INB383 AI for Games
Assignment 1 - Reactive Agents

Prepared for: Dr Rune Rasmussen
Prepared by: Thanat Chokwijitkul n9234900

Queensland University of Technology (QUT)
22 April 2016

TABLE OF CONTENTS

STATEMENT OF COMPLETENESS

This statement is to confirm that the assignment 1, an exploration of reactive agents using finite state machine (FSM), has been completed with all the components required in the assignment specification as follow:

- **Player**: The first person shooter (FPS) is this game is represented by the aspect of the main camera.
- **Non-player Characters (NPC)**: This game includes 4 NPCs represented by metallic spheres. Each NPC is designed and implemented based on the deterministic finite automata (DFA) with Moore machine policies and has at least 4 states.
- **Triggers and Place Markers**: This game includes 15 triggers in total. Each trigger is represented by a transparent cube. Each place marker in this game is represented by invisible terrain.
- **Navigation Structure**: Grid system is used in the pathfinding algorithm.
- **Pathfinding Algorithm**: The A* search algorithm is the main pathfinding algorithm applied to some of the NPCs in this game.

PATH FINDING

This game utilises the concept of the A* search algorithm as the primary pathfinding algorithm for some of the NPCs. The A* pathfinding algorithm is considered fairly flexible in a wide range of contexts in terms of usage. The algorithm applies the concept of breadth-first-search so that it can use a heuristic to guide itself to the desired destination (Reddy, 2013). Regarding the A* search algorithm with its standard terminology, the exact cost of the path from the starting point to the vertex n is represented by the function g(n), and the heuristic estimated cost from the vertex n to the final destination is represented by the function h(n). The algorithm evaluates the vertex n with the lowest f(n) = g(n) + h(n) each time through the main loop and then returns the shortest path as a result at the end of the iteration (Patel, 2016).

The reason that the A* pathfinding algorithm is more efficient when comparing with the greedy search algorithm is that the algorithm itself is guaranteed to be admissible, meaning it always returns a minimal path if such path exists. On the other hand, in the case of greedy search problems, it is possible that the function may return more expensive path when the heuristic function is inaccurate (Rasmussen, 2016).

According to the assignment specification, this game only requires a pathfinding algorithm to be applied to one of the NPCs. However, three of the four NPCs in this game utilise the A* pathfinding algorithm for the ease of navigation.

When each NPC is patrolling in its own area, instead of moving between static points using place markers, it uses the pathfinding algorithm in finding a path to the next random destination. In addition, due to the fact that each agent does not have any base knowledge about its own area, plus each patrolling position is randomly generated, the algorithm plays the main part in finding the most optimal path along with avoiding the obstacles located in the area, which is very crucial because the NPC agents should not be able to pass through any game objects, especially the labyrinth walls and obstacles.

FINITE STATE MACHINE

This assignment is a prototype of the First Person Shooting (FPS) game and each Non-Player Character (NPC) is a reactive agent based on Deterministic Finite Automata (DFA), which is a Finite State Machine (FSM) that accepts a finite set of input symbols and produces a unique output according to each symbol. As a result of using FSM, each NPC agent is able to perform a task according to its current state, such as patrolling, hiding, chasing, attacking, fleeing .etc. In order to differentiate each NPC in this game, the agents are classified by its unique characteristics and behaviours. The following figure illustrates the locations of the triggers used in the game:

