Voronoi-based Path Planning based on Visibility and Kill/Death Ratio Tactical Component

Ilya Makarov [1] [0000−0002−3308−8825] ⋆, Pavel Polyakov [1], and Roman Karpichev [1]

National Research University Higher School of Economics, Moscow, Russia
iamakarov@hse.ru, polyakovpavel96@gmail.com, rma-karpichev@rambler.ru

Abstract. We present an obstacle avoiding path planning method based on a Voronoi diagram adjusted with tactical component in a first-person shooter video game. We use a visibility measure to aggregate information on cover positions in offline and online game modes. In order to incorporate online learning based on frag map, we introduce a path finding algorithm minimizing the probability to walk along the path through dangerous zones, and on the contrary, choosing the best positions to shoot when observing a map level. Several implementations of collision free path finding are compared under efficiency, team goal achievements, and path length measures.

Keywords: Navigation Mesh, Path Planning, Voronoi Diagram, Frag Map, Tactical Path Finding

CEUR-WS.org/Vol-2268/paper14.pdf

1 Introduction

Constructing an efficient representation of obscured and navigable space of virtual environments, path planning and path finding problems play an extremely significant role in video games. For a long time, a traditional approach to these problems has been mainly focused only on searching the *shortest* routes from A to B while keeping the number of nodes in a navigation graph as low as possible. However, this is not enough for an AI to look intelligent, especially in terms of believable human-like behaviour [1]. Navigation agents should instead move along the most *appropriate* routes, the ones that take into account tactical properties of surroundings, such as cover positions, lines of fire, kill/death statistics around a given location, enemy presence or positions of allies.

Basically, tactical path finding can be implemented by modifying a cost of traversing between nodes in a navigation graph as described in [2], however, it is not actually going to work at the desired quality level without a proper navigation graph itself. In addition, as shown in [3], tactical path finding is often linked to a tactical decision making strongly impacting on general AI's

⋆ The article was supported within the framework of a subsidy by the Russian Academic Excellence Project '5-100'.

logic based on world's representation method. Therefore, the question is, what a structure of a navigation graph should be like?

The common practice is using grid or way-point graph with a reasonably high density of navigation nodes [3], [4], [5], [6], however, it has a lot of disadvantages, such as time and memory requirements, while both of them are actually very poor at representing a navigable area itself. Finally, way-points usually (though not necessary) require a level designer to place them over a map slowing down a game development process and restricting usage of dynamic environments.

Another way of acquiring a navigation graph is using navigation meshes. They are, on the contrary, very efficient at representing navigable regions while being completely inappropriate for performing a tactical path finding without applying certain structural constraints due to the irregular spacing [7]. A common workflow is providing a level designer an opportunity to mark up some specific areas on a navigation mesh with extra details. Although the situation with manual mark-up requirement is much better compared to way-point systems, in general, it has similar problems, especially with dynamic environments support. As an example of something more intelligent, in [8], the authors describe a fully automatic algorithm to construct a navigation mesh with polygons marked as concealed or normal ones in order to achieve stealthy path planning.

Last but not least, there are also several promising hybrid approaches, such as [9] where a regular way-point graph is generated dynamically around navigation agents and places of interest by sampling positions from a static navigation mesh. After way-points are created they can be treated as a lower level of a path finding hierarchy. The downsides of this method are potential performance problems and duplication of navigation information.

This paper presents a special navigation mesh type, which is designed to overcome all problems described above, called Voronoi-based navigation mesh, first introduced in our previous publication [10]. In robotics and game programming, Voronoi diagrams are often constructed based on obstacle vertices' points and used to create a space partition for a collision-free path finding [11], [12], [13], [14]. Nevertheless, they have never been used in a context of a tactical path finding yet. Since Voronoi diagram is a simple case of a k-nearest neighbour classification rule with $k = 1$, we find putting a constraint on navigation mesh polygons to be Voronoi faces as a perfect way to aggregate tactical information of a navigable space while using an acceptable by performance number of nodes in a navigation graph. All the properties are calculated at Voronoi sites' locations and then propagated to all other points in a corresponding Voronoi face.

