Analyzing Stealth Games with Distractions

Alexander Borodovski and **Clark Verbrugge**
School of Computer Science
McGill University
Montréal, Québec, Canada
alex@borodox.com, clump@cs.mcgill.ca

Abstract

The ability to *distract* opponents is a key mechanic in many stealth games. Existing search-based approaches to stealth analysis, however, focus entirely on solving the non-detection problem, for which they rely on static, ahead-of-time models of guard movements that do not depend on player interaction. In this work we extend and optimize an approach based on heuristic search of stealth games to model variation in guard paths as dynamically triggered by player actions. Our design is expressive, accommodating different distraction designs, including remote activation and time delays. Using a Unity3D implementation, we show our enhanced search can solve distraction puzzles found in real games, as well as more complex, multiple-distraction level designs. Our work shows how heuristic search can be applied to dynamically determined contexts, and significantly extends the ability to model and solve stealth games.

Introduction

Distraction puzzles are a common extension of basic stealth game problems. A player is presented with the usual task of finding a path from start to goal, undetected by enemies, with success apparently impossible due to enemy guard motions and fields of view that effectively block the goal. In order to progress a player must throw an object or use sound to inspire guards to investigate, changing their motion and opening a solution path. From an analysis perspective, algorithmically solving distraction puzzles introduces additional complexity in defining the state space—guard positions depend on arbitrary, dynamic player choices, and thus regions of non-detection cannot be computed statically, ahead of time, a property on which existing, search-based stealth analysis relies (Tremblay et al. 2013).

Our work addresses this dynamic property of state evolution, allowing guard positions to be efficiently computed and algorithmic search applied despite the potential growth in state space. We first develop a model of distraction puzzles, identifying the basic components of a distraction that allow for a variety of puzzle designs. We combine this with a guard movement model that allows for different choices in patrol path depending on which distraction was activated, and when and where the guard was at that point. This reduces the combinatorial explosion in state space to a manageable branching of game states, even if the branch points must still be detected dynamically. To make search in this space efficient we then extend the main *Rapidly Exploring Random Tree* (RRT) search algorithm, applying multiple optimizations that improve performance without overly stressing the resource requirements beyond that of a basic non-detection search. Implementation of this design in *Unity3D* shows feasibility, and we demonstrate that we can solve both relatively simple distraction puzzles typical of modern games, and much more complex designs involving composition of multiple distractions. Specific contributions of our work include the following.

- We formalize and identify components of a distraction puzzle that can be used to construct a variety of behaviours. This design integrates with a flexible model of guard movement that allows for natural and reactive guard responses.
- To make heuristic search efficient, we extend the basic stealth space search with multiple optimizations. Although these are motivated by the need to search in the richer state space implied by distractions, they are also applicable to more general use of RRT in game analysis.
- We experimentally evaluate the approach in Unity3D; this acts as a proof of concept as well as evidence of how our optimizations enable search of quite complex distraction puzzles.

Distraction Model

Distractions come in many forms. Operationally, a distraction has a point in space that the player needs to reach in order to use it. Once the player reaches the distraction point, a signal is sent out to all guards that that specific distraction point was activated at that specific time. A subset of guards, depending on their current location and pre-set patrol routes, will then react to the distraction by changing where they are going, and entering a new looped patrol. This new patrol may involve them going to check a point and then resuming their prior pattern, or changing to a new pattern.

In this process we can separate the location that causes the distraction to be activated (Activation Location) and the location a guard heads to upon being distracted (Event Lo-

Copyright © 2016, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

cation). Distractions are often time-delayed, with the signal occurring at a future time instead of the time the player actually visited the distraction point. In table 1, we can describe eight basic variations that can thus be implemented: with the same or different activation and event locations, time-delayed by a $\delta > 0$ or not, and whether the guard returns to a previous patrol route or is permanently altered. There can be any number of different, independent distraction points each affecting different (although perhaps overlapping) sets of guards, or affecting the same sets of guards in different ways. Further variation, such as in duration of distraction, or arbitrarily locating activation events are also possible.

Variations 1 and 5 are commonly seen and occur in games such as the *Splinter Cell* series: stepping on broken glass on the ground makes a noise, which a guard investigates, and either eventually returns to their patrol or enters a new alarmed mode. Variation 2 is seen, for example, in the game *Far Cry 4*: throwing a rock causes a noise where it hits and nearby guards go to investigate, returning to their previous position; variation 4 occurs if there is significant delay in the throwing.

