Love thine Agent: Implementing believable agents using augmented transition networks

Tiziano Riolfo and Alex Whittaker

Psygnosis Ltd.
Tiz.Riolfo@psygnosis.co.uk
http://www.psygnosis.com/

Abstract

We present some observations and lessons learned in implementing agent behaviour through an Augmented Transition Network (ATN) in the Playstation console game *Team Buddies* currently under development in the Psygnosis Ltd. Camden Studio.

1 Introduction

The Emotor engine was born out of the need for a versatile system to control multiple agents on a games console. The CPU on the Sony Playstation runs at 33MHz without floating point operations and with 2MB of main memory accessible through a 1KB cache. We are able to drive over twenty autonomous agents within the processor time allocated to AI control (approximately 0.01 second) in a single frame (screen refresh).

Emotor is presented as a C++ library compiled for a Microsoft Windows or Playstation operating system. It provides the game programmer with the means to associate an agent with a brain object that is a base class for an ATN object. The library is accompanied with a graphical editor that is able to create and manipulate the database that is read by the ATN and controls the agent's behaviour.

From the very beginning, the system has been ideally suited to providing an AI framework for a non-programming designer to take responsibility for coding and tweaking agent behaviour away from the programmer. Moreover, essential game features such as triggering sound effects, animation and changes in iconic information are all controlled by the agent through its state machine.

The game also employs a primitive agent communication language; organised team goals can be achieved through a system of orders issued between agents that override current activity and submit a new plan. This was very easily and effectively achieved with Emotor.

2 Problem Domain

The ultimate goal for a computer game AI designer is to have the player accept an agent as a friend, compatriot or enemy rather than merely an obstacle to success. To help the suspension of disbelief, the designer can give agents 'character' so that not only are they reactive to the user's avatar (the one agent entirely controlled by the player), but they also become recognisable and hopefully likeable for their strengths and weaknesses throughout their relationship with the user. This is the goal we have set for this product and this system.

The discussion presented here is informed by the development of an arcade game for the Sony Playstation console: *Team Buddies*. This is a single or multi-player game that puts up to four teams of one to four agents into a battle arena where one team must be victorious using an arsenal of weapons and vehicles to eliminate all others.

The most important aspect to *Team Buddies* is the collection and stacking of the building resource (crates). When crates are placed on the team's stacking pad, they combine to produce larger crates – when crates are opened they release a new weapon, vehicle or agent. Larger stacks reveal better or more potent rewards. In later levels different types of agent with new special abilities, physical attributes and behaviour are revealed.

Teams can be entirely AI driven as independent opponents and will fight amongst themselves as well as against 'player' teams. The player's team can be made to co-operate with the player's avatar by using the Playstation controller to communicate orders to them.

The player controls their avatar using the directional pad for movement and the buttons for jumping, picking up and putting down crates, shooting, etc. The other agents on the player's team can be swapped to at any time i.e. the player can 'possess' any of the host agents on their team relinquishing control of the previous agent which becomes autonomous. More effectively, team members can be ordered to perform a variety of tasks via a context-sensitive system that allows the player to highlight any object or agent in sight and expect their team members to interpret the player's intentions.

The orders issued to team members instruct them to adopt strategies outside of their normal behaviour - the autonomous teams use these as well. Without the addition of strategy selection, computer controlled teams would build weapons and vehicles and defend themselves but would continue to a stalemate rather than win the game. By allowing them to select from various aggressive and defensive strategies they are able to present a much more dynamic and enjoyable gaming experience.

3 Methodology

The programmer is required to provide the game engine with three types of C++ object that allow the agent to interact with its environment: actions, percepts and conditions. Each of these is encapsulated within a class of its own and cross-referenced to a unique identifier in the behavioural database.

3.1 State Transitions

The ATN allows the designer to describe an agent's behaviour as essentially a network of nodes; each node represents the state of the agent - where a state is the definition of the agent's priorities and each priority is defined by a percept. Percepts measure a feature of the environment and return a normalised value representing the strength of that feature.

The normalised percepts allow a comparison between different priorities; the strongest percept drives a state change that may have an action associated with it. For example if an agent is considered 'idle' (i.e. testing only its most basic priorities) a transition could occur to an 'alerted' state if an aggressor has invaded its personal space. A percept may also trigger a recursive transition to the same state usually driving an action that impacts on the same percept as a negative feedback.

An agent possesses many states that can handle numerous issues; however if the problem is complex enough a number of associated states will be necessary to tackle all the facets and perform all relevant tests.