Although a number of tactical properties that can be taken into consideration is potentially almost unlimited, building complex tactical path finding and decision making models is not a purpose of this paper and only two of them are covered further: visibility calculated in eight possible directions and kill/death statistics at a given location as one for precomputed and one for online tactical properties, respectively. In the Experiment section, we show that using these two properties only results in about $11 - 13\%$ winning rate gain against an AI, which does not use any tactical information.

While a visibility measure is a relatively simple idea to discover cover positions, the math behind a frag map generation process used as the second tactical property is more complex. Our approach is based on a graph diffusion model, which is applied as a mechanism to restore a density of the winning rate distribution over a map in case of holding a specific location. Graph diffusion models are widely used in network analysis, recommendation systems and signal processing applications. As an example, in [15] the authors introduce a method of diffusion filtering to smooth signals defined on the nodes of a graph. They show that it can be used to improve the performance of recommendation systems. In [16] the authors present a novel difference metric based on the Laplacian exponential diffusion kernel for measuring a distance between two weighted graphs with the same number of vertices thus capturing similarity between several graph-based models of navigation mesh.

Coming back to a frag map, a kill/death statistics of a virtual environment is usually unknown to an AI at a game's start. It means that a graph diffusion model together with the use of a Voronoi-based navigation mesh and penalties for a poor kill/death statistics at specific locations during a path finding is actually an algorithm of *online navigation learning*.

2 Voronoi-based Navigation Mesh

2.1 Definition

A navigable surface S is defined as a connected closed surface with no self-intersections when projected onto the horizontal plane omitting the intersections with the obstacles. Holes in such a surface are possible. Let $P = \{p_0, p_1, ..., p_n\}$ be a set of points uniformly distributed over a navigable surface S with a number of calculated tactical properties for each of them. Then let

$$VD(p_i) = \{x : \|p_i - x\|_2 \leq \|p_j - x\|_2, \forall j \neq i, x \in S \subset \mathbb{R}^3\} \tag{1}$$

be polygons of a surface. A union of these polygons constructed in all surfaces and navigation links connecting them is called a Voronoi-based navigation mesh. Examples of virtual environment representations using this type of navigation mesh are illustrated on Figure 1 and Figure 2.

2.2 Navigation Mesh Storage

As soon as a map is decomposed into a set of non-intersecting navigable surfaces, each of them is stored independently. There are several things to hold in a surface: borders, vertices, edges and faces of a Voronoi diagram and a quad tree used for solution of a point location problem in a real time. Voronoi diagrams themselves are stored in a way similar to *DCEL* format that guarantees $\Theta(n)$ memory footprint, because the number of vertices and edges of Voronoi diagrams are not greater than $2n$ and $3n$ respectively. For simplicity of maintenance, border edges are orientated in a way that a navigable area is always on their right side.

