Motivated Reinforcement Learning for Non-Player Characters in Persistent Computer Game Worlds

Kathryn Merrick
University of Sydney and National ICT Australia
IMAGEN Program
Locked bag 9013
Alexandria NSW, 1435
+61 2 8374 5590
kkas0686@it.usyd.edu.au

Mary Lou Maher
University of Sydney, Key Centre for Design Computing and Cognition
Wilkinson Building G04
University of Sydney, NSW 2006
+61 2 9351 4108
mary@arch.usyd.edu.au

ABSTRACT
Massively multiplayer online computer games are played in complex, persistent virtual worlds. Over time, the landscape of these worlds evolves and changes as players create and personalise their own virtual property. In contrast, many non-player characters that populate virtual game worlds possess a fixed set of pre-programmed behaviours and lack the ability to adapt and evolve in time with their surroundings. This paper presents motivated reinforcement learning agents as a means of creating non-player characters that can both evolve and adapt. Motivated reinforcement learning agents explore their environment and learn new behaviours in response to interesting experiences, allowing them to display progressively evolving behavioural patterns. In dynamic worlds, environmental changes provide an additional source of interesting experiences triggering further learning and allowing the agents to adapt their existing behavioural patterns in time with their surroundings.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Learning – *neural nets,* I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *dynamic programming, heuristic methods.*

General Terms
Algorithms.

Keywords
Motivation, reinforcement learning, computer games, persistent virtual worlds.

1. INTRODUCTION
Massively multiplayer online role-playing games (MMORPGs) such as Ultima Online, Everquest and Asheron's Call are defined by a cast of non-player characters (NPCs) who act as enemies, partners and support characters to provide challenges, offer assistance and support the storyline. These characters exist in a persistent virtual world in which thousands of human players take on roles such as warriors, magicians and thieves and play and interact with non-player characters and each other. Over time, the landscape of these worlds evolves and changes as players build their own houses or castles and craft items such as furniture, armour or weapons to personalise their dwellings or sell to other players.

Unlike computer games played in non-persistent worlds, persistent game worlds offer months rather than hours of game play, which must be supported by NPCs. However, current technologies used to build non-player enemy, partner and support characters tend to constrain them to a set of fixed behaviours which cannot evolve in time with the world in which they dwell. Motivated reinforcement learning (MRL) agents offer an alternative to this type of character. MRL use an intrinsic motivation process to identify interesting events which are used to calculate a reward signal for a reinforcement learner. MRL agents are able to continually identify new events on which to focus their attention and learn about. In a game scenario, using MRL agents to control NPCs, produces characters which are continually evolving new behaviours as a response to their experiences in their environment.

In the remainder of this section, we discuss the current technologies used to build NPCs. Section 2 describes MRL agents and the benefits they offer as an NPC technology. Section 3 provides two demonstrations of MRL agents in a simple role-playing game scenario implemented in the Second Life virtual world (www.secondlife.com). The first demonstration shows MRL agents can be used to create support characters that are able to explore their environment and learn new behaviours in response to interesting experiences, allowing them to display progressively evolving behavioural patterns. The second demonstration shows how MRL agents can be used to create partner characters which can adapt existing behavioural patterns in response to changes in their environment.

1.1 Current Technologies for Non-Player Characters
Non-player characters in MMORPGs fall into three main categories: enemies, partners and support characters [5]. Enemies in MMORPGs are characters which oppose human players in a pseudo-physical sense by attacking the virtual life force of the human player with weapons or magic. Partners take the opposite role and attempt to protect human players with whom they are allied. Alternatively, partner characters might perform non-combat tasks such as selling goods on behalf of their human ally. In some games, partner characters may be taught to perform certain behaviours by players. Finally, support characters are the merchants, tradesmen, guards, innkeepers and so on who support the storyline of the game by offering quests, advice, goods for sale or training. The technologies used to create these characters fall into two broad categories: reflexive agents and learning agents.

1.1.1 Reflexive Agents

Reflexive behaviour [6] is a pre-programmed response to the state of the environment – a reflex without reasoning. Only recognised states will produce a response. Non-player characters such as enemies, partners and support characters commonly use reflexive techniques such as state machines and rule-based approaches to define their behaviour. Rule-based approaches define a set of rules about states of the game world of the form: `if <condition> then <action>.` If the NPC observes a state which fulfils the `<condition>` of a rule, then the corresponding `<action>` is taken. Only states of the world which meet a `<condition>` will produce an `<action>` response. An example rule from a warrior NPC in the Baldur's Gate RPG is [13]:

```
IF
  !Range(NearestEnemyOf(Myself),3)
  Range(NearestEnemyOf(Myself),8)
THEN
  RESPONSE #40
  EquipMostDamagingMelee()
  AttackReevalutate(NearestEnemyOf(Myself),60)

  RESPONSE #80
  EquipRanged()
  AttackReevalutate(NearestEnemyOf(Myself),30)
END
```

The condition component of this rule is an example of how such rules are domain dependent as it makes the assumption that the character's environment contains enemies.

