

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/266317808>

Pathfinding in Partially Explored Games Environments

Conference Paper · September 2014

DOI: 10.1109/UKCI.2014.6930151

CITATIONS

0

READS

587

4 authors:



John Stamford
University of Hull

4 PUBLICATIONS 0 CITATIONS

[SEE PROFILE](#)



Arjab Singh Khuman
De Montfort University

16 PUBLICATIONS 55 CITATIONS

[SEE PROFILE](#)



Jenny Carter
University of Huddersfield

44 PUBLICATIONS 278 CITATIONS

[SEE PROFILE](#)



Samad Ahmadi
De Montfort University

44 PUBLICATIONS 637 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Intelligent grey dynamic scheduling for complex products based on sense-information of the “Internet of things” [View project](#)



Deep machine learning [View project](#)

Pathfinding in partially explored games environments

The application of the A* Algorithm with Occupancy Grids in Unity3D

John Stamford, Arjab Singh Khuman, Jenny Carter & Samad Ahmadi
Centre for Computational Intelligence
School of Computing
De Montfort University
Leicester, United Kingdom

Abstract—One of the key aspects of games development is making Non-Playable Characters (NPC) behave more realistically in the environment. One of the main challenges is creating an NPC that is aware of its surroundings and acts accordingly. The A* Algorithm is widely used in the games development community to allow AI based characters to move around the environment however unlike real characters they are often given information about the environment without having to explore. This paper combines the use of the A* Algorithm with the occupancy grid technique to allow Non-Playable Characters to build their own representation of the environment and plan paths based on this information. The paper demonstrates the application of the approach and shows a range of testing and its limitations.

Keywords—A* Algorithm, AI, Games Development, Mobile Robots, Occupancy Grids, Unity3D

I. INTRODUCTION

The paper aims to discuss the application of the A* Algorithm [1] in partially explored environments with the aim of making pathfinding for NPCs in a 3D games engine. The proposed pathfinding system is aimed to simulate more realistic planning and decision making processes based on the information known about the environment. This will result in the proposed system making assumptions about the best path and then having to recalculate the path as it learns more about the environment.

One of the distinct differences between a human player and a NPC has is that often the NPC is given information about the whole environment prior to solving the pathfinding process [2]. This gives the NPC an unnatural awareness of its environment which can result in an unrealistic behaviour when navigating. The proposed system aims to address this issue as the NPC has to plan the path to a given target through an fully unknown environment. As the NPC moves along the path it builds an internal map of the environment and recalculates the best path based on this internal map. The proposed system will build on the work of [3] which discusses the application of occupancy grids in robotic systems. As a result the NPC will store information about the environment and will apply pathfinding to its internal knowledge rather than the environment.

The system was developed in Unity3D and simulates real world interaction through the application of its physics engine. The NPC simulates sensors such as Sonar and LIDAR through the use of a Raycast function. The NPC's destination is set through the use of a placeable target object. The environment

can be constructed from any objects which have a 3D collider that allows the Raycast function to interact with.

II. LITERATURE REVIEW

A. Pathfinding algorithms in games

The A* algorithm is widely used in game development as a way of controlling NPC movement [2][4][5][6]. The A* algorithm is similar to that of Dijkstra's algorithm [7] however it uses a heuristic to determine the optimal search area. The most common implementation of the A* algorithm uses the Manhattan approach where the heuristic is calculated by the number of nodes from the current position to the target. The use of an appropriate heuristic allows the algorithm to find the shortest path to the target and has a distinct advantage over the Dijkstra algorithm.

Typically pathfinding in games is achieved through the use of either predefined nodes [8] or the use of a tile based system [2]. When using a node based system the developer is usually required to position path nodes throughout the environment as shown in Fig. 1. The limitation of a node based approach is that the behaviour of the NPC is predictable and that in some cases the NPC will not have full access to the environment. When using a tile based approach as shown in Fig. 2 the NPC is often given information about the whole environment, again making the behaviour unnatural.