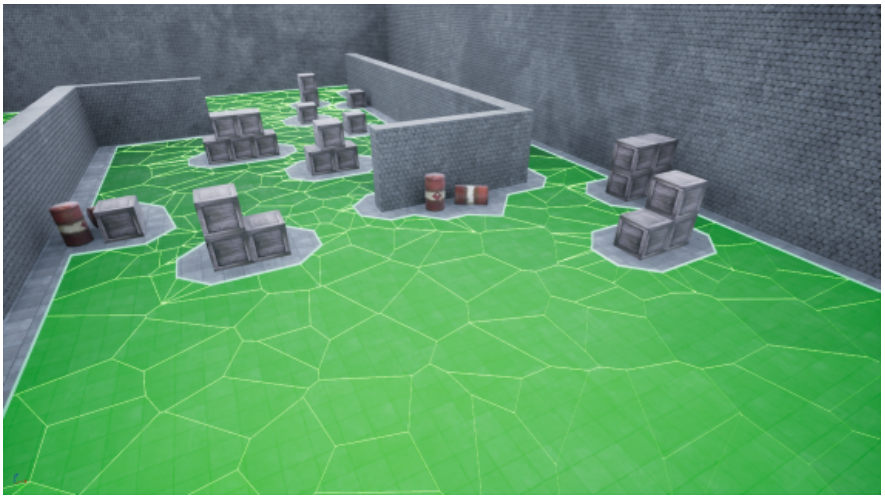

Fig. 1. VD-based Navigation Mesh

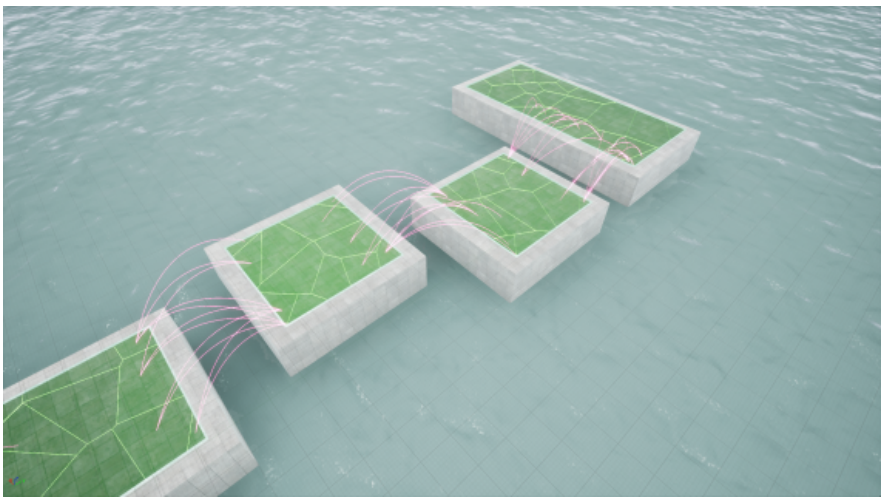

Fig. 2. Navigation Links

2.3 Navigation Mesh Construction

The first stage of constructing navigation mesh is a voxelization of the virtual environment geometry. Stored in an octree, the level geometry can be quickly divided into several groups, each of which is processed independently to construct a height field representing an obstructed and navigable space based on geometry's triangles. The passability of height field's cells is determined by corresponding triangle's normal, height available to a navigation agent and navigation agent's

radius. Once constructed, height fields of each group are merged. In the end of this stage, a grid graph representing a navigable area is obtained.

The second stage is splitting a grid graph into navigable surfaces as they are defined above. Every cell of a grid graph is pushed into a heap ordering elements by ascending order with respect to z-coordinate. Next, cells are picked from the heap one by one. The breadth first search (BFS) with a priority by z-coordinate starts from each of the cells and marks them as belonging to the next surface. An important thing is that during each BFS pass, a cell is considered as visited even if any other cell with the same xy-coordinates was visited during that pass. When such a "visited" cell is encountered, another BFS with a limited depth should be performed from it, in order to discard cells from the current surface being constructed. This strategy is used to produce a better surface split and eliminate float precision problems when simplifying surface's borders. It is also recommended to limit initial BFS depth in order not to end up with too large surfaces as it results in drawbacks in performance. However, it should be taken into consideration that splitting a map into too small surfaces should be avoided as well as it can lead to degenerating of Voronoi structure to a grid. Once the splitting stage is done, several surfaces may be discarded if their area appears to be too small. Eventually, surface's borders are collected and then simplified. This stage takes $O(nlog(n))$ in time, where n stands for a number of cells.

