Finite State Machine in Game Development

Article *in* International Journal of Advanced Research in Science Communication and Technology · October 2021

DOI: 10.48175/IJARSCT-2062

Devang Jagdale
Maharashtra Institute of Technology College of Engineering
1 PUBLICATION **23** CITATIONS
SEE PROFILE

IJARSCT

ISSN (Online) 2581-9429

International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)

Impact Factor: 5.731

Volume 10, Issue 1, October 2021

Finite State Machine in Game Development

Devang Jagdale
UG Student, School of Computer Engineering & Technology
MIT World Peace University, Pune, Maharashtra, India

Abstract: *Finite State Machine is one of the oldest techniques in gaming where it was used in old games like PACMAN and new games like TOM RAIDER also. In all these games one major goal was to make non player characters more intelligent. There are some advance types also available but FSM still is one of the most used technique for non-player characters. The main goal of this paper is to explain how FSM works, how to create FSM and implement it in games using scripting or visual scripting. Hierarchical Finite State Machine is also discussed in this paper as it overcomes the limitations of older simple FSM. Using FSM, we can create intelligent AI agents. We can implement FSM and HFSM in games to make NPS's behave like AI.*

Keywords: FSM, HFSM, NPC, States, Transitions, State Machines.

I. Introduction

Gaming industry is one of the fastest growing industry where all leading technologies like AI, ML, Data Science, AR, VR, etc. are becoming a part of this industry. Gaming industry started in 1960 where developers developed simple games. First game was developed at MIT named Spacewar. Later many companies were started where Atari, Nintendo and Sega were first few companies. Games are into existence from a very long time and AI was and is one of the major parts in games. AI is used in games to depict human intelligence in non-playable characters. Human like intelligence in games mainly implemented the ability to interact with environment and perform some actions according to it. The need of AI in games is increasing as games are becoming more real and perform actions and reactions like actual life. Traditionally AI were implemented in games but they were not accurate or did not perform or react to all situations happening in environment but, now if we see modern games, they all feel like they are using AI for every non player character which are interactable. For creating a modern games AI feels like one of the important parts of development.

In initial days of gaming industry there were games like ping pong, snake, etc. These games did not use much environment so, any interaction with it was not even considered. As these were simple games with very less interactable environment there was no need for AI. But as games started to grow bigger and bigger need for environment and non-player character (NPC) started growing rapidly. NPC were added in games to make interaction with environment more fun and realistic. Basically, NPC is character which are not controlled by player but, is used to interact, guide or assist player in games. As games became more realistic, making NPC perform work like AI and behave realistically became primary goal for developers. In today's games most common role of AI is controlling NPC. Playable character interacts with NPC which makes our game feel like real. If NPC are lagging, it puts a game at a disadvantage.

One of the effective methods to control NPC are Finite State Machine. FSM has been there in gaming industry for a long time and is still being used in today's modern games by many developers. Creating AI agents using FSM which act like Simple reflex agent makes game more interactive and fun to play. FSM is a simple state machines where different states are connected to each other by some conditions. If certain conditions are met then we transition from one state to another state. Inside state we can perform some actions or implement algorithms. At all-time our NPC should be inside a state. Games like PACMAN or Half Life implemented FSM as AI agents in their games as enemies. Many games Battle Field, Call of Duty and Tomb Raider have successfully incorporated the FSM AI design. If we try to capture maximum range of behaviours that our NPC can encounter, we can create a successful FSM AI design.

FSM is used in many games from long time but as complexity of FSM was low people didn't think of it as AI. But as complexity grew developers started to consider more states(possibilities) that might occur and added various algorithms, which made FSM look like AI. Algorithms like path finding algorithms, etc. can make FSM perform as better AI. FSM is easy to implement and without adding any complex algorithms a simple FSM can still mimic an AI partially. FSM is similar as simple reflex agents. There have been some improvements in traditional FSM which can make complex design a bit simple. Many researchers have been made in the field of game development and AI, and merging these two fields to make an advancement in gaming/simulation industry was a major motivation to do research in this field. Soon gaming industry is going to become a part of something big. Application of AI in game has done a lot of good to society. Some applications are, like now student who are going to become doctor can practice surgery using game simulation rather than actual person, simulating test environments for various industries and researches which are costly to perform in real life can also be done using AI in games.