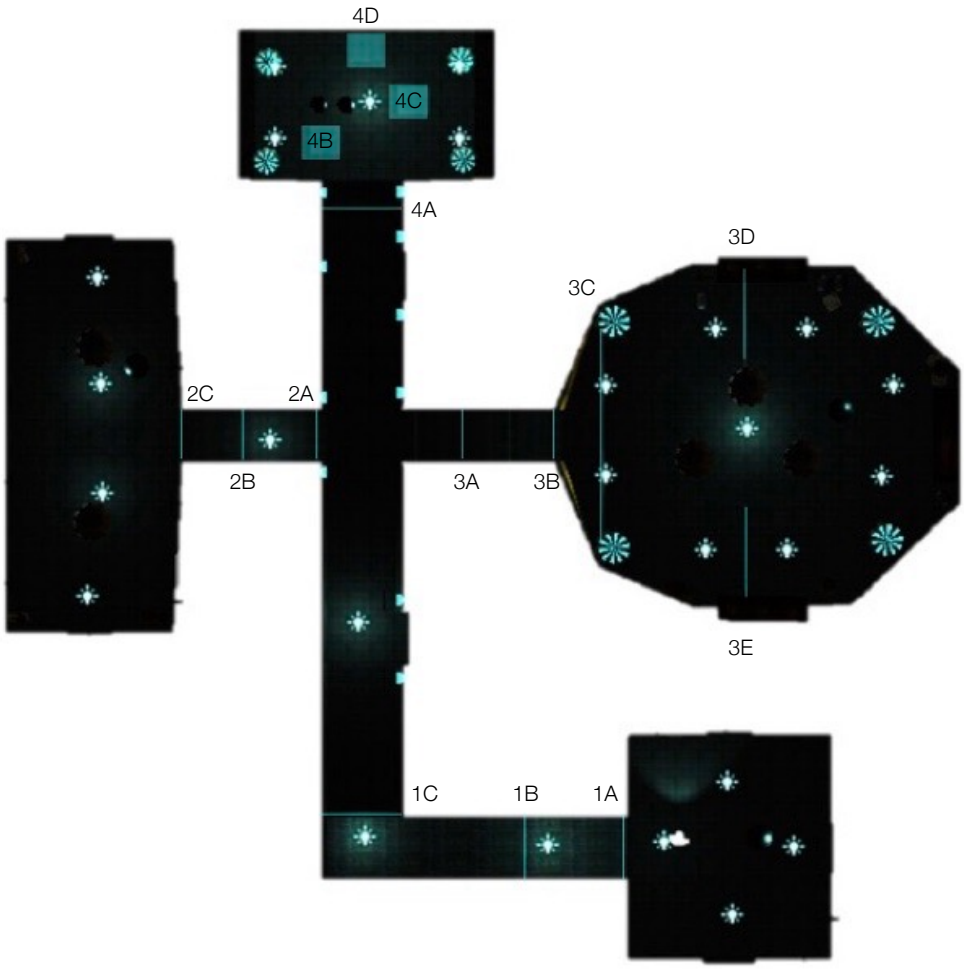

Figure 1: Trigger locations within the

For the ease of management, each trigger is labeled by a room number following by an input symbol. These labels can be used as references when evaluating each NPC's state machine policy. The next section delineates the details of each NPC agent, including the FSM used in defining its policy:

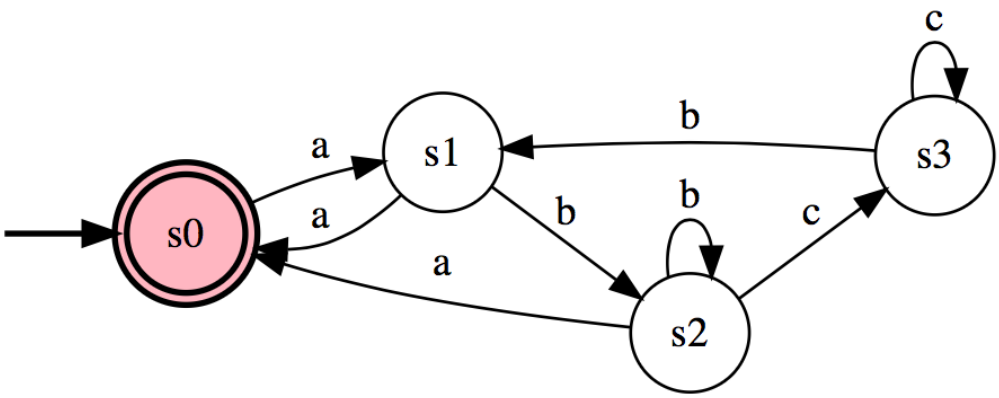

Figure 2: The guard agent's FSM policy

Agent 1: Guard Agent

The FSM of this agent has 4 states, including patrolling, being alert, chasing and fleeing. This agent is the first NPC that the player sees at the start of the game. The following table illustrates the NPC policy of the guard agent:

