An Inquiry on Finite State Machines and their use in Videogames

Micah S. Thompson

University of Advancing Technology

GPE230: Gameplay Programming Implementation

Professor Blake Ratliff

March 21, 2024

An Inquiry on Finite State Machines and their use in Videogames

On the rise in the 1980s and now already dominating the entertainment industry, video games are a form of media that is truly revolutionary. Each passing year seems to be a testament to the development of technology, and games are no different. Video games, without any doubt, have evolved in many ways. From graphics to more and more complex mechanics to game artificial intelligence, it is clear that this phenomenon is not dying anytime soon. With the recent rise of more complex neural networks and artificial intelligence algorithms, note that from here on out, the terms "artificial intelligence" and "game intelligence" will refer to the same thing: the perceived intelligence of entities, actors, or controllers in a video game.

Finite state machines (FSMs) are algorithms designed to switch to a fixed value (a state) based on a determining factor, or a set of factors (Astuti et al., 2022). Video games—like any program—are a series of instructions, variables, functions, and algorithms that produce a digital world. Applying this logic to game objects (entities), finite state machines create an illusion of intelligence inside the game world (Feiyu et al., 2013). Note that state machines are not limited to game intelligence. As Astuti et al. (2022) demonstrate, FSMs can apply to story progression. They also have many other uses outside the sphere of game programming.

It can be surmised that even the most complex intelligence algorithms use some form of state machine. Some of these algorithms are created to play a game itself. However, without direct access to the game data, we must find an alternative solution. Perez-Liebana et al. (2016) describe an algorithm capable of capturing the data rendered to the screen and determining which inputs to trigger while playing a game. While not fitting the exact description of a finite state machine, we can hypothesize that some form of state machine, perhaps one more hierarchical or one using machine learning, is or could be used in such an algorithm.

Combining the idea of using an algorithm to play a game and then applying that to an artificial intelligence that must go head-to-head with the player is why finite state machines have struck a chord with game developers. In essence, it is what summarizes just about any game AI. However, a game intelligence that learns from the player is not a new technique. *Tekken 5: Dark Resurrection*, a fighting game, would store the player's moves inside a file (Feiyu et al., 2013). A game intelligence could use a state machine to determine what combat move it must make to counter the player's attack or go on the offensive. Finite state machines are simple yet versatile. This versatility shows how they have established such importance in game programming and have been the de facto solution for game intelligence programming and development.

While playing a game, it might be easy to piece together how an enemy or another non-playable character (NPC) might work under the hood. However, attempting to map this behavior to an FSM diagram might appear daunting or impossible. With a bit of critical thinking, it can be done. For research, we will investigate two games, *Genshin Impact* and *Honkai: Star Rail,* and determine what a hypothetical finite state machine might look like for the AI of the enemies in both games and Star Rail's auto-battle feature. Both games are published by the company HoYoverse. They may appear similar but, both take an approach to their mechanics differently.

Starting with *Genshin Impact*, it is significant to note a few things. First, there are many characters, each with unique elemental and combat abilities. Some characters may deploy an object on the field. This object (or entity) may serve many purposes. It may deal damage over time (DoT), or maybe it will heal the player, or maybe it could be a distraction for the enemy. Looking at Amber,

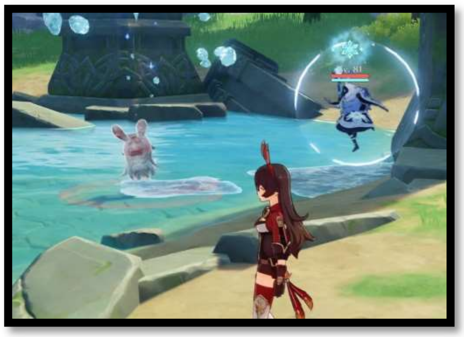

Figure 1. Abyss Mage attacking Amber's Barron Bunny.

one of the first companions you meet in the game, we can see that she will deploy her "Barron Bunny" on the field. This cute little doll will perform a little dance, attracting enemies towards it, allowing our cute bunny-themed archer to snipe them from a distance. Barron bunny will also explode after a few seconds, dealing damage to enemies and setting them ablaze. From this, we can hypothesize that our enemy intelligence has a target variable, allowing them to target the player or another entity.

