Terrain Reasoning for 3D Action Games

William van der Sterren
CGF-AI, Veldhoven, the Netherlands
william@cgf-ai.com

1 Introduction

This paper discusses AI terrain reasoning for 3D action games. Terrain reasoning is the AI capability to take into account terrain in his planning, decisions, actions and communication. Terrain reasoning is a necessity if both AI and terrain play an important role in the game. And that often is the case in 3D action games.

Level designers spend weeks designing a challenging game world with well-thought out battle arenas, multiple access routes, risky approaches to highly valued power-ups, defensive strongholds, and neutral zones.

However, the traditional action game AI was unaware of all the thoughts, efforts and design that went into the level. Instead, he ran around almost blind, via shortest paths and sometimes guided by a few static hints from the level designer. "Don't ask me, I only work here!"...

This paper presents terrain reasoning based on the waypoint graph typically present in 3D action games. It discusses terrain reasoning concepts, and how the waypoint graph can be reasoned about. The paper discusses and demonstrates the relation between tactics and terrain. A case-study shows how to develop off-line and in-game reasoning to pick good sniping spots, and how to use game-play feedback to create adaptive AI with tactical understanding of the terrain.

A short threat prediction example demonstrates one of other ways to reason about the terrain. The paper also addresses the major issues in implementing such a terrain reasoning system.

But first, let’s look at what terrain reasoning can offer.

2 Why Terrain Reasoning?

Today’s action games offer plenty of situations where better AI understanding of the terrain would be valuable to the gamer and level designer. For example:

- *Recognizing key terrain features and communicating about them*
  The player issues a “provide suppressive fire” command while pointing to a door in a distant building. Typically, it is not as obvious to the AI as it is to the player that a door is pointed to rather than the wall next to it. But the player expects to hear the AI confirm his order: “roger, will provide fire on that door”.
- *Distinguishing between good and bad locations*
  All locations are not created equal. Some locations are great to ambush other actors, whereas other locations are not. An AI actor who picks the right location for an action simply is more convincing.

- *Interpreting location based performance*
  After being lured into the same narrow tunnel to receive a rocket for the 3rd time in a row, it would be great if the AI would recognize the tunnel as a bad place to be. The AI should adapt its tactics accordingly, instead of trying to dodge the rocket without the space to do so.
- *Automating part of the level annotation*
  The level designer favorite job probably isn't manually editing dozens of waypoints to tell the AI "camp here", "avoid this location", "here's some cover". An AI capable of automatically annotating the level itself will save the level designer time. And AI provided feedback on its interpretation of the terrain might be welcome as well.

  Automated AI interpretation not only is useful for the professional level designer, but even more for the amateur level designer who is less familiar with the needs of the AI.
- *Including the terrain in tactical considerations*
  Few things are as exciting as engaging a squad of AI actors who coordinate their actions, and alternate suppression fire with grenade lobbing. Part of that excitement disappears if the same squad applies those same tactics at a less suitable location: the AI should understand that attacking from, for example, an elevator is different than from a warehouse full of crates.

Terrain reasoning may be an important ingredient to address the issues, but it cannot do so without enhancements in other parts of the AI, such as the planner, state machines, combat rules, etc. These other parts of the AI have been receiving plenty of attention in literature.

3 Ingredients and Concepts

Terrain reasoning AI subsystem consists of the following components:

- terrain representation; and
- functions to construct, query and manipulate that terrain representation.

The amount of terrain in an action game makes it infeasible to efficiently handle the terrain in raw geometry format, or by means of a rule bases or neural nets: both the size and the geometric detail present in a level prevent this.

For those reasons, terrain reasoning should be addressed with custom solutions, tuned for the specific AI needs and level characteristics of the game.

A popular and custom means to describe terrain are waypoints (and similar concepts such as cells or grids – see [Reece], [Snook]). Waypoints represent the subset of the terrain accessible to the player and AI. The connections between these waypoints denote viable movement. And the graph created by the waypoints and their inter-connections expresses the valid paths.

Often, these waypoints are annotated with the presence of nearby power ups. Sometimes, the waypoints are part of more abstract concepts like areas and portals.

Thus, waypoints are a handy and versatile terrain representation. Nevertheless, the AI typically ignores to reason about them (except for the occasional path finding).

The remainder of this paper discusses how to reason about terrain based on the waypoint graph. While the ideas result from developing tactical squad and individual AI for tactical action game, the concepts apply to a wide range of action games.

3.1 Waypoint Graphs

Waypoint graphs convey a lot more information than just the shortest path from A to B. For example, the waypoint graph in Figure 1 below suggests terrain that is composed of three rooms, connected by a few hallways.

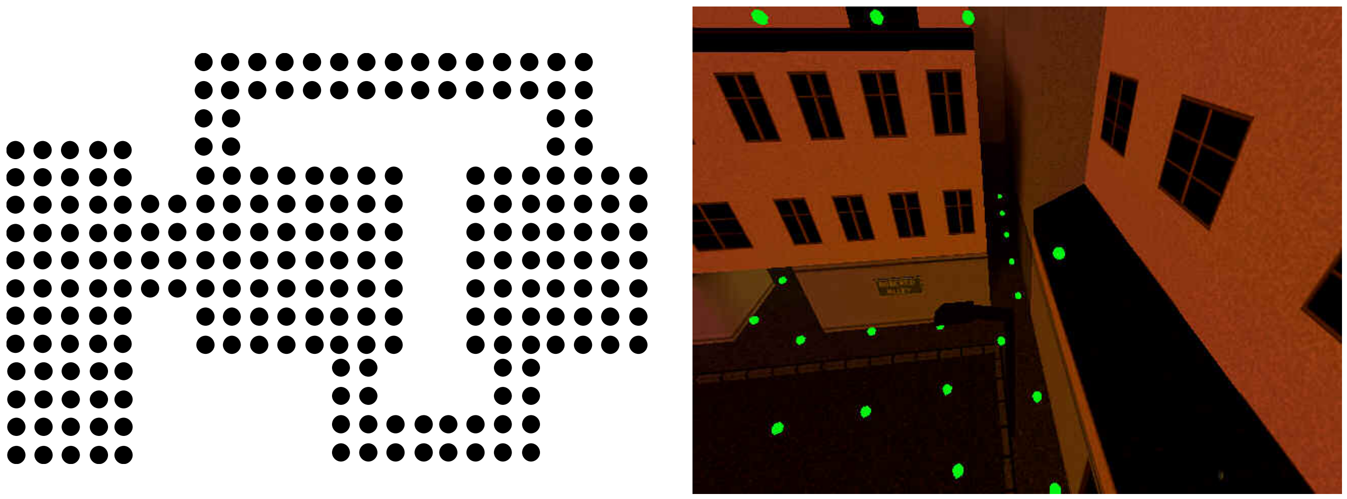

Figure 1. Waypoints expressing the shape of a level (left), and waypoints in-game (right)