State	Accept State	Action	a	b	c
S0	1	0	1	-1	-1
S1	0	1	0	2	-1
S2	0	2	0	2	3
S3	0	3	-1	1	3

States and Descriptions

S0: The state when the NPC is patrolling in the specified area of the first room. This agent uses the pathfinding algorithm to find a path to the next random destination without avoiding any obstacles within that room.

S1: The state when the player leaves the room, resulting in the NPC being alert and jumping in the same spot.

S2: The state when the NPC starts chasing the player.

S3: The state when the NPC flees from the player and goes back to the starting room.

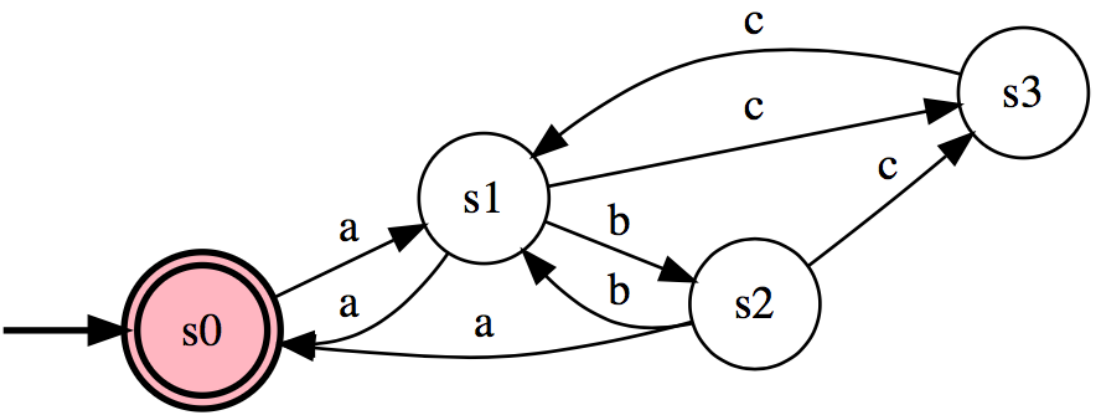

Figure 3: The frightened agent's FSM

Agent 2: Frightened Agent

The FSM of this agent has 4 states, including idling, patrolling, observing (looking) and hiding. This agent will be aware when the player's position is near the room. When the player is in the room, it will be in the "panic" state and try to find an obstacle to hide from the player's sight. If a proper hiding obstacle cannot be found, it will keep running around until the player leaves the room. The following table illustrates the NPC policy of the frightened agent:

State	Accept State	Action	a	b	c
S0	1	0	1	-1	-1
S1	0	1	0	2	3
S2	0	2	0	1	3
S3	0	3	-1	-1	1

States and Descriptions

S0: The NPC is in the idle state, traversing through the remaining path and waiting for another event to be triggered.

S1: The state when the NPC is patrolling in the specified area of the room. This agent uses the pathfinding algorithm to find a path to the next random destination along with avoiding any obstacles within that room.

S2: The state when the NPC moves to the position near the door, observing at the player.

S3: The state when the NPC tries to find an available obstacle to hide from the player's sight. If such hiding obstacle cannot be found, it will be panicking and running around the room.

