Recruits AI: Dynamic Combat Tactics

Jason El-Massih
Commotion Games, jel-massih@hotmail.com
http://www.recruitsgame.com/

Introduction

One of the biggest challenges for Game AI is to create Intelligent and Natural behavior. The AI's ability to adapt to different situations is difficult to achieve with many techniques, such as pre-placed tactic Hints. It becomes easy for a developer to give into the norm and have the AI system rely on pre-decided actions or Scripting. It is essential that tactical games provide significantly smarter and increasingly responsive AI to increase the challenge, rather than simply buffing the health and damage.

In this document, I will describe the design of the Tactical Combat AI for the game "Recruits"; Commotion Games' Independently Developed Top Down Shooter using the Unreal Development Kit. In Recruits, the AI are presented with a wide array of dynamic situations and terrain, and are tailored to react in believable and tactically sound ways. I will start by defining Dynamic Combat Tactics in relation to this article, and how it is incorporated into Recruits AI. Then I will describe the various concepts and mechanics behind the Dynamic Combat Tactics system, Position Evaluation and its application, as well as Suppression and Indirect Fire. Finally, I will talk about my experiences with Dynamic Combat Tactics within Recruits, and the many limitations that had to be worked around due to the use of Unreal Development Kit.

Terminology

Throughout this document I will be making references to the term "goal". When I use this term, it means the current task that the AI Agent is undertaking. Some examples of this are: Shoot at the target, throw a grenade near the target, run to cover, move up and flank the target, etc.
I will be also be using the term Agent to reference the actual physical representation of the AI, such as the Pawn or Bot.

Why Dynamic Combat Tactics?

The new Generation of First-Person and Third-Person combat Games can include a number of the following challenging characteristics:

- Non-Linear Player Fight Paths with a multitude of scenarios
- AI Squads carrying out orders from the player at any location
- Changing Paths and Cover locations from destructible environment
- Advanced, Believable Tactics in Complex and Dynamic Environments

A very common approach to creating Tactical AI in Shooters is a combination of scripts and level placed tactics hints (for example marking, ambush or safe locations). Scripts allow for the AI to react to a very specific situation, and provide a powerful and predictable solution, however, any deviations from the situation can produce less than desirable results (Non Tactical Behavior, for example, such as running from cover or ignoring the player). The need to fight anywhere, from threats that can appear anywhere in a dynamic environment makes such methods unfeasible without placing excessive amounts of hints and creating a slew of scripts. Pre-Defined Hints also cannot provide situation-specific details, such as if the location is a good place for an ambush, or if it is good place to Hide. Likewise, Scripts and their extremely linear approach to situational awareness are not suitable for balancing the dozens of potential inputs from the environment to tailor a tactical decision.

What are Dynamic Combat Tactics?

The AI has Dynamic Combat Tactics when it tailors its decisions based on the current situation and terrain using on-the-fly algorithms and dynamic Inputs. For example, when an AI selects a new position to move to, it takes the various lines of Fire, its Exposure to threats, and its proximity to nearby units into account. Whenever the AI is selecting a path, the path is chosen based on its degree of safety from threats, rather than just simply the shortest distance. This permits the AI's goals to be abandoned or adapted as new threats arise and new possibilities are opened up. The Player will be able to notice this system by the AI acting more responsive to subtle differences in his actions and positioning. The AI will act more tactically sound even in extremely complex situations with multiple moving threats, ever-changing terrain, and incoming threats to avoid. The player is able to experience an advanced, yet amazingly robust, execution of a wide set of tactics anytime and anywhere in the game world.

This System also takes the pressure off of level designers because they can concentrate more on the overall flow of the encounters, and still get generally good AI behavior.

This allows for a significant reduction in the amount of time spent with scripting and placing Hints, if not complete elimination of the need altogether.

Overview of Recruits AI

Recruits is a top down shoot-em-up taking place in the Vietnam War Era. It plays an odd twist on the genre by presenting tactical game-play over the traditional Arcade feeling familiar with most Shootem'ups. It being a Coop centered game, the players friendly AI must behave in a believable, tactical manner that simulates human behavior

To be able to understand where Recruits AI is being made more Dynamic, I will describe the think cycle of an AI agent. At any time, the agent pursues its most desirable goal based on the current situation (ex. Defending position, flanking the player, throwing a grenade, advancing its position). The Agent will pursue that goal until one of numerous events occur, Goal is achieved, Goal is no longer achievable, or a new goal is more desirable.

