DESIGNING A SYSTEM FOR ADAPTIVE COMBAT USING REINFORCEMENT LEARNING

Simon McDonnell
B.Sc. (Honours) in Computing in Games Development
Dundalk Institute of Technology
2012-2013

ABSTRACT

This paper seeks to answer the question "How do I implement an adaptive combat AI in a game using reinforcement learning?" To do this we analyse what adaptive AI is and why it is important. We then determine a set of "good" attributes that we want our system to have. These attributes correspond to our goals for the system (quick adaptation, experimentation, and fair difficulty) and will be used to determine which methods we use for reinforcement learning.
Next we look at reinforcement learning and why it is being used over other machine learning methods. We analyse what reinforcement learning is and also the different advantages that reinforcement learning brings to an adaptive AI solution. Reinforcement learning is stepped through from basics through to more complex methods as the solution is built up. Each method is analysed evaluated using the set of "good" attributes that we previously decided upon. The solution uses the traditional reinforcement learning system of state, action, and reward for its basic structure. It then moves on to implement a value system with rudimentary exploration and value estimates, which improves the overall efficacy of the system. The exploration aspect of this value system is then improved upon further. Both the epsilon greedy and the softmax action selection methods are considered as ways to improve exploration. The epsilon greedy method is chosen as it more closely matches our goals for the system. The system is implemented using unrealscript within the Unreal Development Kit (UDK). The agent can attack and block in three different directions, as well as crouch, kick, and stand up. The system produces an opponent who adapts to fight more effectively as it receives different inputs. It is found that reinforcement learning is complex, and can be taken in many different directions, but is a flexible and efficient way of implementing adaptive combat AI.

KEYWORDS

State, action, reward, value system, exploration, epsilon greedy, softmax action selection

1. INTRODUCTION

AI in games can often be static. It uses a combination of scripting and simple conditional responses (if this condition is met; do this thing, if this other condition is met; do this other thing) to respond to situations. This works well in games like chess, where intuition and human skill takes a backseat to the mathematical problem of an optimal strategy. Where this falls behind, however, is in games where it is less clear what the right option is; games where intuition and reading your opponent are the things which most often lead to success.

As a result using static AI's for these types of games doesn't usually yield good results, the AI can seem transparent and repetitive, easy for the player to exploit in order to win. A better approach for these kinds of games is an adaptive AI; one which, as it fights, adapts its strategy to its opponent. This type of AI learns from its mistakes and is less easily exploited. Utilising an adaptive AI in these types of games can go a long way towards improving the experience of the player, as they are challenged by an adaptive opponent who does not constantly fall for the same strategies.

In this paper I plan to implement an adaptive combat AI into the UDK using unrealscript. Unrealscript is the UDK's programming language and is the best choice for this task due to the in depth programming required. We must use unrealscript because unfortunately there is no Cascade-like equivalent (Cascade being an UDK utility for building particle systems) for building AI systems.

The adaptive system will be implemented using reinforcement learning, which is a type of machine learning. It is a flexible learning method which is based on an AI agents interaction with his environment. It is well suited to the interaction-driven nature of a video game. Reinforcement learning has been implemented into video games before; a system was built for Tao-feng: Fist of the Lotus (Graepel, T. et al, 2004), which is an XBOX fighting game similar to Tekken; and also for Starcraft (Micic, A., Arnarsson, D., Jonsson, V., 2011), which is a real time strategy game the same vein as Age of Empires. These implementations used the existing codebase of the games, and complex methods and systems to produce good results. Our implementation will be more focused on the basics of an adaptive system; answering the question of how to build one from the ground up.

1.1 Aims of this paper

The main aim of this paper is to determine how to best utilise reinforcement learning to facilitate the implementation of an effective adaptive AI system into the Unreal Development Kit. The paper examines the basics of reinforcement learning and then delves into the different methods available and the improvements that could be made to it in order to bring it closer to our goal of an adaptive combat AI. When determining which of the methods to use we also consider the attributes; quick adaptation, experimentation, and fair difficulty.

2. TECHNICAL DETAIL

2.1 Introduction to reinforcement learning

Reinforcement learning is a type of machine learning. Machine learning is the construction and study of computer systems which can learn from data. Reinforcement learning is a system for learning through interaction with an environment. It can be compared with how humans learn to do things as children. We interact with our environments and record the outcomes with our senses; we use these outcomes to modulate our behaviour and influence our future actions. A good example of this is a child touching a hot stove. When the child touches the stove their hand gets burned. They have learned from this interaction with their environment that touching the hot stove produces pain, so they don't do it again. This is indicative of how reinforcement learning works. It is an attempt to model this method in a computational manner.