As FSM states grows, its complexity also grows. It might become difficult to implement and understand the code. There are some advance FSM available which might reduce FSM complexity and increase its performance. One of the solutions to our problem is using Hierarchical FSM where we write FSM inside FSM. Outer FSM states have smaller FSM inside them. HFSM are easy to understand and implement. This research study focuses majorly on how Finite State Machine is implemented in Game Development.

II. Literature Review

In games AI can be implemented by various methods. AI plays different roles in game but most of it is identified as sensing information from environment(input) and taking necessary actions based on goals. These steps can be summarized as sense, think and act. In games FSM as AI is implemented for NPC's. NPC may be friend, foe or just a guide to assist us in games. FSM can be paired with various AI algorithms to make it smarter and less predictable. For example, many developers pair FSM with A* algorithm for making path finding NPC. There are many pre built tools available which make our task as developer easy. Some of them are Unity, Unreal engine, Gadot, etc. These tools use C# or C++ for development but, other languages can be also used like python. Some IDE have visual scripting where we can use FSM instead of coding.

Usually, FSM is used for NPC but there are some other options available too. Machine Learning is one way to build NPC. ML algorithms can learn from their past plays and become better in each passing game. These algorithms can get best at it very quickly. For example, a simple game like snake or ghosts require just less than 200 moves to achieve less than 30% error. This method is very good than FSM but problem is it has so less error percentage that it is almost perfect and playing against a perfect NPC might feel difficult and unfair. Consider a situation where ML algorithms are applied to enemy NPC, here player will face extreme difficulty while playing against it. Another problem is when there is open world game or there is a large environment to interact for our NPC it might take a lot of time to train our models and computation power will also increase. So, implementing this method might not be feasible everywhere. Instead, we can use FSM and if needed we can implement Component Based Hierarchical State Machine where we can implement a complex structure for our NPC with finite set of rules. It works in situations, where our game world has lots of parameter and we want to use only certain parameter for our NPC's.

III. Proposed System

3.1 Finite State Machine

A. System Architecture

To use FSM as AI in games first we need to understand what is FSM and how to create it. FSM is a state machine where we have finite states implemented. FSM is usually designed using graph where node(vertex) is represented as states and edges are represented as transition lines. In FSM state can represent a action or state for our NPC. All states are connected to at least one other state so that it can be reached by some state. Now to move from one state to another we need a connection and it is known as transition. Transition from one state to other happens if a certain condition(s) is/are met. Finally, while building FSM, we should see to that we start from any one state and no state should be unreachable. Game loop keeps FSM updating by checking if transition condition is met or not. Below is a sample structure for how an FSM is designed.

To explain system architecture, we will consider an example where we implement our FSM to enemy NPC. In this example initially enemy is idle and is roaming in free area. Once player is in range of enemy it starts chasing our player and if player is close enough it will also start attacking. While attacking it will also keep a check on its health and if health is too low it will escape from fight and find a potion to heal itself. This loop will keep on going till our NPC health reaches to zero.

Now to build an FSM for above example we need to decide what will be our state and what condition will cause transition from one state to other. In our case Initially NPC is idle so, our first state will be idle where NPC is doing nothing or will perform an idle animation. Our transition condition will be when player in range we chase player. Next state will be Attack state where NPC will attack player. Transition condition will be if player is attacking then move to Defend state. In defend state there will be two transition condition. If health is low for NPC, then escape from fight and transition to escape state and if health is not low then if player is not attacking then move back to attack state or else defend. In escape state NPC will search for health and if health is found then transition to Idle state. FSM for above explanation will look like below figure.

