3 Dual-Utility Reasoning

Kevin Dill

3.1 Introduction

Utility-based approaches to decision making examine the situation in the game at the moment a decision is being made, calculate the *goodness* (which is often called things like utility, priority, weight, or score) of each option using a heuristic function, and then drive decision making using that value. This can be contrasted against purely Boolean decision-making approaches, such as the typical finite state machine (FSM) or behavior tree, which evaluate a series of black-or-white, yes-or-no questions in order to select an option for execution.

The advantage of utility-based AI is that it is better able to account for the subtle nuance of the situation when making its decision. In other words, in situations where more than one option is valid, utility-based AI will base its decision on an evaluation of the relative appropriateness and/or importance of each option instead of picking at random or simply taking the first valid option that it finds. At the same time, because the heuristic functions are hand-authored, the game designers retain reasonably tight control over the behavior of the AI and can thus better ensure that the player's experience will fit with their design.

One point that cannot be overemphasized is the importance of calculating each option's utility in game, at run time. It is not enough to assign fixed weights to the options a priori. Only by evaluating them based on the situation at the moment when the decision is being made can you achieve the responsive, dynamic behavior that utility-based AI promises.

3.2 Dual-Utility Reasoning

There are two common ways of using utility to make a decision. The first, which we will call *absolute utility*, simply evaluates each option and takes the one with the highest score. The second, *relative utility*, selects an option at random, but it uses the score of each option to define the probability that it will be selected. The probability for selecting an option (P_O) is determined by dividing the utility of that option (U_O) by the total utility of all options:

$$P_O = \frac{U_O}{\sum_{i=1}^{n} U_i} \tag{3.1}$$

This approach is referred to as *weight-based random* and is implemented as follows:

1. Add up the total weight of all valid options (i.e., those with a weight greater than 0).
2. Pick a random number between 0 and the total weight from step 1.
3. Go through the valid options one at a time and reduce the random number by the weight for each option. If the result is less than or equal to zero, pick this option. Otherwise, continue on to the next.

Relative and absolute utilities each have advantages and disadvantages. Absolute utility is best when you want to ensure that the AI always takes the most appropriate action available. Unfortunately, it can become predictable since, given a particular situation, it always does the same thing. In contrast, relative utility avoids the rigid predictability of absolute utility. By picking at random, it ensures a certain amount of variation, while still giving preference to the most appropriate choices. Nevertheless, while it *prefers* to pick good options, there is always some chance that an option with very low utility will be selected. This can easily make your AI look stupid. We can reduce the chance that a low-weight option will be selected by squaring the weights or eliminate it by screening out the lowest weight options, but this quickly becomes a balancing act that is hard to perfect and harder to maintain.

Dual-utility reasoning is an approach that combines absolute and relative utilities into a synergistic whole. It avoids the weaknesses of both approaches by combining them together and is also more flexible and expressive for the game designers.

The big idea is that rather than assigning a single utility value to each option, we assign two: a rank and a weight. *Rank* uses absolute utility to divide the options into categories, and we only select options from the best category. *Weight* is used to evaluate options within the context of their category. Once we've found the category with the highest rank, we select from among the options in that category using weight-based random.

The algorithm for actually selecting an option is as follows: First, go through all of the options and eliminate any that have a weight that is less than or equal to zero. These options can't be selected by the weight-based portion of the reasoner anyway, and eliminating them now makes the process simpler. In addition, this gives the designers a convenient way to mark an option as *invalid* or *inappropriate given current circumstances*. If an option shouldn't be selected, simply set its weight to zero and it will be rejected regardless of rank.

Second, find the highest *rank* among the options that remain, and eliminate any options whose rank is less than this. Again, what we are doing is finding the most important category and eliminating the options that don't belong to it.

Third, find the highest *weight* from among the options that remain, and eliminate options whose weight is less than some percentage of this. This step is optional, and the percentage that is used should be configurable on a per-decision basis. Conceptually, what we are doing is finding the options that really aren't all that appropriate (and thus have been given very low weight) and ensuring that the random number generator doesn't decide to pick them anyway. What remains are plausible (not stupid) options.

Finally, we use weight-based random to select from among those options that remain.

Once again, the four steps are as follows:

1. Eliminate the options with a weight of zero.
2. Find the highest rank category, and eliminate options that don't belong to it.
3. Find the best remaining option (i.e., the one with the highest weight), and eliminate options that are much worse (i.e., much lower weight) than it.
4. Use weight-based random to select from the options that remain.

As described earlier, the major strength of dual-utility reasoning is that it allows you to divide your options into categories and select from among those categories using absolute utility. Once this is done, we can use relative utility to pick from among the options within the best category in a random but reasonable way.

3.3 Dual-Utility Reasoning in *Zoo Tycoon 2*

In *Zoo Tycoon 2* [Blue Fang Games 04], the AI for both the animals and guests used dual-utility reasoning. Most of the time, the characters would select options based purely on their needs (i.e., their hunger need, entertainment need, and bathroom need). These options all had a rank of 0, but their weight would vary depending on the current situation.

There were times, however, when characters were in a specific situation that had a distinct set of behaviors, and only those behaviors were appropriate. For instance, if a koala climbed up a tree, then it would have a number of tree-climbing behaviors available (including some that allowed it to get out of the tree), and these behaviors would have a higher rank (typically around 5) to ensure that the animal didn't pick some other behavior and pop out of the tree unrealistically.

Taking it a step further, the *Marine Mania* expansion pack introduced the idea of marine animal shows. These were shows where the player could train dolphins to jump through hoops, seals to play with a ball, and so forth. It was very important that when one of these shows occurred, both the animals and an appropriate number of guests showed up so that the player was rewarded for their hard work training the animals. Furthermore, the behavior of the characters during the show was very heavily scripted. The animals would go through a specific sequence where they would come in to the show tank, swim over to the trainer, wait for the whistle, do their trick, go back to the trainer, get a treat, wait for the whistle again, do their next trick, and so forth. The guests had to come to the entrance, wait in line, go in and sit, react appropriately through the show, and then exit in a timely fashion.

In order to make sure that the right behaviors were picked at the right time, we simply gave them appropriate ranks (which were all between 98 and 102) so that the *best* action at any given moment would play. These ranks were coordinated by a simple FSM, although they could have been scripted or set in other ways.

Finally, there was one behavior that took precedence over every other behavior and that was dying. We didn't want any animals coming back from the dead, so the die behavior had a rank of 1,000,000, ensuring that it trumped anything else that might happen.

3.4 Conclusion

Utility-based AI is a term used to describe approaches that, rather than using purely Boolean logic, use a heuristic function to evaluate the appropriateness of each option given the moment-to-moment situation in the game and base their decision on the resulting score. These approaches typically either take the option with the highest score or use weight-based random to pick in a way that is nondeterministic but still gives a greater chance of being selected to the *best* options.

In this chapter we introduced the idea of dual-utility reasoning. This approach combines absolute and relative utilities. Absolute utility is used to divide the options into categories and ensure that only options from the most important or most relevant category will be selected. Once that is done, relative utility is used to pick at random from among the options within that category. This helps to prevent both the predictability of absolute utility and the occasional poor choices of relative utility. In addition, it provides a bit more flexibility and expressiveness to the game designers who are responsible for configuring the AI.

Reference

[Blue Fang 04] Blue Fang Games. 2004. *Zoo Tycoon 2*. Redmond, WA: Microsoft Games Studio.