Adversarial Navigation Mesh Alteration

D. Hunter Hale and G. Michael Youngblood
The University of North Carolina at Charlotte
Game Intelligence Group, Department of Computer Science
9201 University City Blvd, Charlotte, NC 28223-0001
{dhhale, youngbld}@uncc.edu

Abstract

Game environments are becoming more and more mutable from the actions of both Players and Non Player Characters (NPCs). However, current generation AI agents do not take advantage of the tactical abilities these mutable worlds provide. We propose a method to make the game agents aware of the mutability of the world by extending their repertoire of abilities to include world alteration commands and some evaluation functions, which determine when and where to alter the world for the greatest tactical gain. Primarily, our work focuses on the Adversarial Navigation Mesh Alteration (ANMA) algorithm, which evaluates potential changes to the map in adversarial environments from an attacker and defender point of view. We present an empirical evaluation of the ANMA algorithm in a Capture The Flag (CTF) simulation environment with several teams of agents. One group of agents (adaptive) lacks the ability to initiate world deformations, but they can respond and re-plan to take advantage of world modifications. The second team of agents (builders) can only generate additional paths through the world using the attacker portion of ANMA. The third team of agents (universal) is able to fully deform the world by generating new paths or removing existing paths using both the attacker and defender sections of ANMA. We evaluated these teams and observed that builder agents beat adaptive agents at a rate of 1.33 to 1. The more advanced universal agents beat adaptive agents at a rate of 2.75 to 1 and builder agents 1.4 to 1.

Introduction

In modern games and simulation environments there has been a continuous move towards more and more realism. In particular, there have in recent years been considerable improvements in the fidelity of physics models and dynamic interactions occurring between agents and the world geometry throughout both games and simulations. Game engines now support the ability to dramatically alter the geometry composing a world or simulation level at runtime without any appreciable slowdown in rendering (Jurney and Hubick 2007). Large scale changes to the world geometry usually take the form of physics based interactions (e.g., vehicles being used to punch a hole in the side of a building to create a doorway, or explosives collapsing a large quantity of rubble into a formerly passable street). Such physics based interactions require very little planning to initiate, but more planning to have the resultant changes in the world turn out as expected. There have been several recent and highly successful commercial games that utilized highly dynamic environments (e.g., Battlefield: Bad Company 2, Company of Heroes, Fracture, and Red Faction: Guerilla). Adapting to such large scale changes in the environment poses a challenge to embodied agents moving around in that environment.

Traditionally, work has focused on addressing two problems associated with agents in highly dynamic environments. The first is how the highly dynamic world is represented to the agent. Generally, most games present some form of simplified representation of the world to agents (Tozour 2004). This representation contains a listing of empty space (negative space) as well as another listing of areas of the world which are occupied by obstructions (positive space). Agents use this representation to reduce the computational burden of calculating paths throughout the game world. Without this representation, an agent would have to consider every possible point in negative space when creating a path. Using a navigation mesh (the most common form of such a representation (McAnils and Stewart 2008)), an agent can localize its current position into a single negative space region and its target location to another region. Then the agent can view the representation of walkable space as a graph where each negative space region is a node on this graph and negative space regions that share a common edge (a gateway) are connected via an edge. The agent can then quickly calculate paths by searching this graph for the shortest route from its origin to its destination.

The second problem is one of ensuring that the agent is still able to accomplish the task it was working on regardless of changes that occur in the world. Generating agents which are highly adaptable is the first step in dealing with dynamic environments. In particular, when navigating along a preplanned path or completing another task, an agent needs to be able to adapt and take advantage of potentially better ways to accomplish their goals. Conversely, if the planned path to a goal is no longer viable then they need to be able to quickly re-plan with a minimum of overhead to accomplish their original goal in the revised world.

These are several solutions available for both of these problems. Spatial representations in dynamic worlds can be generated using Dynamic Navigation Meshes (Hale and Youngblood 2009; Axelrod 2008) , Adaptive Voronoi Diagrams (Sud et al. 2008), subdividing the world representation in small deformable cells (Jurney and Hubick 2007), or Adaptive Roadmaps (Sud et al. 2007). Producing agents capable of adapting to dynamic changes in the world also has several commonly used solutions: Planners (Fikes and Nilsson 1971), Task based systems (Rickel and Johnson 2000), or On Demand Replanning (Berg 2006). However, we submit that there is a third problem with agents in highly dynamic environments. Namely, at what times and how should the agent initiate an alteration of the environment?