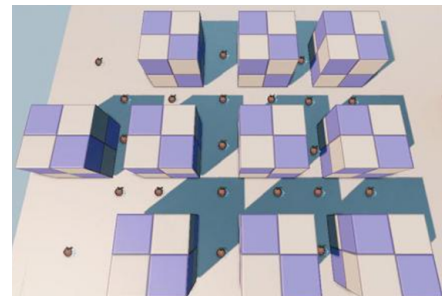


Fig. 1. Node based methods [8]

In addition to the A* algorithm further pathfinding approaches include various D* alternatives such as [9][10] and further work such as [1][11] which are more widely used in mobile robotics.

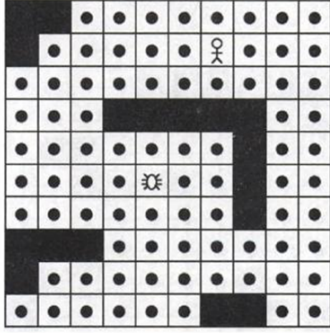


Fig. 2. Tile based methods [2]

B. Mobile robots approach

The main mobile robotics feature this paper aims to implement is the use of an occupancy grid [3]. In mobile robotics the occupancy grid allows a system to create an internal map of the environment which can account for noise from the sensor readings. The implementation aims to take readings from the environment in a similar way to Sonar and LIDAR however noise would be almost non-existent. Therefore the occupancy grid that the system builds will be used to convert high resolution readings into a low resolution map which the NPC can use for navigation.

With mobile robotic systems the positioning can be achieved through two key aspects [12]:

- Relative Position Measurements (dead-reckoning).
- Absolute Position Measurements (reference-based systems)

Within the games environment the NPC will have the distinct advantage over a mobile robot as it will know its exact position in the world.

III. IMPLEMENTATION

A. Architecture

The proposed system uses a hybrid approach [13] as the system will sense the environment, plan a path and move. If the NPC detects a change in the environment the system will re-evaluate the path.

The system was developed in Unity3D and uses the Raycast function to act as sensors. The NPC simulates three sensors as shown in Fig. 3. The sensors sweep through a 90 degree angle giving the NPC a viewable range of 270 degrees.

The Raycast function acts in a similar way to a Sonar or LIDAR based system in that a single point is projected outwards. If the Raycast collides with an object the system records the collision point. The NPC is able to detect any object in the environment which is set as a physics collider as shown in Fig. 4.

When the simulation starts the only information the NPC is given about the environment is the position of the target destination.

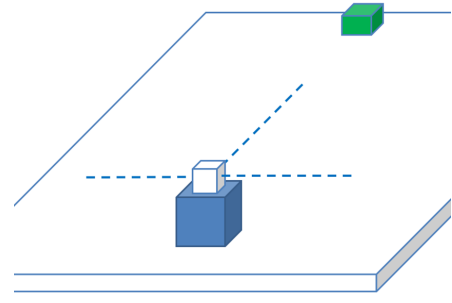


Fig. 3. Simulated sensor positioning

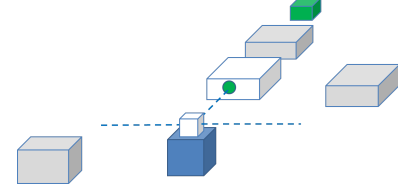


Fig. 4. Simulated sensor detection

B. Occupancy grids

The occupancy grid is stored as a multidimensional array that represents a grid of 100 x 100 double data types and represents the certainty or uncertainty of objects in the environment. This gives the NPC a memory space of 10,000 doubles which requires 9.8 Kilobytes of storage. Each value represents a given area of the environment and the size of each area can be adjusted through the use of a resolution variable. The values stored represent the probability that the square is occupied and with the NPC not been given any prior information about the environment each square is defaulted to 0.5. This represents that each square is neither occupied or empty.

