Skeleton-based multi-agent opponent search

Wael Al Enezi
School of Computer Science
McGill University
Montréal, Canada
wael.alenezi@mail.mcgill.ca

Clark Verbrugge
School of Computer Science
McGill University
Montréal, Canada
clump@cs.mcgill.ca

***Abstract*—In many games, players may run away and hide from NPC enemies that have previously observed them, either to avoid combat or as part of pursuing a stealth-based solution. Rational NPC response then requires searching for the hidden player, which for maximal realism should build on the last known location, and consider the relative likelihood of a player hiding or reaching each searched location. Unfortunately, search behavior is not usually systematic, and in practice is either limited to randomized goals within a small region, or exploits global information on the player position that should be unknown. In this work, we introduce a real-time method for directing a multi-agent search utilizing the environment's topology. This approach allows for more natural and wider-scoped search behavior. Experimental results show that this method scales to relatively large game maps, and performs better than or close to a naïve team of agents fully aware of the player's position.**

***Index Terms*—Artificial Intelligence, stealth, search, multi-agent, games**

I. Introduction

Variations on hide-and-seek or pursuit/evasion are common game mechanics, particularly in the action and stealth genres. Players may aim to avoid observation by enemy non-player characters (NPCs) as part of a game requirement, or in order to avoid combat, with the latter more common outside pure stealth contexts. Having been observed, however, subsequent pursuit and search by enemy agents is often based on very simple strategies: enemies may be given full information and know the player position despite occlusion, which is easily perceived as unfair, or they demonstrate unrealistic search behavior, moving or looking randomly or only in pre-scripted locations, potentially trivializing the challenge.

In this work, we propose a novel searching behavior for enemy agents. Our design is based on a geometric decomposition, exploiting the reachability properties of a structured road map (straight skeleton) to enable efficient propagation of probable player locations. This avoids discretization artefacts and scaling concerns common to grid-based approaches, and ensures search behavior follows the general "shape" of the space, guiding location probabilities better than through random sampling approaches. It has the additional advantage of enabling effective separation heuristics, which allows multiple enemy agents to search independently while still avoiding overlap, and thus more efficiently cover a space. In addition, this method can be expanded to robotic contexts with tracking systems [1].

We evaluate our method experimentally, comparing different parameterization, as well as comparing behavior with an existing probability-based search model in multiple game levels, modelled in Unity3D. Our results show that for most maps, our method produced a better search performance for multi-agent scenarios. We further show that this method is achievable in real-time and can be deployed in real-time systems.

The major contributions of this work consist of:

- We describe a novel method for tracking an opponent's position that can be utilized in commercial games. This method can also be expanded into guard patrol, or for guiding NPC exploration behavior.
- We empirically evaluate the method's performance on several maps modelled from currently existing commercial games. We also describe the limitations of our method based on qualitative assessment. [1]
- Finally, we provide the framework we used for this method as an open-source to make it possible for other researchers or game developers to replicate our results. [2]

II. Problem Formulation

Our goal is to create an efficient multi-agent behavior to search and find an adversary opponent in an enclosed game environment. The multi-agent team has partial observable game states, where the opponent's position is unknown. Each agent is granted a Field of View (FOV) of a limited range and angle. Once the opponent is in the agent's FOV they are considered to be detected. The agents' goal is to navigate to the opponent's most probable position after the line of sight is broken.

III. Literature Review

The problem of searching for an opponent is determined by the ability to track and predict an opponent's position once it is out of the visual field. This problem was approached in many fields, for example, to track enemy positions in military surveillance [2], to monitor people's movement trajectory in

This work supported by the COHESA project, through NSERC Strategic Networks grant NETGP485577-15.

[1] A real-time video example of the result can be viewed at https://streamable.com/774173

[2] The sourcecode can be found at https://github.com/wralenezi/Stealth-Simulator.git

robotics [3], and in games to allow Non-Playable Characters to exhibit a human-like motion in First-Person Shooters [4]. Prior work has also included a Turing "hide-and-seek" test, held to evaluate agent performance in playing such a game in a room [5].

Third eye Crime, a game well-known for implementing opponent tracking as a core mechanic of its gameplay [6], used a modified version of *occupancy maps* [7]. It overlays a grid on top of the game map. Each node of the grid has a numerical variable between 0 and 1 that represents a probability. The probability reflects the likelihood of an opponent occupying the corresponding node. In a typical game scenario, the opponent is detected by a guard's field-of-view. After that, all guards simply take the shortest path to the opponent's position. However, once the opponent is out of their sight, the node that corresponds to the position the opponent was last seen on will have a probability of 1. As time progresses, the update function consists of uniformly diffusing probability across the neighboring nodes and normalizing them. Finally, the guards' role is to cover the node with the highest probability, where once a node is seen by a guard its probability is zeroed. This method produced interesting guard behavior that served as the core mechanic of *Third eye Crime*. Since this method relies on discretizing the space into a grid, however, the accuracy of tracking the opponent's position depends on the granularity of the grid. A low-resolution grid will result in inaccurate tracking, and a high-resolution grid will increase the required computation and memory costs [8].

