1.4

Artificial Stupidity: The Art of Intentional Mistakes

Lars Lidén

larsliden@yahoo.com

Everything should be made as simple as possible, but no simpler.

—Albert Einstein

What makes a game entertaining and fun does not necessarily correspond to making its computer-controlled opponents smarter. The player is, after all, supposed to win. However, letting a player win because the artificial intelligence (AI) controlling the opponents is badly designed is also unacceptable. Fun can be maximized when mistakes made by computer opponents are intentional. By finely tuning opponents' mistakes to be intentional but plausible, programmers can prevent computer opponents from looking unintelligent, while ensuring that the player is still capable of winning. Additionally, by catching, identifying, and appropriately handling genuine problems with an AI system, one can turn situations in which computer opponents would otherwise look dumb into entertainment assets.

A common mistake in designing and implementing computer game AI systems is that they are often over-designed. As an AI programmer, it is easy to get caught up in the excitement of making an intelligent game character and to lose sight of the ultimate goal; namely, making an entertaining game. As long as the player has the illusion that a computer-controlled character is doing something intelligent, it doesn't matter what AI (if any) was actually implemented to achieve that illusion. The hallmark of a good AI programmer is the ability to resist the temptation of adding intelligence where none is needed and to recognize when a cheaper, less complex solution will suffice.

AI programming is often more of an art than a science. Knowing when one can apply cheap tricks and when to apply more sophisticated AI is not always straightforward. For example, as a programmer with full access to game data structures, one can easily cheat by making non-player characters (NPCs) omniscient. NPCs can know where their enemies are, or know where to find weapons or ammunition, without seeing them. Players, however, often eventually detect cheap tricks of this type. Even if they can't determine the exact nature of the cheating, they might report feeling that NPC's behavior seems somehow unnatural.

A Few Tricks

Creating an NPC that can beat a human player is relatively easy. Creating one that can lose to the player in a challenging manner is difficult. The challenge lies in demonstrating the NPC's skills to the player, while still allowing the player to win. The following are a few tricks that allow a game to show off NPC intelligence and help to ensure that a game is fun. The tricks are primarily for the first-person shooter (FPS) genre, but some can be applied in other arenas.

Move Before Firing

Nothing is more disturbing than walking into a new room or arena and being immediately shot by a computer opponent. A player entering a novel location is likely to be overwhelmed with new textures, new objects, and new geometry. Trying to recognize the unique textures associated with an enemy in the morass of background textures is not inherently a pleasurable experience. This is particularly true as polygon and texture budgets have grown enormously. Any NPC attacks that take place during this time are aggravating. Such cheap shots should be avoided at all costs. In such situations, the player has no warning of the impending danger and often no way of knowing where the shot came from.

One of the simplest concepts for decreasing the frustration factor is that opponent NPCs should move the first time they see the player rather than shoot at them. Making an NPC run from a vulnerable open position into a location that is covered before attacking cues the player that a combat situation is about to begin. In situations with multiple opponents, it is usually sufficient to have only one of the opponents move as long as the remaining NPCs wait before attacking the player.

Cueing the player that combat is about to commence is particularly important in action-adventure and FPS games as they typically consist of two primary modes of play. The player is usually either in an exploratory/puzzle-solving mode, during which little or no combat takes place, or they are actively engaged in a combat situation. Warning players that they are about to enter combat mode is vital for game pacing as the player switches from a slow, deliberate form of game play to a fast-paced one.

Be Visible

Although in real-life combat situations, opponents do their best to remain invisible, in the game world great camouflage makes for bad gameplay. Pixel scrubbing while searching for opponents is not an enjoyable experience. Making opponent NPC textures high-contrast with the background allows the player to more readily spot enemies and start the real game experience. Opponents can still have camouflage-like patterns on their uniforms, but their color or brightness should contrast strongly with those of the environment.

Have Horrible Aim

Abundant gunfire is desirable, as it keeps players on the move and the tension high, thus increasing the pace of the game. However, abundant gunfire is undesirable if the player dies too quickly. One of the simplest tricks for dumbing-down NPCs and to amplify the pace of the game is to give computer opponents horrible aim. By doing so, one can have abundant gunfire without making the game prohibitively difficult for the player. FPSs often use a bullet spread as wide as 40 degrees.