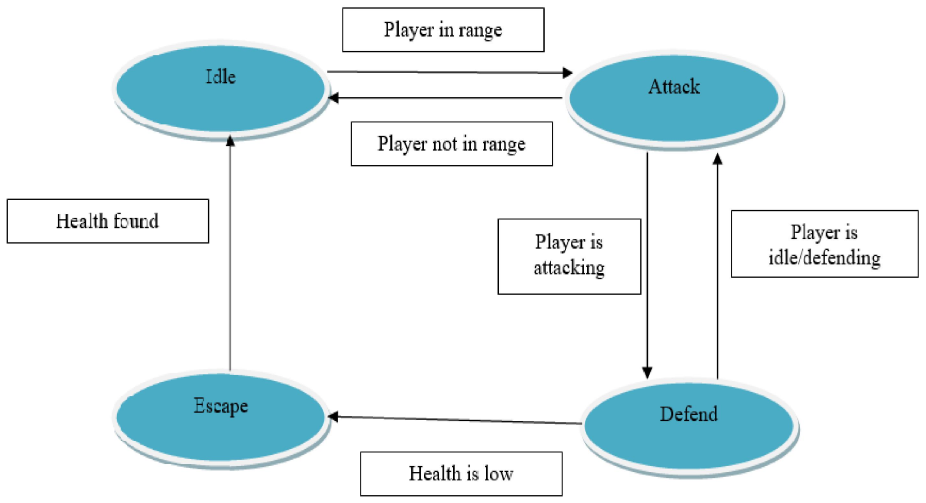

Figure 1: Simple FSM

Above diagram can be also written in state transition table. The table has three columns where first is current state, second is action/event and third is resulting state. Below table represents how we can convert our FSM design to a state transition table.

Table 1: State Transition table

Current State	Action/Event	Resulting State
Idle	Player in range	Attack
Attack	Player not in range	Idle
Attack	Player attacking	Defend
Defend	Player is Idle/Defend	Attack
Defend	NPC health is low	Escape

B. Implementation

FSM can be implemented using simple class and functions. Whole FSM can encapsulate in one class where our FSM states can act as methods or functions, we can use variables to store state data and if else can act as transition and check if right condition are met using variable to transit from one state to other. Below is an example class for FSM

```
public class FSM {
        private var activeState ;
        public void FSM() {
```

IJARSCT
ISSN (Online) 2581-9429
International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)
Volume 10, Issue 1, October 2021
Impact Factor: 5.731

```
                }
                public void setState(state) {
                        activeState = state;
                }
                public void update(){
                        if (activeState != null) {
                                activeState();
                        }
                }
        }
```

In above example we implemented a class named FSM which represents a whole state machine. Inside it we have FSM as empty constructor which can be used as needed. Next, we implemented a private variable named activeState which stores the current active state. It can be accessed only by states inside out FSM class. We created a function named setState which is used to represent a state in FSM. Inside setState we can perform some action and at last we can set a new activeState variable to another state. At last, we implemented a function update. It is function in game which is called for every frame. In each iteration we check if transition is possible and if it is possible, we set active state as new state stored inside variable activeState. In this way we can implement FSM as scripting in games.

3.2 Hierarchical FSM

A. System Architecture

Till now FSM seems to be useful and almost perfect for NPC agents but there are some flaws which make FSM a bit complex. In actual games NPC's need to consider a lot more states than our above example. It reacts to many players response which makes it a better agent. So, as complexity grows number of states start to increase rapidly which makes it difficult to manage. If we want to make even a small change, we need to consider all of its dependent transitions. To solve this problem, we use Hierarchical State Machine.

HFSM involves hierarchy where FSM are nested in another FSM. By grouping states together with the same outgoing transitions, it's possible to cut down on the number of transitions. This is the initial interpretation of a hierarchy, on the state level. Nested states will accept only a subset of the input/output alphabet. If we want to make changes we only have to edit in a particular level of hierarchy and below it. This makes it easy to make changes and understand code and reduces complexity. Below figure is an example of how a HFSM design structure might look like.

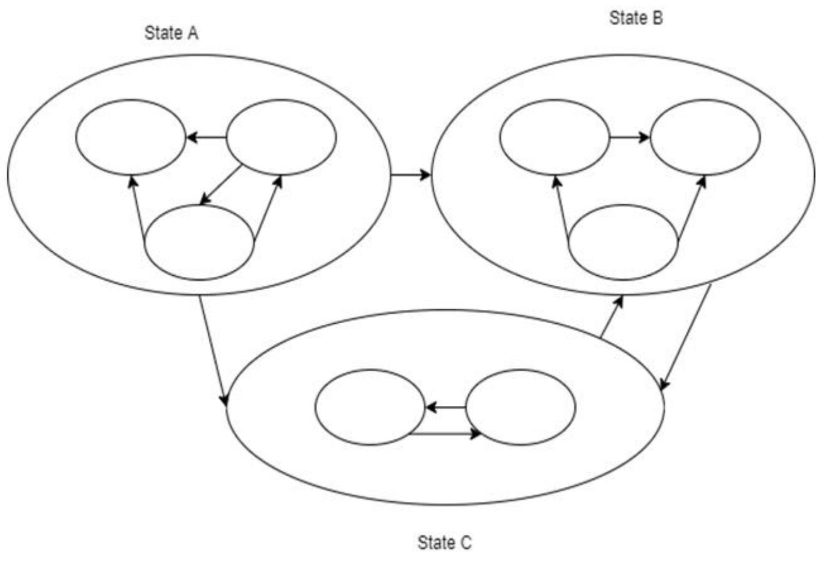

Figure 2: HFSM representation

For above diagram we can see nested FSM inside each other we combine and form one big state. We can zoom in if needed to look whole but if not needed we can zoom out and only see high level FSM. In high level we will only see state A, B and C and nested structure is hidden for simplicity. To understand HFSM better we will consider the same example considered in FSM and build a state diagram for it. For above example we can group Idle and Escape state as they can be considered as Survival task and Attack and defend state can be considered as Fight task.

