International Journal of Management Science and Information Technology (IJMSIT)
Vol. 3, No. 1, January – June 2023.
E-ISSN: 2774-5694, P-ISSN:2776-7388
DOI: https://doi.org/10.35870/ijmsit.v3i1.819

TOWER DEFENSE GAME BASED ON 2D GRID USING GOAL-BASED PATHFINDING METHOD

Genta Sahuri[1], Rosalina[2], Hardwin Welly Tulili Panandu[3]
[1,2,3]Faculty of Computing, President University, Cikarang, Bekasi, Indonesia
Email: [1]genta.sahuri@president.ac.id, [2]rosalina@president.ac.id

Article Info

Article history:

Received October 28, 2022
Revised November 17, 2022
Accepted December 02, 2022

Keywords:

2D Grid Based
Pathfinding
Tower Defense

ABSTRACT

At the moment, agents cannot choose their own path with any flexibility in tower defense games. There may be a lot of enemies in one level of a tower defense game. The majority of in-game characters have a habit of moving in the direction of goals or objectives, though most have distinctive numbers and behaviors. The pathfinding method can be used to determine the route between the sources coordinates and the destination coordinates in an AI movement system. In this study, an objective-based pathfinding technique is used in a tower defense game where players can choose their own route. Based on the test results, the game can change the destination, which forces the adversary to alter their course to reach the new location. By placing units that can block these paths, this game also has the capacity to alter the available paths on the map.

Corresponding Author:

Rosalina
Faculty of Computing, President University,
Jl. Ki Hajar Dewantara, Cikarang, Bekasi, Indonesia.
Email: rosalina@president.ac.id

1. INTRODUCTION

A tower defense game is a strategy-based game genre in which the player's goal is to prevent enemies from reaching objectives or capturing target cores by strategically placing stationary units that can deal damage to enemies or support other units. The enemy will appear from a specific location and stroll towards the goal/core via a predetermined path or the shortest path. Players must use resources to place units. Resources can be obtained through enemy destruction, generated over time, or generated by specific types of units. Tower defense games require unit placement strategy and resource management skills.

At the moment, many computer science researchers are focusing on computer games with intelligence [1],[2]. The tower defense genre, which is still relatively new, has grown significantly in popularity as online mobile versions have become more widely available to casual gamers. To prevent enemy troops from reaching the end point, players in "tower defense" strategy games must place static "towers" that can attack opposing units. When compared to other game genres, tower defense video games have a few distinguishing characteristics. Many tower defense games, in general, have characteristics such as a path that enemies can travel down, different types of enemies, upgradeable towers, and a defensive plan [4]. A tower defense game is an excellent research subject for computational intelligence (CI) [5],[9].

The majority of tower defense games offer various path designs, and maps are one of their key gameplay elements. A new tower defense game with circular tower placement was suggested by [10]. Two scenarios—one with hurdles and the other without—were suggested for experimentation by Sabriet al. additionally, the sizes of the maps differ slightly. Real-time or turn-based tower defense games allow resource allocation and tactical deployment through the position of towers. There could be numerous enemies in one level of a tower defense game. Although enemy agents typically have distinctive numbers and behaviour patterns, the majority of them have a tendency to move in the direction of goals or targets. To achieve this, pathfinding algorithms can be employed to analyze the behavior of enemy soldiers, also known as enemy AI in video games (AI). Pathfinding is a method or algorithm used to determine the relatively short path between source and destination points.

The enemy AI's path is sometimes predetermined to take an open path in tower defense games. Some other tactic is to have each enemy be using a pathfinding algorithm which controls the agent to the target point. It was advised that the pathfinding algorithm be more efficient and flexible in order to make the gameplay as dynamic as possible while granting players an opportunity to adjust enemy routes as an extra layer of strategic plan. Every route from every location leading to the destination is calculated in the destination-based pathfinding method, also known as Vector Field pathfinding [12]. Instead of employing an algorithm to compute the route of a single AI component from one location to the destination for any particular AI character, this method determines the required path length to the destination from each available position. This process offers all AI characters with the insight they need to determine on the most effective way to achieve their goal. This method has been implemented in research games [13],[17].