Apart from Amber's Barron Bunny, enemies will also target Ganyu's (another character) Ice Lotus. However, they do not attack Guoba, Xiangling's companion. Therefore, we can hypothesize that certain entities cannot be targeted by enemies. Not all enemies will go after these entities, but for simplicity's sake, our diagram won't account for those higher-level enemies. When an enemy spots a target or is interrupted by an attack, they will chase after the target. Therefore they must have a chase and attack state. Enemies also are either idle in one location, or are patrolling an area, so they must have two passive states: idle and patrolling. If an enemy no longer sees the target, it will return to its starting position and return to the idle or patrolling state. An enemy will only attack the player if they are close enough. If not, they will return to chasing the player.

Figure 2. Ganyu attacking a Lawachurl from too far away.

To lay it out, the enemy will start in an idle or patrolling state where it will sit still or will patrol an area. If it sees a target or gets attacked, it will enter the chase state and move towards the target's position. If the enemy does not see the target, it will re-enter the idle or patrolling state. Likewise, if the target is within range, the enemy will begin the attack animation. If not in range, the enemy will continue to chase the player.

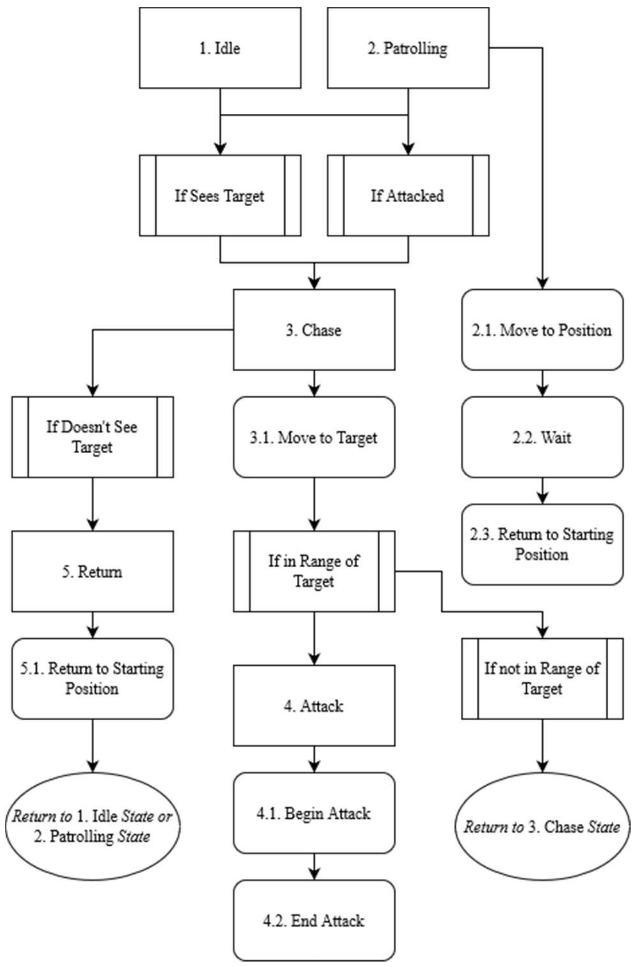

Figure 3. A basic hypothetical FSM Diagram for Genshin Impact's AI.

With *Honkai: Star Rail* being a turn-based game, the AI will function differently. There is a basic form of intelligence for enemies in the open world. Enemies are either set to an idle or patrolling state. If the player enters a specific range, the enemy will notice them. A meter above their head will slowly fill up. Once the player is detected, the enemy will chase the player. Once our enemy has caught up to the player they will attack, and if they hit the player, a combat instance will occur. If an enemy is attacked by the player, the game will also open a combat instance.