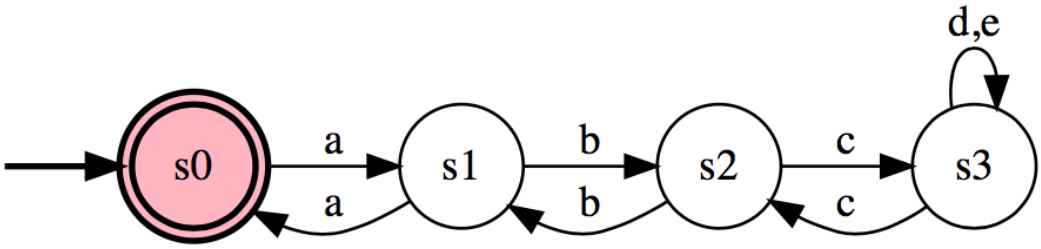

Figure 4: The strategic agent's FSM policy

Agent 3: Strategic Agent

The FSM of the agent is relatively straightforward compared to both of the previous FSMs. The state machine has 4 states, including idling, patrolling, hiding and attacking. This agent is in the room full of obstacles where it will keep chasing and attacking while the player is staying in that room. The following table illustrates the NPC policy of the strategic agent:

State	Accept State	Action	a	b	c	d	e
S0	1	0	1	-1	-1	-1	-1
S1	0	1	0	2	-1	-1	-1
S2	0	2	-1	1	3	-1	-1
S3	0	3	-1	-1	2	3	3

States and Descriptions

S0: The NPC is in the idle state, traversing through the remaining path and waiting for another event to be triggered.

S1: The state when the NPC is patrolling in the specified area of the room. This agent uses the pathfinding algorithm to find a path to the next random destination along with avoiding any obstacles within that room.

S2: The state when the NPC is hiding behind one of the obstacles in the room.

S3: The state when the NPC keeps chasing and attacking while the player is staying in the room.

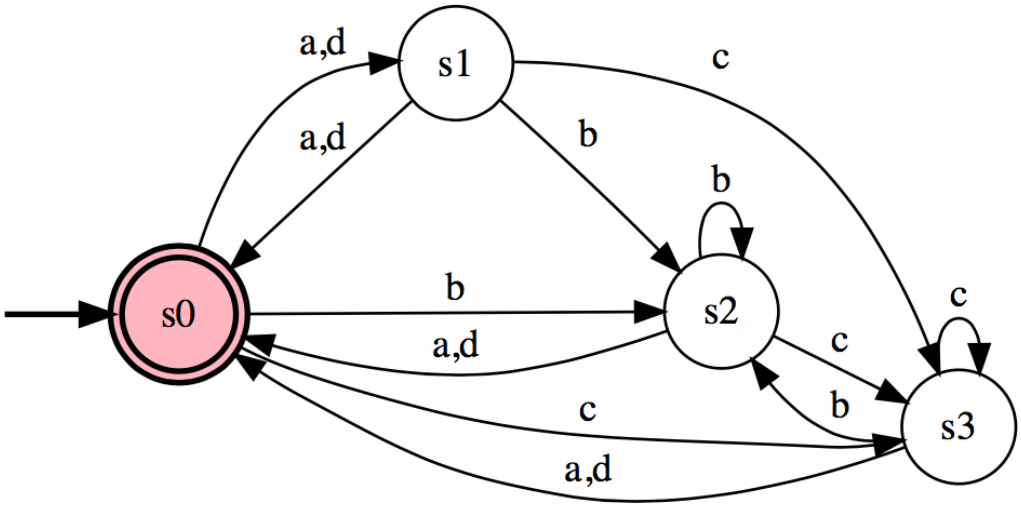

Figure 5: The entity agent's FSM policy

Agent 4: Entity Agent

The FSM of this agent has 4 states, including terminating, starting, chasing, fleeing. The entity agent is represented by a pair of metallic spheres bouncing against the walls within the specified area. This is the only agent with the pathfinding algorithm applied to it. Therefore, each of its action will be solely based on the triggers located within the specified area. The following table illustrates the NPC policy of the entity agent:

State	Accept State	Action	a	b	c	d
S0	1	0	1	2	3	1
S1	0	1	0	2	3	0
S2	0	2	0	2	3	0
S3	0	3	0	2	3	0

States and Descriptions

S0: The state when the NPC is terminated. The movement of the entity agent will be instantly frozen.

