Fuzzy Fitness Scoring for Companion AI Strategy Selection

Carlos Gutierrez, Squirrel Eiserloh

***Abstract*— The primary aim is to demonstrate the effectiveness of Fuzzy Fitness Scoring for Companion AI characters. Through this method, AI characters can select their highest scoring strategy from a list of potential actions, without the need for behavior trees or state machines.**

***Index Terms*— Companion AI, Fuzzy Scoring, Game Awareness, Actions**

I. Introduction

As games continue to advance, developers strive to create progressively more dynamic and engaging worlds for players to experience and explore. A key component of creating these dynamic worlds lies in improving their inhabitants: AI driven Non-Player Characters (NPCs). As games, worlds, and the stories within them continue to evolve, developers continue to closely involve NPC characters with the player, often as allies, companions, or escort subjects. However, these efforts have not been without problems.

Throughout the history of digital games, players have often viewed companion or escort AIs as an annoyance, or at worst, hindrances. Asking video game enthusiasts about companion AIs typically provokes strong responses, including primarily:

- The companion fails to understand player intention;
- The companion obstructs/gets in the way of the player;
- The companion mismanages resources;
- The companion behaves in ways that confuse the player;
- The companion requires high maintenance and constant attention.

What these common threads of thought ultimately boil down are two main ideas: Players feel that the AI is either getting in their way, or they feel they have to babysit the AI. Both of these scenarios can drastically reduce player enjoyment.

Attempts to solve these issues have a tendency of masking the problem by making the companion AI overpowered, invisible to enemies, or altogether invincible.

Carlos Gutierrez is a graduate student at the Guildhall at Southern Methodist University specializing in Softward Development.

Brian "Squirrel" Eiserloh is a Software Development professor at the Guildhall at Southern Methodist University.

Despite these issues, games with prominent AI companions will continue to increase in popularity. In 2013 alone, two of the most critically acclaimed games (The Last of Us and BioShock Infinite) featured companion AIs, and received much of their praise due to the presence and use of companions.

In order for companions to be progressively more successful at "behaving correctly," they need the following criteria:

- The companion understands player intentions and selects strategies appropriately;
- The companion is able to synergize with the player;
- The companion uses resources efficiently and appropriately;
- The companion's motivations are conveyed clearly.

The first steps required to reach this point, and the focus of this project include:

1. Gathering data and information
2. Evaluating and interpreting the data
3. Selecting a strategy to execute.

II. Research Review

A recent example of companion AI that felt aware of the player was in "The Last of Us." The player have a companion, Ellie, for most of the game, who engages in dialog with the player avatar and attempts to assist in combat.

Ellie follows the player by evaluating potential points to move towards relative to the player. Potential points are eliminated if objects are in the way so that Ellie does not get stuck following the player. Ellie was allowed to "cheat" by teleporting at crucial times when the player wasn't looking at her, and being immune to enemy detection during stealth. Ellie's damage output is also reduced while offscreen, to ensure that the player is dispatching most of the enemies [1]. The issue addressed by this reduction is cannibalization of the player's fun. When Ellie's damage was "correct," she became too powerful, which drastically reduced the number of enemies that players could take out. A key component of developing companion AIs is ensuring that they allow the player to have as much fun as possible, even if it means sacrificing some "correctness."

Another method of making AI appear more aware and intelligent was demonstrated in Valve's "Left 4 Dead" [3]. The game features four characters that speak lines of dialog based on what their current circumstances. A challenge the developers faced was to ensure that characters could speak about their scenario without requiring excessive amounts of special code and without the characters repeating the same lines

Valve's solution came up with was use of "fuzzy pattern matching." The AI polls its field of view every now and then to scan for objects. Each object has a corresponding entry in a database which dictates which lines characters can say upon seeing it. When the AI queries the database an appropriate line of dialog, it uses query rules to find appropriate lines. Rules also include factors such as the number of times an object has been seen, so the characters do not repeat dialog, and can say unique lines based on their circumstances. For example, if a character sees a barrel, they will say a line of dialog pertaining to the barrel. When they see a second barrel, they will comment specifically on the second sighting of a barrel.

Although this implementation was utilized for the game's writing, the ideas behind fuzzy pattern matching can potentially be utilized for AI behavior in general. NPCs could use this kind of database to make better informed decisions when planning its next move.