The shortcomings of occupancy maps were addressed by using particle filters, a technique originally developed in robotics [9]. By using this method, there is no need to discretize the space into a grid, like in occupancy maps [10]. In general, this method works by sampling a finite set of weighted particles that represent the possible positions of the opponent. The weight of each particle corresponds to the probability the opponent is in that position. Once a particle is seen, it is removed and unseen particles are resampled. After that, the new particles are updated by a random walk by choosing a direction, then it is moved in that direction at max speed for one time step. The memory requirement for using particle filters is, unlike occupancy maps, independent of the size of the map. Alternatively, the computational expense relies on the number of particles sampled; a high number of particles results in high accuracy in exchange for performance, and a low particle count will result in lower accuracy.

Particle filters may provide lesser computational and memory requirements than occupancy maps, but these two methods share a weakness where they uniformly give weights to unlikely opponent positions due to the randomness of particle filters and the omnidirectional diffusion of probability in occupancy maps. In other words, they don't make use of the map's layout to aid in weighing possible positions. This shortcoming was refined by using a motion model called "simulacra" [8]. In a simulacrum, the particles are defined on the navigation graph used for agent pathfinding. The restricted sampling is intended to create particles that more accurately represent opponent movements on the navigation graph.

The simulacrum approach has been described algorithmically, but has not had sufficient research investigation to evaluate its properties, or understand the impact of relying entirely on a navigation graph for both agent movement and probability propagation. In our design we follow a similar approach, however, we expand on it by using a separate graph that captures the map properties, regardless of the agents' pathfinding system. In addition, we further define a multi-agent framework for this problem.

IV. Methodology

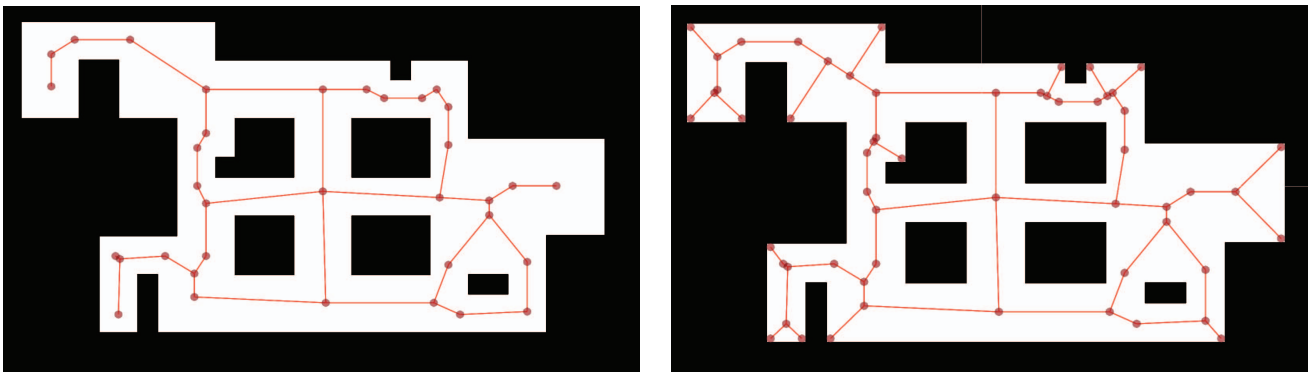
In this section, we describe our method for predicting the opponent's position. In our design, agents use a *navigation mesh* for pathfinding instead of a navigation graph or roadmap. This reflects modern game design, where agents have more free-form movement potential. Probability diffusion, however, depends on a graph that captures the topology of the environment. This graph (which we also refer to as roadmap) is designed to reach any spot in the walk-able space. A numerical value that represents the probability the opponent is at a certain position is propagated through the topology graph. For testing various approaches, we defined two methods that dictate the manner the probability is transmitted through the graph. In the first, the probability is simply propagated through the graph. The second is our interpretation of using an occupancy map approach on a graph instead of a grid. After that, the agents choose a part of the graph by using a heuristic feature function. For an adversary, we specified three simple opponent behavior to test the performance against.

First, we create a graph that serves as a backbone of the map and captures its topology.

A. Topology graph

1) Graph creation: The purpose of the graph is to create a simplified representation of the map and utilize that representation as spatial knowledge for the agents. We use the scale axis transform (SAT) [11]. We choose the SAT graph because it captures the main features of a polygon in a simpler skeleton graph, well covering a space and extending naturally into corners or other areas that may be useful for searching behavior. We implemented the SAT graph based on this method [12]. Note that the algorithm to build this graph is based on a discrete grid, but it can also be constructed directly from the level geometry (with a more complex implementation). Figure 1.a shows the result on an example game map. After that, to ensure our graph is connected to the corners, we expand on the graph by connecting each convex angle of the map to the closest corresponding projection on the graph. Figure 1.b further shows our extension.

