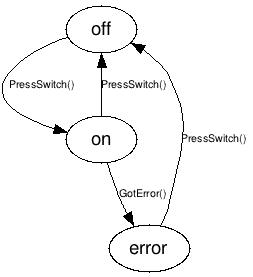
**Artificial Intelligence  
Introduction**This document will outline the reasoning behind the decisions that were made whilst designing and implementing a restaurant simulation in C++ using techniques such as finite state machines, steering behaviours and path finding.

**Finite State Machines**A finite state machine (FSM) can be described as a device that has a finite number of states that it can transition to and from. In regards to software engineering, an FSM is commonly an implementation of what is known as the state design pattern. ­

Design patterns are solutions to common and reoccurring software design problems. The state pattern allows an object to change its behaviour relative to its current state (Gamma et al, 1995, p.305), Figure 1 illustrates a state diagram that represents the states and behaviours of a light switch.

Figure 1 – Light Switch State Diagram



(Light Switch State Diagram, 2015)

Game agents frequently act in a limited set of ways and­­­ change their behaviour in response to influences in their environment (Millington and Funge, 2009, p.309). FSMs provide a simple and logical way of managing these behaviours, they are quick to setup, easy to debug and have little computational overhead and as such have been widely used in the development of computer games for many years. (Buckland, 2015, p.43).

The restaurant simulation application contains three types of agent that each fulfil a specific role, customers, waiters and chefs. In a bid to simulate their real-world equivalents, three FSMs were created and each role was broken down into its respective states and behaviours (see Figures 2).

Figure 2 – Agent State Diagrams



By using the state design pattern and building FSMs that model these behaviours, the agents used in the simulation were developed quickly and issues were easily identified and resolved. FSMs are a simple yet powerful organisational tools that help break problems down and make implementing artificial intelligence systems an easier task. (Schwab, 2008, p.261)

**Path Finding**The term path finding refers to algorithms that plot the shortest path between points whilst accounting for and circumventing untraversable areas. Game characters often navigate throughout their environments, agents that wander freely may appear to be aimless and unintelligent. Path finding algorithms are used in games and artificial intelligence programming to create more realistic and seemingly intelligent behaviours such as obstacle avoidance (Buckland, 2005, p.333).

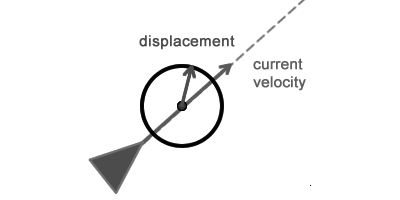
A\* is a path finding algorithm that is popular in games programming due to its speed and accuracy. A\* extends Dutch computer scientist Edsger W. Dijkstra’s famous algorithm (known as Dijkstra’s) by using heuristics to guide its search (Millington and Funge, 2009, p.197).

The restaurant simulation required that agents would be able to navigate the restaurant whilst avoiding objects such as tables and chairs. The restaurant was designed with an underlying grid system, each grid cell represented a node that could be flagged as untraversable. The A\* algorithm was implemented on top of this to provide the path finding functionality to the agents.

**Steering Behaviours**The term steering behaviours refers to the flocking simulation behaviours graphics expert Craig Reynolds began experimenting with in 1986. Steering behaviours simulate natural movement effectively and are relatively simple to implement and are therefore popular in games and artificial intelligence programming (Millington and Funge, 2009, p.437). There are many different steering behaviours such as seek, flee, arrive, pursue and evade.

The restaurant simulation features a rat agent that uses a steering behaviour known as wander to move around the restaurant. The wander behaviour is often used to simulate searching, foraging or as the name suggests - wandering behaviours in video games. The wander behaviour works by placing an (imaginary) circle in front of the agent and picking a random point within the circle to move towards each frame (See Figure 3) (Peters, 2009, p.82).

Figure 3 – Wander Behaviour



(Wander Behaviour Diagram, 2012)

**Bibliography**

Buckland, M (2005). *Programming Game AI by Example*, Massachusetts: Wordware Publishing.

Freeman, E., Robson, E (2004). *Head First Design Patterns*, California: O’Reilly Media.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*, Massachusetts: Person.

Light Switch State Diagram (2015). Available at: http://stackoverflow.com/questions/5923767/simple-state-machine-example-in-c (Accessed: 15th December 2016)

Millington, I., and Funge, J (2009). *Artificial Intelligence for Games*, 2nd edn, Florida: CRC Press.

Peters, K (2009). *Advanced ActionScript 3.0; Animation*, California: Friends of ED.

Schwab, B (2008). *AI Game Engine Programming*, 2nd edn, Cengage Learning.

Wander Behaviour Diagram (2012). Available at: https://gamedevelopment.tutsplus.com/tutorials/understanding-steering-behaviors-wander--gamedev-1624 (Accessed 17th December 2016)