The third stage is a construction of Voronoi diagram in each of the generated surfaces. Voronoi sites are placed at random points on the surface and at each vertex of surface's borders. The latter condition is very important for eliminating float precision problems in the further steps of the algorithm and some degenerate cases, such as a border completely lying inside of a Voronoi face. In addition, a geometry information such as a height above a Voronoi face is copied to Voronoi sites from corresponding grid cells and is then used by a navigation agent to determine whether it is possible to crouch or jump in a particular Voronoi face. Voronoi diagrams themselves are constructed using Fortune's algorithm presented in [17] running in $O(nlog(n))$ time. Once the diagram is built, Voronoi faces turned out to be outside of a surface are discarded. The faces with border intersection are cut off and then are split into convex polygons. Location of a Voronoi site for these polygons is fictive due to the fact they are no longer Voronoi faces and is set to their center. For each constructed diagram a quad tree is built, ending the third stage.

The fourth stage consists of adding additional navigation links between faces lying in different surfaces. Building them between faces adjacent to the same border line can be done in linear time, however the interesting part is the exploration of jump-points on a navigation mesh. This step requires $O(n^2)$ ray casts in the worst case. In practice, this number can be greatly reduced to $O(n)$ on real maps with bounded gravitation field. For each of the polygons lying near the surface border, a set of polygons lying within a range d, that is derived from gravity field, is found, which can be done using a quad tree built on the previous stage. In what follows, a link candidate is eliminated if a segment connecting polygons' sites intersects a non-border edge. Finally, each link candidate is evalu-

ated by one of two possible variants: falling from a ledge or jumping. The process of jumping can be checked using ray casts and physics calculations. Walking off a ledge requires minimizing undesired jumping by comparing of ground heights between nearby polygons.

From this point, a navigation mesh is actually ready to use. Table 1 shows time values required to construct a Voronoi-based navigation mesh on a sample map (without tactical properties computation) depending on a site density (sites placed at vertices of surface's borders are not affected by this parameter).

Table 1. VD-based Navigation Mesh Construction Time

Site Density, n/m^2	0.1	0.2	0.3	0.4	0.5
Time, ms.	248	260	274	292	305

The path-finding based on this step requires specifying several parameters of Voronoi diagram affecting particular smoothness and precision of obstacle avoiding path, seen at Figure 3.

The last stage is a calculating static tactical properties. In this paper, only visibility calculation process is described. We define visibility as a value from 0 to 1 indicating the amount of area visible from a specific polygon within a given range. It is calculated in eight possible directions with a maximum of 0.125 for a single direction. Figures 4 and 5 show examples of an aggregated and directional visibility tactical property respectively.

3 Kill/Death Statistics

3.1 Definition

We define K_i and D_i as numbers standing for the amount of kill and death events occurred in the i-th Voronoi face, respectively. Each new event is processed using a softmax blurring around the exact event's location L:

$$E_i \leftarrow E_i + \frac{\sigma(-C \cdot \|S_i - L_i\|_2)}{\sum_j \sigma(-C \cdot \|S_i - L_j\|_2)}, \quad \sigma(x) = \frac{1}{1 + exp(-x)} \tag{2}$$

where S_i is a location of a Voronoi site corresponding to the i-th Voronoi face, C is a constant indicating the preferred scatter and E_i denotes either K_i or D_i depending on the type of the event. Collected K_i and D_i statistics are then combined into a single number indicating a probability of being killed instead of killing an enemy while occupying a given location using the following formula:

$$P_i = \frac{D_i + B}{D_i + K_i + 2B} \tag{3}$$

where $B > 0$ is the algorithm's sensitivity to new events.