2) Discretizing the graph: We divided each edge of the graph into smaller edges with a fixed length, which we refer to as segments. This will serve in further decomposing the graph to simplify and improve the precision in determining possible locations of the opponent. Another advantage of using segments is to ensure probabilities degradation to match the



(a) SAT (b) Modified SAT

Fig. 1: SAT graph on Docks level from the game Metal Gear Solid.

opponent's movement. Each segment is associated with a numerical value that represents the probability an opponent is present near it. Figure 2 shows the final shape of the graph.

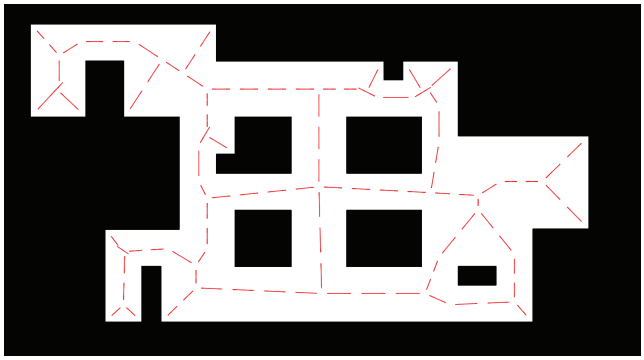


Fig. 2: The divided graph of the Docks map of Metal Gear Solid.

The segments are affected by the guards' Field of View (FOV). Once the two end-points and mid-point are in the FOV, that segment is considered to be seen.

B. Search behavior

While the opponent is in the FOV of at least one guard, all guards simply navigate to the opponent's position. Once the opponent is out of the FOV of all guards present on the map, the closest segment in the direction of the opponent's velocity is assigned with a probability of 1. After that, the probability is propagated through the neighboring segments. We define the following two methods for the probability flow.

1) Probability propagation: In this method, the probability is propagated to neighboring segments while deteriorating for each segment. The probability deterioration serves in prioritizing the search of closer segments. The amount of deterioration depends on the size of the map's area. To match the opponent's movement, the propagation pace is set to match the opponent's maximum movement speed. At each time step, the probability of each segment is incremented only if it satisfied at least one of the following conditions:

- It has a nonzero probability.
- It is adjacent to another segment that has a nonzero probability.

To find the opponent, the guards scan the segments and start moving toward the midpoint of a chosen segment. A segment is chosen by a fitness function that guides the search; see section IV-C below. Once a segment is seen by a guard, its probability is set to zero, and then its probability increments at a fixed rate and the guard chooses a new segment. However, if at one point during the search, the max probability of the segments is less than a specified threshold, then the probability of all segments will be (re-)normalized. This method can be also used as a patrol behavior, which is an alternative approach to the one in [13].

2) Probability diffuse: In this method, the probability is similarly diffused to Isla's implementation of occupancy maps [6]. However, in our implementation, the discrete Gaussian distribution is allocated to the segments instead of pixels. At every time step, the probability is diffused to the neighboring segments using equation 1. After diffusing the probabilities, they are normalized over the segments.

$$P_{t+1}(n) = (1-\lambda)P_t(n) + \frac{\lambda}{|Adj(n)|}\sum_{n' \in Adj(n)} P_t(n') \quad (1)$$

where $\lambda \in [0,1]$ is the diffuse factor, *Adj(n)* are the neighboring segments to n.

C. Choosing a segment

The goal of the guards is to efficiently investigate the segments to find the opponent. Once a guard queries the available segments, it chooses its next destination by a fitness function, which is described in equation 2. After the segment is chosen, the guard uses the NavMesh to find the shortest path to see it.

$$f_i(g) = p_i * w_p + d_i(g) * w_d + min_{t \in G}(d_i(t))) * w_g \quad (2)$$

Where i is a segment, g is a guard, p_i is the probability assigned to i, w_p is the weight assigned to the probability variable, $d_i(g)$ is the segment i distance from the guard, w_d is

the weight assigned to the distance variable, $min_{t \in G}(d_i(t)))$ is the path-distance of the closest other guard to the segment i, and w_g is the weight assigned to that variable. The weights range between [-1,1].

After testing the performance, we fixed the weights $[w_p, w_d, w_g]$ to $[1, -0.8, 1]$.

D. Opponent behavior

To assess our method, we need a suitable hiding opponent. For the opponent's behavior, we relaxed the problem into having the opponent choose a hiding spot from a set of predefined hiding positions on the map to navigate to. We populated the set of hiding spots by allocating one hiding spot on the normal vectors of edges forming a convex interior angle, and two hiding spots on the sides of an interior reflex angle, using a small fixed offset. Figure 3 shows the two cases.

(a) Reflex Angle (b) Convex Angle

Fig. 3: Possible hiding spots based on the angle of the corners of obstacles.