Table 1: Possible Distraction Combinations

Variation	1	2	3	4	5	6	7	8
Activation Loc Event Loc	=	$\neq$	=	$\neq$	=	$\neq$	=	$\neq$
Delay	0	0	δ	δ	0	0	δ	δ
Old Path New Path	=	=	=	=	$\neq$	$\neq$	$\neq$	$\neq$

To implement all these variations we assume guard paths are based on a waypoint system. Each waypoint has a location, an action for the guard (wait, move, rotate), and a set of pointers to subsequent waypoints, one for normal movement, and one for each distraction. A guard reaching a waypoint performs the indicated action and then follows the normal pointer to the next one. If a distraction is activated during that action, a guard stops wherever they are, rotates toward the corresponding distraction waypoint, and moves toward it. Upon arrival they again return to normal movement, following waypoints within this (possibly) different list. Waypoints on all routes include these pointers, allowing a guard to react to a second distraction even while still resolving the first.

This design is aimed at modeling guard behaviours as seen in many stealth games, but ensuring we can always predict a guard position at a given time. The key property is that while a guard's location depends on dynamic game state, it is nevertheless a deterministic function of a small set of player actions (distraction activations). More complex guard AI could be used of course, as long as any branching in behaviour still permits us to map (time $\times$ history) to a precise guard position.

Searching

To find possible player solutions in this context we use a *Rapidly Exploring Random Tree* (RRT) search to heuristically generate a detection-free path from the initial location to the goal. Details on applying RRT to stealth can be found in (Tremblay et al. 2013), but RRT is conceptually simple: from the initial node (state) we incrementally grow a search tree by (repeatedly) randomly sampling the state space and attempting to connect the sampled node its nearest neighbour in the tree, terminating once the goal state is reachable. The sampled state in each node includes both space (x, y) and time (t) dimensions, and in our case nodes also carry the time each distraction (if any) was activated.

Sampling is performed in $\langle x, y, t\rangle$ only, with nodes implicitly activating a specific distraction if they have the exact same x, y as that distraction. Serendipitously sampling precise locations is unlikely, so we bias the sampling process. We force 20% of samplings to pick the location of a random distraction point at a random time and try to connect to the tree; other bias rates are possible of course. Nodes that subsequently connect to such a distraction node inherit the fact that that distraction was activated.

When adding sampled nodes to the tree we need to compute whether or not a given state in the tree can reach the sampled state by straight line movement. Feasible connections require maximum player speed is respected, no intersection with obstacles, and no intersection with the fields of view (FOV) of any guards. The obstacle collision is done using the built-in Linecast in Unity. For the FOV check we need to locate each guard given the time and set of activated distractions, and intersect the connecting line-segment with their FOV. Our waypoint design allows us to determine guard position, but as constructing the full FOV polyhedra is complex we perform the intersection heuristically, checking player–guard visibility discretely every 3rd frame, again using Unity's Linecast to determine non-occluded sight and filtering it by guard orientation and distance.

Optimizing RRT

RRT will eventually always find a solution if there is one, but as it is stochastic this can take arbitrarily long. In particular, tree density increases faster than tree coverage in RRT searches, and for the same number of samples it can be advantageous to do more, smaller searches rather than one larger search, especially for goal-directed contexts such as finding game solution paths. We also make use of several additional optimizations to improve scalability and allow us to solve more complex game levels.

Distraction Stacks Biasing the sampling of distraction activation points is important to ensuring they are frequently considered in the search. Doing so naively, however, has a negative impact on the RRT search. RRT samples are connected to the tree through nearest neighbour searches, and having added a sample of a distraction event location to the tree, subsequent samplings of the same distraction event at different times will tend to find prior activations as nearest neighbours. The result is that the tree builds a dense line of samples, extending in time, but not in space, over-focusing the search on the player lingering at the distraction location for different amounts of time.

We improve spatial coverage by extending the set of nodes considered in adding a sample to the tree. Rather than the single nearest neighbour, we instead find the three closest nodes, and look for one that is in a different location, rejecting the sample if none are found. Lingering nearby is still possible by sampling points arbitrarily close, but as our biased sampling is only on the exact distraction location this prevents the search from building trees that over-concentrate nodes on a player remaining at the exact distraction point after they have already used the distraction.