Because making intelligent companion AI shares similar problems with Dynamic Difficulty, it was worth considering implementations of those kinds of systems. An example of dynamic difficult can be found in "SiN Episodes" Personal Challenge System [5]. In this system, vast quantities of statistics were gathered and funneled to several "Advisors," each of whom was written to keep track of a specific aspect by analyzing selected metrics and providing an overall heuristic score. These scores determine how "happy" or "unhappy" the Advisor is. Unhappy Advisors make recommendations to the Director, who adjusts the game.

Valve's "Left 4 Dead" also touched on this aspect with its Director system [4]. The Director measures the game's current "intensity" by keeping track of several metric, such as the number of enemies encountered, how often the players become incapacitated, and number of items obtained. Based on gathered data, the Director adjusts the amount of enemies and items spawned.

III. Methodology

Base Game

For this project, we created a small Action Role Playing demo (similar to Blizzard's Diablo series) which was implemented in a custom game engine using C++ and OpenGL. The demo is single player, with simple tile based level construction, but with characters moving in real time. Graphics are intentionally limited to simple, abstract 2D shapes (figure 1).

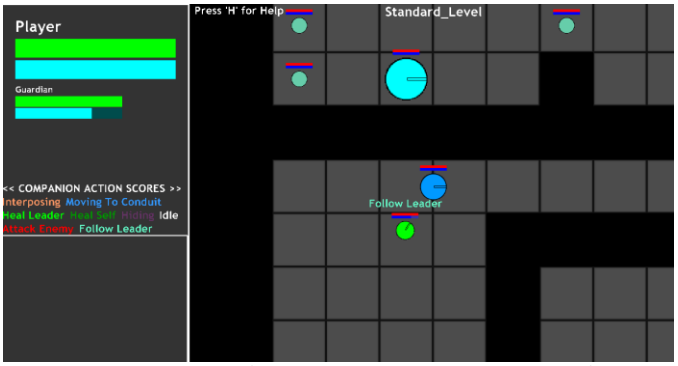

Figure 1: A screenshot of the demo. The player and companion find enemies on the other side of the wall

The premise used to test the Thesis is that of the player, a Wizard, and his companion, a Guardian trying to clear a series of small dungeons full of monsters (figure 2). The player character has a couple of potent spells to use, but is vulnerable in melee combat and does not have very much health. The Guardian has no ranged attack spells, but can heal the player and "tank" (take the hits of enemies so they do not attack more vulnerable allies), in addition to having more formidable melee attack capability. The player and companion cast spells using Mana, a slowly regenerating resource.

Figure 2: The characters in the demo and basic stats.

To move through the room, Enemies and the Companion AI utilize A* pathfinding with opportunistic shortcuts, and use capsule traces to check for line of sight. Rooms can have as many as 30 enemies in varying combinations.

Companion AI

In order to implement an AI that would react dynamically based on the current situation, discrete Actions were created that the AI can consider each frame. These actions are implemented as C++ classes with virtual functions. The AI considers and scores each potential action, then picks the highest scoring action to execute.

Each Action provides following basic functions:

CanBeTaken is a Boolean function that simply checks if the action should be considered for scoring at all. It is used to prevent the AI from taking actions that are completely suboptimal or illegal. Where scoring functions convey "soft" consideration, *CanBeTaken* allows authorship of a "hard" check that will definitely remove the action from

consideration. An example of *CanBeTaken*'s use is when the companion considers healing the player. If the companion does not have enough mana to cast the spell, *CanBeTaken* returns false.

For actions that do not have resource requirements, CanBeTaken can take other factors into account – for example, when the AI considers attacking, the CanBeTaken function will return false if there are no enemies within vision range and line of sight.

CalculateScore determines how "hot" or appealing an action currently is. This function returns a float that is within a range of [0, 1], where 0 is the least desirable score, and 1 is the most desirable score. In this way, actions are judged from a distinct "cold/hot" perspective, with the AI picking the "hottest" action each frame. In order to calculate each Action's score, several Action-specific factors were calculated and normalized to the [0, 1] range.

All of the factors for a given action are multiplied together to calculate the action's score. Since each value is normalized, then factors less than 1 will bring down the overall score for the action. If a factor is 0, it will zero out the entire score. If design demands a factor that affects the score without ever being able to zero it out, "soft" factors can be created by clamping their range to exclude zero.