Defining good hiding heuristics remains a research challenge in itself. We thus defined 3 simple behavior patterns for the opponent to choose its hiding spot:

- **Heuristic**: The opponent is aware of the guards' positions, and so it chooses the hiding spot that has the furthest path distance from the other guards. The opponent stays in that position until it is spotted, then it moves and finds another spot to hide.
- **Heuristic with Movement**: This is similar to the previous behavior, however, after the opponent arrives at a hiding spot, it waits for a random time interval and then relocates to a new spot using the same heuristic, irrespective of whether it has been seen yet or not.
- **Random with Movement**: This behavior is also constantly moving between hiding spots with a random wait time, however, it instead of randomly choosing a new destination using the base heuristic, it chooses the next hiding spot randomly.

V. Experiments

For our experiments, we defined a scenario to start by randomly allocating the guards on a map. After that, the opponent is spawned in front of one of the available guards by random. The guards' goal is to reduce the distance between them and the opponent. Since our goal is to study our search implementation, we relaxed the scenario by preventing it from ending once the opponent has been caught. Each experiment thus ends after a fixed amount of time is passed, sufficiently long that multiple separate opponent sightings and searches will have been performed.

We ran 40 episodes for each combination of the following variables:

A. Game maps

We focused on using maps from commercial games. We recreated maps from First-Person Shooter, Action/Stealth, and Role-Playing games. Figure 4 shows an illustration of the maps used, which are:

- The "Docks" level from *Metal Gear Solid 1* (as shown in figures 1 and 2).
- Two maps from the *Dragon Age* series. They were taken from MovingAI collection [14].
- An excerpt of the "San Cristobal Medical Facility: Basic care unit" from *Alien: Isolation*.
- A map we designed to be similar to a warehouse with a high number of intersections.
- The "Ascent" map from the Multiplayer FPS game, *Valorant*.
- "Vacant", a multiplayer map in the FPS game *Call of Duty: Modern Warfare*. This is the largest and most complex map we included in our experiment. We included it as a stress test.

B. Search behavior

We considered three search behaviors:

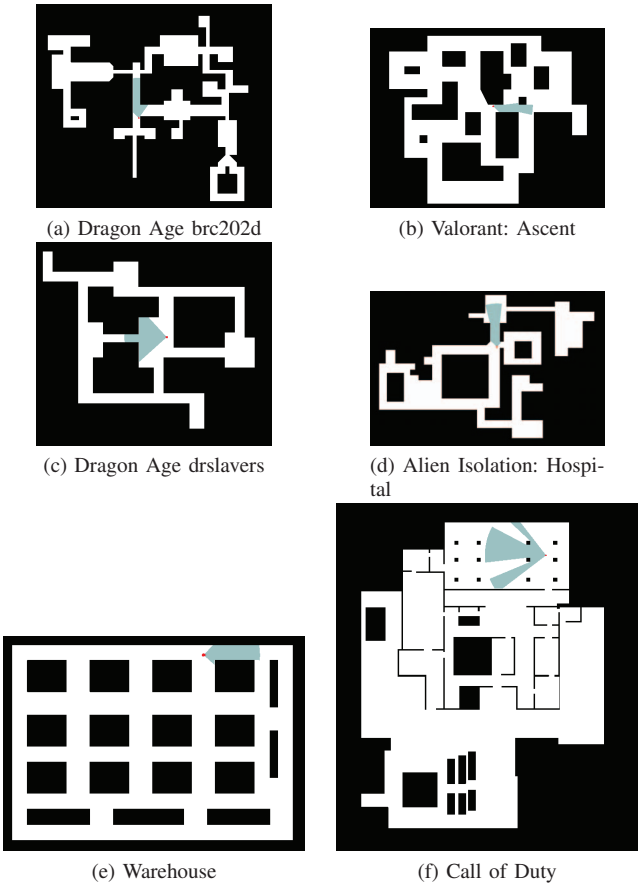
- **Cheating**: The guards were cheating by knowing the opponent's position at all times, even if occluded. This was intended to serve as an upper-bound on possible performance, and to reflect common game practice.
- **Probability Propagation**: Our main model, as described in section IV-B1.
- **Probability Diffuse**: An occupancy map approach adapted to our road-map model, as described in section IV-B2.

C. Guard FOV

The radius of the guard's FOV depends on the map's dimension. It is proportional to the largest side of the bounding box of the exterior polygon of the map. The default radius we defined is 20% as a reasonable fit for the maps we investigated, however, we also considered the effect of changing radius on the search performance.

D. Number of Guards

Finding an opponent can be accomplished easier when the guards cooperate in tracking the opponent. We used a default of 3 guards, but also consider the search performance for different numbers of guards in a team.



(a) Dragon Age brc202d (b) Valorant: Ascent

(c) Dragon Age drslavers (d) Alien Isolation: Hospital

(e) Warehouse (f) Call of Duty

Fig. 4: Game Maps with one guard in each map, the light blue area is its FOV.