Alternatively, one can reduce the player's difficulty by making opponent bullets do very small amounts of damage. However, in doing so, one loses some of the secondary benefits of bad aim. One of the fortuitous by-products of bad aim is the tension created by bullet tracers flying past the player's head or puffs of concrete dust or sparks exploding from projectiles impacting walls next to the player. Additionally, there is a feeling of reward imparted by having just been missed by a bullet. Players will often attribute near misses as an affirmation of their clever movement skills.

Miss the First Time

For weapons that do more damage, such as those that kill with one or two shots, something more than bad aim is required. In general, it is not fun to suddenly and unexpectedly take large amounts of damage. Players often feel cheated in such situations. One can alleviate this frustration by intentionally missing the player the first time. Doing so gives the player a second to react and maintains a high level of tension.

In addition, intentional first misses can be strategically placed. One of the irritating aspects of being shot from behind in an FPS is that the player has no idea where the shot came from. Several FPSs have attempted to alleviate this problem by adding screen cues (such as flashing red icons) that indicate the direction of attack. Such cues (usually justified as being part of a heads-up display) tend to break the illusion of reality and are surprisingly not as salient to players as one might expect.

Intentional misses, particularly those coming from behind the player, can alleviate this problem by indicating the direction of attack without breaking the illusion of reality. A laser beam or projectile with a tracer that is strategically placed to hit the floor or a wall just in front of a fleeing player conveys the direction from which the attack commenced. Additionally, the directional information conveyed by lasers or tracers is more informative than simple flashing screen cues, allowing the player to respond to the attack in a more appropriate manner.

Another valuable way to employ intentional misses is to hit particular environmental targets. For example, as the player nears a water barrel, porcelain statue, or glass vase, rather than aim for the player, the NPC aims for the nearest breakable object, preferably one that shatters as dramatically as possible. Remember, the goal of a good AI programmer is not to kill the player, but to create tension.

One final and more advanced use of intentional misses is that they give the game designer a method for herding the player. Carefully placed rounds of projectiles shooting next to the player can compel the player to move in a direction determined by game design.

Warn the Player

Another effective method to increase game enjoyment is to warn the player before attacking. This can be done visually, by playing a short "about-to-attack" animation, or aurally by playing a sound (a beep, click, etc.), or having a human opponent announce "Gotcha!" or "Take this!" before attacking. Aural cues are particularly important when the player is being attacked from behind. They give the player a chance to react to the attack without feeling cheated.

As humans in real combat situations would never pause to warn their enemies, one might think this would make the computer opponents appear to be less intelligent. To the contrary, such warnings can actually be used to draw attention to other more remarkable aspects of the AI. For example, an intelligent NPC can be programmed to discover routes to flank an enemy or locations to set up an ambush [Lidén02]. If the players are immediately killed as they fall into the ambush, they won't have time to comprehend their opponent's crafty tactics. The intelligent behavior will be more salient if the NPC makes a warning sound so the player has time to turn and see the NPC hunkering down in the clever ambush location before the player is attacked. If the ambushing NPC consistently kills the player, the game will no longer be enjoyable. Warning players gives them time to see the clever AI, yet have to time to react to it without being killed.

Another advantage with aural cues is that players readily become conditioned to react to specific sounds as long as they are consistent [Madhyastha95]. For example, if a particular phase ("Gotcha!") precedes an attack, after enough occurrences, the sound will elicit a physical reaction in the player. Such conditioning can dramatically increase tension in a game. Furthermore, if the phase "Gotcha!" is only used when an enemy employs a particular AI combat strategy (such as figuring out how to flank the player), after witnessing the flanking maneuver/"Gotcha!" pairing several times, the player will eventually infer that a sophisticated flanking maneuver has occurred even if it wasn't witnessed.

Attack "Kung-fu" Style

In many games, the player is in the role of "Rambo" (in other words, one man taking on an army). Although for some genres, mowing down large number of enemies with single shots might be appropriate, opponents in such games are like lambs to the slaughter with little in the way of artificial intelligence. If our opponents are intelligent, however, then taking on more than a few at a time will prove to be too much of a challenge to the player. On the other hand, having many enemies makes for a more exciting, dynamic and fast-paced game.