This method of learning through interaction suits the interaction-driven nature of video games.

2.1.1 Basic computational representation

At its most basic level reinforcement learning is represented through the use of states, actions, and rewards (Csaba S, p.3, 2009). The state contains all the relevant information about the agents environment (things like their health, their distance from the enemy, and whether they are prone or not). The actions are all of the things that the agent can do from this state; performing an action transitions the agent from their current state to a new state. Rewards are a scalar quantity associated with each state. The goal of the agent is to accumulate as much of this reward quantity as possible. The agent uses this method of reward accumulation to determine which action it should take at any given moment.

These three elements form the basic structure of a reinforcement learning system, without them it would be impossible to implement. However, there is an improvement that can be made which will much improve the efficacy of the adaptive system; adding a value system.

2.1.2 Value systems

If an AI uses reinforcement learning but only utilises the basic system of rewards as incentives, then it will become very predictable. This is because the agent will always choose actions greedily. That the agent will choose greedily means that the agent will always choose the reward with the highest number. It will always take the route that it thinks will lead to the highest reward. This shows a lack of planning or ability to adapt over the long run. What we desire from our adaptive system is for the agent to maximise its reward in the long term. This is where the value system comes in.

A value system is a way to maximise an agents reward over the long term, as opposed to the short term scenario faced when only using rewards to determine choices. Values themselves are scalar quantities which are used in lieu of rewards when determining what actions to take. Because of this values can seem identical to rewards at first glance. One of the key differences, however, is that values can be changed by the agent; whereas rewards cannot. (Sutton, R., Barto, A, p.7, 1998) Values are also different from rewards in that they indicate the long term viability of choosing that particular action, not just the immediate return. An example of this long term reward maximisation is if an agent is confronted with two possible actions; one has a low immediate reward, but leads to many high reward actions, whereas the other has a high immediate reward, but leads to many low reward actions. If an agent were to simply choose greedily then they would choose the high reward action, but if they were trying to maximise their long term reward they would take the low reward action, because it would lead to the most accumulated reward in the long term.

Value systems and the optimising thereof are a big part of reinforcement learning and a large field in their own right (Harmon, M. E. & Harmon, S. S., 1996). We will be focusing on two aspects for our value system; coming up with the actual values for the system, and refining these values to better suit the needs of our game.

Coming up with the values

There are many ways to assign values to our actions. The first and most basic is to simply assign an arbitrary, user-chosen number, which stands as the value for that action. That number would then be modulated up or down depending on an agents interactions. This is not desirable, it does not improve over the base reward system, the numbers do not truly reflect the long term accumulation of reward.

Another way is to simply obtain the total reward gained when trying an action. This number is then divided by the amount of times that action has been performed. This results in an averaged value which incorporates the reward for each action and the long term results from choosing that action. This is a neat and simple value estimate which satisfies our criteria for a value system.

There are some points in the implementation where we add to an actions value despite the agent not performing that action. An example of this is when the agent is hit in a direction it is not blocking in. We exploit our own knowledge of this and have the agent add to the value of the block function for whatever direction it was attacked in. These small modulations do not impede the overall structure of the adaptive system, they simply improve upon an area where the system is weak.

This is a good starting point for our value system; it has a strong basis in reward and modulates according to how actions have performed in the past. However its ability to maximise reward over the long term is still fairly weak, and it still falls into the problem of constantly choosing the action with the highest number associated with it, even if this number does modulate over time. These are problems we must fix if we are to fulfil our goals for this adaptive system.

2.2 Refining the solution

The basic system of reinforcement learning (with the improvement of a value system) is now in place, but it does not fulfil the goals that we have laid out for our system by itself. The goals we laid out for our adaptive system are as follows: quick adaptation, experimentation, and fair difficulty.

So far we have adaptation through our value system; the values for each action change up or down depending on the average reward obtained when performing that action. This causes the agent to adapt, but it does not do it very quickly.

We do not really have any experimentation yet, the agent will simply choose the state with the highest value number every time it has to take an action.

Fair difficulty is an agent that will fight at a level that is competitive with the current player, it won't outstrip him and it won't be a pushover, it will be a challenge. This is also not in the current system, the agent will always try to find the optimal path for his current situation. It is possible that adjusting the other two attributes (quick adaptation and experimentation) will provide this fair difficulty, but that is yet to be determined.