E. Opponent speed

Guards that move as fast or faster than players are much less likely to lose track of a player, and can easily reduce the testing situation into a pure chase. Since our goal is to test search performance, we set the opponent's movement to be 150% faster than the guard's, a value found to be effective without being excessive in initial testing. We also considered the three opponent behaviors in section IV-D.

VI. Results

For evaluating the search performance, we ran 40 episodes of 250 seconds long for 3 guards chasing an opponent for each combination of the search behavior, opponent behavior, and maps. We measure performance in terms of relative "alert time", which is the proportion of time an agent is observed versus staying successfully hidden: a lower alert time indicates better performance. We first evaluate the search performance over the different maps, and the effect of different parameters on the search performance, then We describe interesting limitations that affected the performance on our maps. We then consider the effect the opponent behavior has on the search performance of the probability propagation method. Lastly, we report the worst-case computation for the probability propagation method.

A. Overall search performance

Figure 5 shows the percentage of time the opponent was in the line of sight by at least one guard over all our maps, keeping opponent behavior constant (base heuristic). Noticeably, as the map size and complexity increase, we notice a decline in performance across all methods. In other words, all methods could not keep the opponent detected for more than 30% of the episode's time. This could be caused by the fact the opponent is faster by 1.5 times than the guards, so the opponent can easily move out of sight, however, further experiments is required to confirm this. Another factor is the map complexity, which we will describe more in detail in the following section. Interestingly, we found that the cheating guards were often outperformed by our methods. We quantitatively assess a possible cause in the following section.

B. Search method performance

We noticed that the probability propagation performed the best in most maps, in many cases exceeding our cheating "upper-bound". While this result might seem counterintuitive, figure 6 provides a possible explanation to why a group of guards unaware of the opponent's position outperformed the ones who had full data of the opponent's whereabouts. It can be seen that in cheating, guards spent more time being close to each other during the scenario. This is because the opponent's position was known, and all guards simply navigated towards them. Even if they started in random positions, they would easily end up converging to the same path after a certain time passed. In our method, since we gave a weight that motivated the guards to navigate to segments that are further from other guards, they ended up dispersed across the map. This wider distribution of guards resulted in a higher chance for the opponent to be spotted or intercepted as they moved around the maps.

After observing the performance of the "Probability diffuse" method, we also observed several factors that led to its relatively weaker performance. A core factor seems to be due to the omnidirectional approach to probability diffusion: segments that were on the opposite side of the opponent's velocity had the same probability as those in its direction. This caused the guards to go back and investigate unlikely segments. This observation confirms a similar observation made in [8], and which motivated specific tuning concerns in Isla's *Third eye Crime*. On the other hand, this limitation is handled in the probability propagation method by preventing the probability of being transmitted to segments that the opponent cannot possibly enter. Figure 7 shows an example where the opponent is out of sight and the probability is not propagated behind the guard since, assuming the FOV is large enough, the opponent cannot pass by the guard unnoticed. This allows the guard to search in more reasonable locations rather than re-investigating previously seen areas.

The cheating did manage to outperform our method in a few maps, notably the "Warehouse" and "Call of Duty" levels. The latter is a large map, but the former is not, and so we attribute this outcome to the relatively high count of path-crossings in