3.2 Percepts, Interrupts and Compound variations

Percepts are described by graphs where the variable quantity (e.g. distance, time) is on the X-axis and the normalised strength is represented on the Y-axis. The shape of a percept graph is crucial to the 'believability' of the agent, as the best agents are those that react with apparently analogue responses. An agent's 'character' originates from its variety of response to clearly defined stimuli.

Percepts can be combined together as compound percepts using operators such as **MIN** (the minimum value of two or more percepts) and **MAX** (the maximum) logically equivalent to the conjunction and disjunction operators. This is useful because more relevant percepts can be brought together to make a more informed decision to change state and/or undertake a new plan. More interestingly, a simple measure such as proximity to an opposing agent can be combined with a superficially unrelated measure such as low health to give us a complex emotional percept such as fear. Layering these more emotive 'life-like' responses provides us with a less predictable and more believable agent.

Another useful function set measures a percept across an entire team and the normalised value is the result of the maximum or minimum value of each team member's observed quantity. For example, if you wanted to test whether every agent has a weapon, using the minimum value across the team for a percept measuring weapon strength will return a value reflecting the least armed team member.

3.3 Thresholds and interrupts

In order to keep the agent's attention focussed on the goal that it is currently pursuing, an activity threshold is set whenever the agent begins executing a new action. This threshold provides an upper limit that percepts need to exceed before they can trigger a second state change (and consequent change to the action being taken). When the agent completes its action plan, this threshold drops until the first percept to exceed it drives a new action selection.

The percept activity responds in the range 0 to 100, the default threshold is set to 75 at the onset of an action execution and drops to 25 at its completion. Percepts that can return a value of 75 or greater are termed interrupts, as they are able to cause an agent to stop whatever action it is executing and respond to the stimulus. Interrupts are critical to describing responsive intelligent behaviour.

3.4 Actions and Registers

Actions exist and are triggered along the transition arc from node to node. They have a variety of uses that are specific to the type of game using the Emotor library. Most actions are used to tell agents to go to a particular location or store information – they embody the response to a stimulus. Actions are intrinsically linked with registers, as it is the actions that allow information to be stored into the registers from which percepts can subsequently read.

Registers offer temporary storage allowing an agent to make reference to objects in its environment for later action. The temporary storage extends the representative power from that of a simple transition network to an ATN. In *Team Buddies* we use a stack register to hold way-points in a route-plan, and normal registers to hold other states and pointers to other agents, vehicles, toys and buildings.

Actions can either be primitive or compound. Primitives tend to be just one simple instruction, such as 'commit this information to memory' (into a named register) or 'travel to the co-ordinates stored in a named register'. Compound actions always contain more than one action within. It's perfectly possible to have compound actions within compound actions, however they all break down to a string of primitives that are executed sequentially.

For example, the frequently used compound action in *Team Buddies* named **GO_ATTACK_TARGET** breaks down to three primitive actions:

1. **GO_WITHIN_RANGE_OF_TARGET**	causes the agent to move within firing distance of its chosen target
2. **FACE_TARGET**	rotates the agent to face the target
3. **SHOOT_TARGET**	fires the equipped weapon

All three actions refer to a 'target' the assumption is that an object in the game environment has been selected and stored in the appropriate 'target' register.

A compound action is not executed instantly, component actions are executed sequentially over consecutive screen refreshes (40 milliseconds), so that at any given time actions may be being executed (carried over) from the last state transition.

3.5 Pre-and Post-conditions

As some actions are continuous over several frames we need to be able to test the environment in order to determine when they are complete. A database of conditions is built, each one cross-referenced by unique identifier to the code, and these can be attached as a post-condition to an action. For example the condition **AT_TARGET** would be supplied as the post-condition to **GO_WITHIN_RANGE_OF_TARGET**, the agent would continuously attempt to execute the action over many frames until the post-condition was true.

Similarly, we need to test that the preconditions for executing an action are met, this can avoid the agent attempting to execute an action that is patently futile. For example the precondition for the action **GO_WITHIN_RANGE_OF_TARGET** would be to test that the target register referenced a valid target: **TARGET_REGISTER_NOT_NULL**.

4 Strategic Behaviour

4.1 Triggering Strategies – AI and Player control

Strategies represent the need for activity other than the reactive behaviour described by the core state machine. Strategies have been organised as detached states that are only entered when conditions are favourable. On the computer-controlled team, this may depend on one agent ordering team members to change from whatever state they are currently in to a strategy state.

For teams controlled by a player the autonomous members will not execute a strategy; rather it is incumbent on the player to determine when a strategy will be effective. Pressing or holding the appropriate button on the Playstation controller orders a single agent or the entire team to adopt a strategy selected according to the focus of the avatar's attention.