An example of score calculation is when scoring "Heal Leader" in a scenario where the player's health is 75%, the companion's mana is 90%, and healing replenishes 30% of max health. One of the factors that features prominently is the player's health. This factor is calculated by taking 1 – the Player's health percentage, which in this case results in the factor becoming 0.25.

The companion's mana factor becomes 0.9 due to mapping linearly with its mana percentage. The total range of overhealing is [0, 30% max Health], which was mapped to a range of [1, 0] to create the overheal factor. In this scenario, the overheal factor is 0.66 due to the 10% overheal.

The final score for the Heal Leader action is determined by multiplying all of the factors together, in this case is 0.9 * 0.66 * 0.25 = 0.14 out of 1. In this case Heal Leader is fairly "cold," but will be picked by the AI if little else is going on. The result is that while the companion will not waste mana in combat, it will select the action at a safer time.

In addition to the factors used to calculate the Action's score, the Action's weight (typically another normalized value) is multiplied against the total score before returning. This allows AIs to have their own personalities, due to how likely they are to take the action in question.

PerformAction is called during the AI's Update function when the AI executes the highest scoring action. It serves as the "update" function for the Action by handling any code that is required to perform it. Upon completion of the function, it returns a Boolean value indicating if the action has finished or not (such as the AI reaching its target destination). If an action has completed, this value

When considering its next move, the AI looks through its list of legal actions, and scores each one. Once the AI has picked the best potential action then it checks the score against the current action's score.

To ensure that the AI does not constantly switch between actions in a sort of livelock situation (figure 3), a thermostat threshold was added. When the AI is considering a new action, it must score higher than the old action's score plus a thermostat value. If an action scores higher than this combined total, then the new action takes over (figure 4).

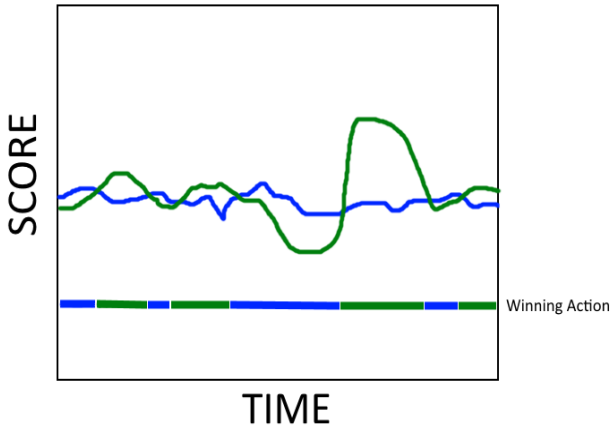

Figure 3: A scenario in which the companion would switch between the blue and green actions rapidly.

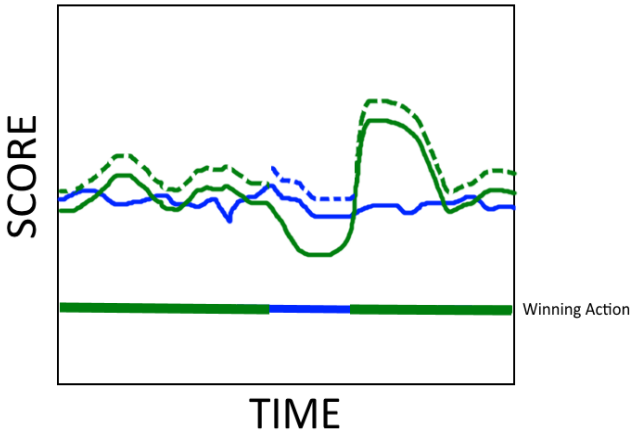

Figure 4: The same scenario as figure 3 with a thermostat added (dotted lines). The result is the AI behaves much more smoothly.

By adding the thermostat, the equivalent of a car's shocks were added to avoid small local maxima from creating spastic behavior.

The thermostat for this project was fairly small – 0.05. This value was selected by observing the degree of "noise" that was produced when two actions competed closely, then setting the

thermostat to a value slightly greater than the typical noise range – about a standard deviation.

Potential issues with the thermostat arise at either extreme – thermostats that are too small or too large. A thermostat that is too small is ineffectual based on the size of the noise it is supposed to suppress. For thermostat values that are too large, actions can get "stuck on" or switched too infrequently.

An exceedingly large thermostat can create a scenario in which it becomes impossible to switch actions. Consider an action that scores 0.85 plus a thermostat of 0.2; because the total is greater than 1.0, no action can possibly beat its score.