Using goal-based path-finding, this study creates a tower defense game. A destination-based pathfinding process generates a transformation matrix or distance vector based on the locations of the endpoints. Each tile or node in the vector field includes the direction of the next tile or node that is closest to the destination based on the available path. The distance to the target is calculated for each tile or node in the distance field based on the potential path. Rather than developing a pathfinding algorithm for each enemy, the research's enemies only need to read the values previously written on the tiles or nodes where they are placed. This study seeks to address the following issues:

a. How to apply effective pathfinding algorithms in strategy games like tower defense games to manage massive groups of enemies.
b. How to instated an adaptable pathfinding algorithm that can adapt to variations in the map that might impact the path the opponent must hold in a tower defense game.

2. RESEARCH METHOD

Initialization, preparation, and processing are the three main steps of the research methodology used to create tower defense video games. Each stage of the development process analysis is explained in the following direction:

a. Initialization: This is the phase at which the player starts the game. The title screen and main menu will appear.
b. Preparation: This stage determines when the main menu appears. There will be several buttons on the main menu for players to click, including the Play and Exit buttons. Players can exit the game by pressing the Exit button. When the player presses the Play button, the level options are displayed. Following that, the player will be pointed directly to pick the agent with whom they wish to interact.
c. Processing: At this level, the player chooses a level and an Agent to play. The game will load all of the required assets. The level, the Commander's specific position, and any enemy coordinates are then displayed. To determine the path the enemy takes from each spawner to the commander, objective-based pathfinding will be applied. The game will then run the logic for the enemy spawner to find a suitable opponent. Based on the player's choices, a list of Agents that can be highlighted by the player will be showcased.

Tower defense game with a grid-based map if the area can be represented as a grid of tiles or a graph of nodes. Goal-based pathfinding consists of two major activities. The major tasks are as follows:

2.1. Create a Heatmap

Heatmap is a concept in which each tile represents the distance from that tile to a target. It is also referred to as distance fields. Euclidean distance won't be the result of the computation (straight line distance). The paths that are accessible will be used to compute the distance. Any blockage, like a wall, will have an impact on the distance measurement. The heatmap would be generated using a Breadth First Search (BFS) approach. The BFS method determines the range of distances that can exist between each tile and the selected tiles. The method is carried out as shown in the subsequent steps:

a. The algorithm's beginning point, designated as the endpoint, is marked with a distance value of 0.
b. The algorithm then visits all of a marked tile's unmarked neighbors. Mark the tiles with the same distance from the preceding tile, then multiply that distance by 1.
c. It is necessary to repeat the previous step until every tile has been marked.

2.2. Creating Vector Field

According to the "Vector Fields" concept, each tile has a vector value that indicates the direction resulting from it to the end destination. Vectors can be calculated using the dissociation between adjacent tiles. The AI character can travel by simply ticking the path of the current tile since each tile's direction is calculated. The following is a possible way to express the route vector calculation for each plot:

Vector moveDirection = new Vector2 (left neighbor distance – right neighbor distance, top neighbor distance – bottom neighbor distance)

3. RESULTS AND ANALYSIS

There are three different views in the Grid-Based 2D Tower Defense game: The Main Menu, Level Selection, and Gameplay.