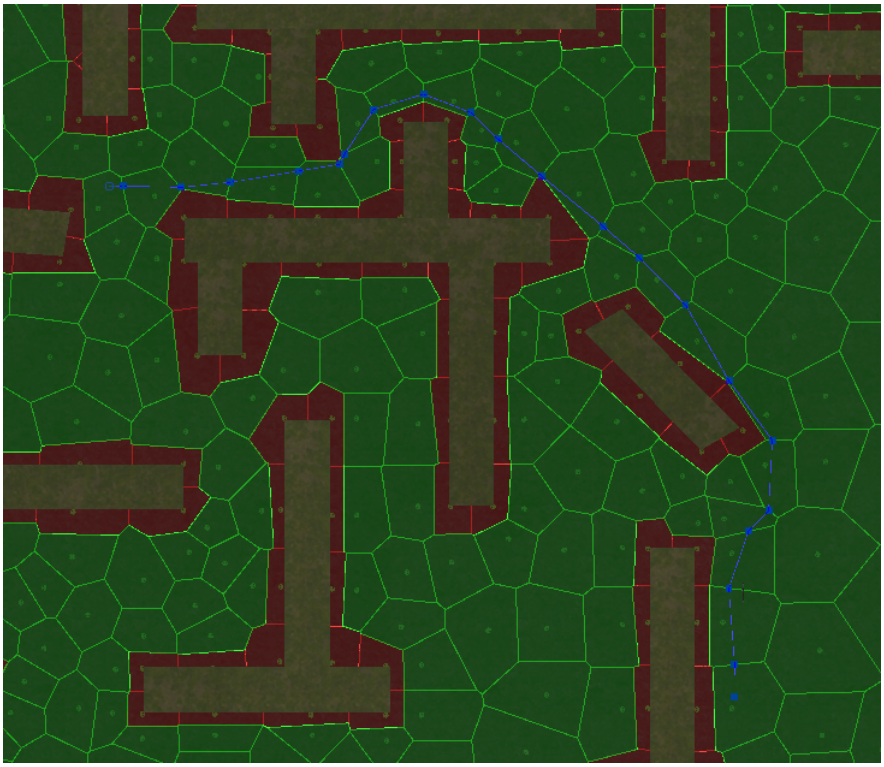

Fig. 3. VD-based Path Finding

3.2 Graph Diffusion

It can take a considerable amount of time before probabilities P_i defined above converge for every Voronoi face of a navigation mesh. In addition, there is a need to somehow discard the statistics an AI gets in the initial stage of the algorithm. To overcome these problems we use the graph diffusion model, as it have been stated in the Introduction section. The following formula is applied:

$$E_i(t + \delta t) \leftarrow E_i(t) + Q \sum_{j \in N_i} \|S_j - S_i\|_2 \cdot (E_j(t) - E_i(t))\delta t \tag{4}$$

where Q is the diffusion coefficient. In fact, double buffering of kill/death statistics should be used for this formula to work correctly.

3.3 Penalty in Path Finding

We utilize the A* algorithm for path finding on a Voronoi-based navigation mesh. The additional penalty for traversing between i-th and j-th polygons in order