State machines can be used to divide an NPC's reasoning process into a set of internal states and transitions. In the Dungeon Siege RPG, for example, each state contains a number of `event` constructs which cause actions to be taken based on the state of the game world. Triggers define when the NPC should transition to another internal state. An example of part of a state machine for a beast called a Gremel is [10]:

```
startup state Startup${
     trigger OnGoHandleMessage$
        (WE_ENTERED_WORLD){
        SetState Spawn$;
   }
}

state Spawn${
   event OnEnterState${
     ...
   }
   event OnGoHandleMessage$( eWorldEvent e$,
      WorldMessage msg$ ){
         ...
         if(master$.Go.Actor.GetSkillLevel
            ("Combat Magic") > 0.01){
            Report.SScreen(master$.Go.Player.Ma
            chineId, Report.Translate(
            owner.go.getmessage("too_evil")));
         }
         else{
             ...
             SendWorldMessage( WE_REQ_ACTIVATE,
             Master$, newGoid$, 1 );
             PostWorldMessage(WE_REQ_ACTIVATE,
             Master$, newGoid$, 2, .2);
             Physics.SExplodeGo( owner.goid, 3,
             MakeVector(0,3,0) );
         }
    }
}

state Finish${
}
```

As only characters which multiply would require a spawn state, this example shows how the states are character dependent. .In addition, the condition components of the rules within the states are again heavily domain dependent, assuming for example that the environment contains characters that have a combat magic attribute.

In a departure from purely reflexive techniques, the support characters in some RPGs, such as Blade Runner, have simple goals. However these have also tended to be fairly narrow, supported by only a limited set of behaviours.

1.1.2 Learning Agents

Learning agents are able to modify their internal structure in order to improve their performance with respect to some task [14]. In some games such as Black and White, non-player characters can be trained to learn behaviours specified by their human master. The human provides the NPC with feedback such as food or patting to encourage desirable behaviour and punishment to discourage unwanted actions. While the behaviour of these characters may potentially evolve in any direction desired by the human, behaviour development relies on feedback from human players, making it inappropriate for characters such as enemies or support characters.

Researchers from Microsoft have shown that it is possible to use reinforcement learning to allow NPCs to develop a single skill by applying it to fighting characters for the Xbox game, Tao Feng [3]. Reinforcement learning agents [10] are connected to their environment by sensation and action. On each step of interaction with the environment, the agent receives an input that contains some indication of the current state of the environment and the value of that state to the agent. This value is called a reward signal. The agent records the reward signal by updating a behavioural policy which represents information about the reward received in each state sensed so far. The agent then chooses an action which attempts to maximise the long-run sum of the values of the reward signal. In Tao Feng, while NPCs using reinforcement learning can adapt their fighting techniques over time, it is not possible for them to identify new skills to learn about as they are limited by a pre-programmed reward for fighting.

2. MOTIVATED REINFORCEMENT LEARNING AGENTS

Motivated reinforcement learning agents are meta-learners which use a motivation function to provide a standard reinforcement learning algorithm with an intrinsic reward signal that directs learning. Unlike existing NPC technologies, the motivation function uses domain independent rules based on the concept of interest in order to calculate an intrinsic motivation signal. Skill development is dependent on the agent's environment and its experiences rather than on character or domain specific rules or state machines. This means that a single agent model applied to different NPCs will develop different skills depending on the NPC's environment. These skills are developed progressively over time and can adapt to changes in the agent's environment.

Our motivated reinforcement leaning agent model is depicted in Figure 1. In this model, $W_{(t)}$ represents the state of the agent's environment at time t. $S_{(t)}$ represents the state sensed by the agent at time t. The sensation process $\mathcal{S}$ accepts the current sensed state from the sensors and computes events as changes in the world since the last sensed state. Events represent the dynamics of the agent's environment where sensed states provide information about the current state. An agent remembers two sensed states, the previous $S_{(t')} = (s_{1(t')}, s_{2(t')}, \ldots s_{L(t')} \ldots)$ and the current $S_{(t)} = (s_{1(t)}, s_{2(t)}, \ldots s_{L(t)} \ldots)$. A comparison $S_{(t)}$–$S_{(t')}$ of these states produces the difference variables $\Delta(s_{1(t)}, s_{1(t')})$, $\Delta(s_{2(t)}, s_{2(t')})$, … $\Delta(s_{L(t)}, s_{L(t')})$ …. An event function defines the combination of difference variables an agent recognises as events. In this paper, we assume an event $E_{(t)} = (e_{1(t)}, e_{2(t)}, \ldots e_{L(t)} \ldots)$ contains all non-zero difference variables after a numerical subtraction of sensation values. The motivation process $\mathcal{M}$ uses the current event $E_{(t)}$ and the agent's experiences of all events $\mathbf{E}_{(t-1)}$ to produce a new representation of experiences $\mathbf{E}_{(t)}$ and a reward signal $R_{(t)}$.