The Actions used by the AI are:

ACTION NAME	SCORE FACTORS
Interpose	- High Companion health, low Player health, - High Player proximity to Enemies
Move to Conduit	- Enemies within range, - Player's current action.
Heal Leader	- Low Player health, - High companion mana, - Low overheal amount, - *High cast percentage*
Heal Self	- Low Companion health, - High Companion mana, - Low overheal amount, - *High cast percentage*
Hiding	- Low Companion health, - High Player health, - High proximity to enemies
Idle	- Low, constant score
Attack Enemy	- High companion health, - Close proximity to enemies
Follow Leader	- High distance from Player

Figure5: Table of Actions and the Factors used to calculate their Scores

Do Nothing: The base line action, has a tiny, constant score. This action is considered "idle" and is only taken when the AI has no other options.

Attack Enemy: Find and attack the enemy threatening the player most. The companion defines the most threatening enemy as the one that is closest to the player. *Attack Enemy'*s only factor in *CanBeTaken* is the distance to potential targets. *CanBeTaken* only returns false if the AI has no line of sight to any enemies within its vision range.

When calculating *Attack Enemy*'s score, there are two factors that are considered. The first is the AI's distance to the most threatening enemy. This factor is calculated as 1 minus the enemy's distance from the AI divided by the AI's vision range. As a result, the factor scores better when the AI is closer to the target. This factor is clamped on range [0, 1].

The second factor is the companion's health. This factor is intended to make melee combat less appealing if the AI is at low health. This factor was clamped between two points to prevent overreactions to lowering health. Above half health, the AI was fully comfortable attacking, but below half health, it drops sharply down to zero at about a quarter health. The score was range mapped between the two points to create the drop. By doing so, the AI would not behave in a suicidal manner by attacking at dangerously low health (figure 6).

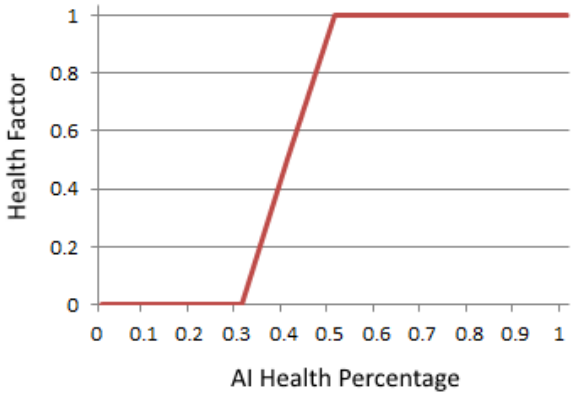

Figure 6: Health Factor for Attack Enemy. The factor is clamped and range mapped to be slightly gated.

Heal Self: Casts a heal spell on itself. This action is taken when the AI feels that its health it at a dangerous level. *CanBeTaken* is simple for this action: return true if the AI has sufficient mana to cast the spell.

When calculating the score for Heal Self, several factors are considered. The first and simplest factor is the AI's health. The AI is more inclined to heal if its health is lower (figure 7). The next factor is the AI's mana: if the AI has mana to spare it will be more inclined to spend it healing. This factor only zeroes out if the AI does not have enough mana to cast.

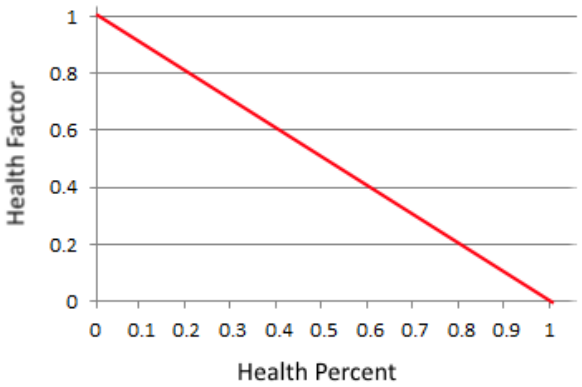

Figure7: Health Factor for Heal Self. The factor has an inverse relationship with the health percentage

In the interest of mana efficiency, a factor to govern overheal was introduced. This factor lowers its value in direct relation to how much of a potential heal is overheal. An added

soft factor was the "is casting" factor, which would give a slight bump to this action's score depending on its cast progress, so the action would be interrupted less when nearly finished.