All waypoints together not only communicate the shape of the accessible terrain. If enhanced with line-of-sight and line-of-fire information, waypoint graphs can be used to express many ingredients that make up tactics.

3.2 Picking Tactics Apart

Tactics in 3D action games frequently deal with the terrain. For example, in deathmatch games, the winner typically is the one able to seize and hold the location where the most powerful weapon, item or power-up respawns (such as the “quad damage”).

In ‘capture-the-flag’ (CTF) team games, the team able to identify and defend the bottleneck areas near the home base will stand a good chance.

In a tactical shooter, the squad that manages to reach the objective via an concealed approach will have good chances of surprising the enemy.

In all these examples, terrain plays a big role. Actually, the relation is often big enough to enable tactics ingredients to be expressed in terms of waypoints and their relations.

Look at the game terrain, pick two locations p_1 and p_2, and ask the question:

- why is p_1 a better (or worse) location than p_2 (for a specific tactic)?

3.3 Relating Tactics, Terrain and Waypoints

In deathmatch games, p_1 often is better if it is closer to a key power up than p_2, or if enables to the actor to see and lay fire on power up locations.

“Closer to” can be measured in travel time along the shortest path, and the presence of a line of sight between two waypoints often equals the availability of the line of fire between the corresponding locations.

The bottleneck area in CTF is typically on the access route to the base. Access routes can be identified from the number of paths to the base that pass through that area.

The bottleneck area also offers little cover from observation and incoming rockets. Expressed in waypoints, that means that waypoints in or near the bottleneck area all have a line-of-sight and line-of-fire from a number of waypoints in the base area.

A concealed approach consists of a path to the destination that cannot be observed for long periods from locations near the destination. In other words, solely a small number of path waypoints can be observed from waypoints near the destination, and it never involves a long string of consecutive waypoints.

Note that the quality of a tactical expression in terms of waypoints depends on the density of waypoints: the denser the waypoint graph covering the terrain, the more accurate the expression (see also section 6.1).

The expressions also make assumptions about the effects of weapon, and player movement abilities.

3.4 Waypoint Calculations

The direct relation between tactical concepts and the waypoint graph enables us to turn these concepts into computable properties. For meaningful results, those computations should resemble the human tactical interpretation.

A waypoint has four different kinds of characteristics that all should be taken into account:

1. *the local environment.*
   Waypoints describe, for example, how dark the location is, which types of movement are required (crouching, swimming, using a ladder), whether a door or button entity is present.
2. *the membership(s) of higher-level terrain concepts*
   Waypoints may be grouped to represent terrain concepts such as a room, a lake, or the blue team’s base.

3. *the relations with other waypoints*
   Waypoints have relations with other waypoints: is there a valid line-of-sight, how long does it take to get from one waypoint to the other.
4. *the focus (in the relations with other waypoints)*
   A waypoint whose relations, such as line-of-sight or easy access, are clustered primarily in a single narrow direction has focus in that direction.

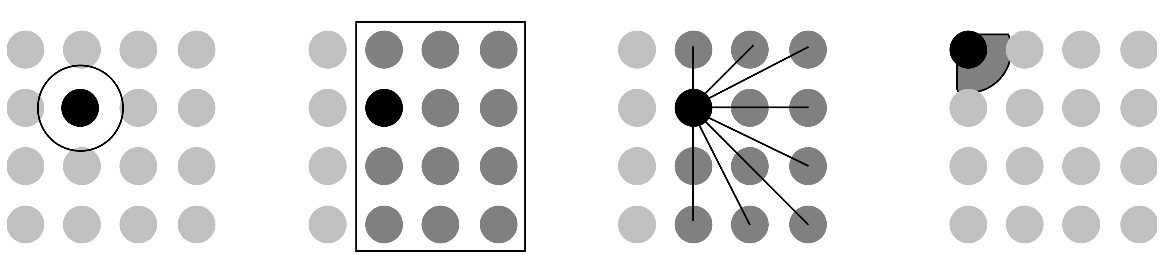
Figure 2. Waypoint characteristics: local environment, group membership, relations and focus

Of all the four characteristics, focus, is the most complex to grasp. The following case studies will illustrate why it is needed.

4 Analyzing terrain and adapting tactics

Terrain reasoning based on waypoint graphs, its value, and the concepts introduced here are best explained via an example.

This section discusses a concrete case, ‘picking good sniping spots’, in detail and illustrates:

- how to translate tactics such as “sniping” to waypoint graph properties;
- how to create an algorithm that turns these properties into a waypoint sniping suitability score;
- how to improve the pre-computed sniping scores with sniping performance feedback collected in-game (thus learning);
- how to adapt the algorithm to include game play experience in its tactical considerations (thus adapting to game tactics)

4.1 Problem: Picking Good Sniping Spots

Assume the following scenario (in a tactical 3D first person action game):

The player points to a building some one hundred meters in front of him, and summons the nearby AI actor: “get into position to snipe that building”.

The AI establishes the direction to the building.

The AI determines all waypoints that are within 2 seconds distance of him. He quickly ranks these spots for “sniping” suitability towards the target building. For the most promising spots, he checks if a clear line of fire is available.

Once a candidate spot has been found, the AI checks if it good enough for sniping purposes. The AI then reports back to the player whether he is able to execute the instruction. If so, he moves into position.

This example focuses on computing the waypoint's suitability for sniping.