The learning process $\mathcal{L}$ uses the Q-learning reinforcement strategy [11] shown in Equation 1 to incorporate the sensed state into a behavioural policy $B_{(t-1)}$ to produce the updated behaviour $B_{(t)}$ which is stored in memory M.

$$Q(S_{(t)},A_{(t)}) \leftarrow Q(S_{(t)},A_{(t)}) + \beta[R_{(t)} + \gamma \max_{A \in \mathbf{A}} Q(S_{(t+1)},A_{(t+1)}) - Q(S_{(t)},A_{(t)})] \quad (1)$$

Finally, the activation process A uses an exploration function with the Q-learning action selection rule in Equation 2 to select an action $A_{(t)}$ to perform from the updated behavioural policy $B_{(t)}$. We used ε-greedy exploration for our experiments with $\varepsilon = 0.1$, $\beta = 0.9$ and $\gamma = 0.9$. The chosen action $A_{(t)}$ triggers a corresponding effector $F_{(t)}$ which makes a change to the agent's environment.

$$A_{(t)} = \arg\max_{A \in \mathbf{A}} Q(S_{(t+1)},A_{(t+1)}) \quad (2)$$

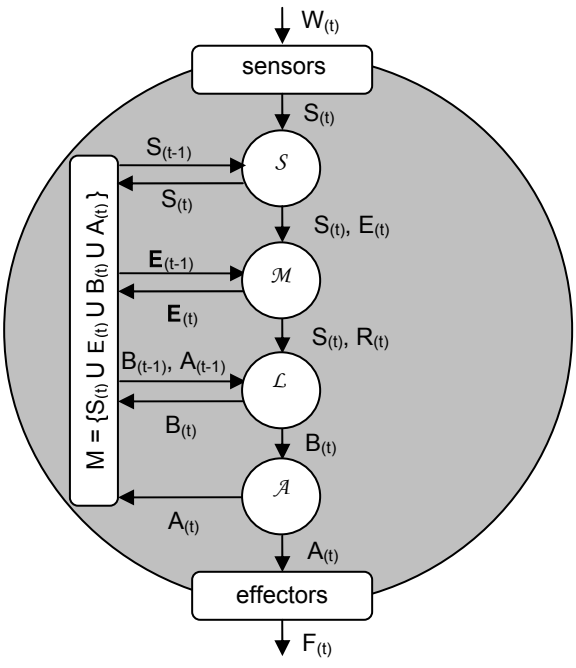

Figure 1. A motivated reinforcement learning agent model.

The key process that differentiates motivated reinforcement learning from existing NPC technologies, is the motivation process. Where existing NPC technologies relied on domain specific rules, state machines and rewards, motivated reinforcement learning uses a task dependent motivation process to reason about the agent's experiences $\mathbf{E}_{(t-1)}$ and produce a reward signal $R_{(t)}$ to direct the learning process.

A number of computational models of motivation have been developed for use in artificial agents. These include models of biological theories of motivation such as drive theory [2] and cognitive theories such as curiosity and interest [9]. Cognitive theories about phenomena such as curiosity and interest explain this search in terms of constant adjustments and adaptations to a baseline level of stimulation from the environment which in turn defines some moderate, optimal stimulation level. As we are interested in building agents that can adjust and adapt their behaviour to learn multiple tasks in response to their environment, these cognitive theories make an ideal starting point for motivation functions.

Saunders and Gero implemented a computational model of interest for social force agents by first detecting the novelty of environmental stimuli then using this novelty value to calculate interest. The novelty of an environmental stimulus is a measure of the difference between expectations and observations of the environment where expectations are formed as a result of an agent's experiences in its environment. Saunders and Gero model these expectations or experiences using an Habituated Self-Organising Map (HSOM) [7]. Interest in a situation is aroused when its novelty is at a moderate level, meaning that the most interesting experiences are those that are similar-yet-different to previously encountered experiences. The relationship between the intensity of a stimulus and its pleasantness or interest is modelled using the Wundt curve [1] shown in Figure 3.

An HSOM consists of a standard Self-Organising Map (SOM) [4] with an additional habituating neuron connected to every clustering neuron of the SOM as shown in Figure 2. A SOM consists of a topologically structured set **U** of neurons, each of which represents a cluster of events. The SOM reduces the complexity of the environment for the agent by clustering similar events together for reasoning. Each time a stimulus event $E_{(t)} = (e_{1(t)}, e_{2(t)}, \ldots e_{L(t)} \ldots)$ is presented to the SOM a winning neuron $U_{(t)} = (u_{1(t)}, u_{2(t)}, \ldots u_{L(t)} \ldots)$ is chosen which best matches the stimulus. This is done by selecting the neuron with the minimum distance d to the stimulus event where d is calculated as:

$$d = \sqrt{\sum_{L} (u_{L(t)} - e_{L(t)})^2}$$

The winning neuron and its eight topological neighbours are moved closer to the input stimulus by adjusting their weights using the update equation:

$$u_{L(t+1)} = u_{L(t)} + \eta\,(e_{L(t)} - u_{L(t)})$$

where $0 \leq \eta \leq 1$ is the learning rate of the SOM. The neighbourhood size and learning rate are kept constant so the SOM is always learning. The activities of the winning neuron and its neighbours are propagated up the synapse to the habituating layer as a synaptic value $\sigma_{(t)} = 1$. Neurons which do not belong to the winning neighbourhood give an input of $\sigma_{(t)} = 0$ to the synapse. The synaptic efficacy $N_{(t)}$, which represents the novelty

