[[1]](#endnote-2)

Implementation of a Non-Deterministic Fixed State Machine Using Assembly to Control a Multi-tiered Game.(December 2008)

David B. Moss  
[albatross4242@yahoo.com](mailto:albatross4242@yahoo.com)

*Abstract*— Discussion of the game flow of the military game, with a look at finite state machine types, implementation and the use of assembly to speed up processing speed.

# INTRODUCTION

I

N this paper I will be discussing the basic game flow of the yet unnamed military game that has been designed in SG410. More specifically I will be discussing how the game transitions between the different levels of game play and how this can be handled using a finite state machine by using assembly programming as well as why this should be done.

# Game Flow

## Game concept and Flow

### Basic Rule Set

First and foremost, this game is a game of objectives and achieving the objectives set forth. Upon completion of an objective, new objectives are given. The more objectives a player completes and how efficiently the objectives are completed are stored in a weighted system that determines advancement through ranks. At any time, a player of a higher rank can temporarily demote themselves in order to play a lower ranked mode.

### First Person Shooter - Grunt

The player begins as a foot soldier or grunt in a first person shooter environment with objectives such as “defend a location” or “assault a location”. As the player progresses through the game, they are capable of rising in rank such that they are given the option of becoming a squad leader.

### First Person Shooter – Squad Leader

A squad leader is given very similar objectives to a grunt; only in this game mode they can command up to 10 grunts to organize a more strategic accomplishment of objectives. When a player progresses far enough in the squad leader game, they can then advance to Captain.

### Real Time Strategy – Captain

At this point in the game, the player’s interface switches from a first person shooter, to a real time strategy game. The player’s objectives may now be something along the lines of “capture enemy command centers” or “destroy all structures at this location.” The player is now shown a bird’s eye view of the battlefield and can command squads of units to do specific actions, which is then filtered down to the Squad leaders and so on. When a player is ready for advancement, they can then advance to General.

### Risk – General

The interface changes again, this time to a more abstracted view of territories. The objectives given in this mode will be more generalized such that they may be tasked with “eliminative hostilities in a territory” or “secure natural resources from a territory.” In this mode, the player must move armies around and give them more specific tasks, which then get filtered down to the real time strategy players. From here a player can eventually advance to the final phase of the game, President.

### Space Command – President

The space command game is a game of defending a country from alien invasions. Resources to do this are obtained via the other game modes. The more efficient the other game modes are, the more resources the president has at his disposal.

## Potential Problems

With five different game modes and vast amounts of data related to the player, the game must have some method of managing when a player can advance, as well as which mode they are currently playing. If the information was processed linearly at every pass through the game loop, many processing cycles would be wasted as the game attempted to figure out what to do for game modes that are not being used by the player. A solution to this problem is by using a finite state machine.

# Finite State machines

## What is a finite state machine?

“A model of a computational system, consisting of a set of states (including a start state), an alphabet of symbols that serves as a set of possible inputs to the machine, and a transition function that maps”

## Why should a finite state machine be used?

Due to the nature of finite state machines, there are pre-defined states that the machine can be in. This will allow us to design a machine that will only use the information needed at the specific game mode, handle the gating from a game mode to a higher mode, as well as handle the transitioning from a higher mode back to a lower mode. Overall this will allow for a vastly improved efficiency in processing, as well as a very simplified way of representing the game for the programmers and artists.

# Types of finite state machines

## Deterministic Finite State Machines

A deterministic finite state machine is a machine in which each state has only one possible state in which it can transition into. For example let us imagine a light switch. A light switch has only two states, on and off. From the “on” state, the switch can only transition to the “off” state, and vice versa (Figure 1). While a deterministic finite state machine can have many more states, they are very linear and offer no flexibility.