In Recruits, the AI computes its destination based completely on dynamic information gained from its environment such as the amount of cover from threat, various lines of fire to friendlies and enemies, goal assignments and proximity to other agents. Even when not performing a movement operation, like throwing grenades or indirect fire, the AI still relies heavily on Position Evaluation for completing its task in the most tactically sound manner.

Dynamic Combat Tactics: Concepts and Examples

Position Evaluation is the Keystone of my approach to getting tactical AI. Almost every potential goal of an AI Agent incorporates the use of a position evaluation function. It enables the agent to get both static and dynamic information about its environment and combine it into one number. This allows a robust and universal method of allowing the agent to decide on the most tactical action or goal.

Position Evaluation Functions

Position Evaluation Functions are a well-known AI technique. They allow the AI to determine which position is the most attractive one to move to, the most useful goal to pursue or which threat to prioritize. An example would be if a Position Evaluation Function in Recruits is computing the presence of cover in nearby locations from a given threat. This gets taken into account when deciding on a location's overall desirability.

One of my objectives for Recruits AI was to get the agents to behave as close as possible to how a normal human would behave. It is also not feasible to have the AI predict every possible hostile action in a 3D environment due to the amount of calculations and performance cost it would have, with the AI only having access to limited resources. Therefore, Instead of using a mini max tree search or similar to go through every possible action beforehand, the AI compensates for small hostile actions with its representations of Line-of-Fires and generalized condition checks. This provides for a fairly intelligent system that still makes the occasional mistake, similar to how a player would.

Tactical Position Picking

During combat, an AI Agent will frequently need to change position to achieve its goal. Some examples of this are for the AI to pick a new position to move up to, flank the player to get better line of sight/firing angle and Move away from a grenade. Generally, The AI will attempt to pick a new location that is suitable to meet its goal, and provides the greatest tactical advantage.

To pick a position to move to, the AI considers all possible positions within an area around the agent know as its Area-of-operations. From these positions, it then eliminates any positions that are already occupied by another agent and any positions outside the Area of Operations. A position Evaluation function is then invoked on each remaining position within the pool, assigning a score of desirability. The Higher the score, the more attractive the position is for the agent.

A Post Process Step is then performed to verify which of the highest scoring positions is suitable for the agent's current goal. An Example of this is if the Agent is trying to find cover in the middle of an empty field. Even the highest scoring position will not provide cover. For Recruits this is solved by Level Placed Nodes that are given a priority, however in many cases the AI may simply abort its search for cover.

All weights are non-negative values to allow solely for the inputs to reflect the attractiveness of a position. For Example, rather than subtracting the distance weight from the score, the value is increased based on the proximity the position is to the agent.

Table 1 below shows several examples of functions used in Recruits.

Table 1 Examples of inputs for the position evaluation function.

Position Evaluation Input	Description: (Rewards higher score for…)
Proximity to current Position	Being closer from current Position
Proximity From Goal	Being closer to goal (if applicable)
Cover From Threat	Does Position Provide cover from threat
Line-Of-Fire to Target	Can Shoot at Target
Within Optimal range from target	Being Within an optimal range
Outside Danger Zone	Outside potential blast areas
distance from friendlies	Being within optimal range of friendlies
Wall Hugging	Being Close to a wall or other obstacle

Unique Behavior Types Through Parametrization

Having Configurable Position Evaluation Functions allow for the creation of distinctive behaviors between different character types. The Position Evaluation Function can be tweaked and tuned so that different inputs can carry heavier weights. For Example, You can adjust it so that positions closer to target carry a heavier weight, which would allow for enemies with shotguns to have specialized behavior that allows for efficient use of their weaponry.

Applying Dynamic Combat Tactics in Recruits

One of the biggest issues that I constantly ran into when developing the Dynamic Combat AI is that Unreal script is SLOW. Since Recruits is being made with the Unreal Development Kit, I do not have access to the Source code, and thus must do all of my coding in their scripting language, Unreal script. This is very problematic when trying to create a system that is capable of performing advanced calculations on the fly to adapt to dynamic situations. I was able to perform a little trickery however, which allowed for the AI to be capable of the functionality I wanted it to have. One of the biggest resource savers I had was to simply only check what was necessary, when it was necessary. This is to say that rather than have the AI reevaluate its position every tick, there is a general logical Tick that gets called every 2-5 seconds. What this logic tick does is very quickly check for any changing conditions within the environment and adjust its current goal accordingly. An example could be that the AI is trying to reload, however the target ran around and flanked the agent, giving it better LOS. Instead of the AI completing its current task of reloading and just remaining in its current position like a sitting duck, if the agent was able to see the target move on its position, it will abort the reload, seek a new destination that provides cover from the threat, and then complete its reload action. By having this logic tick at an adjustable rate, it allows for varying difficulty of AI through having them play more tactically and aware, rather than simply buffing their health or damage. By allowing the agent to check its environment more often, it will allow for more difficult and intelligent AI.