We propose a solution in adversarial contexts for this third problem through the use of the Adversarial Navigation Mesh Alteration (ANMA) algorithm, which builds off work both in dynamic replanning and navigation mesh regeneration. This algorithm contains two simple sub-algorithms (attacker, defender) that are derived utilizing path planning algorithms and spatial representations. These sub algorithms allow agents to initiate some world modifications for the purposes of improving or degrading the pathfinding abilities of the other agents and players moving around in the environment. In particular, ANMA focuses on the usage of world alterations to assist agents with path planning towards a given goal location, and the destruction of passable areas in the world representation to assist an agent or team of agents with the defense of predefined goal locations.

Related Work

This question of how and when an agent should initiate alterations to the world has been traditionally addressed through the use of planners. In particular, Jeff Orkin's work to extend the STanford Research Institute Problem Solver (STRIPS) planner (Fikes and Nilsson 1971) into the Goal Oriented Action Planning (GOAP) (Orkin 2004) system has proved to be especially interesting to us. This system has obvious applications to games and simulations as it was later integrated in the agents present in the game F.E.A.R (Orkin 2006). GOAP presents an interesting approach to planning in dynamic game environments in that every agent has a goal it wants to accomplish. Objects and locations in the world have a list of actions the agent can perform on them. There are pre-conditions that govern when a given action or item can be utilized, and after activation, all items and actions yield a result which alters the state of the world. Using GOAP an agent will search the space of potential actions and results until it finds the minimal cost chain of them that would result in the fulfillment of its final goal. For instance, an agent that wants to enter a room could determine that breaking a nearby window would be faster than entering through the more distant door. This approach is similar to the one used by ANMA in that it uses a search-based approach to generate plans for agents that can then interact with and deform the environment based on these plans, but unlike ANMA the GOAP planner searches through a list of pre-defined actions provided by the objects present in the world. ANMA searches through the automatically generated representation of world space looking for areas it would be beneficial to the agents to alter.

The ANMA algorithm also draws heavily from two general areas of previous research. The biggest contributor to ANMA comes from the work to convert navigation meshes and other spatial data structures from static and immutable structures into highly dynamic objects that can be updated in real time to reflect the changes in the world environment. In particular, the ANMA algorithm works best when executed on some form of Dynamic Navigation Meshes (Hale and Youngblood 2009; Axelrod 2008) instead of other spatial representations, since unlike other spatial representations, the navigation mesh provides a completing listing of negative space and positive space that proves helpful to the algorithm. Secondly, the ANMA algorithm is reliant on fast dynamic searches of the navigation mesh (Stentz 1994) in order to determine which regions to convert from positive to negative space to create shortcuts for the agent or which regions to convert from negative to positive space in to better defend an area.

Methodology

The Adversarial Navigation Mesh Alteration (ANMA) algorithm contains two basic sub algorithms. The first subcomponent (attacker) of the ANMA algorithm slots into most huristic-based path finding algorithms to improve agent navigation through negative (open) space via the conversion of positive (obstructed) space into additional negative space regions when it would benefit the agent. The second sub algorithm (defender) is a higher level extension to the agent's reasoning ability that highlights and suggestion regions in a navigation mesh or navigation graph representation, which it would be advantageous to the agent to remove. This is removal is designed to reduce the number of potential approaches to a position the agent is supposed to protect. These algorithms work best, and in the case of the defender sub algorithm require, an adversarial context for the agents to fully utilize them in planning.

Path Planning Through Positive Space

The first part of the ANMA algorithm (attacker) attempts to answer the question of "When should an agent plan to use its ability to destroy positive space to assist with path planning." To answer this question the ANMA algorithm provides a simple extension to popular path planning algorithms such as A* or D* which allows them to consider the effects of altering the navigation space representation when determining the optimal path the agent should take to reach his goal. Doing so allows the agent to create new paths which are often shorter than paths entirely in existing negative space regions.

Consider the example of a building that the agent wants to enter, which the agent is standing next to. The area around the building is fully described in the agent's navigational representation, so it is rather trivial to plan a path around the building to the door and then enter it. However, consider what would happen if the building is rather large and the agent is on the other side of the building from the only door. In that case, the agent would take considerable longer to enter the building. However, what if the agent did not have to go around the building, but had some capability to create its own doorway in the building near where it currently is (e.g., a sledge hammer to take out a wall, a brick to break a window, or an explosive charge to put a bigger hole in the building). Now the question becomes whether it would be faster to walk around the building to an existing door or to install a new door (this is also what tends to limit this algorithm to adversarial games or simulations since non-adversarial contexts tend to frown on impromptu doorway installations via explosives).