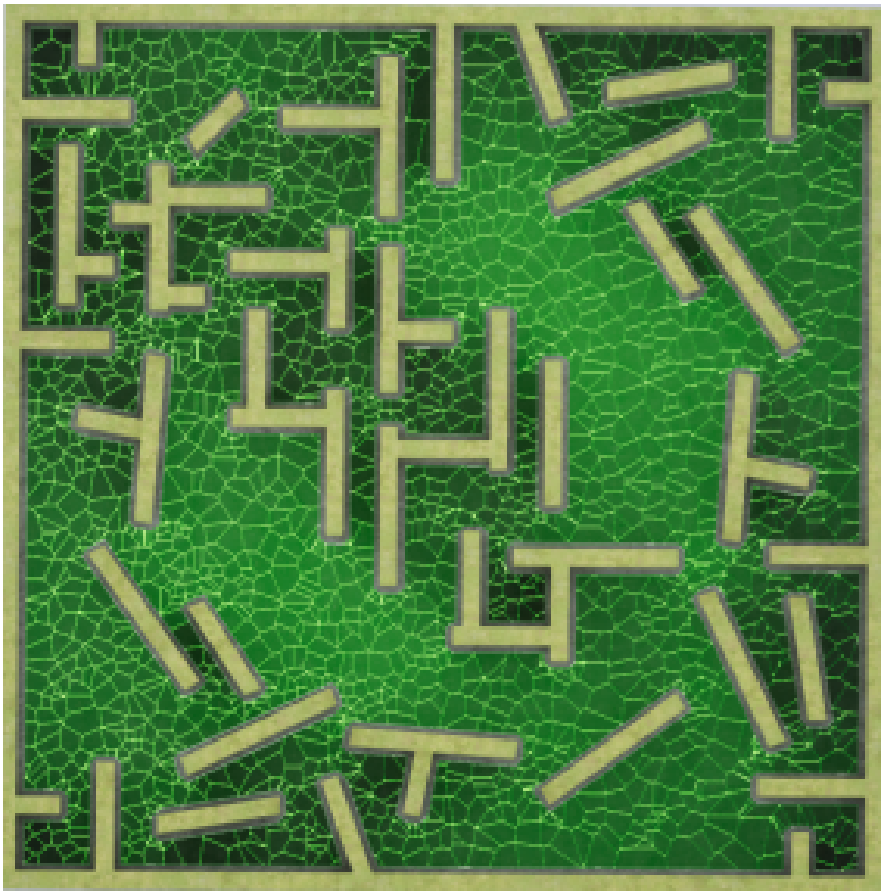

Fig. 4. Aggregated Visibility

to incorporate tactical component is defined as follows:

$$Penalty(i,j) = V \cdot \|S_i - S_j\|_2 \cdot (P_i + P_j) \tag{5}$$

where $V \geq 0$ is an importance of the tactical property. A penalty value for directional visibility is calculated in a similar way. In fact, the only difference is considering a direction to an enemy to choose one of precomputed visibility values if his position is known and using aggregated visibility otherwise.

4 Experiment

In this section, three tactical path finding algorithms are compared: the one using only position estimation by the kill/death statistics, the one using only the directional visibility and the one using both of them. The first AI utilizing these

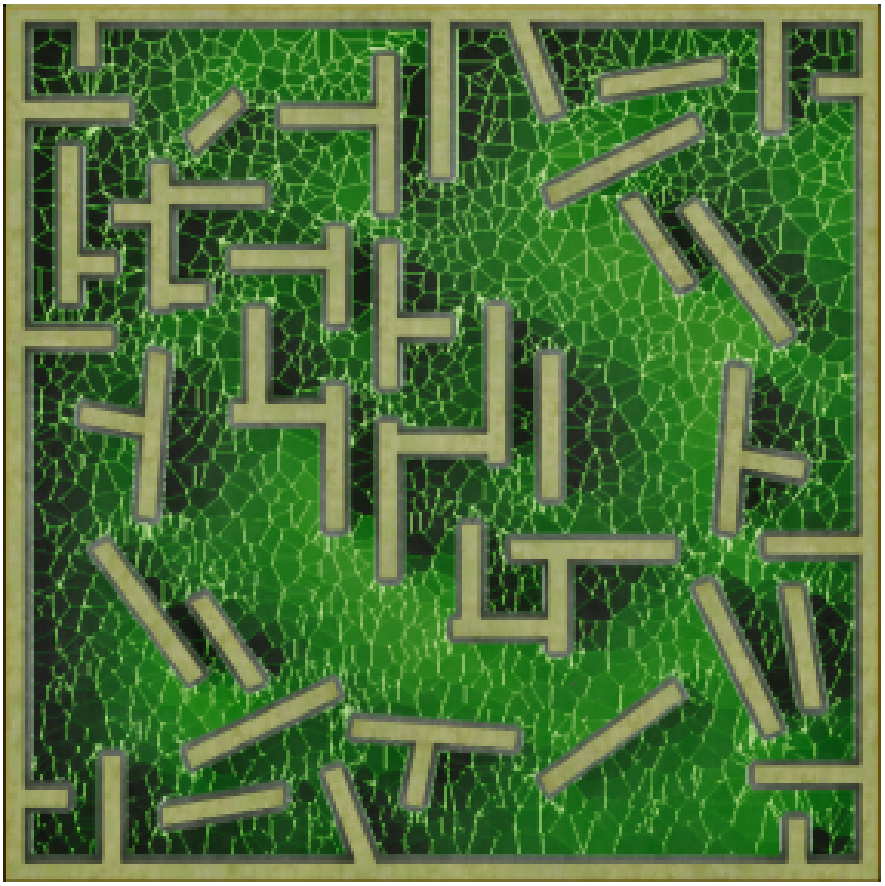

Fig. 5. Directional Visibility

algorithms plays against an AI ignoring any aggregated tactical information in 1000 consequent 1 vs 1 matches. The map used for the experiment is symmetric and the AI's inventory is limited to a single weapon.