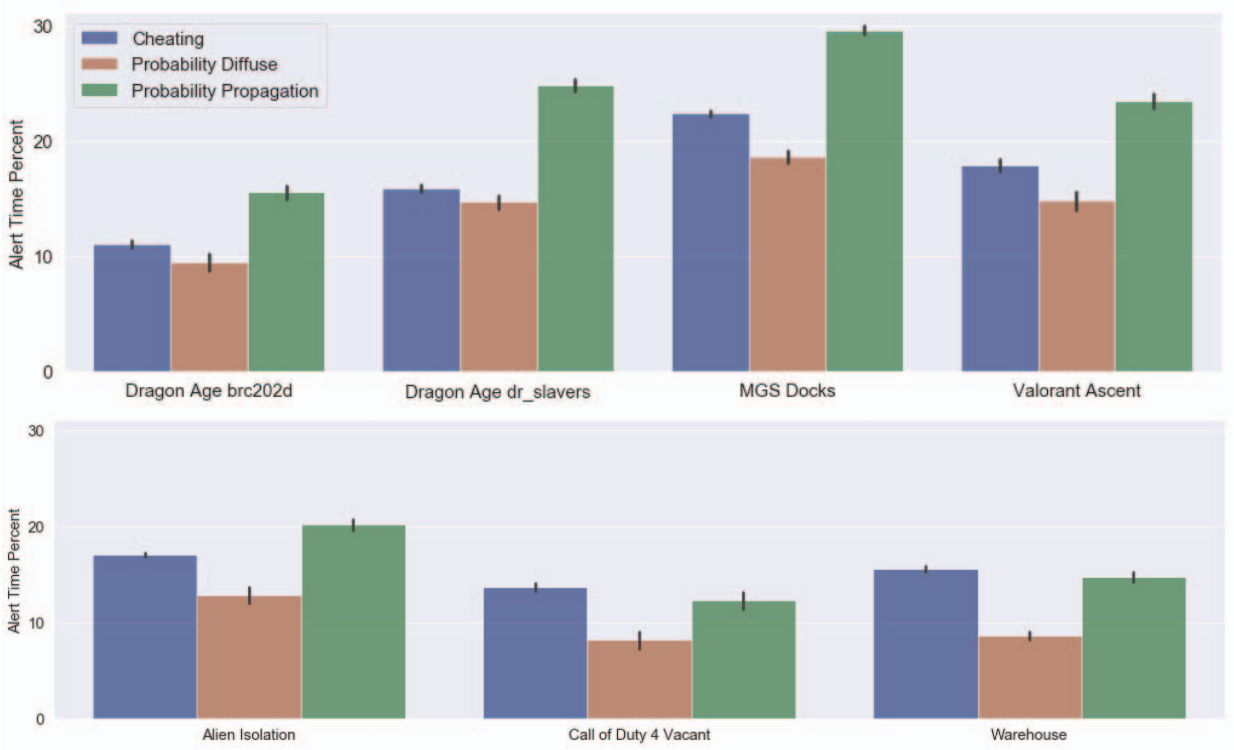


Fig. 5: The percentage of the time the opponent was detected by at least one guard to the episode's total time.

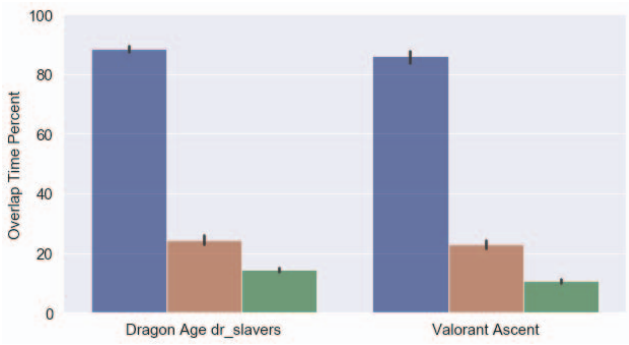


Fig. 6: The percentage of time the guards were 0.5 meter close to each other to the total episode time.

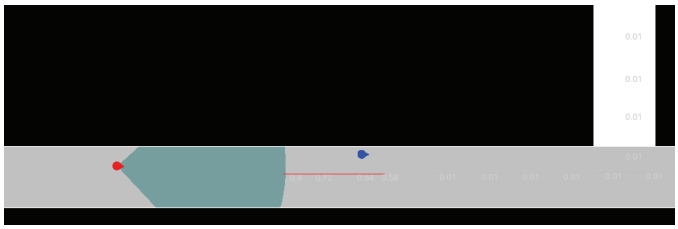


Fig. 7: A guard (red) looks for the opponent (blue), who is out of sight. The red line represent the segments with nonzero probabilities; note that the probabilities behind (left of) the guard are 0.

these maps. The guard spends more time searching for the opponent since there are a higher number of possible paths the opponent took. On the other hand, the cheating team of guards simply navigates to the opponent's position.

C. Number of guards vs FOV radius

In commercial games, defining the adversary's characteristics, like the number of guards in a level, the sensory parameters of the guard, the health of the guards, etc., are usually determined by human testing. We consider the possibility of quantitatively assessing the impact of tweaking such parameters on the game as a preliminary step before human testing.

For example, to provide a reasonable degree of difficulty for players in stealth games, they should be faced with a challenging, but still doable task of staying hidden. Guard speed is trivially manipulated, but two other major factors that affect the difficulty of this task are the number of guards in the level, and the radius of their FOV.

Here, we compared the impact of changing the radius of the guards' FOV and the number of guards on a map. Figure 8 shows the results of our comparison on two maps. It is evident that the impact of changing such parameters can have different effects on the search performance depending on the map. In both cases we observe an increase in performance in rough proportion to the number of guards, but changing the FOV has almost no impact on the "Call of Duty" map, while it has a quite noticeable impact in the "Warehouse" level. The

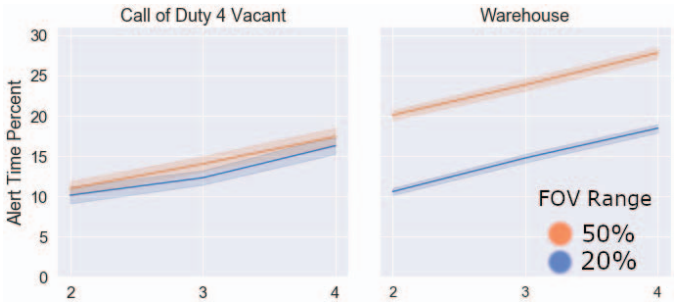


Fig. 8: The alert time for different guard count (x-axis) with different range of FOV (y-axis). Results for two maps.

dense occlusion in the latter map may be responding better to increased guard vision. This does not necessarily mean the effect is unimportant in the "Call of Duty" map, as even though the setup of two guards with a 50% radius range offers similar performance to the setup of 4 guards with 20%, these two setups might provide a different experience for a human player.