Triangulation and Distance Although wide-open, outdoor locations are sometimes used, stealth games tend to favour complex, obstacle-dense maps that offer multiple opportunities for occlusion. The larger the proportion of space consumed by obstacles the larger the number of RRT samples rejected, wasting significant search effort. We solve this by ensuring all sample points are viable, using a triangulation of the reachable space within the game level. Sampling is then done by first selecting a random triangle in a list weighted by triangle area, and then a random point within it.

Even outside of obstacles, convoluted room/corridor designs common in games mean that sample points far away from the search tree are unlikely to be able to connect to the tree, at least not through simple straight-line movements. This can be improved by constraining the sample range, restricting sample points to ones within a fixed distance from the current tree's bounding box. To fit this in with our triangle-based selection, and as Euclidean distance can be quite inaccurate in game levels, we base our constraint on a triangulation distance measure, computing the distance between points in two triangles as the sum of center-to-center distances along the triangle adjacency path.

Multi-midpoint Compositional Search More complex levels may include multiple distractions that must be solved to reach a goal position. From a given puzzle, however, the RRT search will tend to spread slowly, filling in variations in the solution to one puzzle before proceeding to the next. Excessive resources may thus be required to solve even simple levels if they are large or composed of multiple puzzles.

Our approach to this is to try and break up an overall level search into multiple, smaller searches. We do not of course know where precisely to connect such searches, but we can approximate a division geographically, taking advantage of the fact that any solution to distraction puzzles must also include a basic path solution from start to goal. We can thus find a "midpoint" expressed only in two dimensions, and once we have reached the midpoint in a first RRT search, we then do a second search to try to reach the end, assuming the midpoint as a starting point, and including the state given by the result of the first search.

The success of this approach depends on finding a good midpoint, more or less half-way through the search, and along an actual solution path. We first find all simple paths from the start to the end based solely on the geometry of the level, ignoring guards since their behaviour can change. Each path generates a midpoint, and since different paths may share midpoints we eliminate any duplicates. At this point we perform the two part search for each midpoint in turn until we either fail with all of them or find a solution. This approach naturally extends to splitting it into more than two parts, or for use with different ways to calculate the waypoints that are not in the middle.

Experimental Results

Here we describe experiments applying RRT to 5 different game levels. We use a combination of levels based on actual games and synthetic tests; the former to be representative of actual games, and the latter to investigate more complex level designs that are not well represented in available, fan-based depictions. For all of the experiments, the search was implemented in Unity3D, and the levels were created in it as well. These experiments were all run on a Windows 10 machine, using a Intel i7-6700HQ processor operating at 2.60GHz, with 16 GB of RAM using Unity3D 5.3.1.

Simple Tests

For feasibility tests we used two levels taken from actual games. Figure 1 is modeled on a portion of a game level from *State of Decay* as seen in a Youtube clip (GruntShieldInc 2016). There is a starting area on the left that is fenced off from stationary enemies near a house. The player throws a firecracker from a small tower in the fenced off area, all the enemies go to it, and then he sneaks into the house. We modeled this using the throwing point as the activation location, and the place where it was aimed as the event location. The distraction activation point and the distraction event points are represented with light green circles, labelled "A" and "E" respectively. Orange lines indicate direction and span of enemy FOVs, and the red line represents a possible solution. The actual level is an instance of variation 8 from table 1, however we implement it as an instance of variation 5, as the delay between when the player throws the distraction and when it lands is minimal, and mainly just the length of the throwing animation.

Figure 1: State of Decay Level

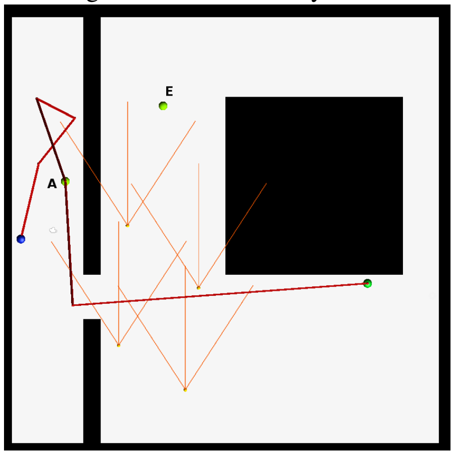


Figure 2 is based on a portion of a level from *Thief* as seen in another Youtube clip (Centerstrain01 2016). The starting area is in the bottom left corner. There is a stationary guard (1) in the top left, and another stationary guard (2) in the