4.2 Single and Team Strategies

The strategies in *Team Buddies* divide into two distinct groups, 'single' and 'team'. The approach is quite different as single types are triggered by specific observable values (e.g. vehicle available nearby might trigger the joy-rider strategy) that apply to only one agent. A team strategy is a more complex proposition, as a decision has to be made on a team basis using percepts measured across the team.

4.3 Affecting strategic decision-making

The system adopted in *Team Buddies* uses four percepts to modify the probability of a particular strategy being performed. These percepts modify the 'general' behaviour required for a game level, and represent the amount of aggressiveness, defensiveness, hindrance and stupidity expressed by all enemy agents. For enjoyable gameplay purposes, at the beginning of the game, levels of defensiveness and stupidity modifiers are high, and the reverse is true towards the end. These percepts modify the triggering conditions for the appropriate strategy: an aggressive strategy that throws caution to the wind will typically be driven by the percepts for aggressiveness and stupidity.

5 Lessons learned

The *Team Buddies* game is currently in the beta release phase, and we are now in a position to point out some of the pitfalls inherent in describing an agent using an ATN.

5.1 Data Management

Whilst complex behaviours can be emergent from a simple ATN, in general more intelligent, responsive agents require more complex ATNs occupying more memory. In *Team Buddies* the behavioural database occupies approximately 70KB of the 950KB of memory available for game data. This competes for space with the game world description, including animations, speech, model descriptions etc.

We have found that more correct and concise use of the grammar leads to more robust behaviours described in fewer states. Assumptions made in earlier states allow superfluous testing to be removed from subsequent transitions, making decisions faster and reducing the amount of environment sampling. For example if we enter a state as a reaction to being shot then we can assume that the aggressor is within weapon range and do not need to test it.

5.2 Limitations imposed by non-recursion

Despite there being many collections of states that perform very different tasks, there are some subsets within them that are repeated - often they are identical. In a combat-based game like *Team Buddies* attacking an aggressor is very much the same procedure which implies the same set of states. These sets of states are repeated because they are an intrinsic part of very different strategies and cannot be shared.

Due to the restrictive nature of the Playstation platform we have been obliged to develop an extension of a finite state transition network (FSTN). If we were able to store the unique identifier for a strategy into a stack register, we could model a recursive transition network (RTN). The RTN has a higher degree of notational adequacy, and is able to model a functionally equivalent FSTN with fewer states. We have implemented a single state register (rather than a stack) allowing a single level of recursion which has provided considerable improvements in clarity and memory use.

5.3 Simplification

Effective management of any state machine is important to maintain the designer's clarity of vision. Each state should deal with a set of conditions that give the state a specific purpose. Often a situation requires a collection of states to deal with a particular problem, so each substate is used to tackle a facet of the problem. This is generally the case where a particular stimulus is intended to cause a complex or unpredictable response.

For some bonus levels we have expected a considerably different behaviour from an agent, which would never be needed outside of that domain. Here we have tailored specific state machines focused on that level alone.

5.4 Interrupts

Percepts that return values at the maximum threshold level, referred to as *interrupts* due to the immediacy of their effect, present a unique problem because of their ability to stop what an agent is doing and intercede with a state change and new action. Problems occur where the state changes describe a cyclic component to the FSTN graph, where a state loops to itself or between states in a short loop. The actions that are driven by the interrupt are themselves interrupted by the same percept before they can complete. This is most notable when a string of actions is being executed, there is no guarantee that all actions will be completed if the destination state contains a potentially interrupting percept. The best solution is to use more states to ensure that actions do complete, however this adversely affects the size of the state machine. Another solution has been to extend the ATN model, allowing actions to set their own threshold level when executing, making themselves effectively non-interruptible, especially useful in actions affecting the registers.

5.5 Cyclic behaviours

Agents occasionally fail an action such as movement to a location for a variety of reasons. Sometimes the rules for collision detection between cylinders and rectangles become confused and the agent is caught in an inextricable loop. However, the state machine is able to assist by simply monitoring the time taken to complete an action and the number of failures, interrupting when a threshold is exceeded, a simple escape behaviour such as leaping to avoid an obstacle is then executed.

6 Summary

We have used an ATN to describe the behaviours of a large number of reactive agents in real time, on a platform that places severe constraints on processor and memory resource. Agent behaviour has been carefully crafted through a graphical editor and the limitations and advantages of the ATN model in agent behaviour description have been explored and presented. A data-driven model for agent behaviour is vital for developing complex agent behaviours in the development time scales available for computer games. Reactive behaviour is vital for real-time implementations within the constraints of a games console development platform.