S1: The state when the NPC start moving within the specified area.

S2: The state when the entity agent tries to flee away from the player.

S3: The state when the entity agent is bouncing against the player.

CONCLUSION

For this game prototype, all the NPC agents have been designed and implemented using the state machine design pattern based on the concept of Deterministic Finite Automata (DFA) with Moore machine policies. Consequently, if given an input symbol and a machine state, the output or the state transition can be predicted based solely on the input and its machine policy.

The development process of the game prototype can gain various kinds of benefits by applying the concept of DFA. Its simplicity can assist inexperienced developers in designing and implementing reactive agents using FSMs. Its predictability allowed the game to be easily tested without the complexity of unforeseen events because each state transition is predictable based on a set of inputs (Brownlee, 2016). Additionally, due to the fact that FSMs and similar modelling concepts have been around for decades, it is guaranteed to be well proven with a myriad of well-designed examples to learn from.

Even though FSMs are useful when dealing with some kinds of problems that need to be predictable, such as the scenario when an NPC agent whose behaviours alters based on its internal states, or an agent that reacts to a set of inputs which has an effect its state transition (Nystrom, 2014), its functionalities are still considered limited compared with other artificial intelligence modelling techniques, such as behaviour trees or knowledge-based agents. In addition, FSMs are not considered turing complete due to the computational limitation (Champandard, 2007). Since FSMs are turing incomplete, unique designs may be needed for some special cases of problems, and this directly impacts on the applicability of the state machine concept.

Despite the concept of FSMs, pathfinding is another essential requisite of this game prototype. Without pathfinding, all the transitions in the game would be static and predictable, which make it less interesting and cumbersome in terms of the game development. By understanding the concept of the A* search algorithm and applying it to the game using the classes abstraction concept and some other useful concepts of Object Oriented Programming (OOP), each NPC agent can easily utilise the pathfinding algorithm to enhance its navigation system, which results in the game being less predictable and also heavily aids in the implementation process.

In conclusion, the design and implementation of FSMs are idiosyncratic and problem specific. Therefore, FSMs are suitable for problems which require behaviours of NPCs to be determined by states and transitions can occur when one state changes to another according to well-design rules that govern the state transitions (Warden, 2012). In addition to the state machine utilisation, another important concept used in this game is the pathfinding algorithm. By applying the A* search algorithm to the game, it mainly reduces the predicability of some behaviours and enhances the ease of navigation of each NPC agent along assisting the developer in the game design and development process.

REFERENCES

Brownlee, J. (2016). *Finite State Machines (FSM).* Retrieved April 13, 2016, from AI Depot: http://ai-depot.com/FiniteStateMachines/FSM-Background.html

Champandard, A. J. (2007). *10 Reasons the Age of Finite State Machines is Over.* Retrieved April 13, 2016, from AIGameDev: http://aigamedev.com/open/article/fsm-age-is-over/

Nystrom, R. (2014). *State: Game Programming Patterns / Design Patterns Revisited.* Retrieved April 13, 2016, from Game Programming Patterns: http://gameprogrammingpatterns.com/state.html

Patel, A. (2016). *Introduction to A**. Retrieved April 13, 2016, from Red Blob Games : http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Reddy, H. (2013). *PATH FINDING - Dijkstra's and A* Algorithm's.* Retrieved April 13, 2016, from Indiana State University - Computer Science, Math & CS Department: http://cs.indstate.edu/hgopireddy/algor.pdf

Rasmussen, R. (2016). INB383 AI for Games: Lecture 2 — Navigation Structure: Basic Graph Theory and Introduction to Path Finding. Lecture, QUT.

Warden, J. (2012). *Finite State Machines in Game Development.* Retrieved April 13, 2016, from Jesse Warden: http://jessewarden.com/2012/07/finite-state-machines-in-game-development.html