The lack of measurable impact of an increased FOV in "Call of Duty" can also be partly explained by the nature of the map structure. As we previously mentioned, the decision to use a straight skeleton for probability propagation is to capture the map's topology. However, using this graph for modelling possible opponent trajectories may not be practical for maps with large, wide and open spaces, which the "Call of Duty" map does contain. Figure 9 shows an example of this limitation. When the guards lose sight of the opponent, the closest segment in the direction of the opponent's movement is set with a probability of 1. After that, the probability is propagated in the direction of the opponent's movement along the straight skeleton. However, in some locations on the map, an opponent may navigate through an area where the skeletal road-map coverage is reduced or does not fully reach, and probability diffusion diverges from the opponent's actual path. A denser variation on the road-map may reduce this limitation.

D. Opponent behavior

As figure 10 shows, the 3 different opponent behavior types are all able to hide effectively, although the different choices of hiding behavior did not have a significant impact: preemptively moving to a different hiding spot may result in slight benefit or degradation in performance, but is otherwise relatively ineffective, with variance across maps higher than across behavior type. These behavior types, however, are simple and meant to provide a benchmark to compare the search methods; more complex opponent behavior remains future work.

E. Computation costs

The limited computation budget for AI in most games requires an efficient implementation for any practical design. We evaluated the performance of the probability propagation method by measuring the maximum amount of time it takes to complete one loop of execution. We measured it over several

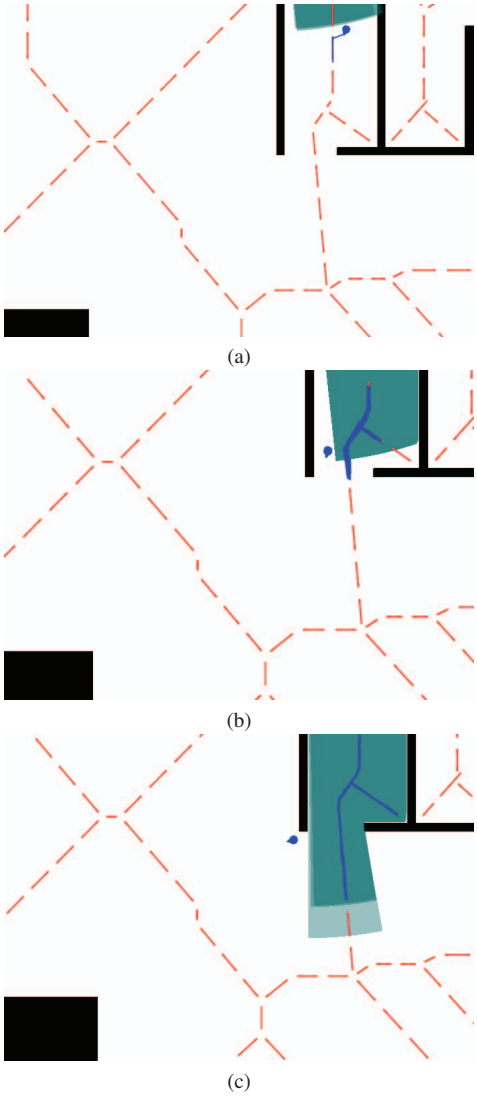


Fig. 9: An Example of a limitation in our method. The dashed red line is the straight skeleton of the map. The blue line is the segments that had a non-zero probability. As the FOV zeros out the propagated probability, the enemy escapes.

maps in a worst-case situation, when all segments have non-zero probabilities. Table I shows the method's performance over the maps for 3 guards.

We notice that this method is relatively cheap, and realizable in real-time for normal-sized maps. Unsurprisingly, the performance is slower on larger maps, such as the "Call of Duty" map. Performance could be improved by increasing segment length, reducing the update rate, or through hierarchical or other spatial partitioning approaches that focus on local area over more distant locations.

VII. Conclusion

NPCs that show rational, intuitive behavior can be more interesting to play against in games. In this work, we introduced a method that allows guards to exhibit a real-time search

Map	Map Area (M^2)	Segment Count	Make a Decision (ms)	Update loop (ms)
MGS Docks	236	99	4	3
Dragon Age brc202d	550	274	26	9
Dragon Age dr_slavers	362	136	3	4
Valorant Ascent	1020	219	13	7
Warehouse	504	192	14	4
Alien Isolation	466	183	9	8
Call of Duty 4 Vacant	3970	727	210	25

TABLE I: The worst-time decision time and update loop time for different game maps. The experiments were done on a CPU Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz with 16 GB RAM.