4.2 The Terrain

A rather simple and small level serves to illustrate the terrain reasoning (whereas the reasoning can deal with any terrain size). The level consists of:

- two buildings on opposite ends of the terrain, each having two windows with a view to the bridge, and a rotating door;
- a creek separating the two sides;
- a bridge, providing a fast way to cross the creek, and offering some cover (half-height walls);
- two stairs, enabling players to leave the creek. Both stairs provide some cover (half-height walls);
- two full-height walls, providing additional cover.

In this example, assume that the AI sniper is in the northwest corner of the level, and is instructed to snipe towards the building in the northeast.

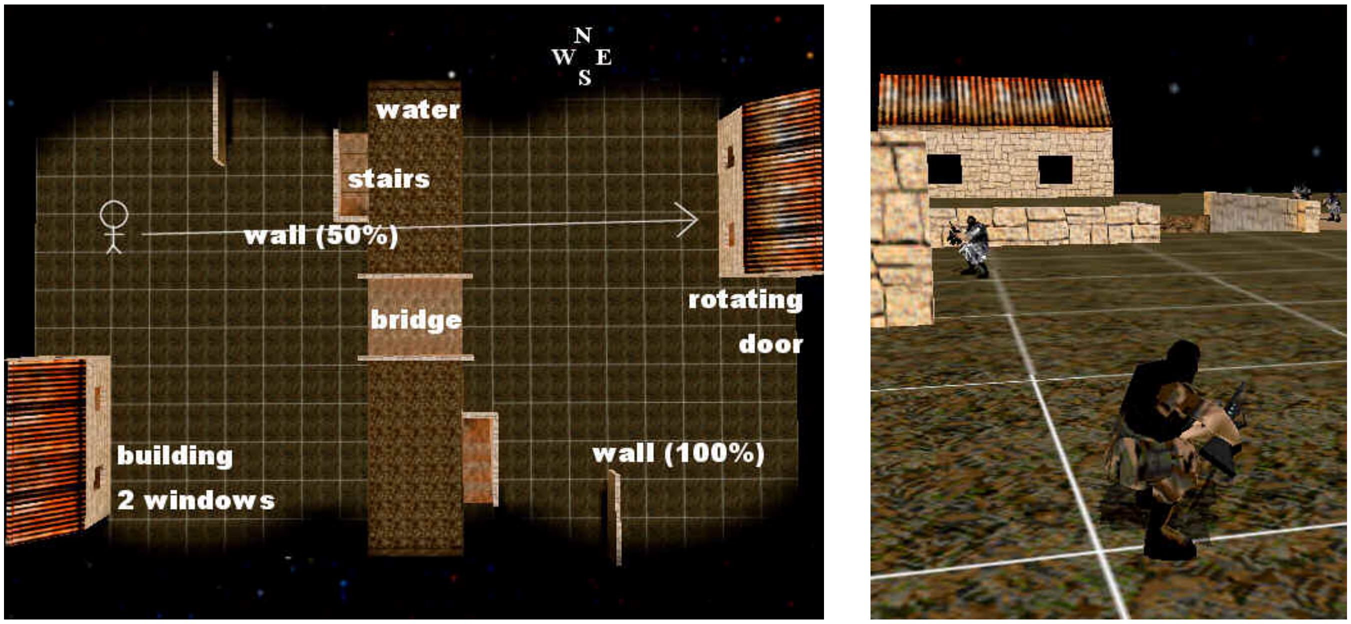

Figure 3. A simple, symmetric level, to demonstrate reasoning about sniping

The terrain is covered by some odd 480 waypoints[1].

[1] The number of waypoints is about 10% larger than required for such a level; additional waypoints have been introduced to work around peculiarities in Excel 97's surface charts.

4.3 Step 1: Sniping Tactics and Good Sniping Spots

The suitability of a position for sniping depends on many qualities. Many of these qualities are easily identified when trying to describe the differences between an obviously good and an obviously bad sniping position.

The following qualities make for a good sniper spot in a tactical shooter game:

a. the sniper is inconspicuous at the spot;
b. the sniper can overlook many distant locations from the spot;
c. the spot is hard to reach from the overlooked locations;
d. in the direct surroundings of the spot, the sniper can find cover from threats at overlooked locations;
e. the overlooked locations do not provide nearby cover from observation and attack by the sniper (at his spot);
f. the spot overlooks many key locations (such as doors, windows);
g. threats cannot easily approach the spot without being observed;
h. the spot has focus, that is, it offers a great view in one single direction, in combination with one or more protected flanks;
i. the spot allows the sniper to move freely, without requiring special movement skills, and without having to deal with obstacles such as doors

4.4 Step 2: Relating Sniping Spot Qualities and Terrain

These sniping spot qualities are easily related to the following waypoint graph properties. For example,

a. the sniper is inconspicuous at the spot;

can be expressed using a waypoint property:

- the (average) light level at the waypoint is below a minimum value.

And a directional property (since reflects a relation between waypoints):

d. in the direct surroundings of the spot, the sniper can find cover from threats at overlooked locations

can be translated to:

- for all waypoints overlooked from the sniper spot waypoint:
  given a presumed threat at the overlooked waypoint,
  an easily reached waypoint that offers cover from such a threat
  is available near the sniper waypoint

Also, a complex property as focus:

h. the spot has focus, that is, it offers a great view in one single direction, in combination with one or more protected flanks;

can be translated to:

- the ratio of waypoints visible in one direction vs. waypoints visible in other directions

The following table lists the terrain properties used to express the sniping spot qualities:

	terrain property description	light level	obstacles	special terrain	line distance	travel time	line of sight	line of fire	portal member	game activity		property	directional	higher level	focus
a	**inconspicuous**	✓										✓			
b	**overlook distant spots**				✓		✓	✓					✓		✓
c	**hard to reach**					✓							✓		✓
d	**cover from targets**					✓	✓	✓					✓		✓
e	**target without cover**						✓	✓					✓		✓
f	**overlook key spots**			✓			✓	✓	✓				✓	✓	✓
g	**observed approaches**					✓	✓						✓		
h	**focus**														✓
i	**local movement**		✓	✓								✓			
j	***overlook traffic***			✓					✓				✓		✓