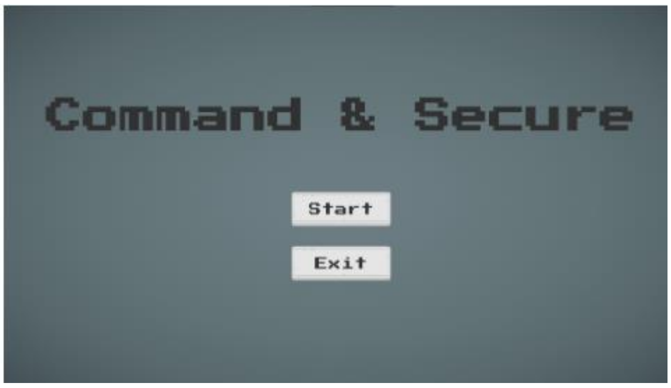


Figure 1. Main interface

Figure 1 depicts the 2D Grid-Based Tower Defense's main interface. Two options are available to the user in the main interface: 1 and 2 are the start and exit buttons. The Exit button is used to end the game, while the Start button initiates it. GameManager first calls the Awake() method after loading the game. Its purpose is to allow GameManager to determine which level the Player has chosen from the Level Selection Menu. When the game is being debugged, if no level is chosen, the system will typically choose level 0, which is the default level. The system loads the Select level interface after the user clicks the Start button (Figure 2). The user can select a level in the Select Level interface. The levels will be generated using the GridManager.

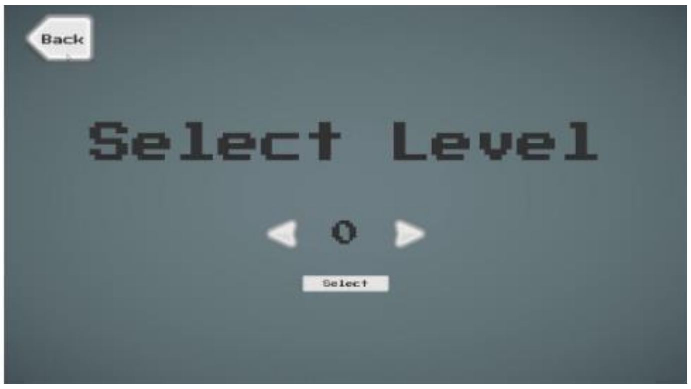


Figure 2. Select Level Interface

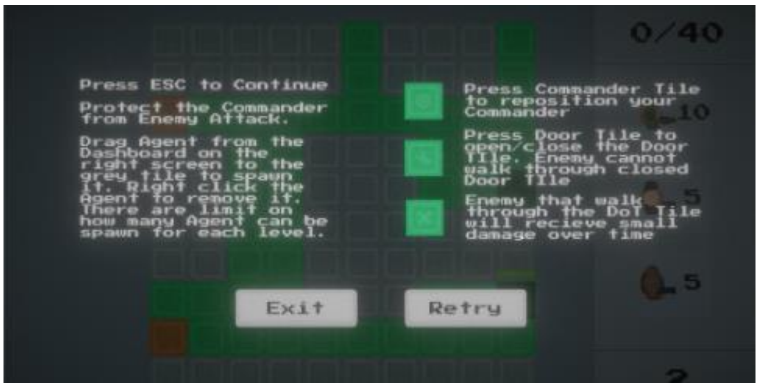
GridManager is started in order to generate the map. Using a 2D character arrangement that represents each of the game's tile types, this function creates a map. Based on information from the database, this function will also offer an index for the commander's starting location. (2) The AddTilesToArray() method stores all generated tile tiles in a 2D array after they have finished generation. Later, when the program generates destination-based pathfinding, it will make use of this 2D array. After that, to make managing the EnemySpawnerTile easier, all opponent spawners will be registered in AddEnemySpawnerTileToList().All DoorTiles will also go through the same procedure for the same reasons. The Commander object will be moved to the position of the CommanderTile currently being used after all of these operations using UpdateCurrentCommanderTile(). InitiateNeighborsForTiles should be called after that (). This method iterates over each tile, setting the attributes of their neighbors in accordance with the tiles that are right next to them.

Destination-based pathfinding can be produced once all the techniques required to create the map and its parts have been analyzed. GridManager will execute the two methods required to create a destination-based pathfinding on the created map using the GoalbasedPathfinding() method. These are the methods GenerateFlowField() and GenerateHeatmap().