of the stimulus $E_{(t)}$, is then calculated using Stanley's model of habituation (Stanley, 1976):

$$\tau \frac{dN_{(t)}}{dt} = \alpha [N_{(0)} - N_{(t)}] - \sigma_{(t)} \quad (3)$$

where $N_{(0)} = 1$ is the initial novelty value, τ is a constant governing the rate of habituation and α is a constant governing the rate of recovery. In practice, it is desirable to split the habituation constant τ into τ_1 and τ_2 where τ_1 governs the rate of habitation in neurons in the winning neighbourhood and τ_2 governs the rate of habitation in losing neurons. Using $\tau_2 > \tau_1$, the novelty of an event will tend to increase at any time-step more slowly than it can be decreased by the occurrence of other events allowing the HSOM to learn more quickly than it forgets. $N_{(t)}$ is calculated stepwise at time t by using the value $N_{(t-1)}$ stored in the habituating neuron to calculate the derivative from Equation 3 and then approximating $N_{(t)}$ stepwise using:

$$N_{(t)} = N_{(t-1)} + \frac{dN_{(t-1)}}{dt}$$

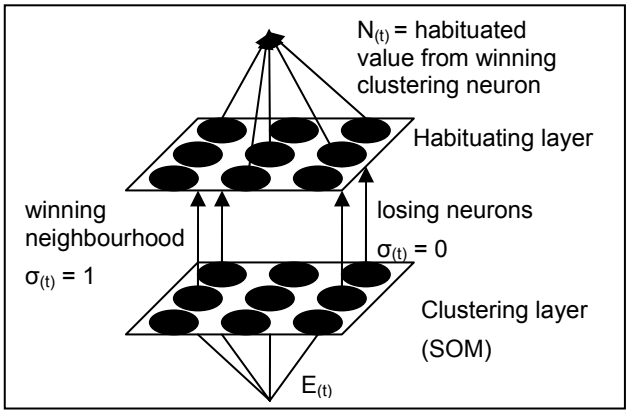

Figure 2. A novelty filter. A clustering layer (SOM) is connected to an habituating layer.

Habituation has the effect of causing synaptic efficacy or novelty to decrease with subsequent presentations of a particular stimulus or increase with subsequent non-presentations of the stimulus. This represents forgetting by the HSOM and allows stimuli to become novel more than once during an agent's lifetime. Once the novelty of a given stimulus has been generated, interest I of the is calculated using the Wundt equation:

$$I(N_{(t)}) = \frac{F^{+}_{max}}{1 + e^{-\rho^{+}(2N_{(t)} - F^{+}_{min})}} - \frac{F^{-}_{max}}{1 + e^{-\rho^{-}(2N_{(t)} - F^{-}_{min})}}$$

The first term in the Wundt equation provides positive feedback for the discovery of novel stimuli while the second term provides negative feedback for highly novel stimuli. It peaks at a maximum value for a moderate degree of stimulation as shown in Figure 3, meaning that the most interesting events are those that are similar-yet-different to previously encountered experiences.

F^{+}_{max} is the maximum positive feedback, F^{-}_{max} is the maximum negative feedback, ρ^{+} and ρ^{-} are the slopes of the positive and negative feedback sigmoid functions, F^{+}_{min} is the minimum novelty to receive positive feedback and F^{-}_{min} is the minimum novelty to receive negative feedback. We used $F^{+}_{max} = 1$, $F^{-}_{max} = 1$, $\rho^{+} = 10$, $\rho^{-} = 10$, $F^{+}_{min} = 0.5$ and $F^{-}_{min} = 1.5$ in our experiments. The interest value $I(E_{(t)})$ is used as the reward $R_{(t)}$ which is passed from the motivation process $\mathcal{M}$ to the reinforcement learning process $\mathcal{L}$. The reward function is defined as follows:

$$R_{(t)} = \begin{cases} I(N_{(t)}) \text{ if } E_{(t)} \text{ not empty} \\ 0 \text{ otherwise} \end{cases}$$

This reward function is based on the agent's expectations of its environment represented by an HSOM. The structure of the SOM component of the HSOM is determined over time as a response to the agent's experiences of events in its environment. Thus, the motivating force behind the agent's actions is dependent on its environment rather than on a task specific reward signal.

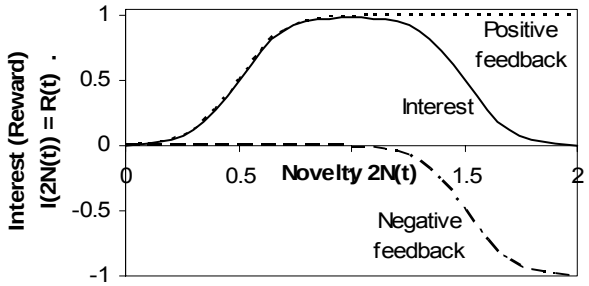

Figure 3. The Wundt curve is the difference between positive and negative feedback functions.

