Autonomous Agents - Project

Tadhg Riordan
12309240

April 2017

1 Introduction

In this project I implemented a finite state machine for a west world style game using Unity. This document will outline my approach. Supplementing code is provided.

2 High Level Overview

The game consists of agents which can freely move around a game grid and locations where the agents can travel to. There are three agents: an Outlaw, called Jesse, a sheriff, called Wyatt, and an undertaker, called Grim. Each agent is instantiated at a particular location. There are six locations: a bank, a saloon, an outlaw camp, a sheriff's office, an undertakers and a cemetery. Each agent starts at their base location. Jesse starts at the base camp. He travels between the outlaw camp and the cemetery and lurks in each for some time. Occasionally, he will go and rob the bank, taking a certain amount of gold in doing so. Wyatt patrols around each location. He can visit any location except the outlaw camp. As a sheriff, his job is to stop the bad guys, and so when he encounters Jesse, he shoots him on site, taking his gold in the process. He then goes to the bank to return the gold, assuming there was some, and then drops by the saloon for a drink. However, his work is not done, as Jesse respawns back at the outlaw camp, while his dead body remains at the scene. Grim is notified by Wyatt when there is a body to collect. Grim will wait in the undertakers until receiving such a notice, and on doing so, goes to the scene of the shooting, picks up the dead body, drops it off at the cemetery, and returns to his base.

3 Classes

3.1 Managers - Board and Game

GameManager is the root class of the project that handles instances of scripts and the functionality of the game. It contains an instance of itself which can be

used by other scripts to access global variables for the game. The *Awake()* function instantiates the game, connecting instances of the the scripts to *GameObjects* using the *AddComponent* function. An Update function is executed every frame which is used to transform the positions of the agents, executes a short wait between frames, and performs a function that checks where agents are co-located.
BoardManager is the primary way in which I handle the objects within the game and contains the global variables. The game grid is initialised by choosing random tiles from an array for the floor and outer wall tiles. I then initialise the mountains, locations, and finally the agents, all at random locations. Costs for the agents to move across each type of tile for the path-finding algorithm is also set; I set it so that mountains could be traversed infrequently, and that locations would never be traversed by setting an arbitrarily high cost for them.

3.2 State classes

The class *State* was left with it's default abstract functions for *Enter*, *Execute* and *Exit*. An abstract *Travel* class was implemented which inherited from an instance of the State class. This class was to handle the travelling of the agents between states, and is the cleanest way to do so. The *StateMachine* class was extended to include variables to handle global states and state blips.
The default Agent class was extended to include some extra variables associated with each agent: a value for *gold*, an (x,y,z) *position* variable and a *Location* variable. I also made it so that the class inherited from *MonoBehaviour* so that I could implement classes that derived from *Agent* that could make use of the behaviour of that class.

3.3 Location

I included a *Locations* class which performed functions related to the position and location of various things within the game. An Enum is defined for each location. Two two-dimensional arrays for the cost associated with each tile and the number of dead bodies at each tile is defined, indexed by grid position. A *LocationPositions* variable contains the position of each variable when it has been initialised in the Board Manager.

3.4 Agents

The agents are where the bulk of the functionality is implemented. I initially have definitions for the abstract classes from Agent. In *Awake()*, each agent is is initialised with a *StateMachine* of it's type, which places the agent in it's starting location.
Each agent implements a travel class which is a definition of the Travel class discussed earlier. Parameters are passed in the constructor of this class which set up the goal position and goal state for the agent when travelling. On entry to the class, the path is found by calling the *AStar* class. Then, for each cycle,

execute is called which sets the agent to be the next position in the path, until the path is complete. following this, the agents state is set to be the goal state, while the previous state is set to be the state before the travel state, to allow for state blips to still work.
The Outlaw agent has definitions for five states - *OutlawCamp*, *LurkCemetery*, *LurkBank* *OutlawTravel* and *OutlawGlobal*. Most of the functionality for this state is handled in the global class. in both *OutlawCamp* and *LurkCemetery*, on entry, the location for the agent is set, as well as the amount of lurk time he will have in each. On each cycle, a message is printed indicating what the agent is doing, and his lurk time decreases. In the global state, it is checked if the outlaw is bored by calling a function which returns the lurk time. If zero, it starts to travel to either the outlaw camp or the cemetery. The *LurkBank* class is called about 10% of the time, and only when the agent is at a location and not a random position. In this class, the outlaw takes a random amount of gold and stores it in his 'gold' variable.
The Undertaker class has states for *PickupDeadBody*, *DropDeadBody*, *LurkUndertakers*, *UndertakerTravel* and *UndertakerGlobal*. On *Awake()*, the class adds an event subscription to the Sheriff's Greeting event. The undertaker agent is initialised in the *LurkUndertakers* class, where it waits for a message to tell it that there is a dead body available. When it does, it traverses the *deadBodies* array defined in Locations and grabs it's position. it then changes to the travel state, moving towards the *PickupDeadBody* state, for the position of dead body. On entering that state, it increments it's *deadBodyCount* variable, and starts travelling to the *DropDeadBody* state for the Cemetery location. Once there it decrements its *deadBodyCount* variable and returns to the Undertakers. If any more outlaws were killed in the mean time, it sets off to repeat the procedure again.
The sheriff implements three classes: *Patrol*, *DropGoldAtBank* and *SheriffTravel*. The sheriff starts in the Patrol state. A location is picked at random that is not the Outlaw Camp, and the sheriff starts travelling there by creating a new instance of *SheriffTravel*. In the *Colocation* function in *GameManager* discussed earlier, the Sheriff's position is checked with the other agents. If the Sheriff is co-positioned with the outlaw, the sheriff increments its gold counter from the outlaw's gold, sets the outlaw to reset, increments the dead bodies array at it's position, notifies the undertaker and starts travelling to the *DropGoldAtBank* state. If the sheriff is co-located with the undertaker, it triggers a function which greets the undertaker. I realise that this functionality would be better implemented in the Sheriff class, however I could not get it so that it worked satisfactorily that way.