Algorithm 1: The ANMA algorithm looking for potential positive space regions to convert to negative space during a search. The example is in the context of a node expansion in an A* search.

```
// Sorted List of Regions to consider
List OpenList;
// List of Regions that have already
   been considered for path planning
List ClosedList;
// Standard checking of the first
   node in the open list
FirstNode in OpenList;
for Neighbor of FirstNode do
    if Neighbor.isPositiveSpace then
        // Calculate the cost to convert
           this node to negative space
        float CostToConvert =
        Neighbor.findConversionCost();
        // Generate normal heuristic
           costs to traverse this node
        Neighbor.f = FirstNode.f +
        Neighbor.findH();
        Neighbor.f += CostToConvert;
        OpenList.append(Neighbor);
    else
        // The node is negative space
           treat it normally
```

Our algorithm to accomplish this task of path planning through positive space is quite simple and can be included as a few extra function calls in most existing search algorithms as shown in Algorithm 1 in the context of A* search (for a good survey of search techniques see Artificial Intelligence: A Modern Approach (Russell and Norvig 2003)). When a search algorithm is evaluating a potential path to a goal, it calculates the benefits of moving to any given region using a heuristic. This heuristic generates an approximation of how far a potential move would leave the agent from its destination, and when added to a stored value indicating the cost to get to that potential location, allows an agent to rank potential paths through the environment. This system works very well for movement through negative space, and it makes sense to extend it to movement through positive space. There are two parts to this extension: the first is calculating the raw cost of moving through the positive space region as if it were negative space—this can be done using the same calculation as for negative space. The second, harder part comes from calculating the cost of converting a positive space region into negative space. This is where having a navigation mesh spatial representation (or some other representation with a listing of positive space) is very useful. Using this representation, the size and composition of the target positive space region can quickly be determined. These factors combined with the agent's ability to manipulate positive space (agents with explosives can perform faster manipulations than agents with wrecking bars) can be used to generate a cost to convert for any given region of positive space. This cost of conversion is then added to the cost of movement through the positive space region, and this total value can then be used in the path planning algorithm like a normal negative space region.

Planning for the Obstruction of Negative Space

The second, defensive half of the ANMA algorithm deals with a slightly higher level problem which occurs somewhat less often. This portion of the algorithm answers the question "How can an agent tasked with defending an area from other agents or players alter his local environment to restrict the number of potential access points?" Our approach to this problem uses conventional search algorithms on the world space representation to locate potential entry ways into the area the agent is in charge of defending and then prioritizing the order in which these negative space regions should be converted into positive space regions through the introduction of obstructions.

Consider the example of a building which an agent has been assigned to defend from other agents and players. In particular, there is a single room on the second floor of the building which the agent must defend at all cost. By examining the navigation representation of the environment, the agent can determine the listing of entry ways into the area they need to defend, in this case, let us say there is a ladder connecting to a second floor window from outside, the obvious front door entrance to the building, and finally a hole some impatient person blew in a wall instead of coming in the front door. Now the agent needs to determine which of these potential enemy attack routes they should close off first (patching the hole in the wall, locking the front door, or hiding the ladder leading to the second floor entry).

In order to the use the defensive sub-algorithm of ANMA, we first require three pieces of information. First, we need to know the extents of the area the guard should remain in. Secondly, we need to know which direction the other agents or players we will be defending against will be approaching from. Finally, we need to know the main point we are supposed to be protecting. Using this information, we are able to determine which sections of the navigation representation it would make the most sense to alter and invalidate to prevent or delay the enemy moving through them. These requirements are also what limit the defensive portion of the algorithm to adversarial situations. While the attacker portion of ANMA can be used to create non-adversarial path planning agents (albeit ones who have little respect for the condition of the world as they move through it) it is unlikely a non-adversarial scenario could supply the three requirements to implement defensive-ANMA in an agent. The