3. MOTIVATED NON-PLAYER CHARACTERS

In this section we apply the MRL model described above to a number of NPCs in a simple role-playing game scenario implemented in the Second Life virtual world. The first demonstration shows MRL agents can be used to create support characters that are able to explore their environment and learn new behaviours in response to interesting experiences, allowing them to display progressively evolving behavioural patterns. The second demonstration shows how MRL agents can be used to create partner characters which can adapt existing behavioural patterns in response to changes in their environment. In both demonstrations we use the same MRL agent model. Only the agent's environment and effectors differ. In this way, we show practically how the agents develop different behaviours as a result of their environment and experiences rather and not because they have domain specific or character specific programming.

3.1 The Game World

In order to experiment with MRL agents, we implemented a village scenario in Second Life (www.secondlife.com). Second Life is a commercially available persistent virtual world in which users can build and act in real time. The village, shown in Figure 4 has a carpenter shop and a smithy. There are various tools and other artefacts hidden about the village including an axe, a pick, a lathe and a forge. To the north of the village are a forest and an

iron mine. Objects in the environment contain scripts written in Second Life's Linden Scripting Language (LSL) defining the mechanics of their use. For example, the pick object places iron in an NPC's backpack when the NPC is holding the pick near the mine and chooses the "use" action. Unlike the smart terrain concept used in The Sims, it is not necessary to define details such as when to use the pick inside the pick and what to do with the resulting iron inside the iron. This will be learned by the MRL agent controlling the NPC as it explores its environment. Second Life includes a physics engine so objects may optionally obey laws of gravity and friction.

Second Life avatars can be controlled by an agent program written in a programming language such as Java using the framework shown in Figure 5. The Java agent program consists of sensor stubs, MRL agent reasoning processes and effector stubs. The Java sensor and effector stubs act as clients which connect via XML-RPC to corresponding sensor and effector stubs written in the Linden Scripting Langauge (LSL) and residing on a Second Life server. The LSL sensor and effector stubs are associated with a backpack object worn by the avatar the Java agent is to control. As well as enabling the Java agent to control the position of the avatar and sense the surrounding environment, the backpack also acts as a repository in which the agent can place objects it picks up and carries.

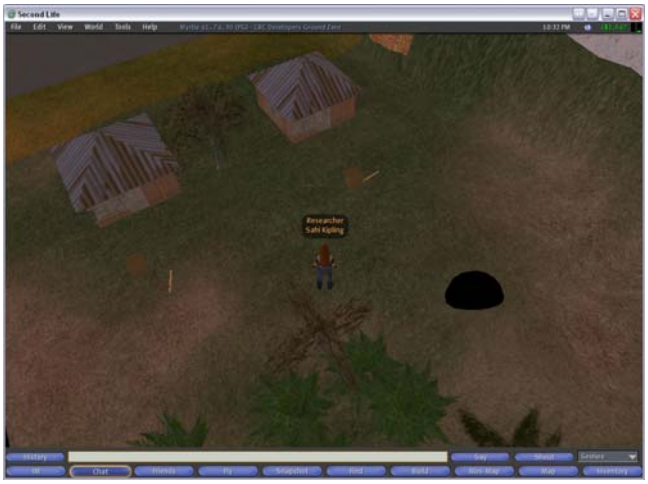

Figure 4. A village scenario implemented in Second Life.

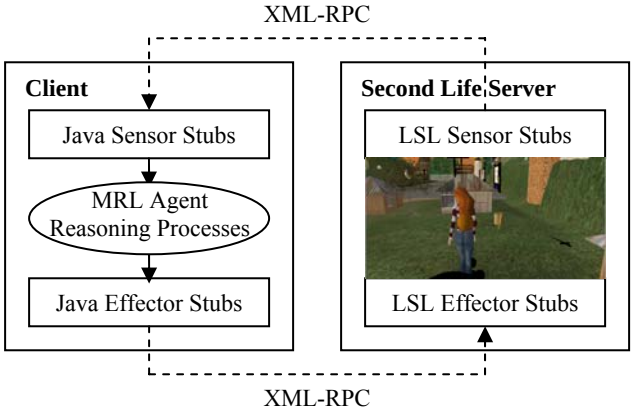

Figure 5. System architecture for Second Life agents.

Rather than using a fixed length vector representation for this environment as is common with standard reinforcement learning, we use a context free grammar (CFG) [8]. Each world state $W_{(t)}$ and sensed state $S_{(t)}$ is a string from a context-free language. While CFGs can represent any environment that can be represented by a fixed length vector, they have a number of advantages over a fixed length representation. Using a CFG, only objects that are currently present in the environment need be present in the state string for the current state. This means that the state string can include any number of new objects as the agent encounters them, without a permanent variable being required for that object. This is desirable in game environments where players can build while the game is in progress as it is not known at design time what objects may occur and what variables may be needed.