2.2.1 Basics of exploration

Exploration is how we will implement experimentation into our adaptive AI system. With our current system the agent will continuously adapt as the fight goes on, but they will do so greedily. This will bring it closer to a successful outcome but will mean it ignores other, initially lower value states, which might eventually lead to the optimal solution.

The agent needs to occasionally choose a different state than the one it thinks is best; in order to see what overall reward it can gain from alternate paths of action. This method is called exploration. It allows the agent to explore other, initially less optimal routes in the hopes of finding an ultimately superior option. Exploration works in conjunction with the value system from before, adding to it and allowing it to maximise its long term reward through experimentation. Without exploration this would be very difficult to implement.

The simplest way to implement exploration into our current solution would be to choose a random action every set number of turns, say every second turn. A turn in this case being a point where the agent has to make a decision on which action to perform next. This is better than purely greedy action selection, but is still too predictable to best serve our aims.

In addition to this we can make one small adjustment which will improve the results of our exploration actions. The adjustment lets them directly influence the preceding steps that led to them, using this equation:

$$V(s) \leftarrow V(s) + \alpha\Big[V(s') - V(s)\Big],$$

Fig. 1 (Sutton, R., Barto, A, p.12, 1998), "V(s)" is the value of the action after the exploratory action, "V(s')" is the value of the exploratory action and "a" is a small positive fraction called the step-size parameter.

The equation in Fig. 1 is used the turn after every exploratory action in order to modulate the value of the exploratory action. It brings the values of the two actions (the exploratory one, and the one after it) closer together. How much closer they get depends on the step-size parameter. The higher it is the quicker the "learning" takes place. So if the step-size parameter is one, then the change will be as abrupt as it can be, making the two values identical to each other. Upon implementation this was found to increase the efficacy of the adaptive system, allowing the agent to better maximise its long term reward gain.

2.2.2 Balancing exploration

Exploration is an integral part to our reinforcement learning system, but it is not perfect at the basic level we have implemented it at. Right now there is no explicit balance between exploration and exploitation (exploitation being choosing the greedy action); exploration is simply something that happens occasionally to a random action. We must have a way to choose the optimal action. Two methods for balancing exploration through action selection are considered, the epsilon greedy method and the softmax action selection method.

Epsilon Greedy

Epsilon greedy is an action selection method in which the selection process is mostly greedy, but which will occasionally select a random action from those available, independent of the value of any action (Coggan, M., p.4, 2004). This random factor is controlled by the variable epsilon. It is typically a low value, as otherwise the rate of exploration would be too high and the solution would act sub-optimally. This is because

as epsilon increases the rate of exploration increases, and the rate of exploitation decreases. More time exploring means less time actually selecting actions which provide the most reward, which ends up giving a poor result. Having a high epsilon would not suit the aims of our system. With a high epsilon the agent would explore so much that it would not truly adapt to an opponent. This directly contradicts our goal of an adaptive system, so we must be wary not to set epsilon too high when utilising this method.

Epsilon greedy is a fairly simple method to implement (Munshi, J., 2012), but it solves the problem we faced with having a predictable exploratory component. It also gives equal priority to each action, which would be bad in cases where some actions are very undesirable. This is not the case for our system, however, as no action is ever so bad a choice that it must not be picked. The lack of discrimination due to value is good, in fact, as it aids our goal of experimentation. Because of these reasons epsilon greedy is a good choice, it solves our issue with correctly balancing exploration.

Softmax Action Selection

The epsilon greedy method for action selection chooses randomly from all available actions. It gives as much priority to the almost-best actions as it does to the worst value options. This is not ideal if we want to keep the agent from choosing terrible options, infusing a little more intelligence into its actions. If an option is very bad then no human would realistically pick it, thus prioritising the higher value options will lead to a more human seeming agent.

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}},$$

Fig. 2 (Sutton, R., Barto, A, p.30, 1998), shows the softmax action selection equation.

Softmax uses the value of an action (Qt(a)), the sum of the values from the considered actions (the bottom row), and a parameter called temperature (r) to determine the probability of choosing the state with the highest reward. As the temperature increases actions become equally probable, as the temperature decreases the action with the highest reward becomes more and more certain to be chosen (it moves closer and closer to the purely greedy approach to exploration).