Algorithm 2: This is the defensive half of the ANMA algorithm which determines the most important negative space nodes to seal to protect an area.

```
// Initially the algorithm starts
   with its 3 requirements
$Region\ regionToDefend$;
$Region\ attackerOrigin$;
$List\ PossibleRegions$;
// We will also assume we have path
   finding algorithms on hand
$PossibleRegions = findPath.(regionToDefend, attackerOrigin)$;
// Validate the target region is in
   the assigned guard area
repeat
    // Select the most connected
       region using distance from
       agent as a tie breaker
    $DestroyRegion = PossibleRegions.getHighestDegree()$;
until $!DestroyRegion.isInArea()$ ;
```

defensive sub-algorithm works by calculating the optimal path(s) from the location the agent is supposed to defend to the areas the enemy is expected to advance from using a search algorithm as shown in Algorithm 2. It then finds the negative space region along this path(s) with the lowest degree in the navigation graph representation of world space (the negative space region with fewest neighboring regions). If two or more regions have equal degree then the closer one to the defender is selected. Additionally, this search for a target region is restricted to the areas of the world where the agent is supposed to be guarding. This is to ensure that our defensive agent does not go running off to try and block the exits of the enemy base and in fact stays inside the area he is supposed to be defending. After locating this target region, the agent will move to it and attempt to use its ability to alter the world geometry to block passage from this node towards the enemy approaches by introducing new obstructing positive space areas. A final restriction on the possible target regions for the agent to alter is that the agent needs to have the ability to introduce sufficient positive space to block off an area. This can be determined in advance by comparing the data on the negative space region stored in the spatial representation of the world to the agent's ability to manipulate negative space regions (e.g., does the agent have concrete barriers it can deploy to block roads, plywood it can nail up to block doorways or windows, or just some rope it can string across a path). The agent then repeats this process of finding a low degree negative space region on the approach to the area it is supposed to guard and sealing the area off until there are no open approaches or it runs out of blocking materials, at which point it can fall back to it's non-ANMA behavior.

Experimentation

We performed a series of experiments using our agents running the Adversarial Navigation Mesh Alteration (ANMA) algorithm to determine their effectiveness against more traditional forms of agent behavior. To do this evaluation in an adversarial context, we decided to use the Capture The Flag (CTF) game type with two teams of agents. Our implementation of the CTF game was as follows: A CTF game is played on a field containing some randomly distributed quantity of open space which players can move around in, as well as obstructions which block line of sight and movement. This space is broken down into two evenly distributed halves, with each team having "control" of one half. Additionally, each team possesses a base inside the area they control that contains a flag. Each team was composed of 10 agents. Of these 10 agents half of them were assigned to attempt to capture the enemy's flag. The other half of the team was tasked with defending their own flag. A flag capture occurred if an agent picked up the other team's flag. If an agent entered the same negative space area as an opposing agent on the side of the field the other team controlled, then they were captured. Captured agents were removed from the game and after a short delay returned to a random location on their own side of the playing field. Games were concluded after a single capture was scored. In order to prevent stalemates, agents were required to move from one area to another each turn if possible.

Instead of spending time implementing a fully fledged graphical CTF game to test our agents in, we designed and built an agent test bed to simulate CTF games that allows us to perform agent testing and comparison very quickly. In this simulator, we use a representation of the world instead of an actual world model; this allowed us to rapidly and procedurally generate many random worlds. In this representation, negative space regions are represented as nodes. The gateways between negative space regions are represented in this simulator as edges between the nodes. In this manner the simulator represents and maintains the navigation graph of the playing field in question without having to worry about perfectly updating the underlying navigation mesh. Additionally, positive space obstructions between any two given areas of negative space were also maintained and represented in the simulation. These positive space obstructions were represented as inactive links, which are not traversable to agents. Using this simulator, we are able to procedurally construct random levels to evaluate our agents in. Our procedural world generation algorithm works as follows. First, eighty to one hundred negative space regions are seeded randomly in the world with a bias to ensure that they are not clumped too closely together. Then all of the neighboring negative space regions are determined using a simple distance metric. After this determination, neighboring regions are randomly determined to be either connected one to another with a gateway, or adjacent but obstructed one from another by positive space. The results of this determination are then stored as links for the simulation. Finally, the two bases are selected, one for each team from the available pool of negative space regions, such that the base is within the back 10 percent of each team's territory (remember that each team is considered to be in control of half the map) and the chosen base has at least three adjacent negative space regions connected to it via gateways to help ensure it is accessible. This simulation representation allowed us to evaluate our proposed agent design on a larger number of worlds than would have been possible if we were using a full fledge CTF game engine. In particular, we were able to evaluate our agents on 180 unique test levels. One of the test levels generated by our tool is shown in Figure 1.