The agent sensors are capable of assigning labels L to their sensations of the world, allowing sensations with the same label to be compared. Some example sensed states in this environment in label-sensation (L:s) format are:

$S_{(1)}$((locX:36)(locY:71)(locZ:30)(Pick:1)(Iron:1))

$S_{(2)}$((locX:37)(locY:76)(locZ:30)(Iron:1)(Mine:1))

A SOM, and thus a HSOM, can be modified to accept CFG representations of states by initialising each neuron as an empty vector and allowing neurons to grow as required. Likewise, a table-based reinforcement learner can be modified to use a CFG representation by storing strings from the CFG in the state-action table in place of vectors.

3.2 Progressive Emergence of Behavioural Patterns in Motivated Support Characters

Support characters in role-playing games frequently play the role of tradespeople such as blacksmiths, and lumberjacks. This enables them to provide materials to human player characters as well as training in their skills. Such interactions with human players are usually facilitated through a graphical or text based user interface. While we assume the continued existence of a user interface to facilitate communication between the support character and human players, we use a MRL agent to control the behaviour of the support character, in place of typical rule based or state machine approaches.

In this scenario the MRL agent has three sensors: a location sensor, an object sensor and an inventory sensor. These allow it to sense its x, y and z co-ordinates, the objects within a 7 metre radius and the objects in its backpack (not including the sensor and effector stubs). The agent has three effectors: a move to object effector, a pick up object effector and a use object effector. The pick, when used on the mine, will produce iron which can be converted to weapons when used near the forge. Similarly, the axe, when used near a tree, will produce timber which can in turn be converted to furniture when used near the lathe.

While the MRL agent is capable of running continuously for long periods of time, in the following paragraphs we analyse the first five hours of its life. Figure 6 shows the actions performed by the MRL agent and some of the interest values (rewards) which motivated it. Specifically, Figure 6 includes line plots against time of the interest values for mining iron and making weapons, as well as a scatter plot showing the actual actions performed against time. The key for the actions is shown in Table 1.

Table 1 – Enumeration of agent actions for Scenario 1.

Action ID	Description
1	Move to forge.
2	Move to pick.
3	Move to lathe.
4	Move to tree.
5	Move to mine.
6	Move to axe.
7	Add pick to inventory.
8	Add axe to inventory.
9	Use pick.
10	Use axe.
11	Use iron.
12	Use wood.

In the first 3.5 hours the action scatter plot shows a preference for actions 1, 3, 4 and 5: the move forge, lathe, tree and mine actions. In these periods the agent is performing a travelling behaviour between some subset of the locations in the environment. Noise on the scatter plot is a result of the random action selection in the ε-greedy exploration strategy. At t=3.5 hours, the interest curves show the interest values for forging weapons and mining iron increase. At this time, the action scatter plot shows a preference by the agent for actions 11, 9, 5 and 1: using the iron and the pick and moving to the mine and the forge. This is an example of how MRL agents are able to progressively evolve new behaviours over time in response to their experiences in their environment.

In the first five hours of its life the MRL agent does not pursue furniture making more than a few times. However, later in its life it may develop an interest in that task and learn the appropriate behaviour to perform it. Thus, when multiple MRL agents are introduced into the world, each agent will develop behaviours based on their experiences in the world, resulting in a number of different characters including blacksmiths, carpenters and travellers.

3.3 Adaptable Behavioural Patterns in Motivated Partner Characters

Another role commonly played by NPCs is that of a partner character while the human player is not online or to perform repetitive tasks on their behalf. For example, in Ultima Online players can set up vendor characters to sell the goods they have crafted. These vendors stand in one place and player characters have to come to them in order to trigger a user interface allowing them to buy goods. While we assume the continued existence of the user interface, MRL agents can offer a dynamic alternative to these static vendors.

In this scenario the agent has two sensors: a location sensor and an object sensor. These allow it to sense its x, y and z co-ordinates and the objects within a 7 metre radius. The agent has one effector: a move to object effector. The object sensor can detect the smithy, the carpenter's shop, the mine, the forge, lathe, pick, axe and a number of trees as shown in Figure 7. In a real game, there might be human controlled avatars at any of these locations. We ran an MRL agent for 2 hours in this scenario. The actions performed by the MRL agent in this time are plotted in Figure 8. A legend is shown in Table 2.