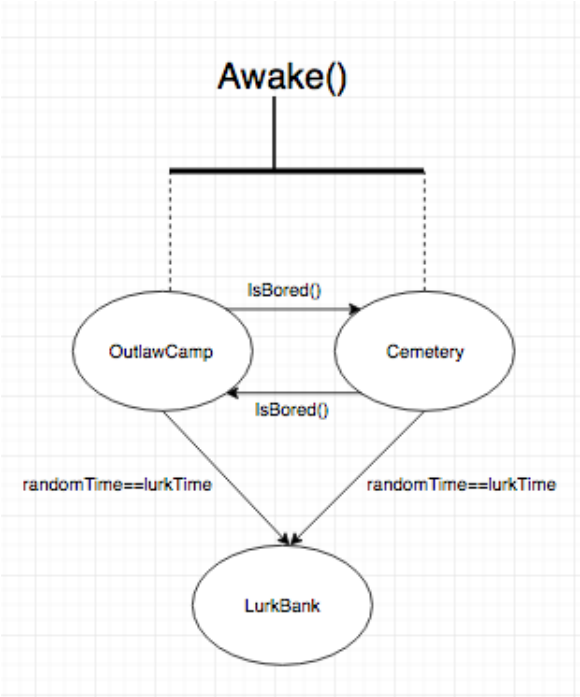
3.5 AStar

This class implements the path finding algorithm A*. This is necessary to find the best path from a position to another position, taking the costs of each into account. As mentioned, each tile is instantiated with a particular cost; basic floor tiles have a cost of one, mountains have a cost of three, and locations are

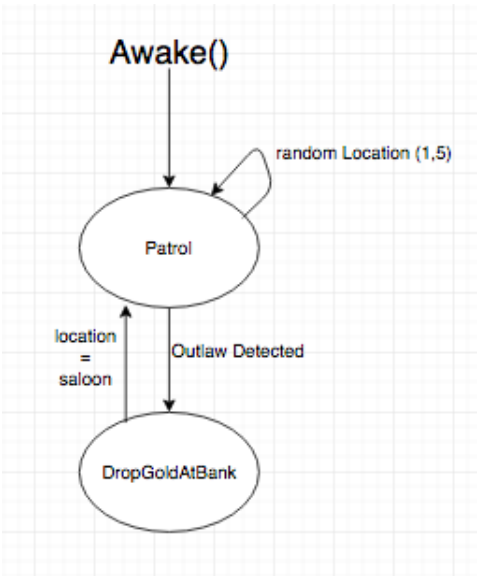
set with an arbitrarily high cost so as to avoid agents passing through these. An A* path consists of a set of nodes. Each Node contains values for the position of the node, the Node parent, the node cost, and values for G and H, which represent the exact and estimated cost to the goal respectively. A function *getF* is defined which represents the lowest cost from any node to the goal; it is calculated by adding G and H. On instantiation of a Node, these values are set from the parameters passed; cost functions for G and H sets those values.In *GetPath*, the goal position and starting position are passed, and for each node, it examines what the F value is to the goal - if it is lower than the current value for F, then it sets the current node to be that node. The current node is checked for the goal position, and if it has not yet reached the goal, it checks the four tiles to the north, south, east and west of the tile, assuming the tiles are available. It returns the best path once the open list reaches 0.

4 State Diagrams

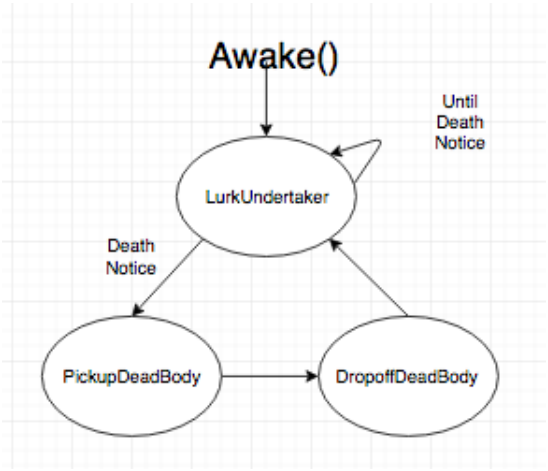
4.1 Outlaw States



4.2 Sheriff States



4.3 Undertaker States



5 Screenshots

5.1 Undertaker

Here the undertaker is lurking in it's base location. Both the sheriff and the outlaw are moving to new locations.

Here the undertaker is moving towards a dead body that it has been notified about by the sheriff. The sheriff and outlaw are both hanging in their base locations.

Here we can see that the undertaker is dropping off a body to the cemetery. following this he will return to his base location.

5.2 Outlaw

Here the outlaw is robbing the bank. Meanwhile, the undertaker is collecting a dead body, while the sheriff continues his patrol.

5.3 Sheriff

Here the sheriff is co-located with the outlaw. Following this cycle, the outlaw will respawn at his base location, and the undertaker will move towards the current position of the outlaw and sheriff.

Here the sheriff and undertaker are in the same location, which happens to be the undertakers base location. The Sheriff will issue a greeting to the undertaker.

Here the Sheriff is returning gold to the bank that he recovered from the outlaw after killing him. He will make the trip to the saloon following this. The outlaw has respawned in his base location and will move out of here once bored.