In this game, the distance to the commander's position based on the available path is set using the GenerateHeatmap() method, which sets the distance for each tile that an enemy agent can walk from the target location. The tiles LowgroundTile, DamageOverTimeTile, Opened DoorTile, EnemySpawnerTile, and CommanderTile are all able to be traversed by the enemy. All of these tiles have a green block, with the exception of the Enemy Spawner, which has an orange block, and they are all marked with the Boolean variable isLowground as true. The function begins with the CommanderTile, zeroes out the Tile Spacing value, and marks the tile as having been visited. The CommanderTile was then added to the queue. This function will check the tiles above the queue, including the unmarked isLowground tile, even if the queue is not empty. Set the distance value for that Tile's neighbor to the previous Tile distance value plus 1, then reposition the tile in the queue. The procedure is finished when the queue is empty.

It is necessary to set the direction values for each tile after the distance values for each tile have been set. In this game, the target location is the Commander's current position, so the direction value is a 2D vector that implies the path that should be taken from each tile to reach that position. By figuring out the difference between the plots' and adjacent tiles' distance values, the distance value is produced. Users can create the shortest path using that value as a starting point. This time, the computation is carried out iteratively for each Tile in the 2D array that stores all of the Tiles.

GameManager will carry out the Start() method execution after the Awake() method has been accomplished. On the Start() method, several operations must be carried out. GameManager must first check to make sure the game has not been paused or ended. The database must then be loaded with information specific to the level that was chosen. The player must then load the Dashboard with all of the Agents they have chosen. The player must then begin the Enemy Counter, which by default starts at 0 and counts both the number of enemies present overall at the level they have chosen and the enemies they have already defeated. The player's initial number of Photons must then be updated based on the level they have chosen, while also updating the Photon counter. In the future, players will occasionally receive Photons. The Pause panel (Figure 3) must be closed by GameManager before proceeding.


Press ESC to Continue

Protect the Commander from Enemy Attack.

Drag Agent from the Dashboard on the right screen to the grey tile to spawn it. Right click the Agent to remove it. There are limit on how many Agent can be spawn for each level.

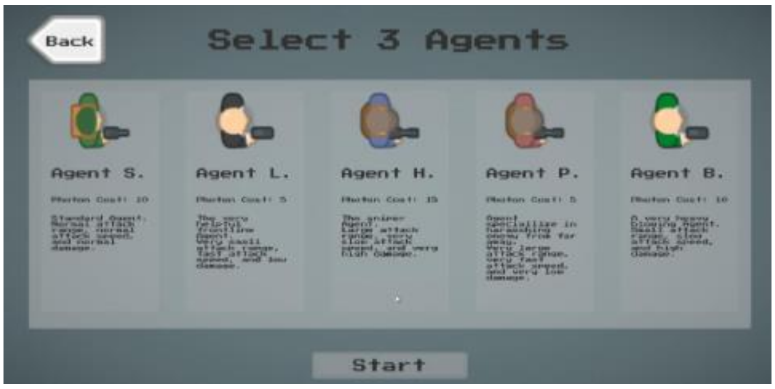
Press Commander Tile to reposition your Commander

Press Door Tile to open/close the Door Tile. Enemy cannot walk through closed Door Tile

Enemy that walk through the DoT Tile will recieve small damage over time

Figure 3. Pause Panel

The first thing GameManager does is determine if the player has chosen an agent from the Agent Selection Menu or not (Figure 3). GameManager will replace the default Agent selection with the player-selected Agent if a player has chosen an Agent (Figure 4). Then, based on each Agent that will be played, GameManager will create a new object from the AgentUI class. These items will be displayed as options on the Dashboard so that the player can drag them to spawn Agents.


Select 3 Agents

Agent S.
Photon Cost: 10
Standard Agent. Normal attack range, normal attack speed, and normal damage.

Agent L.
Photon Cost: 5
The very helpful frontline Agent. Very small attack range, fast attack speed, and low damage.

Agent H.
Photon Cost: 15
The sniper Agent. Large attack range, very slow attack speed, and very high damage.