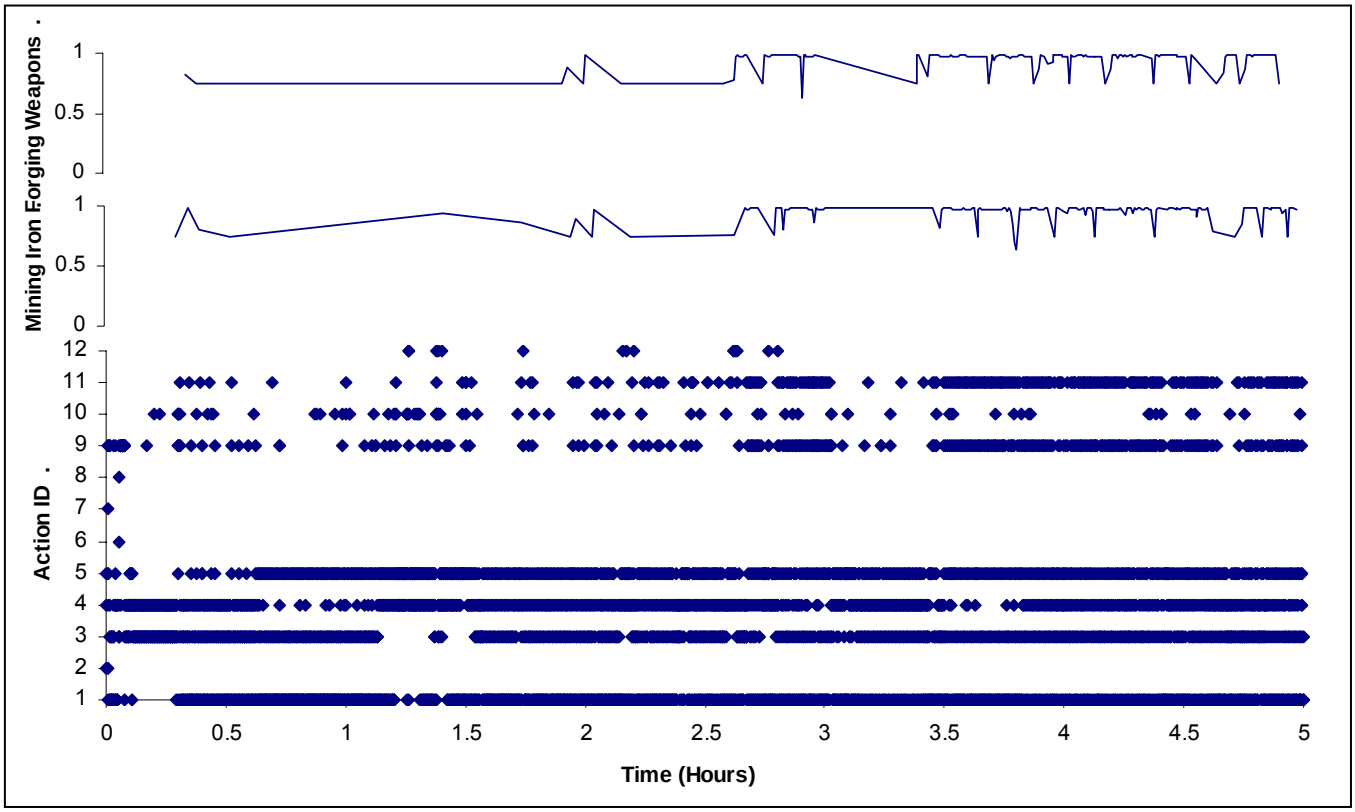

Figure 6 – Actions performed by an MRL agent and some of the interest values (rewards) which motivated them, including (top to bottom) interest in forging weapons and iron mining.

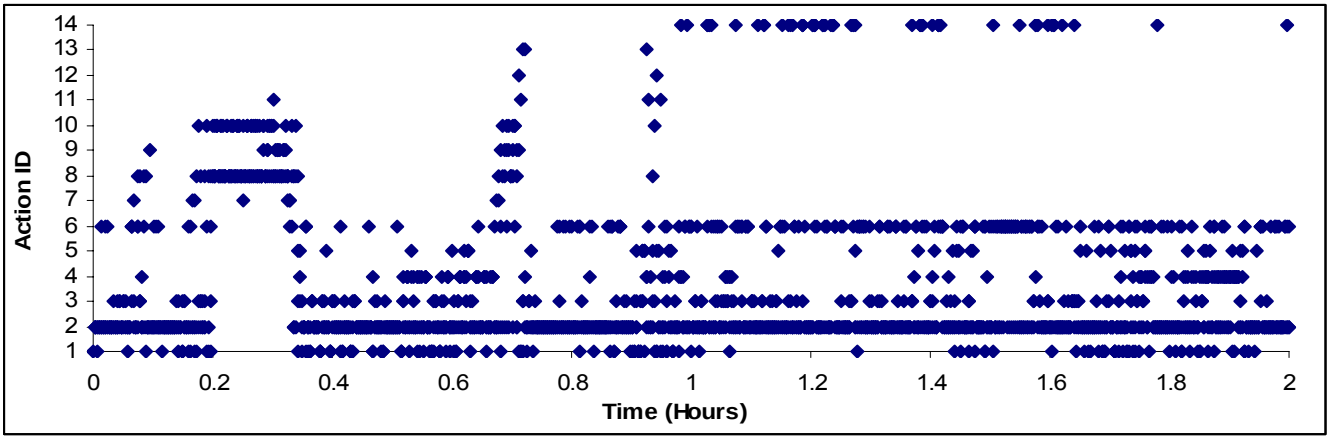

Figure 8 – Actions performed by a travelling vendor MRL agent.

Figure 7. The village scenario with additional objects.

Initially the vendor character was located near the mine and the iron. These two locations became recurrent destinations over the course of its lifetime. During the first hour the vendor initially focused its attention on moving between the mine and iron and the forge and pick. Around t=0.2 its focus of attention shifted and a new behaviour evolved for moving between the wood and the chair.

Table 2 – Enumeration of agent actions for S 2.