The softmax action selection method seems like a good general approach to building an agent who responds reasonably to different options of varying value. The agent will often choose the next highest option given a low temperature value. While seemingly good in general, in our case this is not a good thing, as it will hamper the agents ability to experiment. This is because of the limited amount of actions available to the agent. Even if the temperature value is adjusted to make all actions more viable, the softmax equation does not adequately meet the needs of our system.

Of the two options, epsilon greedy and softmax action selection, epsilon greedy was proven to be more in line with our goals. Implementing it into the system gives a noticeable improvement to the exploratory aspect of our implementation.

2.3 Implementation of system

Now that all the necessary attributes of our system have been analysed and decided upon, this paper will explain how it was implemented using unrealscript in the UDK.

The code obtained from following the lessons in Game Programming with Unrealscript (Cordone, R., 2011) was used as a base upon which to build the required functionality. Utilising multiple classes, a simple system of interaction was set up between the player and the enemy in the scenario. Each had access to a number of commands, which were to attack and block in three different directions (up, left, and right), the enemy could also crouch, kick while crouched, and stand up from the crouch. The attacks decremented health from the victim, the blocks countered their respective directions attacks, and the kick stunned the player for a variable amount of time.

Using these interaction abilities, and HUD notifications, the adaptive system was built into the code. The result is an enemy who reacts to your interactions and tries it's best to counter your attacks and defeat you (rewards are given to it when it either hits you or correctly responds to your attacks).

3. CONCLUSION

Utilising adaptive AI for combat in games is important, it provides a living enemy who reacts less mechanically than a normal computer controlled opponent. Adaptive AI can be achieved through the use of reinforcement learning, which is a method of machine learning where the agent learns through interaction with its environment. Reinforcement learning is flexible, and has many different methods to choose from to best achieve a desired goal. It was easily able to accommodate two of the three attributes we wanted our system to have. The third, unfulfilled attribute, fair difficulty, was too complex to achieve within the scope of this paper; it could be a research topic in and of itself. The best we could achieve was an adapting opponent who reacted quickly enough to not be very easily defeated.

A reinforcement learning system must have three things as a base; states, actions, and rewards. We improved upon this base by including a value system, which allowed the agent to maximise his reward over the long term. We then improved upon the value system by adding exploration; which is where an agent will sometimes choose a random option, instead of the one with the highest value. This is so the agent can find out if there are better actions it could take, as well as adding a touch of unpredictability to the agents behaviour.

The last improvement we made was to the exploration. We wanted to balance exploration and exploitation. We analysed the epsilon greedy and the softmax action selection methods and discovered that the epsilon greedy method more closely matched our goals for the system.

The successful implementation showed that reinforcement learning is a viable solution for building an adaptive combat AI. It is flexible and effective, and integrates well within the structure of the UDK. It is necessary to invest some time into analysing different methods and choosing whichever ones best suit the goals of your adaptive system. This research will allow for a better system with fewer weaknesses for players to exploit.

REFERENCES

Book

Sutton, R., Barto, A., 1998. *Reinforcement Learning: An Introduction*. A Bradford Book, Massachusetts, USA.

Cordone, R., (2011). *Unreal Development Kit Game Programming with UnrealScript: Beginner's Guide*. Packt Publishing, Birmingham, UK.

Conference paper or contributed volume

Graepel, T. et al, 2004. Learning to Fight. *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*. Reading, England, pp. 193-200.

Websites

Harmon, M., E. & Harmon, S. S. (1996). *Reinforcement Learning: A Tutorial*. Available: http://www.nbu.bg/cogs/events/2000/Readings/Petrov/rltutorial.pdf. Last accessed 29/12/2012.

Csaba, S., (2009). *Reinforcement Learning Algorithms for MDPs.* Available: https://www.cs.ualberta.ca/system/files/tech_report/2009/TR09-13.pdf. Last accessed 03/01/2013.

Coggan, M., (2004). *Exploration and Exploitation in Reinforcement Learning.*Available: http://www.cra.org/Activities/craw_archive/dmp/awards/2004/Coggan/FinalReport.pdf. Last accessed 03/01/2013.

Munshi, J., (2012). *How to implement epsilon greedy strategy / policy*. Available: http://junedmunshi.wordpress.com/2012/03/30/how-to-implement-epsilon-greedy-strategy-policy/. Last accessed 04/01/2013.

Micic, A., Arnarsson, D., Jonsson, V., (2011). *Developing Game AI for the Real Time Strategy Game Starcraft.* Available: http://skemman.is/stream/get/1946/9882/22934/1/Final_Report.pdf. Last accessed 23/12/2012.