Figure 4. Tactical properties for a sniping spot, and related terrain properties

4.5 Step 3: Sniping Suitability: The Formula

With the sniping spot qualities expressed in terms of waypoint graph properties, a sniping spot suitability score can be computed, for each waypoint and every direction, as follows:

- *Sniping spot quality* (*w* ,*d*) =
  Σ *properties* (*w*)
  + *focus* (*d*) × Σ *directional properties* (*w* , *w'*),
  for all waypoints *w'* in direction *d* from *w*

A directional property, such as, “b: overlooking many distant locations “ is computed as follows:

- *Overlooking distant locations* (*w* ,*d*) =

$$\frac{k \times \Sigma\,(\,\textit{line-of-sight-line-of-fire}\,(w\,,\,w')\times \textit{fardistance}\,(w\,,\,w')\,)}{\#\text{ waypoints } w'' \text{ with a } \textit{line-of-sight-line-of-fire}\,(w\,,\,w'') > 0}$$

  for all waypoints *w', w''* in direction *d* from *w*

with

- *line-of-sight-line-of-fire* ( ) and *fardistance* ( ) returning values in [0..1]

And focus is computed (assuming 45 degrees directions), for example, as:

- *focus* (*w* ,*d*) =

$$\frac{3 \times value(w, d) + 1 \times value(w, d + 45°) + 1 \times value(w, d - 45°)}{\Sigma\ \text{value}(w, d + \alpha) \text{ for all } \alpha \text{ in } \{-135°, -90°, +90°, +135°, +180°\}}$$

with

- *value* (*w*, *d*) = # waypoints *w"* with a *line-of-sight-line-of-fire* (*w* , *w"*) > 0 for all waypoints *w', w"* in direction *d* from *w*

Focus here is the ratio of forward interactions versus non-forward directions.

The role of focus is best illustrated using the example terrain (see Figure 5, left image, below): Positions that have a strong eastbound focus are:

- the positions at the western border, because they only have relations with waypoints to their west, and typically with a lot them;
- the rooms in the southwestern building, and the small alley south of that building have a very strong focus, because their flanks are protected;
- the positions in the northwest, west of the full height wall, the positions on the northern stairs from the creek, and the positions in western side of the creek also have a good eastbound focus, because their back (to the west) is covered.

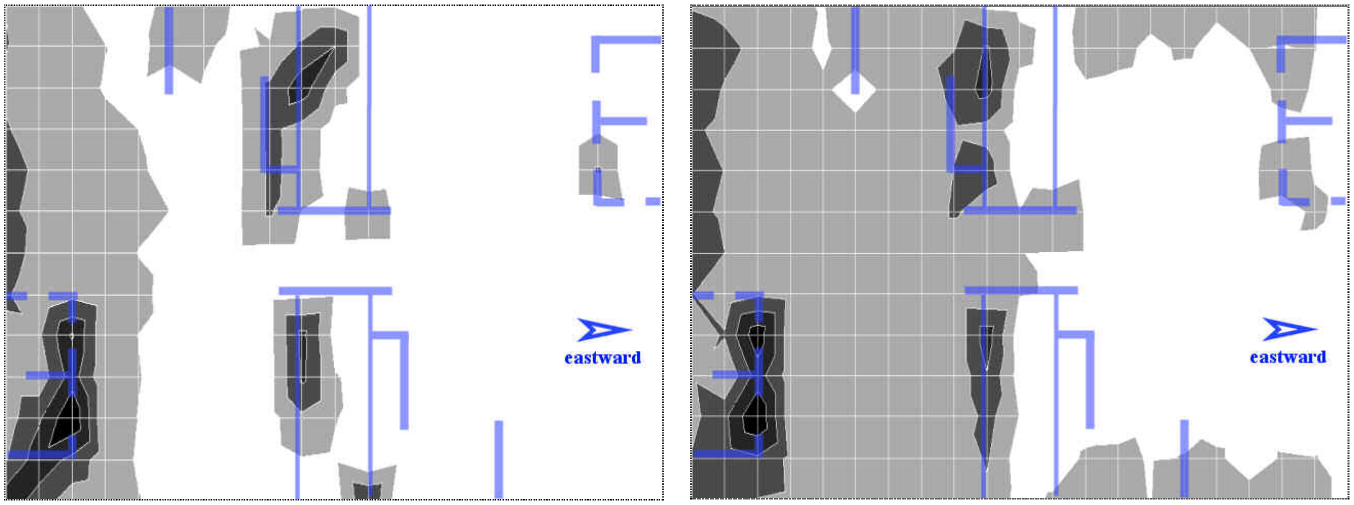

Figure 5. Left: East-oriented focus (darker color is higher focus) for all locations, illustrated as a top view, along with terrain feature overlay
Right: AI computed Sniping Spot preferences (darker is better) in the eastward direction, using static level data only

4.6 Step 4: Ranking Sniping Spots using Static Info

The sniping spot quality formula, computed for each waypoint, in the east direction, generates the result as illustrated in Figure 5, right image, above.

The results demonstrate that using the waypoint graph (and assumptions about game physics and weapon characteristics), the AI really can identify good sniping spots. It identifies as good spots the rooms and windows in southwest building, the western edge of the level, the northern stairs near the creek, and the spot between the northern stairs wall and the bridge wall.

Thus the AI, in the scenario sketched, is able to pick a good sniping spot from a list of automatically ranked sniping locations.

Having an algorithm that identifies great sniping spots using just the annotated waypoint graph has several advantages. The AI no longer is dependent on the level designer to annotate terrain. The level designer himself can generate and inspect the AI's understanding of the terrain, which assists him in properly employing AI in the level.

However, it is also a disadvantage to use only these few inputs:

- locations not properly represented by a nearby waypoint are ignored; and
- actual game play is ignored

The first problem is easily addressed by adding more waypoints at and near key locations: with more waypoints, the AI is less likely not to detect a position offering cover.

The second problem shows in the rating of sniper spots on the western creek bank: these positions receive a rather high rating because they overlook spots in the creek from which it is hard to avoid the sniper. However, in most game scenarios, most activity will not occur in that creek. Consequently, the sniper spots on the creek bank are overrated.