In the combat instance, the AI doesn't necessarily need to think too much. Being a turn-based game, the action order determines who gets to attack at a given time. Characters in the game have skill points and energy, this allows the player to pick from a range of attack options during their turn. Enemies generally have more than one attack they can use, so one could hypothesize that they have a similar system to the player. This unknown variable, or set of variables, will be known as *x*. For example, a *Voidranger: Reaver* ("Reaver") will start by attacking a single character. After the first attack, assuming that *x* has charged up, the Reaver will do an Area of Effect (AoE) targeting multiple characters. It is important to know that *x* itself is different for every enemy.

With each enemy having different effects and abilities, we will create a very rough basis for what a potential finite state machine may look like. This diagram will be refined to provide an example of the game's auto-battle feature. Essentially, Figure 6 represents a simple if-else statement. If *x* is equal to zero, we carry out a default action. If not, we will carry out an advanced action.

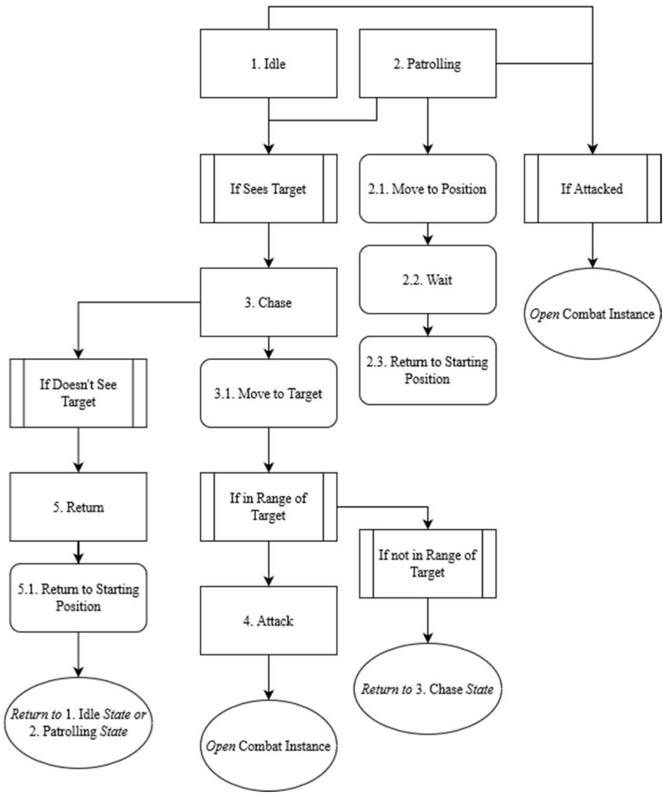

Figure 4. A basic hypothetical FSM Diagram for Honkai: Star Rail's open world AI.

Figure 5. A Voidranger: Reaver attacking both a single character and multiple characters.

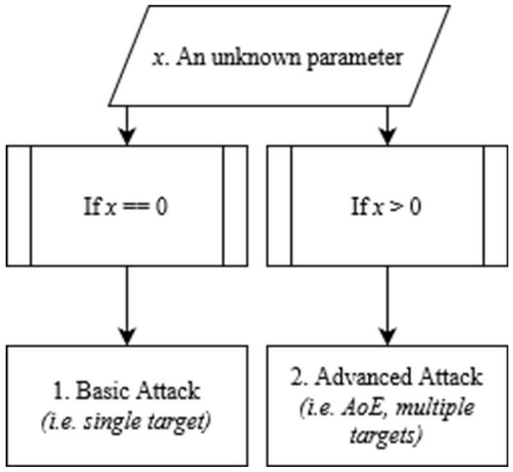

Figure 6. Proposed enemy logic within a combat instance.

Honkai: Star Rail's auto-battle system allows the player to press a button and let a battle play out, without the need to interact with anything. This mechanic is useful for gathering resources and level-up materials, a task that can be very time-consuming. The player has access to character information such as their energy level, health (HP), and other statistics such as attack (ATK), critical attack rate and damage (CRIT Rate and CRIT DMG), and speed (SPD, mainly used by the action order to determine which character or which enemy should act first). As aforementioned, the player also has access to enemy information as well. However, the auto-battle feature does not. This means if an enemy has enough HP left to be killed with a basic attack, the auto-battle will still cast a character's ultimate attack ("ultimate") if they have enough energy.