Agent P.
Photon Cost: 5
Agent specialize in harassing enemy from far away. Very large attack range, very fast attack speed, and very low damage.

Agent B.
Photon Cost: 10
A very heavy blowing Agent. Small attack range, slow attack speed, and high damage.

Figure 4. Select Agents Interface

The Player must drag the Agent from the Dashboard onto the map in order to own it (Figure 5). The Dashboard displays AgentUI components. A component called AgentUI was created to control the drag-and-drop method of spawning Agents. A distinct Agent is depicted by each generated AgentUI. OnBeginDrag(), OnDrag(), and OnEndDrag are the three methods that AgentUI uses to handle drag and drop input ().

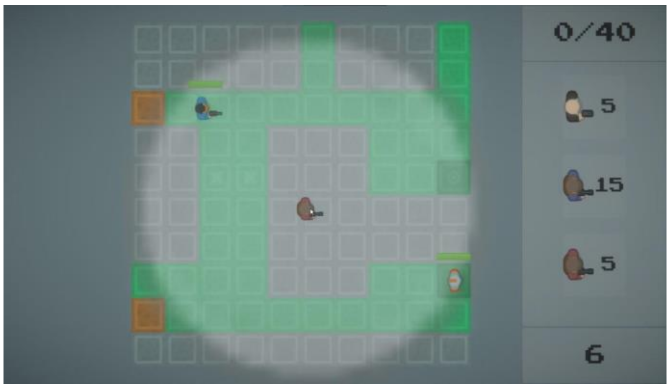


Figure 5. Player Spawn an Agent

A player must right-click an Agent in order to remove it from the game. The Agent's OnMouseOver() method is called whenever the player moves the mouse cursor over the Agent. If the player makes a right-click while OnMouseOver is active, the agent is successfully removed, its number of placements is decreased, and it is removed from the current Tile and destroyed. Players must click on a vacant, not-on-cooldown CommanderTile in order to move the Commander. Its OnMouseDown() method is called when the CommanderTile is clicked. First, the CommanderTile is checked to see if the cooldown has started or not. If not, it requests that GridManager use the ChangeCommanderTile() method to swap out the CommanderTile that is currently in use for itself, moving the Commander to that position, and initiating a rest period for itself.

There are two possible states for each DoorTile: open and closed. In contrast to a closed DoorTile, which is indicated by a gray sprite and isLowground is false, an open DoorTile is indicated by a green sprite and isLowground is true, preventing enemy agents from passing through (Shown in Figure 6). Additionally, every DoorTile has a group id, and multiple DoorTiles can share the same group id. By clicking on the DoorTile, users can make it active. The DoorTile's OnMouseDown() method is called when the tile is clicked. To begin with, this method determines whether or not DoorTile is on cooldown. If not, it will request that GridManager use the ToggleDoorGroup() method to turn on all DoorTiles that share the same group id.