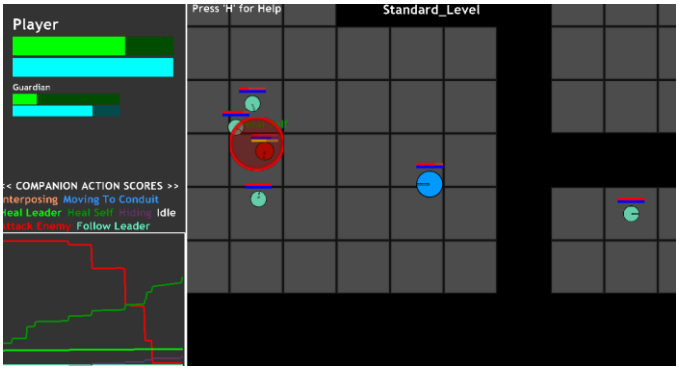

Figure 8: A combat scenario where the AI begins to heal itself. The graph on the left demonstrates how Heal Self (dark green) has overtaken Attack (red).

Heal Leader: Casts a heal spell on the player. This action is scored similarly to Heal Self, but instead of the companion's health becoming a factor, the player's health is used.

Interpose: Moves in such a way to impose between the enemies most threatening to the Player and the Player himself to physically shield the Player. *Interpose* can be taken at any point, so long as the AI has a leader to protect.

The factors used in scoring include the AI's health and the player's health. The AI would be more inclined to Interpose if its health was high and the player's health is low. However, the AI does not need to Interpose if the player is far from danger. For this reason, player proximity to danger was added.

To calculate the player's proximity to danger, the AI determined who was most threatening to the player. Because the enemies are melee focused, this threat would usually be the closest enemy. This factor would increase as the enemy neared the player, which would cause the AI to only perform the action if the player is in actual danger.

When performing the action, the AI moves to the point relative to the player where it can position itself between the player and the greatest threat. That position is calculated as follows.

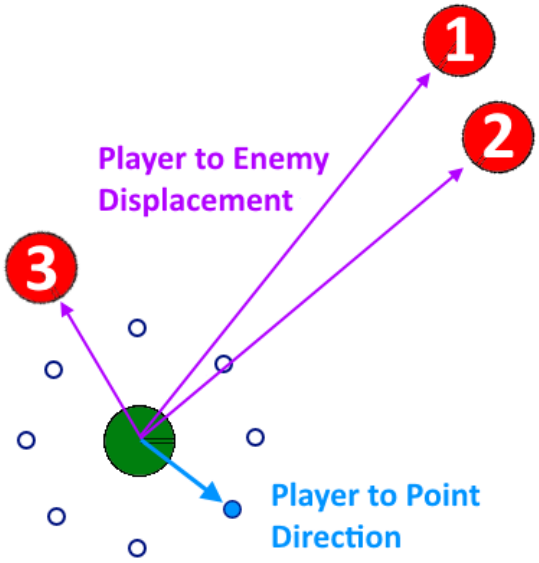

Figure 9: Calculating best position to Interpose given 3 enemies

The blue circles in the diagram are points that the AI is considering. To "score" a point, the AI first calculates the dot product between *Player to Enemy Displacement* and *Player to Point Direction*. The result is then divided the length squared of *Player to Enemy Displacement* and added to the point's score. This formula is used for each enemy, resulting in a total score for a point.

The dot product component of the formula indicates how effectively the companion is interposing between enemies. For points close to enemy 3 in the diagram, the vectors would be aligned, resulting in a greater dot product.

The reason behind the divide in the formula is to allow closer enemies to weigh more in a point's score. In the diagram, enemy 3 is a more immediate threat to the player, so we want the companion to interpose between it first.

Move to Conduit: When the Player is casting Conduit, moves to the best position to maximize damage. Conduit is a spell that does massive area of effect damage centered on the companion. As such, the companion tries to maximize the damage output by moving to the greatest density of enemies.

As such, *Can Be Taken* is simple for this action: return true if the player is casting Conduit. Score is calculated by determining if the companion can reach enemies in the time it takes for the player to cast the spell. If the AI can hit enemies, then the score returns a very high constant. For this reason, when players use Conduit it is the closest thing the AI has to a direct command. The constant, while high, is not 1, to allow for the AI to take emergency actions such as *Hide* if its health is too low.

Hide Behind Leader: Hides behind the Player if the AI's health is row and resources are exhausted, provided the Player is safe and in good health. It is often used to buy time to

regenerate Mana. Given that the AI is supposed to be guarding the player, this action had to be scored carefully.