During every iteration of the simulation the NPC takes sensor readings. If the sensors detects an object the system calculates the occupancy values for each square in relation to the sensor's reading position. To reduce computational costs the system selects a subsection of the occupancy grid as shown in Fig. 5 by calculating the minimum and maximum points of the occupancy grid to calculate.

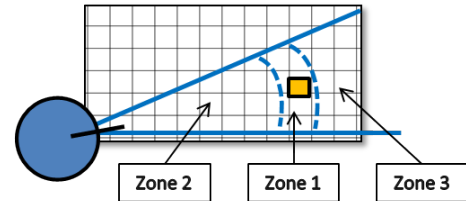


Fig. 5. Subsection of the Occupancy Grid Selected

For the occupancy grid values to be updated each of the squares from the sub selection are calculated based on where they are in relation to the sensor cone as shown in Fig. 5. The calculation requires two variables, a distance tolerance for zone 1 and an angle tolerance. When cycling through each of the squares the system checks if the square is in the sensor cone. If the square is in the sensor cone the system then calculates which zone the square is in. This requires the conversion of

coordinates from Cartesian space to Polar space.

The values to update the occupancy grid are calculated based on a Bayesian approach [14]. If the square is in zone 1 the system calculates the probability that it is occupied as in (1) where R represents the range of the sensor reading, r represents the range of the square, β represents the angle of tolerance and α is the difference in angle. The calculations for zone 1 also limits the occupancy value using a *MaxOccupancy* which is set at 0.98.

$$P(\text{Occupancy}) = \frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2} \times \text{MaxOccupancy} \quad (1)$$

If the square is within zone 2 the system calculates the likeliness of the square been empty. This is similar to the calculations as shown for zone 1 however the focus is on how probable the square will be empty (2).

$$P(\text{Occupancy}) = 1 - \frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2} \quad (2)$$

Zone 3 represents areas of the sensor cone that are not known from the sensor reading as it is beyond the tolerance accounted for in zone 1. The system was designed to limit the number of squares in the sub-section that would be classified as zone 3. However if the system does perform a calculation it simply reinforces the previous occupancy value as in (3).

$$P(\text{Occupancy}) = P(\text{Occupancy}_{n-1}) \quad (3)$$

C. Updating the Occupancy Grid

Once the $P(\text{Occupancy})$ value has been calculated the system updates the stored occupancy value for each of the squares. Reference [14] discusses the application of either Bayesian, Dempster-Shafter or HIMM methods. This implementation uses the Bayesian approach as in (4) where $P(H|s_{n-1})$ represents the previously stored occupancy value and $P(s_n|H)$ represents the calculated sensor value based on the sensor model.

$$P(H|s_n) = \frac{P(s_n|H)P(H|s_{n-1})}{P(s_n|H)P(H|s_{n-1}) + P(s_n|\neg H)P(\neg H|s_{n-1})} \quad (4)$$

As the NPC explores the environment its internal map is updated and can be visually represented as shown in Fig. 6. The map represents the probability of each square being occupied.

D. A* search algorithm

The system applies a cost based search method in the form of the A* search algorithm [1]. The algorithm functions by calculating the path based on the least cost. The cost for each

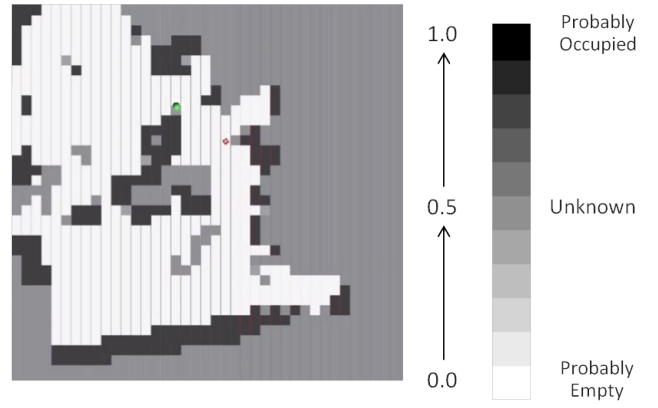


Fig. 6. Example of the NPC's Sensor Map

point in the path is calculated as in (5) where g is the movement cost and h is a heuristic value.

$$f = g + h \quad (5)$$

The cost of moving one square is consistent with horizontal and vertical movements having the same cost value and diagonals costs are based on $\text{cost}^2 = \text{cost}X^2 + \text{cost}Y^2$ as shown in Fig. 7.