Figure 2: Thief Level

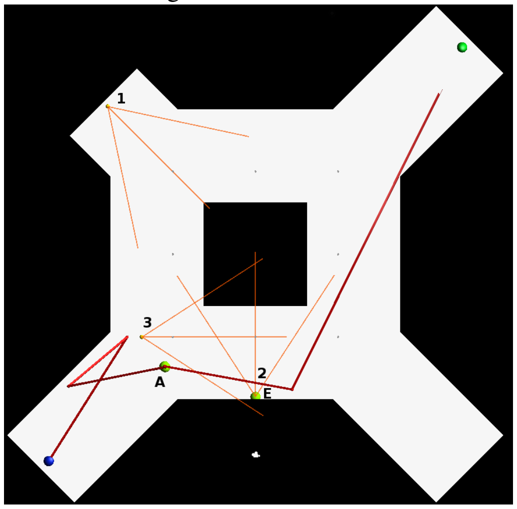
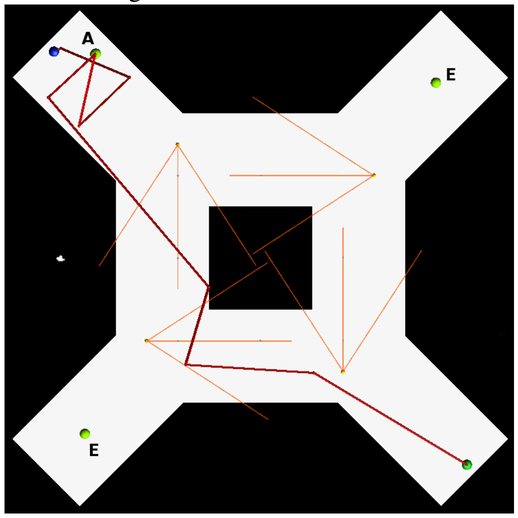


Figure 3: Multi-Alarm Level



middle of the bottom. Finally, guard (3) starts in the bottom left, and patrols in a square around the middle. In the video, the player generates a distraction by shooting a "water arrow," putting out a fire behind the 2nd guard and causing him to turn around to face the wall. We model this abstractly as a distraction, again represented by the light green circles for activation (A) and event points (E). Much like the State of Decay level, this is an instance of variation 8 from table 1 which we model as an instance of variation 5 by ignoring the arrow animation time.

These levels allow us to verify that our RRT search approach is successful in realistic contexts. They also illustrate the value of doing multiple smaller searches over a single large search. For State of Decay, a single (optimized, but non-compositional) search of 2500 samples achieved a success rate of 65% over 100 trials, and using 4000 samples only increasing that to 70%. Using 5 searches, each of 4000 nodes, however, gives a 99% success rate while still taking less than 1s per search. The Thief level is even simpler. A budget of 2500 sample nodes and one search completes in 0.1s with a success rate of 88%; with 4000 nodes and 5 search attempts we had a success rate of 100% with an average time of 0.15s over 100 trials.

Multi-Alarm Level

Figure 3 shows a non-trivial, synthetic level extending the Thief level design. In this level there are four guards patrolling, circling around a central obstacle. The player begins in the upper-left and must traverse to the lower-right. A distraction activation point is near the starting location of the player, with event locations (alarms) in the bottom-left and upper-right. When it is activated, any guard for whom the closest corridor is one containing an alarm point will go to that alarm point before returning to patrolling around the central obstacle.

As well as showing a situation in which the location of the distraction activation does not necessarily match the event location guards go to, here we see that one distraction can lead different guards to different places. Overall this is an example of variation 2 from table 1, although it also works as variation 6, because although the guards that are distracted go back to the same patrol route, they are now shifted in time, and the ensuing pattern of the guard patrols will depend on the exact time the distraction was used.

This complex level is a good basis for understanding the impact of our various improvements to the RRT search. Comparison was made between the basic search (BasicSearch), basic search with prevention of distraction stacks (BasicDSearch), basic search with Euclidean distance bound and distraction stack prevention (DDSearch), triangulated search with distraction stack prevention (SimpTriSearch), and finally, triangulation with distraction stack prevention and the triangle-based distance bound (FullSearch). For each of the experiments 100 trials were done, and the same parameters were used, which were 4000 nodes and five search attempts.