However, by having too many AI agents with an extremely low logic tick rate, performance can begin to be reduced due to the amount of checks that need to be run. To counteract this, I did many optimizations with the position evaluation functions. One of the quickest and most efficient optimization I performed was having the AI have an "Area-of-Operations". This allowed for me to quickly check the position-nodes within the area, and simply discard all the nodes outside. In the event that the agent's current goal lies outside of its area-of-operations, rather than simply expanding the radius, and ultimately the performance hit, I assign bonus weights to nodes that towards the direction of the goal. This allows for the AI to move towards its goal whilst moving tactically and safely rather than running around blindly and dangerously.

Pathfinding for Recruits has been a sheer Nightmare. We are using a mix of navigation meshes and custom position-nodes for all the pathfinding. However, Unreal Development Kit's NavMesh system is an absolute Wreck. Although it has come far since it was first introduced, it is still one headache after another. It is also a major setback with how inaccurate the pathfinding can be, with the Agent taking rather long, extraneous detours to a specified position rather than a nice, simple, clean route. Also,

due to not having access to source, and no native support for pathfinding that takes threats Line of sight into account, a lot of the position finding has to be filtered in the position evaluation function since the agent will always take the shortest route to its target destination. This has been relatively challenging yet exciting to overcome since it's such a major limitation, yet has been circumvented in a rather reasonable fashion. One thing I had to do was to provide higher preference to cover that is closer to the agent rather than the threat. Although this still can have many issues, it does a rather reasonable job of eliminating more blatantly stupid decisions and paths, and the occasional mistake allows for a unique window that the threat can utilize to eliminate the Agent.

Another Key component to having the AI behave as rationally as possible and to limitate any flaws with the pathfinding is the Level Design. It is essential to ensure that there is no bugs in the NavMeshes generation that can cause key areas to be blocked off or become otherwise inaccessible. It is unfortunate that within the Unreal Development Kit, this happens very often and is a headache to fix. It is also very helpful to limit the amount of unnecessarily placed nodes. The way the AI is designed is that it is perfectly usable without any nearby nodes, their fore even if the player goes to a secluded area unmapped by the Level designer, the AI will still be combat effective and not break.

The Accomplishment I am proudest of is not necessarily the AI itself, but rather creating AI in such a way that it is both efficient and effective in such a limited and unsuitable environment such as the Unreal Development Kit. The Creation of this AI has enormously developed my skills at not only pioneering new techniques with the Development Kit, but wringing the limits to get every ounce of potential out of them as possible. Although there is still much work to be done on the AI described in this document, it is my belief that even creating something that has been made is quite an accomplishment.

Future Improvements:

Multi-Threat Tracking

One Major Improvement I wish to make on the system is the use of Multi-Threat Tracking. That is, that rather than simply taking the primary threat's status into account when performing a decision, but to incorporate other secondary threats. If done correctly, this will allow for significantly more intelligent and Responsive AI that will be more capable of adapting to a significantly wider array of situations. One potential problem with this addition however would be having the AI becoming almost too responsive and extremely difficult to defeat.

Conditionalized Heuristics based on Goal.

Another improvement to the system I plan on tackling is to adjust the AI's Heuristic to take its current goal into account. This will allow for the AI to choose its position not tactically alone, but rather on it current goal. Currently, if an agent's goal is to throw a grenade at the enemy position, it will prefer to position itself so that it has a clear line of sight and cover from the threat and then throw the grenade to weed out the threat. An example of a conditionalized Heuristic would be if the agent would move to a location that would provide complete cover, throw the grenade by performing indirect fire checks, and then positioning itself to shoot the threat.

Acknowledgements

I would like to thank Remco Straatman and Arjen Beij for their work at Guerrilla and establishing many of the ideas and concepts used in this document.

Biography

Jason El-Massih is a programmer for Commotion Games. He has recently graduated from High School and is clueless about his future. With little formal programming education, he just runs around doing what he loves and learns as he goes. He likes long walks on the beach, talking in the third person and Turtles.
(jel-massih@hotmail.com, http:\\jel-massih.com)