Luckily, the lack of game play input is not only easily addressed, but also creates a great opportunity.

4.7 Step 5: Using In-Game Sniping Feedback to Rate Sniping Spots

To improve the pre-computed sniper spot ratings, the AI needs to capture and analyze game play, and adjust the ratings accordingly. This sounds more complex than it really is, and even better, at can be done on the fly, at virtually no (CPU) costs.

Capturing and analyzing the required game play data for this case is done as follows:

- record at every way point:
  - the time spent by the AI or player while static at that position and with sniping weapon active;
  - the amount of damage caused by the sniping weapon from that position;
  - the amount of damage taken when sniping at that position

After every game round or mission, aggregate this “experience” as follows:

- *sniping experience at* (*w, d*) =

  player sniping time in direction (*w, d*) + *AI sniping time in direction* (*w, d*)

  + *player directional damage issued* (*w, d*) + *AI directional damage issued* (*w, d*)

  – *player directional damage taken* (*w, d*) -*AI directional damage taken* (*w, d*)

and adjust the pre-computed waypoint sniping rankings with the experience:

- *sniping quality (w, d) =*

  precomputed sniping quality (w, d) + *sniping experience at (w, d)*

The (good) reasons to deal separately with AI activity and player activity are discussed in section 6.4.

Collecting the game play data takes negligible time, especially since most games already determine the nearest waypoint for every actor at every game tick (for navigation purposes). Processing also takes little time, and typically can be done when presenting an intermission screen.

Because the AI can adapt its sniping spot rankings ‘on-the-fly’, it will display varying and adaptive sniping behavior.

For this example, some 40 game rounds were run, with one or two snipers defending the western building, and three or four attackers storming that building from the east side of the level. The player, randomly, acted as a defending sniper or one of the attackers.

The figures below illustrate the “sniping experience data” collected in-game, and the updated waypoint sniping ratings (in the east direction):

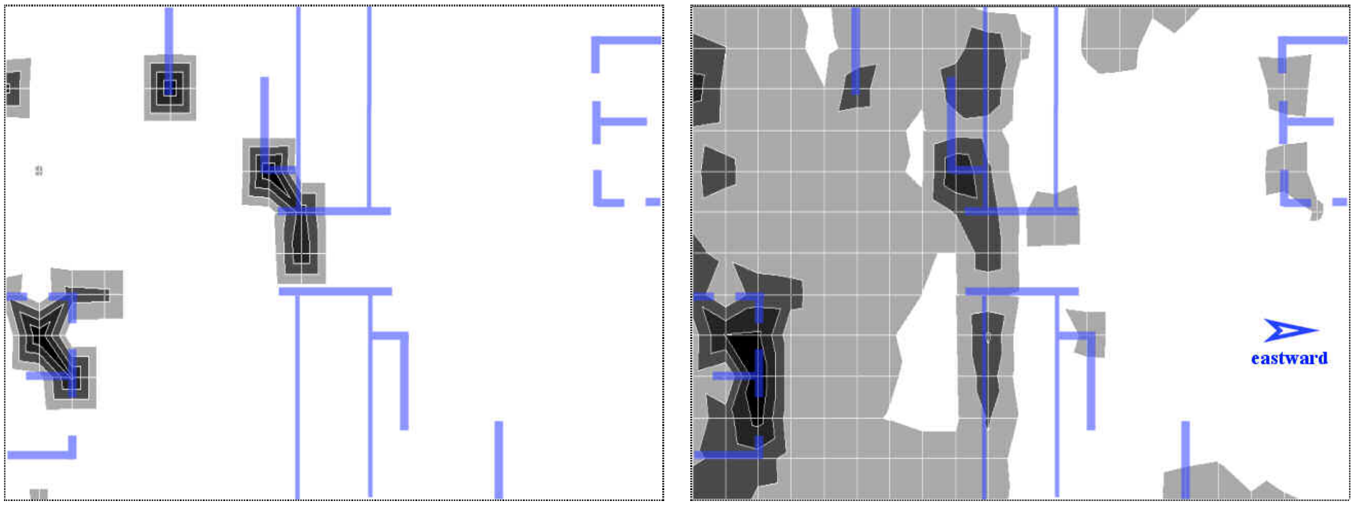

Figure 6. Left: Waypoint sniping experience data (non-directional, darker is better)
Right: Eastward sniping ratings adjusted with the experience (darker is better)

Note that the sniping experience data in Figure 6 (left) largely corresponds with (and thus confirms) the sniper ratings computed from the waypoint graph (Figure 5 (right)). However, the experience data also emphasizes several highly successful sniper spots over other potential sniper spots.

Adjusting the waypoint sniping ratings results in new ratings. These ratings show a decreased importance for the southwestern creek bank, and increased importance for a few northwestern spots partially covered by the full height wall there (compared with the results in Figure 5 (right)).

4.8 Step 6: Better Tactical Understanding From Game Play Experience

Further improvements of the sniping ratings are possible using the game play data. In the previous section, the sniping spot ratings have been adjusted by more or less copying “best sniping practices”, without giving the data gathered further thought.

Giving the data some thought, that is, using the game play data in our reasoning is exactly what needs to be done to improve the sniping ratings.

Game play does not occur evenly across the map. Some locations are more frequently visited than other locations. And sniping spots overlooking many of these “traffic” locations thus are more important.