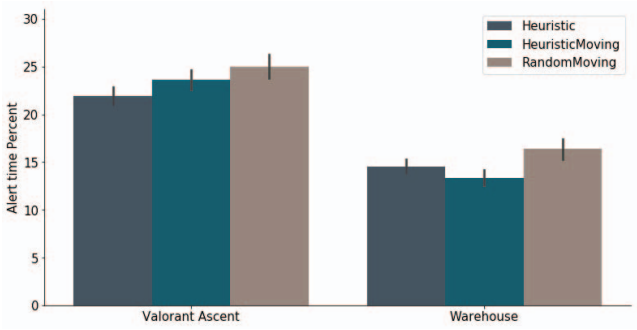


Fig. 10: The percentage of time the opponent, using one of the 3 opponent behaviors, was in sight of guards over two maps.

behavior for an opponent that better exploits awareness of the level geometry. For this, we used a skeletal graph representation to better propagate the probability of potential opponent locations. Our method showed interesting behavior for multi-guards searching for an opponent in several maps taken from existing commercial games, improving even over a full information scenario when guards know the exact opponent location. The approach performs best in relatively occluded contexts, where the skeletal graph well approximates the level shape, and can be tuned through different parameterization, such as extent of guard FOV.

VIII. Future Work

A number of future directions are possible from our work. The effect or our approach, of course, depends on how well an agent may hide or attempt to elude pursuers. This represents an interesting research challenge in itself, but a user study would be ideal to assess relative effectiveness with actual human players, where we can not only observe the impact of different, human-driven evasion strategies, but also the relative appeal and naturalness of our approach—does our design result in more "human-like" searching behavior? Believability may be additionally improved by incorporating "barking," where guards announce their intentions or frustrations based on their success, or lack thereof.

We are also interested in optimizing performance. Evaluation of our method shows it is efficiently realizable on average-sized game maps, but we believe additional benefit is possible by tuning segment size, graph density, and update rate based on map structure. This extends into other motivations for parameter tuning; for example, changing the weights used for choosing the best segment to visit during the gameplay may lead to different behavior, that can be exploited to control relative advantage—weaker players can stay more effectively hidden when we can encourage guards to stick together. Improving guard separation, on the other hand, gives us a more difficult, but also intuitively more natural search behavior, and integrating the full path choices of guards into our propagation approach would allow us to extend the separation heuristic to further reduce the amount of guard overlap, which as we have observed is a major factor in search performance.

References

[1] Á. M. Guerrero-Higueras, C. Álvarez-Aparicio, M. C. Calvo Olivera, F. J. Rodríguez-Lera, C. Fernández-Llamas, F. M. Rico, and V. Matellán, "Tracking people in a mobile robot from 2d lidar scans using full convolutional neural networks for security in cluttered environments," *Frontiers in neurorobotics*, vol. 12, p. 85, 2019.

[2] D. A. Borovies, "Particle filter based tracking in a detection sparse discrete event simulation environment," NAVAL POSTGRADUATE SCHOOL MONTEREY CA, Tech. Rep., 2007.

[3] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thurun, "Learning motion patterns of people for compliant motion," *International Journal of Robotics Research*, 2004.

[4] S. Hladky and V. Bulitko, "An evaluation of models for predicting opponent positions in first-person shooter video games," in *2008 IEEE Symposium On Computational Intelligence and Games*. IEEE, 2008, pp. 39–46.

[5] A. Cenkner, V. Bulitko, M. Spetch, E. Legge, C. G. Anderson, and M. Brown, "Passing a hide-and-seek third-person turing test," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, pp. 18–30, 2013.

[6] D. Isla, "Third eye crime: Building a stealth game around occupancy maps," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 9, no. 1, 2013.

[7] ——, "Probabilistic target tracking and search using occupancy maps," *AI Game Programming Wisdom*, vol. 3, pp. 379–388, 2006.

[8] C. Darken and B. Anderegg, "Particle filters and simulacra for more realistic opponent tracking," in *AI Game Programming Wisdom*, vol. 4. Charles River Media, 2008, pp. 419–428.

[9] S. Thrun, "Particle filters in robotics." in *Uncertainty in artificial intelligence*, vol. 2, 2002, pp. 511–518.

[10] C. Bererton, "State estimation for game ai using particle filters," in *AAAI workshop on challenges in game AI*, 2004.

[11] J. Giesen, B. Miklos, M. Pauly, and C. Wormser, "The scale axis transform," in *Proceedings of the twenty-fifth annual symposium on Computational geometry*, 2009, pp. 106–115.

[12] C. W. Niblack, P. B. Gibbons, and D. W. Capson, "Generating skeletons and centerlines from the distance transform," *CVGIP: Graphical Models and image processing*, vol. 54, no. 5, pp. 420–437, 1992.

[13] W. Al Enezi and C. Verbrugge, "Dynamic guard patrol in stealth games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, no. 1, 2020, pp. 160–166.

[14] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012. [Online]. Available: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf