Open Challenges in First-Person Shooter (FPS) AI Technology

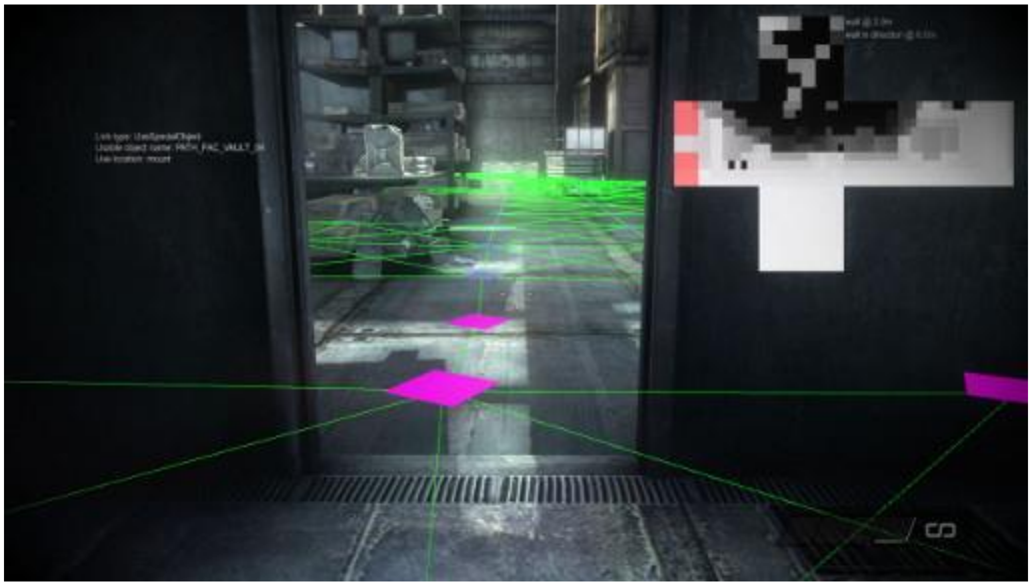
Alex J. Champandard on April 7, 2011

First-person shooters have a reputation for driving graphics technology, and it's no different for AI. Many notable innovations of the past few years have come from FPS games — including the AI Director from LEFT 4 DEAD, the STRIPS-planner from F.E.A.R. the HTN planner in KILLZONE 2, etc. With the latest wave of shooters being released in Q1 & Q2 2011 (e.g. CRYSIS 2, BULLETSTORM, BRINK) there's no better time to stop and take a look at which challenges are still ahead of us.

Over the past month, while preparing the program for our Paris Shooter Symposium on June 22nd, I've been carefully taking notes about open research areas of research — based feedback from the developers at Crytek, People Can Fly, IO Interactive, Ubisoft. The rest of this post includes topics that are still (at least) a few years away from being solved, sorted from low-level to high-level AI.

IMPORTANT: If you'd like to attend the Paris Shooter Symposium 2011 free, we're **looking for (two) volunteers** available on the afternoon June 21st! If you can make it to Paris then and spend the afternoon in preparations, you'll get a free pass in exchange — contact `<alexjc` at `AiGameDev>` by email. If you don't have time to help, there are some tickets left and they are on 15% discount until April 15th!

Challenge A.
Sensory Performance



Screenshot 1: Cover maps in KILLZONE 2, used for efficient line-of-sight.

The Problem

Line of sight (LoS) queries are one of the most common ways for AI to acquire dynamic information about the environment nearby. Unfortunately, LoS calculations are among the most expensive currently too, so every single ray or volume that you trace through the world's collision geometry is going to cost you dearly. Generally, the longer the line, the worse the performance. Obviously, if the AI can't get all the information it needs, then the quality of the behavior suffers.

Suggestions

An approach is to try to make these types of sensory queries as efficient as possible by optimizing common algorithms for the hardware they will run on, specifically PS3 and XBox360 consoles (for instance, taking into account the in-order execution of instructions and the properties of the cache). Adding a job scheduler into the mix to make things run in parallel, means there are lots of opportunities for intelligent batching of these queries to make sure they perform well when calculated in bulk. Alternatively, developers also try to design custom data structures to provide fast approximations that are good enough for FPS games.

Examples

- KILLZONE 2 uses cubic cover maps that are pre-generated and attached to waypoints. (See this INSIDER article.)
- DARK SECTOR uses cylindrical lookup table at two heights to determine visibility. (See this PREMIUM interview.)
- CRYSIS 2 apparently uses a PVS system based on custom data-structures. (Find out more at the Shooter Symposium!)

What's the next step? How can we do better so this works well in dynamic environments too?

Challenge B.
Motion Planning

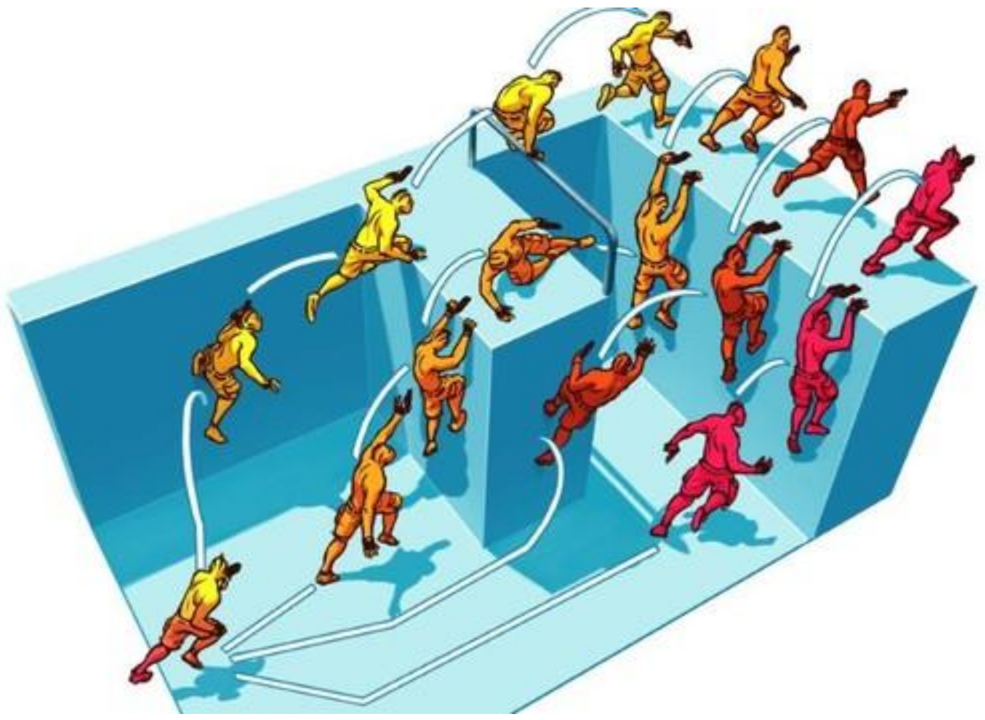


Diagram 2: Parkour movement in BRINK.

The Problem

Most games today have the AI dictating to the animation what should be done, for example when following a path. While this works most of the time, it can result in low quality movement — for example when taking the shortest path means the animation isn't as smooth as going the longer way around. Having the AI in complete control also causes bugs when there's a mismatch between what the AI wants and what the animation can provide.

Suggestions

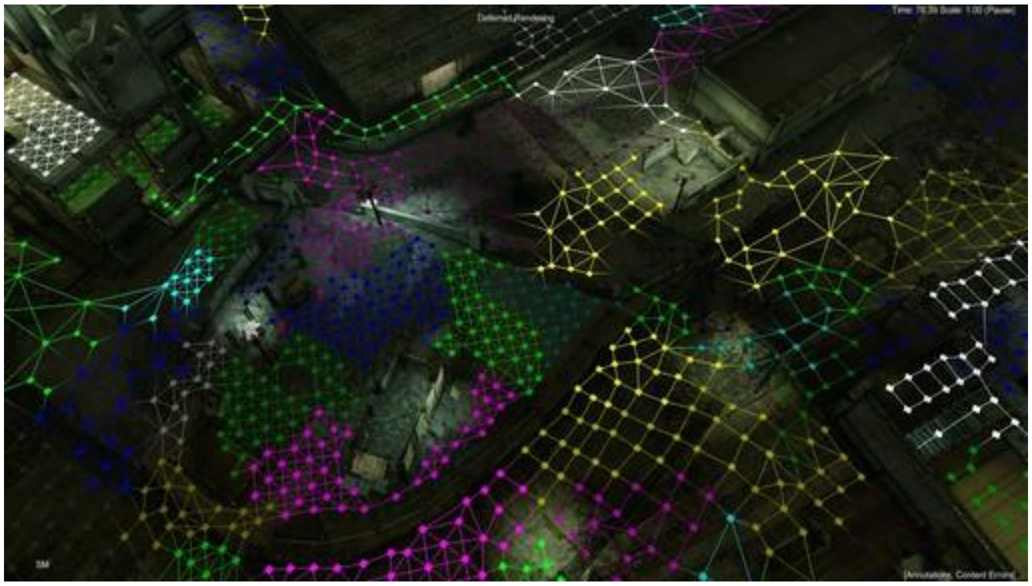
One solution to this problem is to use motion planning. Think of it like pathfinding, but using information about the possible movement, i.e. a motion model. In theory, if you plan your paths using data and/or statistics from the animation, then when you actually start following those paths the results will be much higher quality and fewer bugs will occur. The big challenge is that it requires a lot of computation to search for motions that match, and solutions that reduce that computation can be complex and memory hungry.

Examples

- BIOSHOCK uses an A* search to figure out what possible motions fit on a path.
- FULL SPECTRUM WARRIOR also uses a form of A* to pick which movement animations to use.
- UNCHARTED 2: AMONG THIEVES uses A* to search through the animation states of the player. (See this PREMIUM interview.)

As more games try to ground their characters into the world and reduce foot skating, advances in this technology will become more important.

Challenge C.
Fast Tactical Pathfinding



Screenshot 3: Waypoint network in KILLZONE 2.

The Problem

In combat situations, it's important to be able to move around from cover to cover quickly and effectively as a player would. However, in most cases the shortest path isn't the one you want. I'm sure you remember seeing an enemy try to flee your fire and end up running past you! Using better tactical pathfinding the AI could have figured out that running past you was a bad decision, and use a different route instead.

Most A* implementations use shortest paths with distance-based Euclidean heuristics, that means you get the mathematically shortest route when planning to reach a point. Game developers occasionally used variable A* costs, but this blows up the cost of the search to pretty much the worst case. There are a couple hacks around this, like multiplying the heuristic estimate by a constant factor, but this approach ends up cancelling out your variable costs.

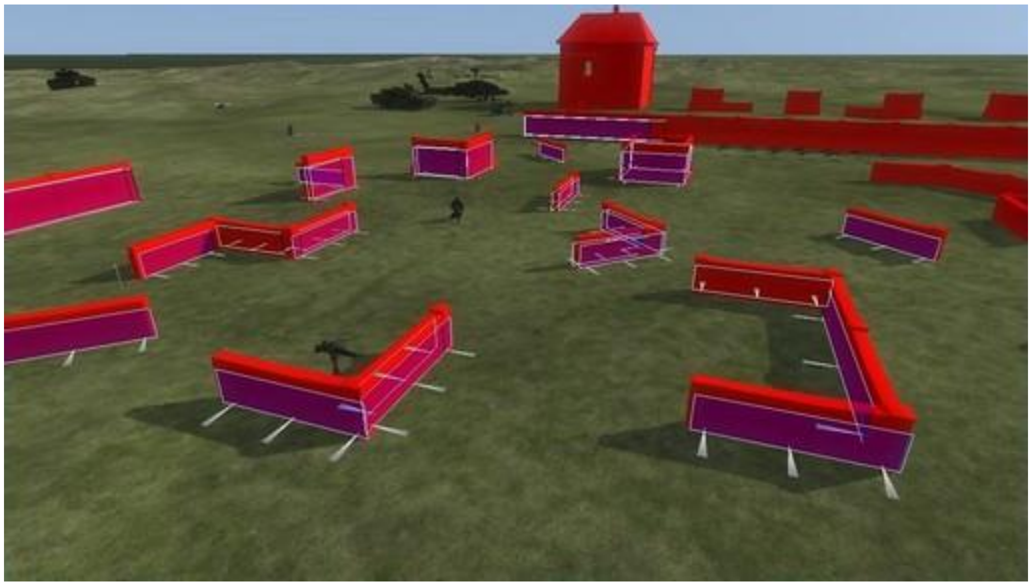
Suggestions

Pathfinding performance problems are often resolved using hierarchies or better heuristics. In this case, both are a challenge due to the dynamic nature of the variable map costs, e.g. for line-of-sight or threat levels. Any algorithm would have to be fast enough to deal with edge costs that change multiple times per second. Area clustering algorithms unfortunately aren't quite fast enough yet (though they'd offer better quality), and it's unclear how what the tradeoff of cost vs. benefits would be for grid-based hierarchies.

Examples

- Read Dynamic Procedural Combat Tactics (PDF) about KILLZONE's AI.
- KILLZONE 2 also uses influence maps to bias the squad pathfinding. (See this PREMIUM presentation.)

Challenge D.
Dynamic Terrain Analysis



Screenshot 4: Dynamic obstacle detection in BATTLEFIELD: BAD COMPANY 2.

The Problem

Many games rely on manual annotations or pre-processessing to create efficient and accurate representations of the terrain. This is often done during development, for example on a nightly basis or everytime a level is changed. The problem, however is that the information becomes invalid once the world changes. Some games store multiple versions of their annotations based on pre- and post-destructed worlds, or rely on some amount of dynamic probing in simple places. Obviously, this doesn't work very well in dynamic worlds.

Suggestions

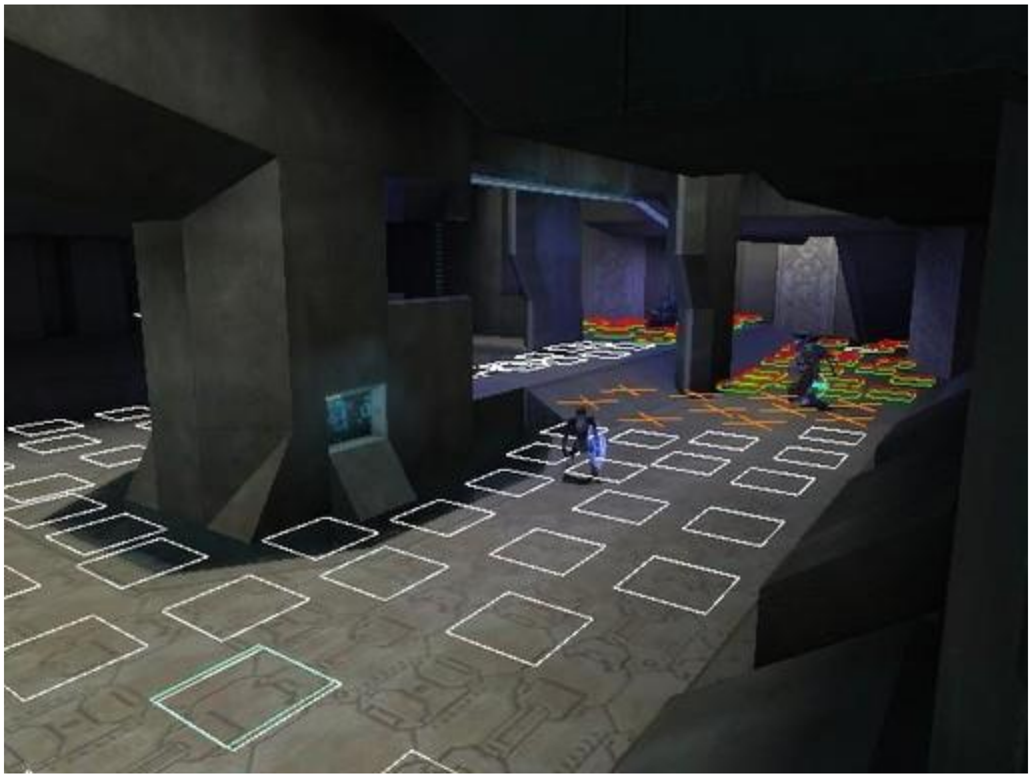
An approach would be to do dynamic terrain analysis at runtime, in order to generate these annotations as the terrain changes. There are many challenges to the implementation though. Firstly, this needs to be robust enough to not require manual annotations, and secondly this needs to be fast enough to work at runtime within the game at interactive rates. In particular, the type of voxel-based solutions that are proving useful for navigation mesh generation may be equally suited to these kinds of analyses.

Examples

- BATTLEFIELD: BAD COMPANY 2 uses dynamic probing to detect cover locations dynamically. (See this PREMIUM presentation.)
- KILLZONE 3 uses voxel-based solutions to find player cover locations. (Find out more at the upcoming Game/AI Conference!)

It's still not clear whether the best way forward is using intersection tests and voxel approximations, or even a combination of the two.

Challenge E.
Efficient Combat Reasoning



Screenshot 5: Combat Reasoning in the HALO series.

The Problem

To make good decisions in combat, you need good quality information taken from annotations, terrain analysis, pathfinding, line of sight, and many other systems. This can quickly get inefficient though, as you need to evaluate many different options for to be able to find combat positions reliably. Apart from building more efficient low-level systems to provide the information faster, we need to be able to make better decisions with less information, and be aware of the cost of each piece of information.

Suggestions

The solution is to improve our AI systems that reason with large quantities of information, each of which has a cost. The cost of information is often proportional to how many requests are made and how similar they are. Two A* pathfinds to similar positions in space are only marginally more expensive, and batching up line-of sight queries also has known performance gains. Our AI reasoning should be able to deal with this, carefully making requests for information which will prune its options down quickly, but also batching up queries efficiently.

Examples

- Also read Dynamic Procedural Combat Tactics (PDF) about KILLZONE's AI.

In a way, this research is about optimizing utility systems when they require lots of information to support their decisions.

Challenge F.
Player Prediction

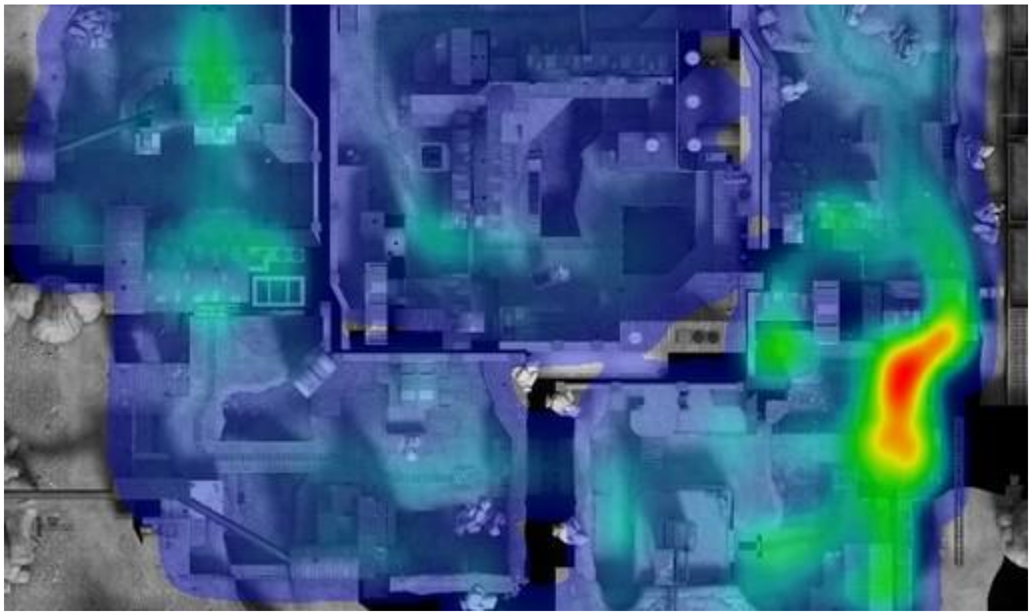


Figure 6: Heat maps in TEAM FORTRESS 2.

The Problem

We've pretty much got the hang of building enemy soldier AI, but when it comes to friendlies that stay on screen for longer than 10s it's quite a challenge. Obviously, building friendly soldiers in a question of emotions and interactive behaviors too, but we still haven't nailed the basic functionality yet. Your buddies often get in your way, get stuck, go the wrong way, do the wrong thing, etc. That layer of basic functionality needs a lot of work still!

Player prediction is a big part of this. We can solve mechanical obstacle avoidance problems, pick sensible positions to stand, or create the behavior for friendly soldiers when the player is not around. The hard part is making the AI perform according to the expectations of the player. Doing this right most likely involves being able to predict what the player is doing and what he'll try to do in the near future.

Suggestions

There aren't many applications of player prediction in games currently, and there's only a handful of research projects on the topic. It's not clear to what extent it's possible to predict player behavior, and even less what the ideal solution is. The two main avenues for research are logic-based, using for example plan recognition, and statistical using various models of player actions. It may also be possible to combine the two!

Examples

- Read "Being a Better Buddy: Interpreting the Player's Behavior" in AI WISDOM 3.
- See Peter Gorniak's research on plan recognition in game-like environments.

Challenge G.
Scripting & Conjunctive Goals

Screenshot 7: NPCs in combat in Red Dead Redemption.

The Problem

Game AI programmers have become pretty good at building AI as adaptive goal-based systems that can deal with unforseen changes in the world and still achieve things. This is a nice interface to tell an NPC what to do, and break down complex behavior recursively. However, this goes terribly wrong when the level designers take over and impose very specific low-level orders that ignore everything else!

Conjunctive goals, or the ability to reason with multiple objectives at the same time, is one possible solution to this. The idea is that the designers would specify what they want to happen as one goal (such as story-driven objectives), and the AI programmer would be able to set other goals (such combat or survival objectives) and let the AI deal with combining everything together! This is difficult to achieve with all of the techniques used traditionally by game AI, since it either requires a lot of manual work or extra computation power.

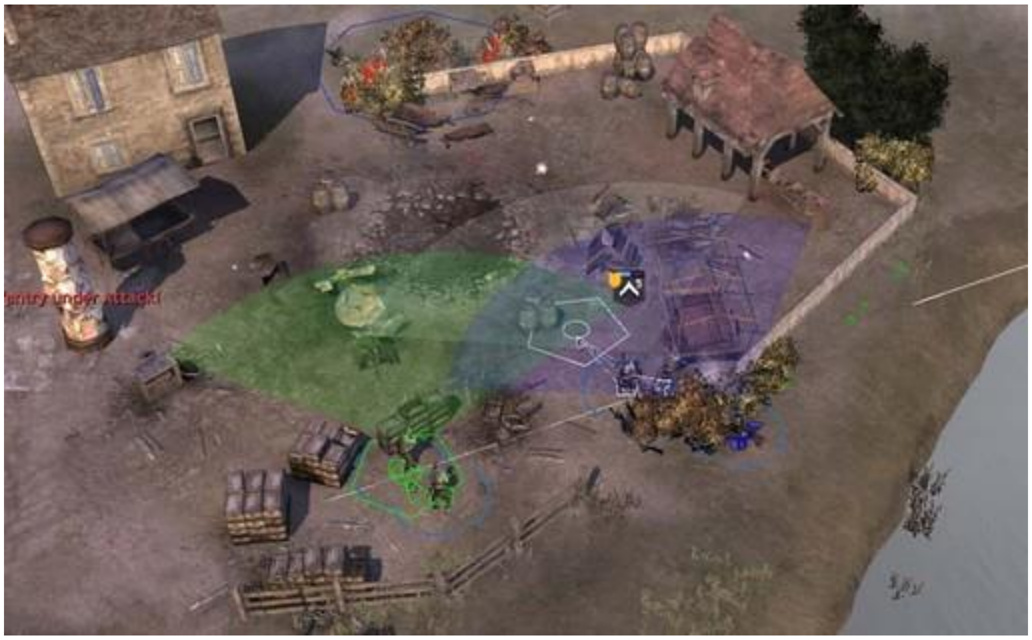
Suggestions

These days, orders are integrated into behavior trees using design patterns to place the orders sensibly within the overall combat tree and reusing behaviors that don't completely supress autonomous behaviors. However, single behavior trees are not well equipped to fulfilling multiple independent goals and using parallel trees is still not very common. On the other hand, while STRIPS-style planners may be well suited to dealing with more complex goal states, they haven't yet been applied to combining scripting with autonomous behavior.

Examples

- Façade uses a reactive planner knows as ABL, which allows multiple trees to cordinate together. (See this BLOG article.)

Challenge H.
Squad Coordination



Screenshot 8: Squad coordination in COMPANY OF HEROES.

The Problem

Often when you see NPCs as part of a squad, they're most given a shared goal and common information so they can achieve objectives together and avoid stepping on each other's toes. When there are NPCs with different abilities, all the behavior you see out of the squad is emergent based on the individual AI. There's very rarely any top-down coordination going on, which limits the kinds of things these squads can do. The coolest and most effective maneouvers need to be coordinated.

Suggestions

A form of AI placed in the squads could help significantly with the top-down coordination to make set pieces happen on purpose rather than let them emerge. The big challenge is being able to detect when and how to enable coordination, instead of letting the individual behavior take over. Then, how specifically should the units in the squads be coordinated; both hierarchical planning and database approach with set-pieces played back like in sports games seem sensible approaches. To what extent does planning the outcome require figuring out the low-level details for each individual? What's a good interface for the squad AI to interact with the individual AI?

Examples

- William van der Sterren has worked on applying HTN planning to ARMA II. (See this PREMIUM presentation.)
- Sony's NBA 09 uses a system based on logical pattern matching to coordinate teams. (Seee this PREMIUM masterclass.)

Challenge I.
Experience Management

Screenshot 9: Horde of zombies in LEFT 4 DEAD.

The Problem

The player is an important part of the game, and the AI certainly reacts to the player's position, his actions, and occasionally his past actions. However, few games go through the trouble of modelling what the player expects from a situation. Once you start doing that, a good designer can taylor the AI to fit those expectations and make sure the game isn't frustrating, unfair or annoying.

Suggestions

The essence of the solution is to have a system that watches the player's actions and collects statistics. Then, the next step is to build a model based on that, to help classify the player and figure out what they are expecting from the game. Casual gamers may expect a specific something very different than your hardcore players.

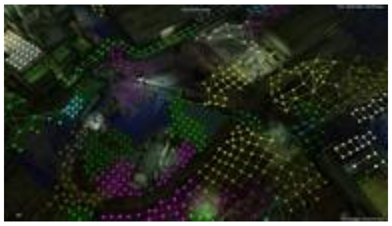
Using pattern recognition, then having an AI Director (implemented using traditional game AI techniques) acting based on those recognized patterns will help craft the a suitable experience for each player.

Examples

- LEFT 4 DEAD uses an AI director to adjust the spawning rates of zombies in the game. (See this PREMIUM article.)
- DARK SECTOR uses dynamic difficulty adjustment for enemies and changes their statistics based on player state. (See this PREMIUMinterview.)

Summary

You may have noticed some general challenges of Game AI are missing! I don't claim for this list to be exhaustive, but I did try to focus it on FPS specific problems. Things like emotions, social interactions, multi-agent navigation, or even crowd behaviors — although challenging — aren't typically a focus of this genre.



Also, you might notice the absense of navigation challenges or even questions of AI decision making or behavior. Even though these domains have many open engineering issues, the technology has matured a lot over the past few years, and there's significantly less room for innovation in those departments. The fact that middleware companies are so established already shows these are rathermore areas for incremental improvements.