A solution is to design combat to occur "Kung-fu" style. In other words, although many NPCs might be in a position to attack the player, only a couple should do so at a time. Others should occupy themselves reloading weapons, finding cover, or changing their position. No opponent should stay in a particular location for too long, even when the current location provides a good attacking position. By swapping who is attacking and keeping the opponents moving, a fast-paced combat situation is created in which the player is confronted by many enemies but only attacked by a few. Surprisingly, players confronted with this scenario usually don't realize that only two opponents are actively attacking them at a time, even when confronted with a large number of enemies.

Tell the Player What You Are Doing

When a player witnesses an NPC's actions, it can be difficult for the player to interpret what the NPC is actually doing. Is a running opponent going for cover, getting reinforcements, doing a flanking maneuver, or just running aimlessly trying not to get shot? Complex NPC behavior is often missed by the player. When this happens, an AI developer has done of lot of work for nothing. One effective way to overcome this difficulty is to literally tell the player what the AI is doing. For example, an opponent might yell "Flanking!" or "Cover me!" or "Retreat!" to his compatriots when performing an action. Such cues can be highly effective and often have the beneficial side effect that players assume intelligence where none exists.

React to Mistakes

Even the most sophisticated AI system makes mistakes; they are inevitable. If not handled correctly, they make NPCs appear to be dumb. By recognizing when a mistake has occurred and reacting to it intelligently, not only can the illusion of intelligence be preserved, but the mistakes can be turned into features.

Consider the computations required to accurately throw a projectile at a target. In a rich 3D world with moving objects (including other NPCs and the player), it's unavoidable that occasionally a mistake will be made despite sophisticated physics calculations. A grenade will bounce off another object or another NPC and land back at the feet of the NPC that threw the grenade. (Note that players occasionally make this mistake too!)

If we simply let the NPC stand there and blow himself up, the AI will appear to be pretty pathetic. However, if we recognize the mistake we can react to it appropriately. If the NPC that threw the misguided grenade covers his head with his arms, shows an expression of surprise and/or fear, and yells "Oh no!" it no longer looks like a failure of the AI. Instead, the mistake has now been turned into a feature, adding personality to the NPC, as well as humor and some interesting variety to the game.

Pull Back at the Last Minute

The goal of an AI system is to create excitement and tension for the player. The ideal is to have the player feel challenged, pushed to the edge, but still win the game. One trick is to directly architect the pushed-to-the-edge scenario into the AI. In this model, NPCs will attack vigorously until the player is near death. The performance of the player is monitored carefully so that the player's health or resources are pushed to the limit, but not beyond. Once the player has reached the edge, the AI will pull back, attack less effectively, and become easier to kill. After winning, a player experiencing this scenario really feels like he or she accomplished something. A caveat to using this technique is that the developer must be very careful that the players are not aware that they are being manipulated in this way. If the trick is perceptible, it unequivocally ruins the game-playing experience.

Intentional Vulnerabilities

Players learn to capitalize on opponent's weaknesses even when they are unintentional. Rather than allowing a player to discover such vulnerabilities, it is often better to design them into an NPC's behavior. For example, a running NPC might have to pause and prepare its weapon, taking longer to attack than a stationary one. An opponent attacked from behind might act surprised and be slow to react. Adding imperfection into NPCs' behavior can also make them seem more realistic and give them more personality. Occasionally, an NPC might fumble when loading a gun. One that knows how to avoid trip mines might occasionally run into one. Planned vulnerabilities make computer-controlled characters seem more realistic. Unintentional mistakes break the realism. Extensive play-testing is required to get the balance right.

Play-Testing

An AI programmer's most important tool is play-testing. Even for a developer with years of experience developing AI, play-testing is the only sure way to determine when one can use cheap solutions and when one must apply more sophisticated artificial intelligence techniques. One can't overemphasize the importance of testing naïve players' reactions to the artificial intelligence. Even expert AI developers are often surprised by the results of play-testing and the interpretation made by players.

The choice of play-testers is crucial. Play-testers should certainly not be members of the game development team and preferably not in the game development industry. Any knowledge of artificial intelligence techniques and tricks will influence the play-tester's interpretation of events. Second, the pool of play-testers must be large. There are two reasons for this. First, as adjustments are made to the AI, a fresh pool of naïve players is required. Play-testers who are used repeatedly will be biased by their previous exposure to the AI. Their interpretation of NPC behaviors and their playing techniques will be different than those of a naïve user. Second, as players vary widely in their skill set, it is important to use a large number of play-testers. Only one player out of 30 might find that fatal flaw in the AI that makes the NPC look dumb or easy to kill. The AI might look flawless when tested with only a handful of players, but get trashed when the game hits the larger market.