Light Switch Finite State Machine  
Figure

## Non-Deterministic Finite State Machines

A non-deterministic finite state machine is a machine in which each state has multiple possible states in which it can transition to. With a non-deterministic finite state machine, based upon inputs, the desired state is reached. A simple example would be eating a value meal from a fast food restaurant (Figure 2). You have the basic state of not eating or drinking, you have the state of eating fries, eating the sandwich and drinking a soda. Additionally you have the states of chewing and swallowing. The final state of being done eating can be meat at any time, even if you are not out of food. Because there are several states in which the next state is not always the same, this is called a non-deterministic finite state machine.

Eating a Value Meal Finite State Machine   
Figure

# Implementation of the finite state machine

In a game such as this, there will be many different finite state machines working together. Each level of game play will be its own collection of finite state machines. In each game mode, a statistic is tracked that determines if the player is able to advance to the next game mode or not. For this example we will call this *experience*. Each mode has the state change option of gaining experience. Once the experience is gained, the game changes states back into the game mode that is currently being played. Once the required amount of experience is gained to allow the player to advance, the player is given the option of changing game modes to the next higher game mode. At any point the player is always given the option of moving back down to the game mode one rank lower than where they are currently. (Figure 3)

With this simple finite state machine, one can envision the entire game play experience in a very simplified form. While each of the game modes will have to have its own finite state machine available, this clearly conveys how the main backbone of the system will function.



Game States  
Figure

With this finite state machine, the game’s progression is very apparent. This allows the programmer to write a simple algorithm based off of the finite state machine.

# Assembly versus High Level Languages

When it comes to actually coding the finite state machine, typically a programmer will use whatever high level programming language they are using. This in turn leads to case statements or lengthy if 🡪 then statements. Such that if we wanted to do a check of the current state and then call the appropriate function for that state in c++, the code may look like this:

switch(state)

{

case 1: // grunt fps

callGruntFunction();

break;

case 2: // squad leader fps

callSquadLeaderFunction();

break;

case 3: // captain rts

callCaptainFunction();

break;

case 4: // general risk

callGeneralFunction();

break;

case 5: // president space

callPresidentFunction();

break;

default:

setState1(); // set state to 1

break;

}

If this were to be written as assembly code, it would look like this:

cmp ebx, 1

je STATE1

cmp ebx, 2

je STATE2

cmp ebx, 3

je STATE3

cmp ebx, 4

je STATE4

cmp ebx, 5

je STATE5

jmp setstate1

While this is as easy to read, the code would process far quicker and take up less memory.

# Conclusion

Non-Deterministic Finite state machines can be used to implement game states such that processes are simplified and made clear to all parties involved in the project. Additionally by selectively implementing assembly code into a high level programming language, specific portions of code can be simplified and sped up considerably, making a program run more efficiently. If this process is propagated throughout a programming project, the entire project will be less difficult to plan out, implement and more efficient from a processing consideration.

# Bibliography

1. **The American Heritage Science Dictionary.** finite state machine definition. *Dictionary.com an Ask.com Service.* [Online] Houghton Mifflin Company, 2002. [Cited: December 5, 2008.] http://dictionary.reference.com/browse/finite%20state%20machine.

2. **Brownlee, Jason.** Finite State Machines. *AI depot.* [Online] [Cited: December 5, 2008.] http://ai-depot.com/FiniteStateMachines/FSM-Background.html.

3. **Wikipedia.** Nondeterministic finite state machine. *WIkipedia.org.* [Online] 2008. [Cited: December 5, 2008.] http://en.wikipedia.org/wiki/Nondeterministic\_finite\_state\_machine.

4. **Microsoft.** Finite State Machines, Wizards, and the Web. *MSDN.* [Online] February 2004. [Cited: December 5, 2008.] http://msdn.microsoft.com/en-us/library/aa478972.aspx.

5. **Gill, Sunbir.** Visual Finite State Machine AI Systems. *GAMASUTRA.* [Online] November 18, 2004. [Cited: December 5, 2008.] http://www.gamasutra.com/features/20041118/gill\_01.shtml.

6. **Koehler, Kenneth R.** Finite State Machines. *Elementary Computer Mathematics.* [Online] 2002. [Cited: December 5, 2008.] http://www.rwc.uc.edu/koehler/comath/34.html.

1. [↑](#endnote-ref-2)