Again, it is easy and cheap to record actor and player movement per waypoint. The formula for sniping rating needs to be extended with yet another factor:

j. the sniper can overlook much of the traffic (preferably at some distance from its spot, because then the sniper has an advantage);

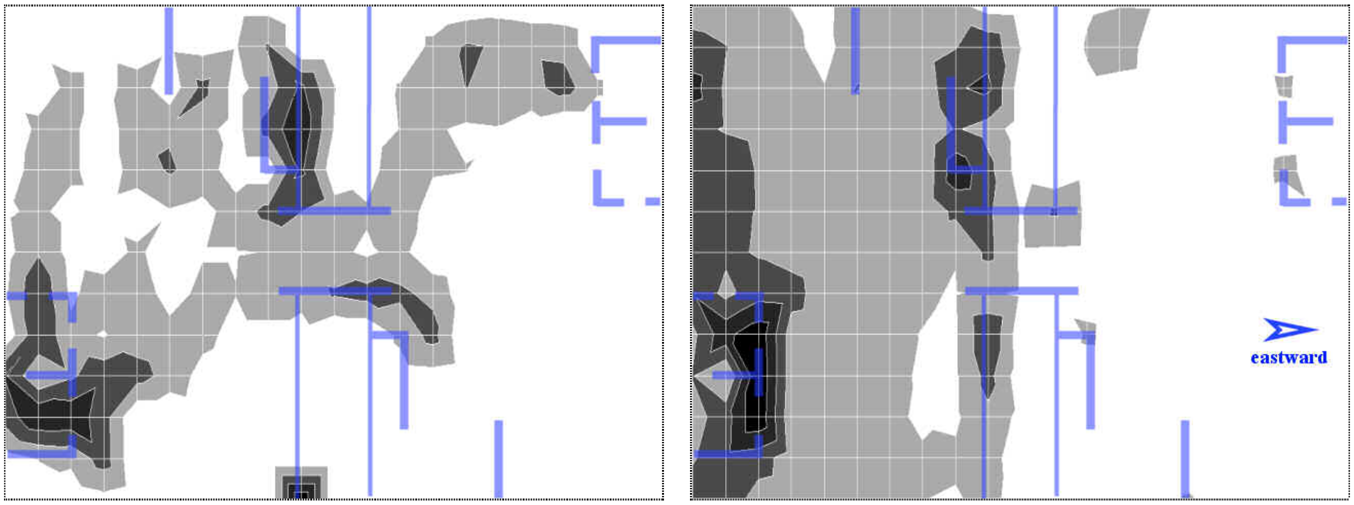

Figure 7. Left: Traffic data per location (non-directional, darker is more traffic)
Right: Sniping ratings now also taking into account traffic and sniping experience (darker is better)

Re-running the algorithm, now taking into account both sniping experience, and traffic data results in the following output as is shown in Figure 7 (right).

The new results again show reduced sniping ratings of the spots on the southwestern bank near the creek. They also show increased ratings for the whole western edge of the map (because these positions overlook many traffic spots).

4.9 Processing Steps

The AI need for the waypoint sniping ratings, as discussed in the scenario above, has been addressed with:

- a tactics based algorithm to rate sniping spots using the waypoint graph;
- an on-the-fly procedure to adapt the sniping ratings to game-play feedback;
- (optionally), a re-run of the algorithm to put more tactical understanding of the game play in the ratings

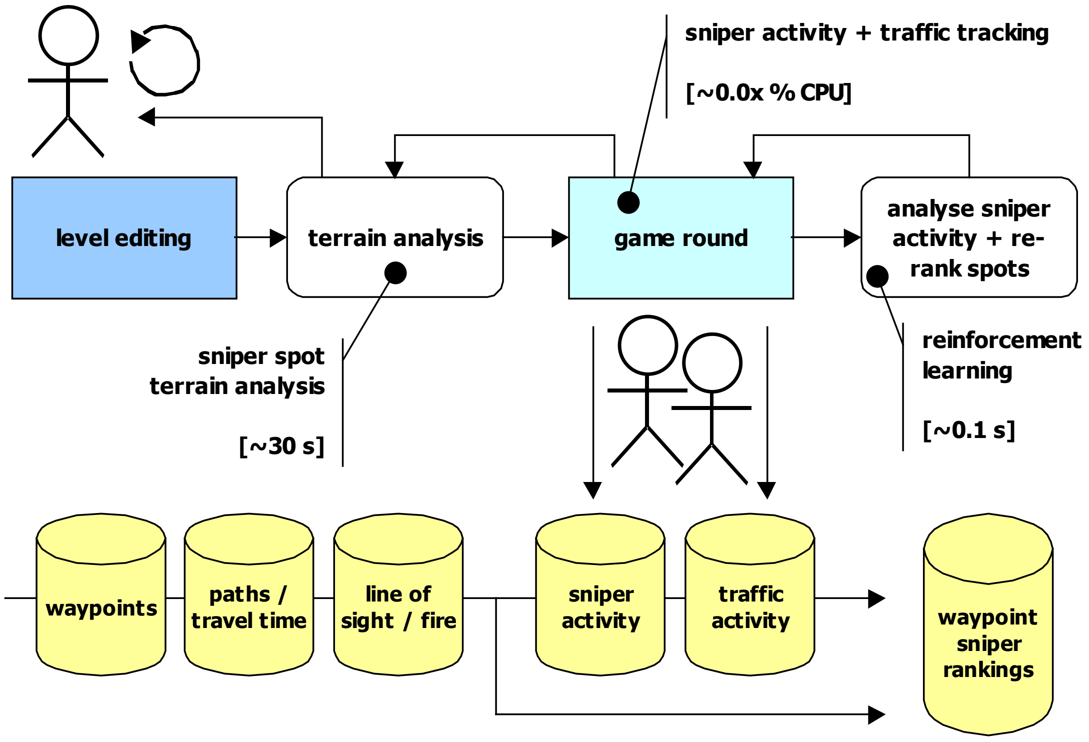

Figure 8. The terrain analysis cycle, with pre-game and in-game processing

The in-game memory consumption can be as small as two tables containing, for each waypoint and each relevant direction, the sniping rating and player and AI activities respectively.

The in-game CPU consumption consists of tracking player and AI activity per waypoint, and a brief computation to adjust sniping ratings (typically done when presenting an intermission screen).

To preprocess a level, the memory and CPU requirements are more demanding. The tables for shortest paths, line-of-sight, line-of-fire easily consume a few MB, and the algorithm takes a few tens of seconds to execute.

There are several alternatives to create good waypoint sniper rankings, as is shown in Figure 8 above.

The approach presented enables the game developer to ship his game with an “experienced” AI that has gained some tactical understanding of the game’s levels.

Optionally, the gamer can be allowed to run the algorithm using on his game play data, so the AI adapt to his style of playing.

4.10 Similar Terrain Reasoning Applications

A problem similar to picking sniper positions is finding appropriate ambush positions. The ambush position typically is hidden from observation in all but a few directions. The ambush position typically overlooks a number of nearby locations, thus providing targets little time to respond.