Play-testing should be done throughout the game development process, not just at the end of production. Determining what will be fun and challenging is a difficult and time-consuming process. Plan on throwing out ideas and starting from scratch multiple times. Additionally, it is important to allow some of the play-testers to play-test throughout the entire development process. Game players get better with time and are more likely to find loopholes in the AI over time. They are also more likely to see through AI tricks after playing the game for a prolonged period.

Multiple observers should be present at each play-test. During the session, observers should take notes about the in-game actions of the player as well as their physical reactions such as posture and facial expression. Notes should also be taken for questions to ask the play-tester, but questions should only be asked after gameplaying is complete. It is important that observers say nothing during the course of the play-test, even when the player runs into difficulty.

After gameplay is complete, the play-testers should be interviewed for their reaction to the game. What is important is to determine what the players think happened, not what AI was actually implemented. There is a tendency for players to assume that complex behaviors are happening when they are not. Conversely, players also fail to notice complexities in the AI. Ask the play-tester about the actions and intentions of the NPCs. Be careful not to ask leading questions. More sophisticated game engines will include a recording mode where one can play back the entire play-test session to the player and ask questions about his or her actions.

A Few Examples

A common mistake of AI developers is that of over-design. Often, a much simpler solution will suffice, and cheaper, more creative solutions turn out to be considerably better. As a first example, one area in which programmers often over-design is in developing squad tactics. Complex communication and relationships between combat units is often lost on the player and not necessary. Valve's *Half-Life* [Valve98] offers an impressive example of how simple behaviors can produce rich gameplay that capitalizes on players' impulse to assume intelligent behavior.

The marines in *Half-Life* used the "Kung-Fu" style of fighting, meaning that regardless of the number of marines that the player is fighting, only two are actually allowed to shoot at the player at any given time. No actual communication exists between the marines. Instead, each squad of marines is given two attack slots; if a marine wants to attack and both slots are filled, he finds something else to do (such as reloading his weapon or moving to a new attack position). When one of the attacking marines runs out of ammo, he releases the attack slot and goes into his find-cover-and-reload behavior. Then, one of the non-attacking marines, finding an empty slot, grabs the attack slot and starts shooting at the player.

A simple rule was added that whenever an attack slot opens up and there is more than one marine present, the marine vacating the attack slot should yell "Cover Me!"

Although there is no communication between the marines, the perceived effect is one of a marine asking for cover and another opening fire to cover him. In reality, an attack slot opened up and was filled by another marine who knows nothing about the one reloading.

During play-testing, it was discovered that occasionally when a player threw a grenade at a group of NPCs, *Half-Life*'s pathfinding algorithm was unable to find a path for all of the NPCs to escape. The behavior of remaining NPCs looked exceptionally dumb as they shuffled around trying to find a way out. Rather than redesigning the pathfinding system (a huge undertaking), Valve's solution was to detect when the problem occurred and play specialty animations of the trapped marines crouching down and putting their hands over their heads. This was very well received by play-testers, as it added to the character of the game.

Conclusion

This article discussed several important concepts. Developers' efforts in the arena of artificial intelligence are often so focused on making their computer opponents smart that they neglect to adequately address how the AI makes a game fun. The goal of a sophisticated AI is not to kill the player but to add tension, to control the pace of the game, and to add personality to nonplayer characters. Simple solutions are often better and more entertaining than complex artificial intelligence. By adding intentional vulnerabilities to NPCs, we can ensure that players capitalize on planned weaknesses in AI rather than discover unintentional weaknesses.

This article also introduced several tricks for creating AI systems that maximize the entertainment factor. However, it can't be overemphasized that even years of expertise developing AI systems will never be a replacement for extensive play-testing.

References

[Lidén02] Lidén, Lars, "Strategic and Tactical Reasoning with Waypoints," *AI Game Programming Wisdom*, Charles River Media, 2002.

[Madhyastha95] Madhyastha, Tara and Reed, Daniel, "Data Sonification: Do You See What I Hear?" *IEEE Software*, Vol. 12, No. 2, 1995.

[Valve98] Valve LLC, "Half-Life," 1998. See *www.valvesoftware.com*