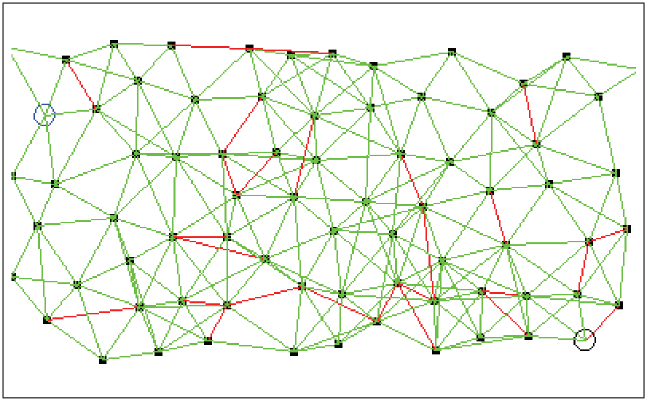

Figure 1: This image shows one of the procedurally generated levels used in our simulated CTF game. Negative Space areas the agent can occupy are shown as black squares, active gateways are shown in green, blocked gateways are shown in red, and the bases with each team's flags are shown as circles.

We evaluated two types of environmentally manipulative agents as well as one non-manipulative adaptive agent. The first type (the Builder) is able to employ the path planning through positive space algorithm to create new gateways and regions in order to facilitate quicker movement through the level. However, the Builder is unable to generate new positive space regions to close off potential routes through the level. The second type of environmentally manipulative agent we call the Universal agent. This agent is capable of both planning paths through and removing positive space obstructions and dynamically placing new positive space obstructions into the world. This allows the Universal agent to close off potential access routes to its flag and set up choke points on approaches to the territory it is defending. We did not evaluate agents who were able to destroy connections between nodes but not build them, due to the fact that these agents trap themselves in their base. The final agent type we tested was designed to be highly adaptable to changes in a dynamic game world. This agent was coded to use D* for path planning (Stentz 1994) through the world representation so that it could take advantage of changes to the underlying navigation graph produced by the more advanced agents, or adapt as best it could to obstacles thrown in it's path by the world manipulative agents which can destroy pathways.

All of the evaluated agents had the same basic behavior patterns depending on the role of the agent. The defending agents regardless of type patrolled their side of the map and attempted to move into and capture enemies they observed in adjacent regions. Additionally, agents of the Universal type used their ability to place positive space objects in the world to barricade off gateways between regions such that in whatever region they are in, the most optimal path back to their own flag will be sealed. Adding this simple logic to take advantage of placing positive space yields a set of agents that automatically build mazes and choke points of narrow or obstructed corridors which approach their flag. Conversely, the attackers all were rather single minded in that they took the best available path to the goal. However, both of the advanced agents (Builder and Universal) will construct paths which require them to cut holes in positive space obstructions or build bridges over empty areas if such a detour would result in a more optimal path to the other team's flag.

We set up test scenarios featuring all possible unique parings of these agents (Universal vs. Adaptive, Universal vs. Builder, and Builder vs Adaptive). Each pairing of agents played sixty matches on our CTF simulator. The results of these matches are given in Table 1 shown below. After 30 of the 60 matches, we switched the side of the map each team was on. We also included the results of running each agent type against itself an additional thirty times as a logical check to verify the integrity of the simulation and show that neither side has a positional advantage, since if the same type of agents compose both teams then the win/loss ratio should be approximately even.

Table 1: Comparing the performance of different types of agents on randomly generated CTF levels (showing wins to loses). Matches vs. the agents own type are provided to verify the integrity of simulation.

vs	Adaptive	Builder	Ultimate
Adaptive	31 to 29	-	-
Builder	34 to 26	30 to 30	-
Ultimate	44 to 16	35 to 25	28 to 32

When examining our results, we see that our hypothesis verifies that agents that can manipulate the environment do perform better in CTF games than basic adaptive agents. Looking through the data, we see that Builder agents win matches against Adaptive agents at a rate of 1.33 to 1. This is a slight performance improvement, and it can be explained by the fact that the only advantage that the Builder agents possess is that they have a shorter path to the opposing team's flag. However, this advantage is somewhat transitory, as the new paths the builder agents create can then be utilized by the simple adaptive agent to attack the Builder agent's flag.

The more advanced Universal agent performs even better against the straight adaptive agent, winning games at a rate of 2.75 to 1. This is in line with expected performance, as the Universal agent design should dominate a purely adaptive agent. However, the possibility does exist that Adaptive agents would be able to capture the flag the Universal agents are guarding before the Universal agents can seal off all of the approaches to it. This is seen in the occasional Adaptive agent's victories. But most of the time, the Universal agents are successful in blocking all approaches to their base (e.g. Figure 2). The Universal agent also fares well when compared to the Builder agent, with a win ratio of 1.4 to 1. These agents are more evenly matched, but the Universal agent gains a slight edge since the Builder agent has to stop and open passageways through blocked paths before traversing them to attack the Universal agent's flag each time they attack. Conversely, when attacking the Builder agent's flag after the first wave of agents goes in, the Universal agents will not have to open or reopen any passageways, since there are no agents creating blockages or obstructions on that side of the map.