As the companion's health drops, the score for the health factor goes up. However, the leader's health is also taken into account. If the leader's health is very high, then the companion is more likely to hide when wounded. However, if both are wounded, the companion will not Hide behind the leader, and will likely make a last stand Interposing.

The companion's mana is also taken into account. If the companion has mana, it can heal itself instead of potentially putting the player in danger by running. As such, the score for Hide improves if the companion has critically low mana, and vice versa.

Follow Leader: Simply follows the Player. Is usually the default state for the AI. Initially this action was a simple constant in the vein of Do Nothing. However, more complex scoring was needed to solve the issue of the AI moving too far from the player when attacking enemies. We needed the AI to follow the player like a soldier would a retreating general, so the AI's distance from the player became a factor. The further a player is from the AI, the more likely the AI will want to rejoin him (figure 10).

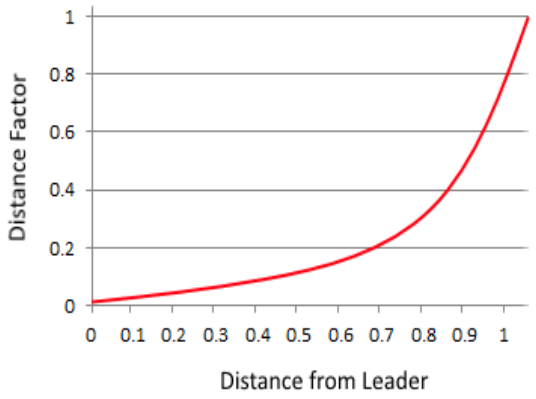

Figure 10: The distance factor for Follow Leader sharply increases as the distance grows

Ultimately, Actions were implemented with the goal of being "free market capitalist," meaning actions should only score themselves based on factors that only positively increase their score. Actions should prove their own merit rather than trying to "undercut" other actions by having special cases or unrelated factors used in calculation. This principle makes developing and adding new actions to the AI more intuitive, since actions do not have cases for every other action and can just focus on their own benefits.

A new subsystem we had to add was allowing "secondary actions" which were small actions that the AI could take while performing another actions. The issue that had to be solved with this system was the fact that actions were mutually exclusive. In particular, the *Attack Enemy* action had sole "governance" over the AI's melee ability. Other actions such as *Interpose* and *Hide* would perform, and the AI would not take any attacks of opportunity despite enemies being nearby.

To create this system, we added another virtual function to the Action class: *CanTakeSecondayActions*. Similar to *CanBeTaken*, this function provides definitive control to when an action allows secondary actions such as attacks of opportunity. For example, *Interpose* only allows attacks of opportunity once the AI has reached the target location.

IV. Results

Ultimately, the artifact demonstrates correctness in utilizing fuzzy scoring to switch actions. The AI would respond in ways that were appropriate for each situation.

However, there were several difficulties I encountered during the development of the artifact that are were significant and had to be solved in order to get the AI to "behave correctly."

The first of these problems was that some Actions would not be utilized when they were obviously needed. The cause was due to the original design of the Action factors used when calculating score. An example of this case was found in two of the earliest actions tested - *Heal Self* and *Heal Leader*. In testing scenarios, the AI would not heal the player when needed if the companion was at low health, because a factor included in the calculation of *Heal Leader* was the AI's health, which would lower *Heal Leader*'s score when low. The original justification for the inclusion of AI health as a factor was that the companion would be less inclined to spend time healing others and would want to focus on healing itself. From the description, it is apparent that the scoring for *Heal Self* began undercutting the scoring for *Heal Leader*.

This scenario is an example of how early Actions were not "free market capitalist." These two actions were not scored purely based on their own merits, but also their potential to affect other actions. The fix was to remove requirements that were scored based on potential for other actions, to ensure that actions are scored purely on their own merits in a given scenario.

A particularly difficult case was close competition between two actions such as *Attack Enemy* and *Hide Behind Leader* In certain scenarios, the AI would continually switch between *Attack* and *Hide*. The reason was that enemy proximity was a factor for both Actions. The result was that when *Hide* was cold, *Attack* was hot, and vice versa. This meant that based on the AI's position, it would continually flip between actions in a sort of live lock.