Figure 4: Success Rate of Different Searches

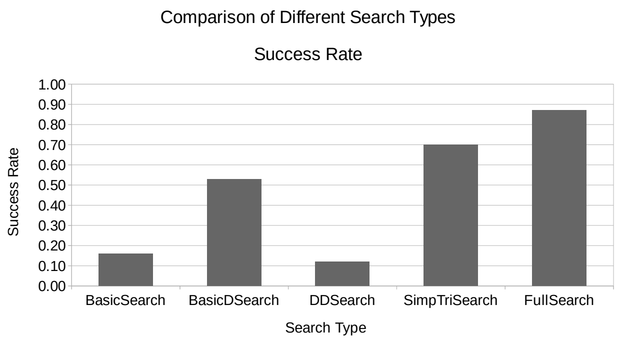
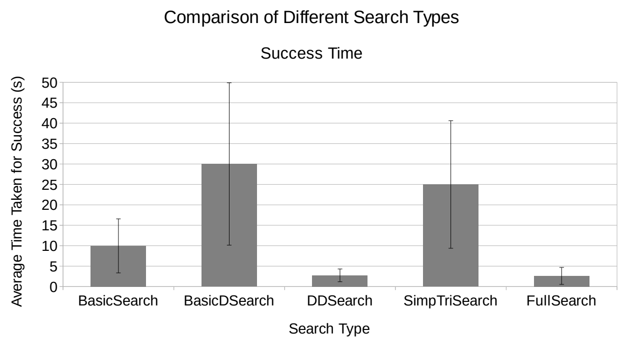


Figure 4 shows success rates achieved under these different execution scenarios. Preventing distraction stacks has the largest increase, improving the 16% rate under basic search to 53%. Distraction stacks not only waste a significant number of samples, in connecting later attempts to use a distraction to earlier ones they effectively prevent consideration of using distractions at a variety of times, and avoiding such stacks has a major impact. Triangulation further improves on this. In analyzing BasicDSearch results we found 56% of samples were within obstacles, a significant waste. Focusing on only viable locations through triangulation raises success to 70%. A final inclusion of our (triangle-based) distance bound in sampling brings success to a respectable 87%. Comparing the latter with the very low DDsearch results (12%) shows a nice synergy in our techniques—a distance constraint should reduce wasted samples that are taken too far away from the main search tree, but this needs to be combined with a technique like triangulation to ensure the constrained sample space is not saturated by infeasible obstacle areas.

Figure 5: Average Success Times in Different Searches



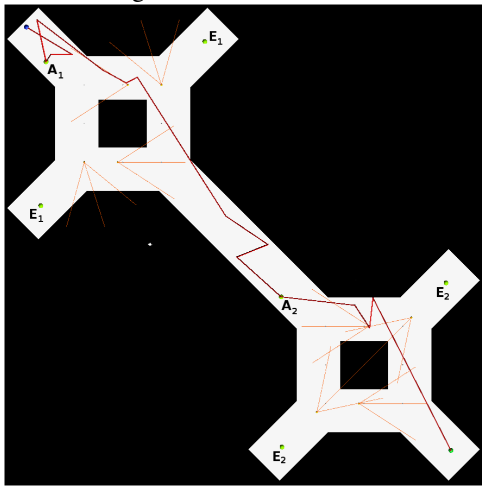
Interestingly, our optimizations are also effective at improving search time. Figure 5 shows the average time of a successful search for all configurations (failed search takes approximately twice as long in all cases, with less deviation). We can see that avoiding distraction stacks slows down the search, use of triangulation has only slight impact, but distance bounds are the major contributor to improving timing. In the end our optimized search is about 3x faster than the naive search.

Two-Part Levels

For more complex levels we further build on our single-distraction puzzles. The two-part level, seen in figure 6, is simply the two copies of the multi-alarm level attached end to end, both of which must be solved to reach the goal. This has a complexity approaching that of a small but complete game level, with multiple stealth puzzles and sets of guards. In much the same way as the multi-alarm level, it is a more complicated version of variation 2 from table 1.