14	10	14
10		10
14	10	14

Fig. 7. Costs of each movement

In addition to the movement cost the heuristic cost needs calculating. The value for each square is calculated using the Manhattan approach as in (6) where dx is the difference between the target's X position and the NPC X grid position, and where dy is the difference between the Y grid positions.

$$h = dx + dy \quad (6)$$

In the simulation the target destination does not move and therefore the heuristic values are calculated at the start of the simulation as shown in Fig. 8.

Using the movement cost and the heuristic values the system explores the best route to the target. To achieve this the system uses open and closed lists of squares. The process starts by adding adjacent squares of the NPC's starting position to the open list. As squares are added to the open list the system also adds a relation to these squares identifying itself as the square's parent. The system then selects the square in the open list with the lowest cost value. Based on this square the system then adds all adjacent squares to the open list. Once all adjacent squares have been added to the open list the current square is removed from the open list and is added to the closed list.

5	4	3	2	1	2
4	3	2	1	0	1
5	4	3	2	1	2
6	5	4	3	2	3
7	6	5	4	3	4
8	7	6	5	4	5

Fig. 8. Heuristic costs

This process repeats until one of the adjacent squares is the target destination. Prior to adding adjacent squares to the open list a check is performed on the occupancy grid value for that square to see if it is classified as an obstacle as in (7). If the square is classified as an obstacle it is not added to the open list and therefore will not be used in the pathfinding process.

$$f(x) = \begin{cases} 1 & x > 0.8 \\ 0 & otherwise \end{cases} \quad (7)$$

	14	13	12		
14	13	12	11		
15	14	13	12		
16	15	14			
17	15				
	17				

Fig. 9. Completed Path based on the A* Algorithm

Once the path has been calculated based on the A* Algorithm, Fig. 9, the NPC will move along the path by moving from one square to the next. As the NPC is moving the occupancy grid is constantly been adjusted based on sensor readings from the NPC. If any of the path squares become classified as an obstacle the simulation will recalculate the next best path as shown in Fig. 10.

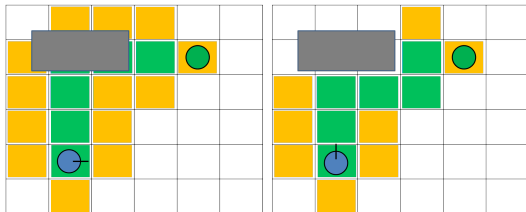


Fig. 10. Example of Path Recalculation

E. Data Structures

For the A* Algorithm to function the system needed to manage more data about the squares in addition to the occupancy values. This involved the creation of a custom data

```
public struct PathSquare {
    public int id;
    public int h;
    public int g;
    public int f;
    public int x;
    public int y;
    public int parent_cost;
    public bool isObstacle;
    public bool isOpen;
    public bool isStart;
    public Vector2 parent;
    public int parent_id;
}
```

Fig. 11. Construction of the PathSquare structure

```
List<PathSquare> oList = new List<PathSquare>();
List<PathSquare> cList = new List<PathSquare>();
```

Fig. 12. Example of how the Open and Closed lists are created

type as shown in Fig. 11. This structure allows each square to contain additional information such as which square is its parent and additionally identify the square as an obstacle.

Based on this custom data type both the open and closed lists were created as lists as shown in Fig. 12.

F. Visual representation

The implemented system in Unity3D displays the occupancy grid through the use of a two dimensional mesh which is generated in real-time. The system builds the mesh based on the size of the occupancy grid and the dimensions for each square is determined by the resolution of the occupancy grid. Each square consists of four vertices and two triangles. Given a grid of 100 by 100 this means that the mesh contains 20,000 triangles. Each pair of triangles that form a square are UV mapped to an appropriate texture. The system currently supports 10 different shades for occupancy values and additional textures to represent squares on the open list, closed list and for squares to be used as the path points.

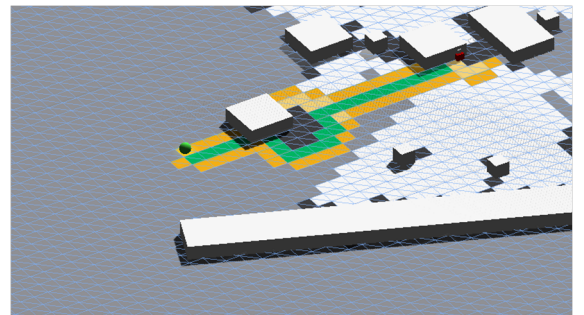


Fig. 13. Displaying the mesh in Unity3D

One of the main limitations of visually representing the occupancy grid is that a mesh in Unity3D is limited to 64,000 vertices which would limit the size of the occupancy grid however the occupancy grid is only displayed for debugging purposes.

IV. TESTING

A. Map Building and Navigation

The system was tested in a variety of different environments ranging from randomly placed objects to environments constructed based on real-world examples such as an office environment. The initial tests focused on the NPC's ability to navigate around randomly placed obstacles as shown in Fig. 14. Regardless of where the target was placed in the environment the NPC was able to effectively build an internal map of the environment and navigate correctly.

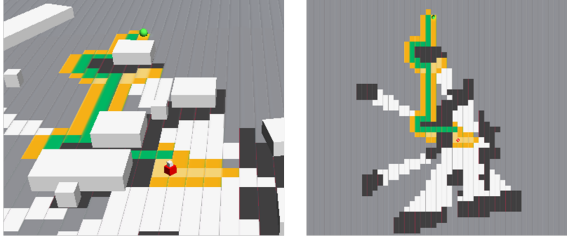


Fig. 14. Test Environment - Random Objects

Further testing involved an environment which would force the NPC to recalculate its path when it discovers more obstacles as it moves through the environment. As shown in Fig. 15 the NPC initially plots a path which would make it collide with a wall however as the NPC turns towards its path the sensors detect an obstacle. This updates the occupancy grid and the path is no longer valid resulting in the NPC recalculating a new path, Fig. 16.

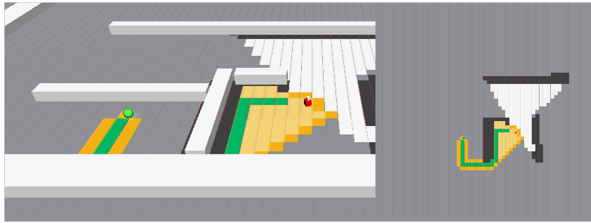


Fig. 15. Navigation Stage 1

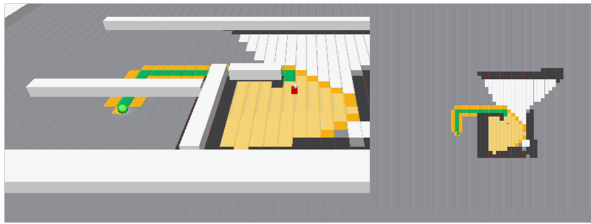


Fig. 16. Navigation Stage 2

As the NPC gets closer to the target it again identifies another obstacle and recalculates the path as shown in Fig. 17. Based on the NPC's knowledge of the environment it is able to correctly identify the best path and complete the task.

An additional test environment was constructed to represent an office environment. The layout consisted of a large room with smaller offices along the sides with the NPC having to move from one side to the other as shown in Fig. 18. Again the NPC was able to successfully navigate the environment.

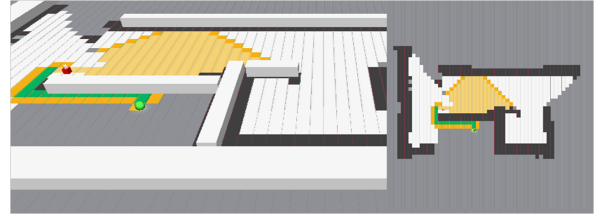


Fig. 17. Navigation Stage 3

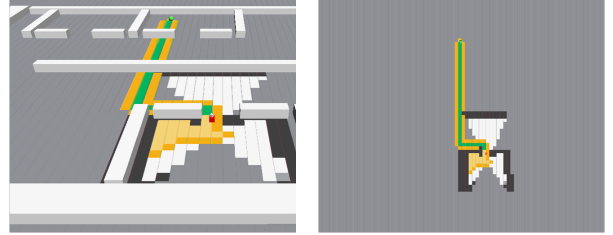


Fig. 18. Test Environment - Office

B. Changing Environment

Further tests included making adjustments to the environment at run-time. Initial testing investigated how the system would adapt its path based on the target moving around the environment. The tests involved moving the target to random places around the map. Additional testing involved moving obstacles around the environment whilst the NPC was moving, this included placing obstacles in the NPC's path.

The observed results found that the NPC could effectively plot a path to the target that had been moved during the run time. The limitation was that the NPC's path had to be blocked to force it to recalculate the path. Additionally if the target was moved while the path was being calculated some times it would not converge on the target. Further observations of the behaviour showed that the NPC could react to dynamically changing environments. If the path was blocked in real time the NPC would recalculate the path accordingly.

C. Different Heuristics

Additional tests involved the inclusion of the occupancy values in the heuristic calculations. Initial tests used the addition of a value based on the occupancy values as in (8). This resulted in the search algorithm favouring explored parts of the environment and resulted in an increased time to converge on the target without improving the outcome of the path calculation.

$$h_n = dx + dy + (OCvalue_n \times 10) \quad (8)$$

D. Performance

Testing of the performance was carried out to investigate the impact of pathfinding on performance. Five tests were carried out on each of the three test maps as shown in Table I with Map 1 relating to Fig. 14, Map 2 referring to Fig. 15 and Map 3 referring to Fig. 18. From the five tests the average Frames Per Second (FPS) were recorded.

TABLE I. PERFORMANCE TESTING

	Map 1	Map 2	Map 3	Average
Moving	29	33	28	30
Pathfinding	50	20	14	28

Measurement is Frames Per Second

The results suggest that on average the pathfinding algorithm had no impact on computational performance however the results for each map suggest that in some cases this is not true. The results for Map 2 and Map 3 show that the pathfinding process significantly reduced the FPS however when the NPC was moving the FPS increased. The results from Map 1 suggest that in some cases the pathfinding process did not impact performance however the observations of the tests highlighted the relationship between FPS and distance to the target. As the number of squares in the open and closed lists increased performance decreased.

V. CONCLUSION

The paper shows the combined use of occupancy grids and the A* search algorithm for pathfinding in unexplored environments. The proposed system was able to function to a good standard in the environments presented to the system. In terms of computational requirements, the system was able to efficiently navigate the environment however the system could be further improved through optimising the path recalculation function.

The system was able to simulate sensor readings through the use of the Raycast function and through the use of the occupancy grid the NPC was able to build an internal map of the environment. The NPC was then able to successfully navigate the environment whilst making adjustments when needed.

From the test results it can be concluded that the system performs to a good standard in terms of computational processing however it is evident that the process could be further optimised. Additional work could be carried out looking at the management of data which represents the environment and potentially taking advantage of the game engine's ability for multi-threaded applications.