The direction of traffic near the ambush position typically is perpendicular to or away from the attack direction.

Again, these properties can be mapped onto waypoint related properties. For ambush positions, it is also simple to capture ‘ambush’ related game play data, and use that to adjust ambush ratings on-the-fly, or with a re-run of the algorithm.

Another problem that is very similar is identifying good positions to position sentry guns.

A more generic waypoint ‘combat suitability’ rating is also valuable. Such a rating will help the AI distinguish between threats that occupy strong or weak positions.

If the game AI implements flocking behavior, that waypoint ‘combat suitability’ rating can be used to stimulate tactical movement.

In a flocking algorithm, squad members determine their position based on attracting forces (for example, stay near the team members) and repulsing forces (stay away from walls, do not get too close to team members). Often, tactical considerations are ignored.

By using the generic ‘combat’ rating as additional force (attracting or repulsing the squad member), concepts such as nearby cover, good overview, and flank protection can be handled as part of the flocking. As a result, the squad will be more likely to move along walls than to move in the open.

In combination with areas and portals, the combat rating of areas and portals can be determined. To do so, for each area, the combat ratings for waypoints are determined using solely the interactions with other waypoints outside the area. This enables the AI to detect strongholds.

5 Other Terrain Reasoning Applications

Terrain reasoning has many other applications, for example dealing with dynamic situations. The following threat prediction case illustrates the use of waypoint graph based reasoning for dynamic situations.

5.1 Predicting Out-of-sight Threats

Image the following scenario (with an AI actor, on defense, having spotted a threat):

> *Upon seeing the AI actor, the threat turns and moves to the right, away from the window and out of sight. The AI actor stays alert...*

To be a useful team member or challenging opponent, the AI should understand where threats are likely to appear and where threats are likely to move to. Though it is often infeasible to fully model the presumed motives and behavior of threats, it is possible to do a good job using the terrain representation.

The following two-step, waypoint graph based approach works quite well. First predict the threat's position using:

- the threat's last observed position and movement;
- the viable paths in the threat's surroundings;
- the (absence of) lines-of-sight to the threat's presumed position.

Typically, the threat's position is extrapolated, under the assumption that the threat continues in the same direction and with the same speed. The resulting position is tested against both the paths feasible, and the line-of-sight. For example, the threat will not move through walls, and in the waypoint graph, the absence of path indicates that. Similarly, if the AI does not see the threat at a predicted location that is in full view, the extrapolation is known to be incorrect. In these cases, the presumed threat position can be corrected to reflect a (more) likely location, for example the last valid extrapolated location.

Second, the AI needs to decide where to aim for. If the AI cannot fire through obstacles, it solely has to consider the visible locations in the immediate surroundings of the presumed threat position. One simple way to pick a likely location for the threat to reappear (and thus a location to aim for), is to construct the shortest path from the presumed threat position to the AI actor. The first location on that path that is within view of the AI, is a good location to aim for. The shorter the time the threat needs to travel to that location, the more likely

If the threat's likely reappear location is quite close to the threat's presumed location, the AI might consider to employ suppression fire or launch a rocket towards that location.

5.2 Terrain Reasoning for Dynamic Situations

A large number of dynamic situations can be interpreted efficiently using terrain reasoning, including locating nearby cover, planning concealed paths, moving in formation, assessment of enemy positions, etc.

Even non-trivial reasoning, such as determining the value of spending a hand grenade to attack or flush out a threat, can be done efficiently provided the data structures are designed to support such a query [Sterren].

6 Implementing Terrain Reasoning

Terrain reasoning means reasoning about tens of thousands of polygons, and millions of inter-waypoint relations. Implementing terrain reasoning thus is a matter of carefully analyzing the AI requirements, the resource constraints, making trade-offs, and executing some experiments.

This section briefly discusses the most important trade-offs to be made:

- waypoint resolution, accuracy and performance
- CPU load, lookup tables, and memory consumption
- Dynamics and pre-processing

6.1 Waypoint Resolution, Accuracy and Performance

Waypoints (and similar AI navigation aids) represent the accessible game terrain. Many action games have their AI use as few waypoints as possible, because:

- fewer waypoints mean less work to the level designer (to place them);
- fewer waypoints mean faster pathfinding and smaller shortest path lookup tables

However, fewer waypoints mean also a coarser representation of the game terrain: each waypoint (on average) describes a larger part of the terrain, and the line-of-sight and travel time relations becomes less accurate.

The waypoint density required to sufficiently describe the terrain depends on the game design and AI needs. For games featuring lethal weapons (“single shot kills”), slow movement, many obstacles, and lack of respawns, the terrain representation should describe as many covered and concealed spots. This typically translates to a high waypoint density.

In games featuring respawns, fast movement, and weaker weapons, a low waypoint density suffices, since terrain details are less important (but the tactics still may be terrain driven, because of power-up and objectives positions).

Note that algorithm such as the one used for identifying good sniping spots assumes a more or less uniform distribution of waypoints across the terrain.

6.2 CPU Load, Lookup Tables, and Memory Consumption

Some terrain-related properties such as paths, and a bouncing grenade trajectory take considerable time to compute. Other properties such as lines-of-sight are inspected so frequently that in total they consume a good amount of CPU.

Now consider the following simple AI query to locate nearby cover from a number of threats:

- given three threats that have line of fire to the AI actor, is there a spot within 3 seconds distance that provides cover?

To determine the result, the AI needs to construct numerous paths and perform a good number of ray traces (taking into account different actor postures, such as sitting and prone).

Today's CPUs and game engines handle in the order of 3,000 A* path searches per second, and 50,000 trace lines per second, depending on level size and geometry complexity. Given an AI budget of 10%, and 10 AI ticks per second, the AI is allowed some 15 path searches and 250 trace lines per tick, for all its actors (typically 4 to 30 of them), and as its sole activity.

Storing part of the computations and looking up the (intermediate) results typically is three orders of magnitude faster for path finding, and two orders of magnitude faster for trace lines. Using lookup tables thus frees up CPU time for more advanced AI or faster graphics.