A cumulative win rate of the first AI during the experiment is shown on Figure 6. As it can be seen, adding a directional visibility tactical property to the model does not result in a significant change, which can be explained by the fact that it is a rather weak position estimator compared to the frag map. Nevertheless, it gives about $2 - 3\%$ of a win rate gain.

In order to prove the efficiency of the developed tactical path finding model a statistical test is performed. A null hypothesis stating that a win rate of the first AI equals 0.62 is tested against a one-sided alternative using the binomial criteria and the last hundred of observations (when a frag map is more or less

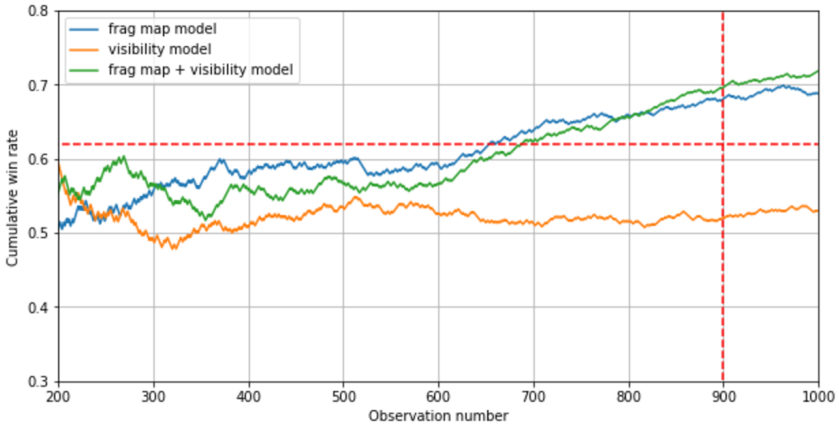

Fig. 6. Cumulative Win Rate

calculated) on 0.05 significance level:

$$\begin{cases} H_0 : p = 0.62, \\ H_1 : p > 0.62 \end{cases} \qquad T(X) \approx 0.038 < 0.05 \tag{6}$$

As expected, the null hypothesis is declined in favor of the alternative which means that the use of Voronoi-based navigation mesh with visibility and frag map tactical properties results in more than 12% win rate gain against the basic AI not using any tactical information.

5 Conclusion

We have developed a new navigation mesh type capable of collecting both tactical and geometry information that can be later used as features in complex path finding [18]. The main importance of our feature generation for VD-based navigation mesh representation is that it can be used in dynamic graph path finding, applying presented in [18] incremental anytime algorithms for fast search to an enemy and incorporating heuristics of kill/death ratio and visibility component.

The proposed approach significantly outperforms our previous decision making model [19], which takes into account only information on the current enemy visibility and based on this information adapts aiming and shooting models. Now, we are able to change BOT behavior based on previous information on enemy encounter and allow it to choose between straight-on attack by the shortest path or choosing backward path in order to outflank the enemy previously seen in a certain location.

Our tactical navigation mesh was also aimed to be used for incorporation in reinforcement learning based on deep video input, already tested in [20] for

respective task. Using our cell penalties and rewards we can train our model to use more peculiar algorithms of navigation and shooting while choosing the safest method based on incoming reward computes in terms of the current player health, enemy visibility and location.

