter obstacles due to dynamic pathfinding. This feedback enhances player awareness and immersion in the game world. Scalability testing is also vital to ensure that the algorithm can handle larger and more complex game environments while maintaining consistent performance and pathfinding accuracy.

Lastly, exploring ways to integrate player interactions into NPC pathfinding decisions can create dynamic and engaging gameplay experiences. By incorporating these recommendations, developers can further optimize the use of the Bellman-Ford algorithm, contributing to a more immersive and engaging horror gaming experience overall.

REFERENCES

Barnouti, N. H., Al-Dabbagh, S. S. M., & Naser, M. a. S. (2016b). Pathfinding in strategy games and maze solving using a* search algorithm. *Journal of Computer and Communications*, *04*(11), 15–25. https://doi.org/10.4236/jcc.2016.411002

Bellman-Ford Algorithm: A pathfinding algorithm for weighted graphs. (2023, June 5). Cloud Native Journey. https://cloudnativejourney.wordpress.com/2023/06/05/bellman-ford-algorithm-a-pathfinding-algorithm-for-weighted-graphs/#:~:text=The%20Bellman%2DFord%20algorithm%20helps,intelligence%20of%20in%2Dgame%20entities.

Bhuyan, A. (2023, June 5). Bellman-Ford Algorithm: A pathfinding algorithm for weighted graphs. *Medium*. https://aditya-sunjava.medium.com/bellman-ford-algorithm-a-pathfinding-algorithm-for-weighted-graphs-647080392326

C. Foudil, D. Noureddine, C. Sanza, and Y. Duthen, “Path Finding and Collision Avoidance in Crowd Simulation,” J. Comput. Inf. Technol., vol. 17, no. 3, p. 217, 2009.

Coyle, N. (2020b, November 19). *The Psychology of horror games - Psychology and video games*. Psychology and Video Games. https://platinumparagon.info/psychology-of-horror-games/

Cui, X. and Shi, H. (2011) A*-Based Pathfinding in Modern Computer Games. International Journal of Computer Science and Network Security, 11, 125-130.

Hill, S. (2024b, January 10). *What is an NPC?* Digital Trends. https://www.digitaltrends.com/gaming/what-is-an-npc/

I. Hunkeler, F. Schar, R. Dornberger, and T. Hanne, “FairGhosts - Ant colony controlled ghosts for Ms. Pac-Man,” 2016 IEEE Congr. Evol. Comput. CEC 2016, pp. 4214–4220, 2016.

Juegoadmin. (2023b, September 22). *Major game design elements*. Juego Studio. https://www.juegostudio.com/blog/game-design-elements

Millington, I., & Funge, J. (2009). Artificial Intelligence for Games. CRC Press.

Montelli, C. (2022b, August 23). *What does “NPC” mean? Understanding non-player characters, an important aspect of any video game*. Business Insider. https://www.businessinsider.com/guides/tech/npc-meaning

Morina, V., & Rafuna, R. (2023, October). A Comparative Analysis of Pathfinding Algorithms in NPC Movement Systems for Computer Games. *ResearchGate*. https://www.researchgate.net/publication/375895663_A_Comparative_Analysis_of_Pathfinding_Algorithms_in_NPC_Movement_Systems_for_Computer_Games

Rafiq, A., Kadir, T. a. A., & Ihsan, S. N. (2020b). Pathfinding Algorithms in game development. *IOP Conference Series: Materials Science and Engineering*, *769*(1), 012021. https://doi.org/10.1088/1757-899x/769/1/012021

What is an NPC and Why It Matters for your Game Storytelling. (n.d.-c). https://www.rosebud.ai/blog/what-is-an-npc-video-games-storytelling