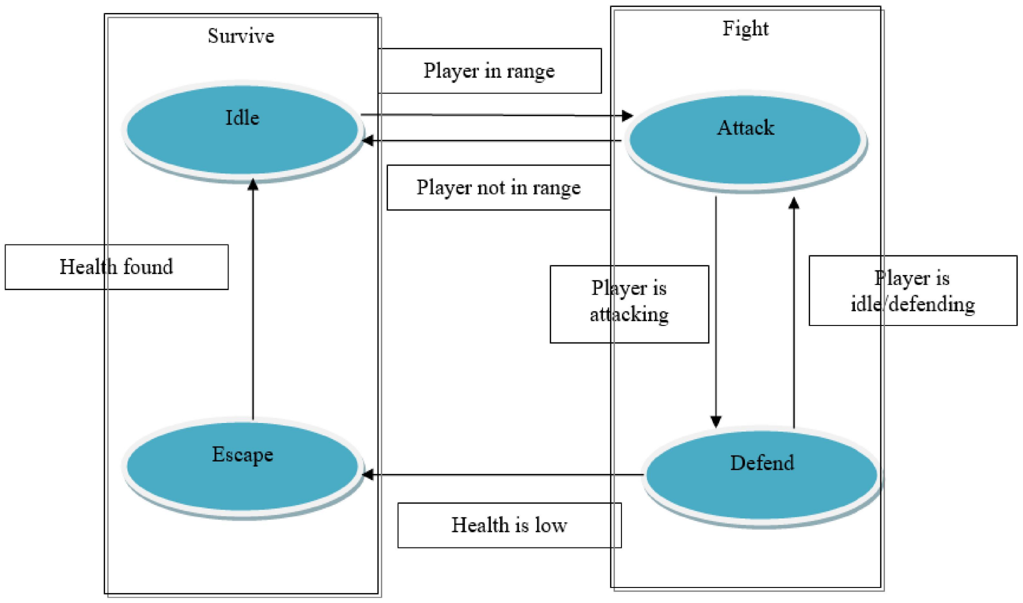

Figure 3: HFSM

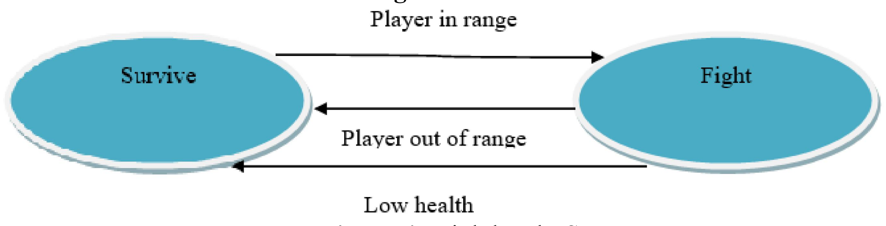

Figure 4: High level FSM

In above figure 4 we can see that by using HFSM we can reduce complexity and have much better understanding just by looking at it. If we want to make changes in any state, we only have to consider the state we want to change and its nested state. For above HFSM state transition table will remain same as we have only grouped state and not added any new state or transitions.

B. Implementation

To implement HFSM we will use Figure 3 as an example. First, we create class FSM which will be used to manage our nested FSM structure. Inside this class we have two main methods. First is used to set new state which takes new state as input and sets current state to new state. Second method is update which calls updates to new state for each frame.

```
public class FSM {
        private var currentState ;
        public void FSM() {
        }
        public void setState(newState) {
                currentState.Exit(); //Exits from old state
                currentState = newState;
                currentState().Enter();
            }
            public void update(){
                if (currentState != null) {
                    currentState().UpdateLogic();
                }
            }
    }
```