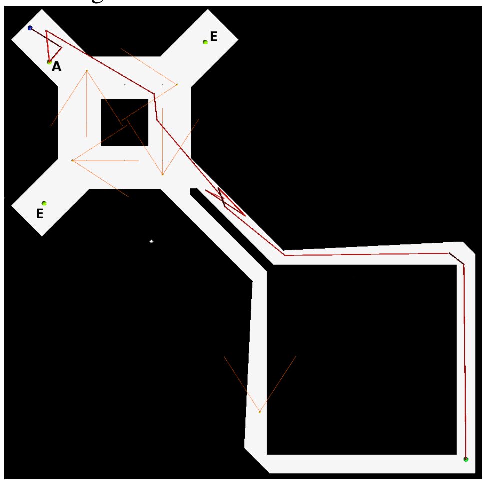
Searching through a large level consisting of multiple puzzles is quite difficult, and the FullSearch mode from the previous experiment was unable to solve the level at all, at least not within 100 trials of our 4000 nodes and 5 attempt searches. Enabling the midpoint compositional optimization allowed the search to find a solution 28% of the time. Although not a high success rate this is important in showing

Figure 6: Two-Part Level



that a compositional version of the search is effective at allowing the search to solve levels too complicated to solve without it. The efficacy comes at some cost, however, and average successful search time is increased to 301.7s. Interestingly, this cost is biased toward the second half of the search—in a successful midpoint search the first half uses an average of 5540 samplings summed over our allowed 5 attempts, while the second half uses 8671. This is likely due to the fact that for the second half of the search, it starts in the middle of the level and then searches in both direction simultaneously. Thus, less of the search resources are used effectively for the second half of the search. We do not restrict the search as in general a solution may involve backtracking or complicated paths, although for some levels it may be heuristically possible to improve success by biasing the geometric positions sampled in the search toward the goal position.

Figure 7: Two-Part Choice Level



Choice of midpoint here is highly heuristic. The two-part level has the advantage that traversing the central corridor is essential in any solution; in general, however, we cannot assume that the middle of a path is necessarily part of an actual solution path, let alone halfway. As a final set of experiments, we thus evaluated our design on the two part choice level shown in 7. The first half of this level is simply a multi-alarm level, but attached to the end there is a choice of two corridors to go down. The clockwise choice leads to the end of the level, while the counter-clockwise direction seems to lead to the end of the level, but has a stationary guard blocking it who cannot be distracted. This level would be difficult for a player to solve in a stealth game, as the choice of which corridor to go down is not necessarily obvious ahead of time, and so multiple play attempts would likely be required. Much like the regular two-part level, the distraction in this level is an example of variation 2 from table 1, and we again use 5 attempts of a maximum of 5000 nodes for each search, averaged over 100 trials.

This level includes two distinct midpoints in terms of pure pathing solutions, one on each side of the corridor divider. A single midpoint search has a success rate of 30%, and consideration of both midpoints raises that to 55%, unsurprisingly about $2\times$ higher. Success timings are relatively comparable though; single midpoint completes in 126.9s on average, while multiple takes 153.3s. This better scaling is likely due to the fact that success time depends more on the first search, as the second merely needs to path to the goal or not, and does not need to solve a complex distraction puzzle.

Related Work

Our work builds on the well known *Rapidly Exploring Random Tree* (RRT) algorithm as a basis for heuristic search of the stealth space. RRT is a random-based pathfinding algorithm commonly used in robotics for navigation planning (Kuffner and LaValle 1999), and has more recently been applied to games research (Bauer and Popović 2012; Tremblay et al. 2013). Numerous works focus on improving the basic RRT algorithm in different ways, although these do not always apply to our context. Our compositional design, for instance, is similar to bidirectional approaches, where RRT searches are done from both the start to the goal and vice versa, aiming to meet more or less in the middle (Kuffner and LaValle 2000). In our case the need to guarantee the full, evolving game state matches where the searches meet means searches cannot be entirely independent. A game context can also make some optimizations simpler. A *retraction-based* RRT planner, for example, improves sampling by pushing samples within obstacles out to the edge of obstacles, where they may still be viable (Zhang and Manocha 2008). We address this problem by using a spatial decomposition to ensure we never sample inside obstacles in the first place.