Action ID	Description
1	Move to pick.
2	Move to iron.
3	Move to forge.
4	Move to sword.
5	Move to smithy.
6	Move to mine.
7	Move to forest.
8	Move to wood.
9	Move to axe.
10	Move to chair.
11	Move to lathe.
12	Move to carpenter.
13	Move to tree.
14	Move to Kathryn's House

At t=0.6 the agent accidentally tripped over the sword, shifting it to a new location. This caused a flurry of new events to become interesting including moving to the forest. Around t=0.8 a player character called Kathryn built a new house next to the iron mine as shown in Figure 8. After a short period of time the vendor adapted its existing behavioural patterns and began to include this house as a destination in its behaviour.

The ability of the MRL vendor to focus its attention on different destinations has two benefits. Firstly, the character is more realistic as it can move around like a travelling salesperson rather than standing in one spot. Secondly, because the vendor can move, its human master is receiving better exposure of his or her goods to potential customers. As the world changes the travelling salesperson can adapt its behaviours in response. This scenario could be further extended to allow the MRL agent to sense information about the goods purchased from it by human players using its user interface. This would allow it to develop behavioural patterns based on its experiences with its customers and further improve the vendor service it offers.

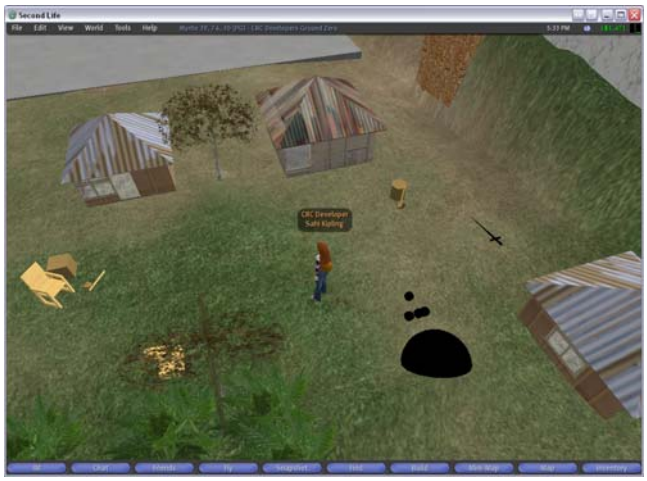

Figure 8. The village scenario after Kathryn has built a house.

4. CONCLUSION

This paper has presented motivated reinforcement learning agents as a means of creating non-player characters which can both evolve and adapt. Motivated reinforcement learning agents explore their environment and learn new behaviours in response to interesting experiences, allowing them to display progressively evolving behavioural patterns. In dynamic worlds, environmental changes provide an additional source of interesting experiences triggering further learning and allowing the agents to adapt their existing behavioural patterns in time with their surroundings. Furthermore, motivated reinforcement learning allows a single agent model can be applied to multiple characters which then develop different behaviours based on their experiences in their environment.

5. ACKNOWLEDGMENTS

This research was supported by a National ICT Australia PhD scholarship. National ICT Australia is funded by the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

6. REFERENCES

[1] Berlyne, D. E. *Aesthetics and psychobiology*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[2] Canamero, L. Modelling motivations and emotions as a basis for intelligent behaviour. In *Proceedings of the First International Symposium on Autonomous Agents.* (New York, NY), ACM Press, 1995, 148-155

[3] Graepel, T., Herbrich, R., and Gold, J. Learning to Fight. *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*. 2004.

[4] Kohonen, T. *Self-organisation and associative memory*. Springer, (Berlin), 1993.

[5] Laird, J., and van Lent, M. Interactive computer games: human-level AI's killer application. In *Proceedings of AAAI National Conference on Artificial Intelligence,* 2000, 1171-1178.

[6] Maher, M.-L., and Gero, J.S. Agent models of 3D virtual worlds, *ACADIA 2002: Thresholds*. (California State Polytechnic University, Pamona), 2002, 127-138.

[7] Marsland, S., Nehmzow, U. and Shapiro, J. A real-time novelty detector for a mobile robot. *EUREL European Advanced Robotics Systems Masterclass and Conference,* 2000.

[8] Merceron, A. *Languages and Logic*. Pearson Education Australia, 2001.

[9] Saunders, R., & Gero, J. S. Curious agents and situated design evaluations. In J. S. Gero & F. M. T. Brazier (Eds.), *Agents In Design*, 2002, 133-149.

[10] Siege University*, 303: Skrit*, http://garage.gaspowered.com, Accessed March, 2006.

[11] Sutton, S. and Barto, A. *Reinforcement learning: an introduction.* The MIT Press, 2000.

[12] Watkins, C., *Learning from delayed rewards,* PhD Thesis, Cambridge University, (Cambridge, England), 1989.

[13] Woodcock, S., *Games making interesting use of artificial intelligence techniques,* http://www.gameai.com/ Accessed March, 2006.

[14] Zoubin G., Unsupervised Learning, In Bousquet, O., Raetsch, G., von Luxburg, U., (Editors), *Advanced Lectures on Machine Learning*, LNAI 3176, Springer-Verlag, 2004.