Now we have FSM manager we create a BaseState class which will be inherited by all other state classes. This class has three main methods. These three methods are used to control state behavior. Enter method mainly is used to set some initial variables , UpdateLogic mainly performs all the logic that the state will perform it also sets the new state and Exit is used if there is any exit animation or variables to be performed. Now if some conditions are met we can implement transitions from one state class to other inside updateLogic or Exit methods.

```
    public class BaseState{
        protected FSM fsm;
        public void BaseState(stateMachine){
            this.stateMachine = stateMachine;
        }
        public virtual void Enter() {}
        public virtual void UpdateLogic() {}
        public virtual void Exit() {}
    }
```

Using above class as template for figure 3 state A,B and C will inherit from BaseClass and create their own Enter, UpdateLogic and Exit methods. Similarly the states inside state A,B and C will inherit from A,B and C.

```
    public class A: BaseState {}
    public class B: BaseState {}
    public class C: BaseState {}

    public class NestedFSMInsideA: A{}
    public class NestedFSMInsideB: B{}
    public class NestedFSMInsideC: C{}
```

In this way we can create a HFSM using scripting.

To make FSM smarter we can add algorithms inside state. One of the most common used algorithms in games is A* algorithm. It is a pathfinding algorithm which finds the shortest path from source to destination. Implementing such algorithms makes our FSM look like AI trained agent. For example, in above example our NPC is in idle state until player is out of range. Once player is in range it transitions from idle to attack state. Consider a situation where if players health is low, he decides to escape out of fight and runs then, we can use A* algorithm and make our NPC chase player. Even when NPC's health is low it can escape and use A* algorithm to search for health revive. In such way we can implement various algorithms inside FSM to make it smarter.

IV. Conclusion

In this paper we explored about FSM and how it can be implemented. We saw how to create FSM state machine diagram and state transition table. We also saw how to create FSM using scripting and created a pseudo-code for it. There were some limitations for FSM so we moved to HFSM which is better than FSM and overcomes its limitation. We used some examples and created state machine diagram for HFSM. We can use FSM or HFSM in games where there are many possibilities that an NPC might have but doesn't need. We can use ML and AI deep learning techniques also but it might make NPC too perfect for players and might feel unfair. Also training and implementing thousands of

NPC with ML might put unnecessary load on systems. Where we need an NPC which only performs few tasks and doesn't need unique interaction each time then FSM might be the right choice.

REFERENCES

[1]. Mela Karim Barznjiێمنەلاکەریمبەرزەنجـــــــیە, Hemn. (2019). Artificial Intelligence and Game Development.

[2]. Syahputra, Mohammad &Arippa, A &Rahmat, Romi&Andayani, Ulfi. (2019). Historical Theme Game Using Finite State Machine for Actor Behaviour. Journal of Physics: Conference Series. 1235. 012122. 10.1088/1742-6596/1235/1/012122.

[3]. Adegun, Adekanmi&Ogundokun, Roseline &Ogbonyomi, Samuel & Sadiku, Peter. (2020). Design and Implementation of an Intelligent Gaming Agent Using A* Algorithm and Finite State Machines. International Journal of Engineering Research and Technology. 13. 191. 10.37624/IJERT/13.2.2020.191-206.

[4]. Aiolli F., Palazzi C.E. (2008) Enhancing Artificial Intelligence in Games by Learning the Opponent's Playing Style. In: Ciancarini P., Nakatsu R., Rauterberg M., Roccetti M. (eds) New Frontiers for Entertainment Computing. ECS 2008. IFIP International Federation for Information Processing, vol 279. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09701-5_1

[5]. W. Hu, Q. Zhang and Y. Mao, "Component-based hierarchical state machine — A reusable and flexible game AI technology," 2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference, 2011, pp. 319-324, doi: 10.1109/ITAIC.2011.6030340.

[6]. Solihin, Ade, Eka Wahyu Hidayat and Aldy Putra Aldya. "Application of the Finite State Machine Algorithm on 2D Platformer Rabbit Games vs Zombies." Jurnal Online Informatika (2019): n. pag.

[7]. A F Pukeng et al 2019 J. Phys.: Conf. Ser. 1341 042006

[8]. K Fathoni et al 2020 J. Phys.: Conf. Ser. 1577 012018

[9]. Reza Andrea, SeftyWijayanti, Nursobah, " Finite State Machine Model in Jungle Adventure Game an Introduction to Survival Skills", International Journal of Information Engineering and Electronic Business(IJIEEB), Vol.13, No.4, pp. 55-61, 2021. DOI: 10.5815/ijieeb.2021.04.05