Previous work on stealth games has also built on the RRT search (Tremblay et al. 2013). This work assumed the movements of guards was fixed, and had no reaction to the player. This was in fact integral to the search process, since the space was pre-populated by both obstacles and the motion of guard fields of view, as projected into the time dimension. In this way search queries could trivially determine whether a position was viable by testing for intersection with the known position and orientation of guards at a given time. Tremblay *et al.* later extended their design to include combat, as a basic form of player–guard interaction (Tremblay, Torres, and Verbrugge 2014). However, this did not affect the movement model, and the guard movements remained deterministic: combat was detected by intersection of a proposed RRT edge with a guard field of view, resolved through an abstract combat model, and resulted in the state space branching, representing contexts in which each guard was alive or not after a point, and thus whether their field of view mattered or not for non-detection or future combat, but not otherwise changing guard motion. Their design also incorporated health packs, incorporated into the RRT search by a biased sampling procedure, a technique we also make use of for selecting distraction points.

Other work on stealth has aimed at creating believable guard search behaviours. *Third Eye Crime* used a form of *occupancy map* in order to have the enemies track the position of the player, showing a probability distribution that becomes more diffuse the longer a player remains out of sight of the enemy (Isla 2013). Planning techniques have also been used to solve game levels based on gameplay constraints, displaying solutions as a storyboard (Pizzi et al. 2010). This work was aimed at level generation more generally, but as it used *Hitman* as a prototype it did involve elements of stealth in addition to combat.

Our work here is specifically focused on dealing with distractions. The use of distractions is well known trope in stealth games, and can be found in perhaps the majority of stealth games, with the gaming website *Giant Bomb* listing 66 examples of game titles that use this mechanic (Giant Bomb 2016). Analysis of the mechanic, however, is much less frequent, with the only prior work we could find in this area being a game studies perspective on indexical storytelling, where distractions are mentioned as examples of how the locations of objects can influence both the player and other non-player characters (Fernández-Vara 2011).

Conclusions & Future Work

In allowing players to dynamically modify the behaviour of NPCs, distraction mechanics introduce a complex, dynamic element into stealth gameplay, complicating ahead-of-time analysis. Our work shows the impact is nevertheless manageable, and by formalizing distraction state into the game state model we can still determine guard positions. We note that our design can be applied in the presence of arbitrarily complex guard responses, as long as the behaviour is constructed as a deterministic function of state.

Our future work is aimed at both further scaling search improvements, and extending the design to other stealth puzzle features, such as sound propagation. As extracting puzzles from actual games is laborious, building a formal suite of stealth benchmarks would also be useful in showing breadth and for formal testing.

Acknowledgments

This work supported by the Natural Sciences and Engineering Research Council of Canada, Application ID #249902.

References

Bauer, A. W., and Popović, Z. 2012. RRT-based game level analysis, visualization, and visual refinement. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*.

Centerstrain01. 2016. Thief: Stealth walkthrough - master - ghost - part 26 - chapter 7 - the hidden city 1/2. `https://www.youtube.com/watch?v=icyKiYFWbBE`. Accessed: 2016-05-02.

Fernández-Vara, C. 2011. Game spaces speak volumes: Indexical storytelling. In *Digital Games Research Association*.

Giant Bomb. 2016. Enemy distraction. `http://www.giantbomb.com/enemy-distraction/3015-5692/games/`. Accessed: 2016-04-02.

GruntShieldInc. 2016. State of decay - distraction. `https://www.youtube.com/watch?v=7pApHmrG6LY`. Accessed: 2016-04-02.

Isla, D. 2013. Third Eye Crime: Building a stealth game around occupancy maps. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 206–206.

Kuffner, J. J., and LaValle, S. M. 1999. Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 473–479.

Kuffner, J. J., and LaValle, S. M. 2000. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, 995–1001. IEEE.

Pizzi, D.; Lugrin, J.-L.; Whittaker, A.; and Cavazza, M. 2010. Automatic generation of game level solutions as storyboards. *IEEE Transactions on Computational Intelligence and AI in Games* 2(3):149–161.

Tremblay, J.; Torres, P. A.; Rikovitch, N.; and Verbrugge, C. 2013. An exploration tool for predicting stealthy behaviour. In *The Second Workshop on Artificial Intelligence in the Game Design Process*.

Tremblay, J.; Torres, P. A.; and Verbrugge, C. 2014. An algorithmic approach to analyzing combat and stealth games. In *IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8.

Zhang, L., and Manocha, D. 2008. An efficient retraction-based RRT planner. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 3743–3750. IEEE.