However, lookup tables may consume a good amount of memory. Especially for waypoint-pair relations, such as paths and line-of-sight, the straightforward matrix implementation has $O(N \times N)$ space complexity. For $N \geq 1000$, these lookup tables really start consuming megabytes.

For path lookup, a hierarchical or multi-level path structure in combination with a cache for the most recently referred to paths (and travel times) offers a smaller footprint while still being very efficient.

The average waypoint sees only a small subset of all other waypoints. In that case, the line-of-sight matrix can be flattened: per waypoint, record solely the waypoints actually seen by that waypoint, and the waypoints able to see that waypoint.

Another concept that helps selecting the right content for look up tables is the so-called "time horizon". Most of the AI decisions, especially under the resource demanding "combat" conditions, have a limited scope. The game world is so dynamic that planning beyond a horizon of 1 to 3 seconds (depending on the game) has little value.

The AI queries therefor limit themselves to the surroundings accessible within the time horizon (and upon arriving there, they will decide again). A per waypoint look up table that contains the info for all waypoints within that time horizon will be very efficient without being large.

6.3 Dynamics and Pre-Processing

Pre-processing the terrain can significantly off-load the in-game AI (as is the case with pre-computing sniping spots). However, the game terrain is not completely static. Dynamic environment such as doors, vehicles, elevators, destructible walls, will change the paths, lines-of-sight, etc.

Dynamic environment does not necessarily invalidate pre-processed terrain information. Whether a distant door is open or closed does not really affect the value of a sniping spot. However, a nearby door being closed typically does invalidate a sniping spot (and is checked for in the sniping scenario).

Apart from checking for changes in the environment, the AI can also try to patch the pre-computed terrain description. The lines-of-sight affected by a door can be pre-computed, and the changes can be applied on the fly when the door changes state. Another option is to have a background threat slowly update the information.

6.4 Implementing Reinforcement Learning

Waypoint based reinforcement learning is easy to implement. Just capture the AI or player activity local to the waypoint, determine whether it is positive or negative feedback for the intended task, and adjust the waypoint value accordingly.

To create an AI that adapts quickly and tactically to the player's tricks, pay some attention to the following issues:

- *attach more significance to feedback from player activity than from AI activity*.
  Player activity is simply more important because the player will be more creative, and have a better tactical understanding of the game and terrain[2]. The player activity should be valued more than the combined AI activities.
  This policy also prevents the AI from honing into its own flaws (for example, the AI repeatedly and unsuccessfully assaulting an area because the area is known for the many frags scored there...)
- *periodically, level all values to be more responsive to changes in tactics*
  After re-computing sniping spots using game play data (see 4.8), the player might see changes in the AI sniping and adapt his tactics. To be more responsive to changes in the player's tactics, the aggregated game play activity values should be leveled. Using a function such as log or sqrt, this can be done while preserving the ordering.

6.5 Visualization and Feedback

The waypoint based terrain reasoning uses an approximation of the terrain, and heuristics to interpret that approximation. Consequently, implementing this kind of terrain reasoning requires tuning.

To efficiently develop and tune a terrain reasoning algorithm, the algorithm is best created incrementally, and using feedback from visualizing its results. Luckily, it is easy to visualize and interpret terrain and terrain properties.

Preferably, the results are visualized as an overlay in the rendered 3D world. For tuning, an export facility to comma separated files and a spread sheet package are recommended.

7 Summary

Terrain reasoning is the AI's ability to include terrain and related concepts such cover, travel time, areas in its plans, decisions, actions and communication. Notably in action games where AI and terrain play a big role, effective terrain reasoning results in a richer game AI.

Terrain reasoning involves a terrain representation, and algorithms to create, manipulate and query that representation. Waypoint graphs, until now primarily used for navigation, are demonstrated to be a powerful means for terrain reasoning.

[2] If not, the player is unlikely to notice that the AI copies his bad influences.

Many action game tactics use the terrain. These tactics then can be related to properties of the waypoint graph. With formulas to express these waypoint graph properties, the tactical concepts such as sniping spots, ambush locations, and strongholds can be computed and identified by the AI.

The reasoning about tactical concepts is easily extended to include player and AI game behavior local to the waypoint. This game activity data enables the AI to adapt its tactics to actual game play. Adaptation occurs because of re-enforcement learning, and because of improved understanding of the role that various locations play in the game.

The waypoint based terrain reasoning can be used in-game, for example to interpret threat positions and select the appropriate tactics. This kind of terrain reasoning can also be used to pre-process terrain in order to off-load and assist the run-time AI.

8 Acknowledgements

Jan Paul vanWaveren (id software), Doug Reece (SAIC), and the numerous talented bot developers in the Quake, Half-Life, and Unreal bot communities for valuable discussions.

The Action Quake II community for their help in turning that “mod” into a custom designed tactics testbed for AI experiments.

9 References and Suggested Reading

[Campbell] Campbell, C.E., McCulley, G., “Terrain Reasoning Challenges in the CCTT Dynamic Environment” in the 5th Annual Conference on AI, Simulation, and Planning in High Autonomy Systems, IEEE, 1994

[Laird] Laird, J.E., “It knows what you're going to do: Adding anticipation to a Quakebot” in AAAI 2001 Spring Symposium Series: Artificial Intelligence and Interactive Entertainment, March 2000, AAAI Technical Report SS-00-02

[Pottinger] Pottinger, D.C., “Terrain Analysis in Realtime Strategy Games” in Proceedings of Computer Game Developer Conference, 2000

[Reece] Reece, D., Krauss, M., Dumanoir, P., “Tactical Movement Planning for Individual Combatants”, in Proceedings of the 9th Conference on Computer Generated Forces and Behavioral Representation, 2000

[Reich] Reich, A.J., “An Efficient Representation of Spatial Data For Terrain Reasoning By Computer Generated Forces”, in Proceedings of the ELECSIM95, SCS, 1995

[Snook] Snook, Greg, “Simplified 3D Movement and Pathfinding Using Navigation Meshes” in Game Programming Gems I, Mark DeLoura (ed.), Charles River Media, 2000

[Sterren] van der Sterren, W, AI for Tactical Grenade Handling, http://www.cgf-ai.com/docs/grenadehandling.pdf, 2000