Two diagrams are proposed. Figure 7 will show a base template on what the auto-battle feature may take into consideration when active. Figure 8 will propose a finite state machine for Fu Xuan, a Preservation unit that can protect and heal the team of characters on the field.

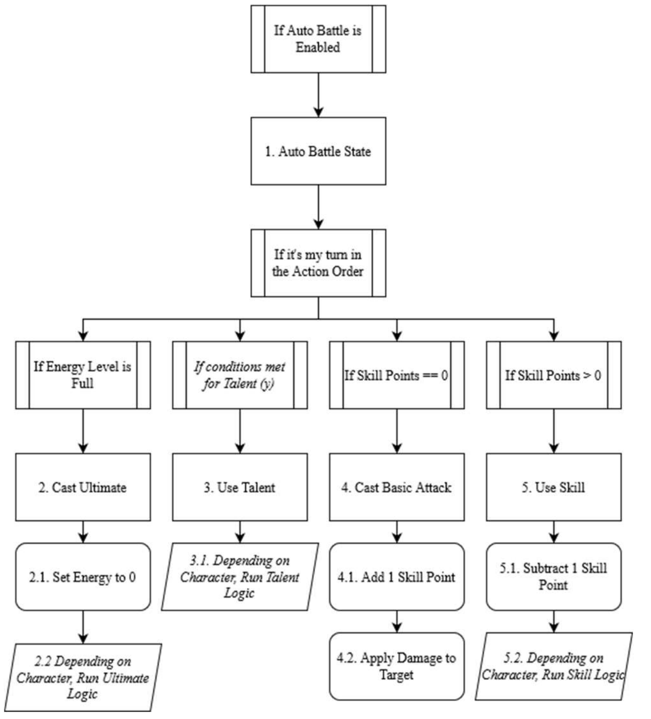
If Auto Battle is Enabled
1. Auto Battle State
If it's my turn in the Action Order
If Energy Level is Full
If conditions met for Talent (y)
If Skill Points == 0
If Skill Points > 0
2. Cast Ultimate
3. Use Talent
4. Cast Basic Attack
5. Use Skill
2.1. Set Energy to 0
3.1. Depending on Character, Run Talent Logic
4.1. Add 1 Skill Point
5.1. Subtract 1 Skill Point
2.2 Depending on Character, Run Ultimate Logic
4.2. Apply Damage to Target
5.2. Depending on Character, Run Skill Logic

Figure 7. Proposed Auto-Battle Feature Diagram.