A solution to the issue was to modify the scoring for the two actions to clamp certain factors. For *Hide*, the health input was clamped to [0, 0.5], which was mapped to [1, 0] to produce the factor number. Attack had the opposite clamp, with the health input clamped from [0.5 to 1], mapping to [0, 1]. As a result, *Hide* is only considered below 50% HP, and *Attack* is only

considered above 50% HP, drastically reducing conflict between the two actions.

In this way, actions could essentially elect to disqualify themselves, leaving other actions' scores unaffected. By balancing the score system in such a way, these actions can utilize a sort of scoring "if" to further determine how hot or cold they are in a given scenario.

A significant consideration remains in rebalancing action score calculation when a new attribute is introduced. For example, if a "stamina" system was added during development, then several actions would have to have their score calculations modified to include stamina. Fortunately, any actions that would not require stamina as a factor remain unaffected.

V. Future Work

Given the way the AI was constructed, it was most adept at handling scenarios as they were in the present. Scenarios that required some looking ahead were not as accurate. In future work, those actions would have more factors to include items such as *health velocity* to utilize the rate of change when judging what is feasible.

Another system to consider would be a planning solution. The AI could either evaluate projections based on the current scenario, or it could utilize the "current plan" as a factor when scoring actions. A more radical addition would be to score potential goals, then use a planner to determine the best course of action.

The currently implemented system has each Action only judging its worth based on its own specific goals. Another interesting direction to consider would be evaluating each Action based on the degree to which it satisfies each of multiple goals. An example use case might be: moving to a particular location. Doing so might put the AI closer to enemies (who he may want to attack), closer to a potion (to pick up), and in a more tactically advantageous position.

Instead of the "goal" being tied to a discrete Action, a number of separate entities representing goals would judge each action, and the action that scores highest across multiple goals would be selected. This system could allow the AI to handle more complex scenarios by evaluating multiple goals simultaneously.

Additional improvements can also be made to the overall system by adding elements of learning for the AI. One example might be adding the player's history with a particular enemy type into action considerations. If a that enemy type has historically given the player trouble, the AI might place higher priority on dispatching that enemy. Ultimately, such a system should integrate well, because it would simply serve to help the AI make even better informed decisions when scoring and selecting actions.

VI. Conclusions

In conclusion, utilizing Fuzzy Scoring to program AI decision making brought a large amount of flexibility to the AI, but it came at an unexpected cost. The closer we came to the conclusion of the artifact, the more that the true nuance of development shifted from programming to design and balancing.

Given how modular the actions are, the bulk of the programming difficulty came from designing and setting up the system itself. Once that was completed, the difficulty from the perspective of making the game fun came almost entirely from deciding which factors to consider for each action and how much to weigh and balance them. To effectively utilize this system in a commercial product, it is likely that staff who are very skilled in balancing and tweaking large amounts of numbers will be required.

Ultimately, the AI was able to appropriately select strategies to handle its current situation. The AI was able to switch smoothly without behaving in odd or strange ways, and without requiring the player to babysit it.

VII. References

[1] M. Dyckhoff "Ellie: Buddy AI in The Last of Us," Game Developers Conference 2014

[2] J. Abercrombie "Bringing BioShock Infinite's Elizabeth to Life: An AI Development Postmortem," 2014

[3] E. Ruskin "AI-drive Dynamic Dialog through Fuzzy Pattern Matching," 2012. Available: http://gdcvault.com/play/1015317/AI-driven-Dynamic-Dialog-through

[4] M. Booth "The AI Systems of Left 4 Dead," 2009. Available: http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf

[5] K. Harward, A. Cole "Challenging Everyone: Dynamic Difficulty Deconstructed," 2007. Available: http://twvideo01.ubm-us.net/o1/vault/gdc07/slides/S3707i1.pdf

Carlos D. Gutierrez is a graduate student at SMU Guildhall, Southern Methodist University's game development graduate program. Carlos received his Bachelor's in Science in Computer Science from Southern Methodist University, and continued to study game development at the Guildhall (email: carlosplusplus@gmail.com)

Squirrel Eiserloh is a game programming faculty Lecturer at SMU Guildhall, Southern Methodist University's game development graduate program. Since he graduated from Taylor University in 1996 (B.A. Physics) he has been working as a professional game developer in the Dallas area, contributing to over a dozen commercial game titles. He co-chairs the Dallas chapter of the IGDA, and coordinates the Math for Game Programmers sessions at the annual Game Developers Conference in San Francisco. (email: squirrel@eiserloh.net)