The paper concludes that a system which has to learn about the environment gives the ability for a NPC to navigate the environment in a more realistic manor, including the ability for the NPC to make mistakes.

A. Future research

Future research could include the use of various Intelligent Systems approaches in the updating process of the occupancy grid values. Additionally Fuzzy Logic can be applied to the sensor model to further improve the interpretation of the environment. It has been considered that the system could also make use of a Artificial Neural Network (ANN) and by using the occupancy grid values as inputs the ANN could potentially be used for navigation or object recognition.

The system currently does not take into account elevation of the environment and therefore could be further investigated. This could potentially be through either updating the occupancy values on the sensor data based on elevation or

alternatively the application of a three dimensional occupancy grid.

With the system being based on mobile robotic techniques the authors would like to apply the system to a mobile robot along with testing various other pathfinding algorithms.

Further refinements to the system could help improve computational performance of the pathfinding process. This could additionally include modifying the system to plan both long and short term paths. This would help with both computational costs and also make the system move more realistically.

ACKNOWLEDGMENTS

The authors would like to thank their colleagues at De Montfort University for their support throughout this work and also for the opportunity to explore this topic.

REFERENCES

- [1] Hart, P.E.; Nilsson, N.J.; Raphael, B., *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics, vol.4, no.2, pp.100,107, July 1968
- [2] Bourg, D.M., (2004) *AI for Game Developers* 1 Edition. O'Reilly Media
- [3] Elfes, A., *Using occupancy grids for mobile robot perception and navigation*, Computer, vol.22, no.6, pp.46,57, June 1989
- [4] Jie Hu; Wang gen Wan; Xiaoqing Yu, *A pathfinding algorithm in real-time strategy game based on Unity3D*, 2012 International Conference on Audio, Language and Image Processing (ICALIP), pp.1159,1162, 16-18 July 2012
- [5] Khantanapoka, K.; Chinnasarn, K., *Pathfinding of 2D & 3D game real-time strategy with depth direction A* algorithm for multi-layer*, SNLP 2009. Eighth International Symposium on Natural Language Processing, pp.184,188, 20-22 Oct. 2009
- [6] Jia-jia Tang; Liang Chen; Ling Yan, *A heuristic pathfinding approach based on precomputing and post-adjusting strategies for online game environment*, Games Innovations Conference (ICE-GIC), 2010 International IEEE Consumer Electronics Society's, pp.1,8, 21-23 Dec. 2010
- [7] Worboys, M.F., (2004) *GIS: A Computing Perspective, Second Edition* 2 Edition. CRC Press.
- [8] Chin, R. (2012) *Beginning iOS 3D Unreal Games Development* 1st Edition. Apress.
- [9] Stentz, Anthony (1994), *Optimal and Efficient Path Planning for Partially-Known Environments*, Proceedings of the International Conference on Robotics and Automation: 3310-3317
- [10] Stentz, Anthony (1995), *The Focused D* Algorithm for Real-Time Replanning*, In Proceedings of the International Joint Conference on Artificial Intelligence: 1652-1659
- [11] Koenig, S.; Likhachev, M., *Fast replanning for navigation in unknown terrain* IEEE Transactions on Robotics, vol.21, no.3, pp.354,363, June 2005
- [12] Borenstein, J., Everett, H.R., Feng, L., and Wehe, D., 1996, *Mobile Robot Positioning: Sensors and Techniques*. Invited paper for the Journal of Robotic Systems, Special Issue on Mobile Robots. Vol. 14, No. 4, April 1997, pp. 231-249.
- [13] Ronny Hartanto, 2011. *A Hybrid Deliberative Layer for Robotic Agents: Fusing DL Reasoning with HTN Planning in Autonomous Robots* 2011 Edition. Springer.
- [14] Murphy, R. (2000) *An Introduction to AI Robotics (Intelligent Robotics and Autonomous Agents)* 1st Edition. A Bradford Book.