In this work, we have only tested and proved its efficiency with a directional visibility property and a new approach to a map position estimation based on kill/death statistics and the graph diffusion model. We have used this approach to incorporate online navigation learning in a first-person shooter video game [21]. This game prototype was presented at demo section of ACM MultiMedia international conference and showed state-of-the-art performance for first-person shooter adjusted for believable BOT behavior in VR video game. The path finding algorithm in this scenario plays important role affecting human perception of the AI in the game. We showed that our method work better than standard path finding algorithms based on either shortest distance for predetermined offline heuristics or manually written rule-based models. We aim to further study the effect of path planning algorithms on FPS players in VR environments while improving the quality of the suggested pipeline of BOT path finding.

References

1. Hingston, P.: Believable Bots: Can Computers Play Like People? Springer (2012)
2. van der Sterren, W.: Tactical path-finding with A*. Charles River Media (2002)
3. Funge, J., Millington, I.: Artificial Intelligence for Games. M. Kaufmann (2009)
4. Pathfinding, B.: Strategic and tactical reasoning with waypoints. AI Game Programming Wisdom (2002)
5. Yakovlev, K., Baskin, E., Hramoin, I.: Grid-based angle-constrained path planning. In: Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz), Springer (2015) 208–221
6. Yakovlev, K., Andreychuk, A.: Any-angle pathfinding for multiple agents based on sipp algorithm. In: International Conference on Automated Planning and Scheduling. (2017)
7. Brewer, D.: Tactical pathfinding on a navmesh. Game AI Pro: Collected Wisdom of Game AI Professionals (2013)
8. Mendonça, M.R.F., Bernardino, H.S., Neto, R.F.: Stealthy path planning using navigation meshes. In: BRACIS. (2015) 31–36
9. Bamford, N.: Situational awareness: Terrain reasoning for tactical shooter a.i. In: AI Summit, GDC 2012. (2012)
10. Makarov, I., Polyakov, P.: Smoothing voronoi-based path with minimized length and visibility using composite bezier curves. In: Proceedings of AIST. (2016)
11. Bhattacharya, P., Gavrilova, Marina L.: Voronoi diagram in optimal path planning. In: IEEE IS on Voronoi Diagrams in Science and Engineering. (2007) 38–47
12. Nagatani, K., Iwai, Y., Tanaka, Y.: Sensor based navigation for car-like mobile robots using generalized voronoi graph. In: IEEE IC on IRS. (2001) 1017–1022
13. Mohammadi, S., Hazar, N.: A voronoi-based reactive approach for mobile robot navigation. Advances in Computer Science and Engineering **6** (2009) 901–904
14. Ho, Y.J., Liu, J. S.: Collision-free curvature-bounded smooth path planning using composite bezier curve based on voronoi diagram. In: IEEE International Symposium on Computational Intelligence in Robotics and Automation. (2009) 463–468

15. Ma, J., Huang, W., Segarra, S., Ribeiro, A.: Diffusion filtering of graph signals and its use in recommendation systems. In: Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on, IEEE (2016) 4563–4567
16. Hammond, D.K., Gur, Y., Johnson, C.R.: Graph diffusion distance: A difference measure for weighted graphs based on the graph laplacian exponential kernel. In: IEEE GlobalSIP. (2013) 419–422
17. Fortune, S.: A sweepline algorithm for voronoi diagrams. In: 2nd Annual Symposium on Computational geometry. (1986) 313–322
18. Makarov, I., et al.: Modelling human-like behavior through reward-based approach in a first-person shooter game. In: Proceedings of EEML. (2016) 24–33
19. Makarov, I., Tokmakov, M., Tokmakova, L.: Imitation of human behavior in 3d-shooter game. Analysis of Images, Social Networks and Texts 2015 (2015) 64
20. Makarov, I., Kashin, A., Korinevskaya, A.: Learning to play pong video game via deep reinforcement learning. CEUR WP (2017) 1–6
21. Makarov, I., et al.: First-person shooter game for virtual reality headset with advanced multi-agent intelligent system. In: Proceedings of the 2016 ACM on Multimedia Conference, ACM (2016) 735–736