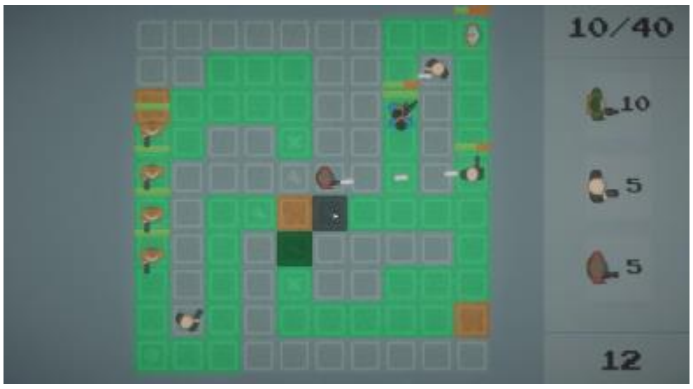


Figure 6. Player Toggle a Door Tile

The enemy agent must be able to walk in any direction toward the Commander and shoot him when he comes within detection range. The FixedUpdate() method deals with both. In the absence of a commander, the enemy will turn to the commander. Otherwise, it will halt its movement and begin shooting the Commander. The enemy must first turn in the right direction before he can move. The Enemy can determine the ideal rotation by using the RotateCharacter() method. Depending on whether they only enter different tile tiles or only pass through the center of the tile, enemies rotate between 2 different directions. An enemy will rotate toward the tiled tile's center after just entering a new Square until he has reached all the way around it. Once it has circled its center, it rotates in the direction indicated by that tile until it comes to another tile. Before converting the direction vector value to a float value that represents the angle required to rotate the enemy in the appropriate direction, normalizing the vector value first determines whether the enemy rotates toward the center of the tile or based on the direction provided by the tile. The Enemy then moves using the MoveCharacter() method once the enemy agent is facing the appropriate way. Based on the selected speed value, the character will move. He must first turn in the direction of the Commander before he can shoot him. The Enemy then uses the Shoot() method to shoot the Enemy after confronting the Commander. The Commander will be fired at predetermined intervals by the Shoot() method. The enemy creates a projectile every time there is a pause, and it is directed in the direction of the commander. Figures 7 and 8 depict the interface of the game over screen after completing the mission and eliminating

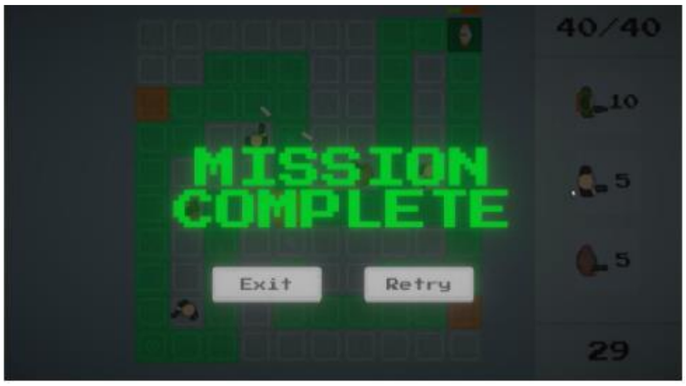


Figure 7. Game Over Screen (Mission Complete)

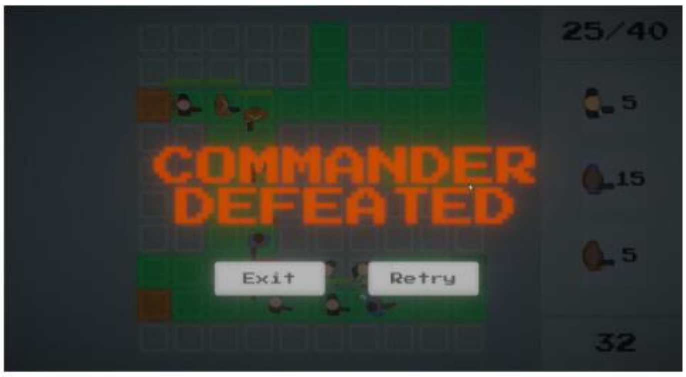


Figure 8. Game Over Screen (Commander is Defeted)

4. CONCLUSION

The creation of this application can be used to draw several conclusions, such as the implementation of objective-based pathfinding by figuring out how far each tile is from the target location. We can then compute the 2D vector value, which represents the direction that needs to be taken from each tile to the target position, using the distance value. Another thing that can be deduced is that this algorithm is very well suited for handling issues in which numerous objects need to find the shortest route to the same location.

ACKNOWLEDGEMENT

Researchers acknowledge the assistance of the President's University LRPM. The President's University Internal grant "Research and PkM Beginner Lecturers" provided funding for this research.