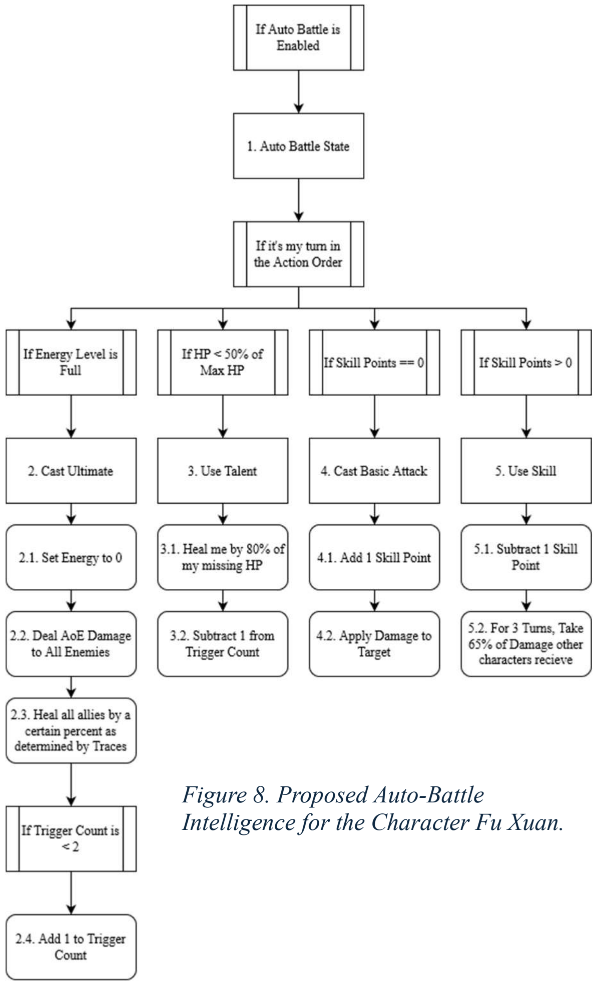
If Auto Battle is Enabled
1. Auto Battle State
If it's my turn in the Action Order
If Energy Level is Full
If HP < 50% of Max HP
If Skill Points == 0
If Skill Points > 0
2. Cast Ultimate
3. Use Talent
4. Cast Basic Attack
5. Use Skill
2.1. Set Energy to 0
3.1. Heal me by 80% of my missing HP
4.1. Add 1 Skill Point
5.1. Subtract 1 Skill Point
2.2. Deal AoE Damage to All Enemies
3.2. Subtract 1 from Trigger Count
4.2. Apply Damage to Target
5.2. For 3 Turns, Take 65% of Damage other characters recieve
2.3. Heal all allies by a certain percent as determined by Traces
If Trigger Count is < 2
2.4. Add 1 to Trigger Count

Figure 8. Proposed Auto-Battle Intelligence for the Character Fu Xuan.

Finite state machines are a highly structured way to map out game intelligence, so it is critical to adhere to strict rules when creating one. Here are three proposed practices to use 1) *avoid redundancy,* 2) *make a base for all NPCs,* and 3) *make detailed documentation.* The practice of avoiding redundancy allows for fewer issues down the line. When writing functions, it is important to avoid repeating the same code. The purpose of functions is to consolidate it. If something needs to be written twice, then collapse it into its own function. Apply this practice to the states in the state machine; if one state is similar to another, consolidate them.

With the many enemies or other non-playable characters a game may have, it is necessary to have a base intelligence class. This class should contain all the functions for these NPCs to function. Sometimes it may be necessary to place a function inside the game intelligence for one specific NPC. But if it is something that can be called by other NPCs, then place it in the base class. Lastly, make detailed documentation. This is not just a courtesy or good practice. It is essential to your future self, other programmers, and even game designers. If something is well documented, it is easy to troubleshoot and tweak.

The use of finite state machines in the games industry proves their robust and critical role in creating a believable game world to immerse players into. They provide a series of checks, balances, and intelligence into the very soul of a game. Finite state machines have proven to be the building blocks of nearly every successful game on today's market, and they will only keep getting more advanced. The use of a finite machine provides nearly infinite possibilities and ways to utilize them in the industry and even outside of the games industry.

References

Astuti, Dwi et al. (2022). *Application of the Finite State Machine Method to Determine the End of the Story Based on User Choice in Multiple Role Playing Games.* International Journal of Computer and Information System (IJCIS)

Feiyu, Lu et al. (2013). *Fighting Game Artificial Intelligence Competition Platform.* Intelligent Computer Entertainment Laboratory, Ritsumeikan University.

Perez-Liebana, Diego et al. (2016). *General Video Game AI: Competition, Challenges, and Opportunities.* Association for the Advancement of Artificial Intelligence

Smolyakov, Ivan Y., & Belyaev, Sergey A. (2019). *Design of the Software Architecture for Starcraft Video Game on the Basis of Finite State Machines.* Saint-Petersburg Electrotechnical University. IEEE.

Honkai: Star Rail Walkthrough Team. (2023). *Fu Xuan Best Builds and Teams: Honkai: Star Rail*. Game8. https://game8.co/games/Honkai-Star-Rail/archives/405760#hl_7