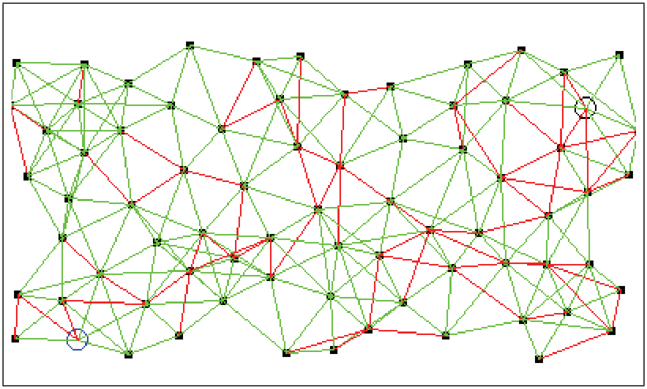
Figure 2: This image shows one of our procedurally generated levels after two teams of Universal agents have obstructed many of the potential gateways between regions, notice all of the red inactive links in the image.

Conclusion

In conclusion, we have shown that through the use of the Adversarial Navigation Mesh Alteration (ANMA) algorithm we can generate agents which can initiate logical alterations to the game geometry in support of their own goals. These geometric deformations are primarily supported and engineered using searches through the navigation space representation instead of more complex procedures and rules in the agent design. This results in very simple agents which can still perform complex actions. These agents are able to dynamically remove world space objects in an intelligent manner to find the shortest path between two points. Additionally, ANMA-enabled agents in a defensive context are able to automatically generate obstructions and choke points in the world, greatly reducing the potential approaches into the agents' defensive area. We showed with our simulator testing that agents running the full ANMA algorithms win against otherwise identical agents without ANMA at a rate of 2.75 to 1 through multiple rounds of Capture The Flag. Overall, we feel that ANMA agents are a good step towards addressing the problem of agent-dynamic environment interaction.

Acknowledgments

This material is based on research sponsored by the US Defense Advanced Research Projects Agency (DARPA). The US Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the US Government.

References

Axelrod, R. 2008. *AI Game Programming Wisdom 4*. Charles River Media. chapter 2.6 Navigation Graph Generation in Highly Dynamic Worlds, 125–141.

Berg, J. V. D. 2006. Anytime path planning and replanning in dynamic environments. In *Proceedings of the International Conference on Robotics and Automation*, 2366–2371.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Hale, D. H., and Youngblood, G. M. 2009. Dynamic updating of navigation meshes in response to changes in a game world. In *Florida Artificial Intelligence Research Society Conference*.

Jurney, C., and Hubick, S. 2007. Dealing with destruction: Ai from the trenches of company of heroes. In *Proceedings of the Game Developer's Conference (GDC)*.

McAnils, C., and Stewart, J. 2008. *AI Game Programming Wisdom 4*. Charles River Media. chapter 2.4 Intrinsic Detail in Navigation Mesh Generation, 95 – 112.

Orkin, J. 2004. Symbolic representation of game world state: Toward real-time planning in games. In *AAI Workshop on Challenges in Game AI*. AAAI Press.

Orkin, J. 2006. Three states and a plan: The ai of f.e.a.r. In *Proceedings of the Game Developer's Conference (GDC)*.

Rickel, J., and Johnson, W. L. 2000. Task-oriented collaboration with embodied agents in virtual worlds. 95–122.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Pearson Education, Inc.

Stentz, A. 1994. Optimal and efficient path planning for partially-known environments. In *IEEE International Conference on Robotics and Automation*. New York, NY, USA: IEE.

Sud, A.; Gayle, R.; Andersen, E.; Guy, S.; Lin, M.; and Manocha, D. 2007. Real-time navigation of independent agents using adaptive roadmaps. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, 99–106. New York, NY, USA: ACM.

Sud, A.; Andersen, E.; Curtis, S.; Lin, M.; and Manocha, D. 2008. Real-time path planning for virtual agents in dynamic environments. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, 1–9. New York, NY, USA: ACM.

Tozour, P. 2004. *AI Game Programming Wisdom 2*. Charles River Media. chapter 2.1 Search Space Representations, 85–102.