REFERENCES

[1] Alayed, H., Frangoudes, F., Neuman, C.: Behavioral-based cheating detection in online firstperson shooters using machine learning techniques. In: 2013 IEEE Conference on Computa-tional Intelligence in Games (CIG), pp. 1–8, 2013
[2] Schaul, T.: A video game description language for model-based or interactive learning. In: 2013 IEEE Conference on Computational Intelligence in Games (CIG), pp. 1–8, 2013
[3] Charlotte. IEEE International Games Innovation Conference (IGIC), 2012. 7~9 Sept. 2012, Rochester, New York, USA. Piscataway, NJ: IEEE. Available online at http://ieeexplore.ieee.org/servlet/opac?punumber=6319458, 2012.
[4] Wright, W. Tower Defense Game Genre: 6 Characteristics of TD Games MasterClass. Retrieved November 1, 2021, from https://www.masterclass.com/articles/tower-defense-game-video-game-

guide#6-characteristics-of-the-tower-defense-video-game-genre], 2021

[5] P. Avery, J. Togelius, E. Alistar and R. P. van Leeuwen, "Computational intelligence and tower defence games," 2011 IEEE Congress of Evolutionary Computation (CEC), (2011): pp. 1084-1091, doi: 10.1109/CEC.2011.5949738.

[6] Brich, J., Rogers, K., Frommel, J. et al. LiverDefense: how to employ a tower defense game as a customisable research tool. Vis Comput 33, 429–442, 2017

[7] Zhang, Baoyi. Exploring the Attractive Factors of Mobile Tower Defense Games. 10.2991/icassee-18.2018.107, 2018

[8] Y. Du et al., Automatic level Generation for Tower Defense Games, IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), (2019): pp. 670-676, doi: 10.1109/ITNEC.2019.8728989, 2019

[9] Firas Safadi, Raphael Fonteneau, Damien Ernst, "Artificial Intelligence in Video Games: Towards a Unified Framework", International Journal of Computer Games Technology, (2015): vol. 2015, Article ID 271296, 30 pages. https://doi.org/10.1155/2015/271296

[10] Asfarian, A., Ramadhan, W., Putra, W., Raharjanto, G., & Frisky, R, Creating a Circular Tower Defense Game: Development and Game Experience Measurement of Orbital Defense X.Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control, (2019): 4(3)

[11] A. N. Sabri, N. H. M. Radzi and A. A. Samah, "A study on Bee algorithm and A algorithm for pathfinding in games," 2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE), (2018): pp. 224-229, doi: 10.1109/ISCAIE.2018.8405474.

[12] Durant,. Understanding Goal-Based Vector Field Pathfinding. Game Development Envato Tuts+. Retrieved November 1, 2021, from https://gamedevelopment.tutsplus.com/tutorials/understanding-goal-based-vector-field-pathfinding--gamedev-9007

[13] Smołka, J., Miszta, K., Skublewska-Paszkowska, M., & Łukasik, E. A* pathfinding algorithm modification for a 3D engine. MATEC Web of Conferences, (2019): 252, DOI: https://doi.org/10.1051/matecconf/201925203007

[14] G. Teixeira Galam, T. P. Remedio and M. A. Dias, "Viral Infection Genetic Algorithm with Dynamic Infectability for Pathfinding in a Tower Defense Game," 2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), (2019): pp. 198-207, doi: 10.1109/SBGames.2019.00034.

[15] D. Kurniadi, A. Mulyani and R. S. Maolani, "Implementation of Pathfinding Algorithm in Sundanese Land History Educational Game," 2021 2nd International Conference on Innovative and Creative Information Technology (ICITech), (2021): pp. 145-150, doi: 10.1109/ICITech50181.2021.9590181.

[16] Subrando, Temmy & Prasetyatama, F.A. & Fitrianah, Devi. Implementation of a* algorithm within navigation mesh in an artificial intelligence based video games. International Journal of Engineering and Technology(UAE). (2018): 7. 3249-3254. 10.14419/ijet.v7i4.15084.

[17] A. Candra, M. A. Budiman and R. I. Pohan, Application of A-Star Algorithm on Pathfinding Game, Journal of Physics: Conference Series, (2021): Volume 1898, Issue 1, article id. 012047, doi: 10.1088/1742-6596/1898/1/012047