Behavior Decision System using Evaluation Trees and Scoring Architecture

Iker Rodrigo and Pablo Riesco

1 Introduction

In the wild animal world, there are species that behave completely differently from each other; they eat different food, would rather eat or drink less, or may opt to attack other animals for territory or survival. But all animals share the same objective: survive.

In this research project, we will explore a system that could be able to give logic to AI to simulate this kind of environment: with the use of a behavior decision system using evaluation trees and scoring architecture. We will explain how this system works, and what modifications can be applied for different animal behaviors.

2 Behavior Decision System's Assumptions

In order for the behaviour decision system to work, some assumptions related to the AI's actions and decision capabilities are established to simplify the process. These assumptions are:

1. At any point, our AI will have a finite amount of actions that it can do.
2. Our AI will not be able to perform more than one action at a time.
3. Each action must be evaluated to give a numerical *utility value*.
4. Two actions can not have the same *utility value*.

Utility Value: The value of an action. The higher the *utility value*, the more priority that action will have.

Taking these assumptions into account, a simple process sequence for the behaviour decision system can be used. Evaluating each possible action by the AI, a score can be given to each, where the most important actions will have the highest value, while the less prioritizing or less possible actions will have the lowest values. Then, the action with the highest utility value will be the one performed by the AI.

Being able to evaluate the AI's actions using the utility value means that a designer of the game can determine which actions will be most important to each AI. This also means that different AI could have the same actions, but a different priority, giving alternate behaviour for a variety of circumstances or for a range of similar but non-identical AI archetypes.

3 AI Structure

As previously established, the AI will have a finite amount of actions, where only one can be executed simultaneously. In order to better explain the behaviour decision system, an example game scenario will be used: a wild animal life simulation.

3.1 Possible actions

In the animal wildlife simulation, every animal can perform the same actions, no more, and no less. The possible actions are:

1. Eat
2. Drink
3. Attack another animal
4. Die
5. Reproduce
6. Spread out
7. Walk

Each action represents a function included in each animal structure:

Listing 1 Struct of an animal AI.

```
struct Animal
{
    void setActionToHighest();
    void eat();
    void drink();
    …
}
```

This means that every animal will have each action available at all times. However, in wildlife, there are some actions that animals of a certain type would prefer more than others. For example, a carnivorous lone lion would be more mindful of other lion's territory and spread out from them, and would rather eat other animals than to eat fruit on the ground. This is where the behaviour decision system comes into play: a carnivorous animal may have the ability to eat a fruit in the ground, and it may be a valid option, but due to the nature of the behaviour system, the carnivore's option to eat an animal will have a higher utility value than to eat a fruit if an animal is around. To get these values, all of the animal's actions evaluations trees are analyzed.

3.2 Evaluation tree analyzer and structure

Each action has one evaluation tree corresponding to it. This tree's only function is to determine how much utility value is added to the action, giving it its priority. However, something else is required to structure and store these results. To accomplish this, a new control flow node is used, that will run all of the evaluation trees without stopping, and store the action's utility value inside the animal once each evaluation tree (each child) is fully processed. The results are stored in a structure which contains the action and its score, while the results of all the actions are stored in a vector of the animal.

Listing 2 Result of a behavior tree of an action.

```
struct evaluation
{
    float score;
    action a;
}

struct Animal
{
    vector<evaluation> scores;
    (...)
}
```

4 Calculating Utility Value

There is one more question which is still unanswered: how do we determine the amount of utility value we are giving to each action?

4.1 Evaluation trees

As previously mentioned, each utility value is computed using an evaluation tree. To compute this utility value we will be using a score system. The evaluation tree is run, processing nodes that answer yes or no questions. Depending on the answer, a score may be added to a total. The final result will be the tree's action utility value.
Here is a simple example of two evaluation trees for the eat and drink action of an animal:

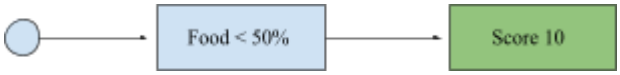

Figure 1 Behavior tree of the eat action

Figure 2 Behavior tree of the drink action

In the case where the animal is both "hungry" and "thirsty", when both trees are processed, the drink action's utility score (15) will be higher than the eat action's utility score (10), meaning that the drink action will be more prioritized.
This might work in the case where all the animals have the same priorities. On the other hand, if we attempt to simulate real animal behaviour, some animals will prioritize one action over the other depending on how long they could resist without doing this action.
To simulate these animal priorities, each animal has what we call *influence values*, which include both the animal's hunger resistance and thirst resistance.
When taking each decision, being hungry or thirsty may not be enough of a question to determine if to complete the action or not. The distance to where the action is performed is also something that should be considered, as actions that can be completed faster should also have higher priority. Adding more questions to the eat action's behaviour tree, we get

something more like this:

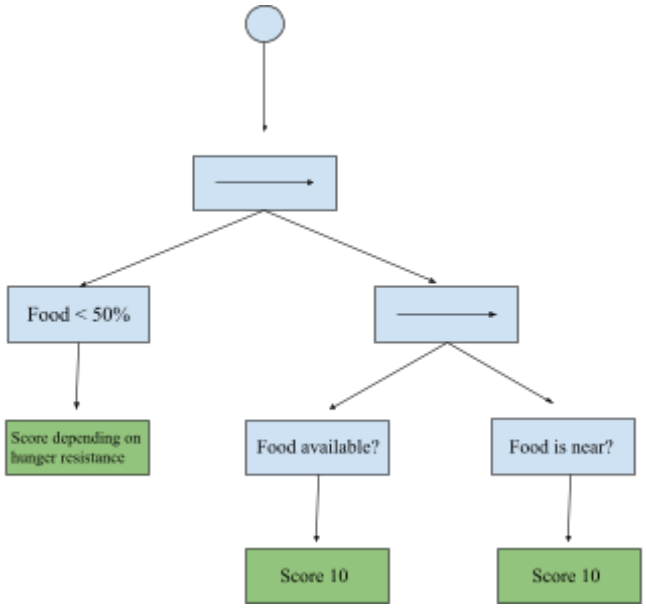

Figure 3 Complete tree of the Eat action

In order to compute the score depending on the hunger resistance, the following formula is used:

$$score = base + (2 * maxStatValue / foodResistance)$$

Note: *maxStatValue* is 100 and *base* is the minimum value that should be added.

The higher the $foodResistance$, the smaller score that will be added, since having a high hunger resistance means the animal can survive more time without eating.

4.2 AI Influence Values

More *influence values* can be added to the animals so that each one can have its own priorities in every circumstance. The following table shows statistics and influence values that could be added to create different behaviours:

Stat	**Explanation**	**Influence Value**	**Action**
Hp	If reaches 0, dies	None, depends on food, water and age	Eat and drink
Food	Slowly lowers over time	Hunger resistance	Eat

Water	Slowly lowers over time	Thirst resistance	Drink
Social	Depends on the number of animals of its own species which are around	Friendliness	Spread out from your species
Population	Depends on the number of animals that are around (all species)	Peacefulness	Reproduce
Confort	If the confort is high the animal will reproduce if is low it will attack other animals	None, depends on the hp, social and population values	Reproduce or attack other animals
Age	Higher the value faster will decrease the hp	None	Die
Adaptation	Determines how much a species influence value can change	Will be less the more the animals reproduce	None

Animals would primarily decide their actions looking at their stats, as the animal's primary goal would be to survive. Alternatively, *influence values* should also determine the action to take, as this is what makes each animal's behaviour different.

5 Execution Step

The action execution is straightforward. After evaluating all the trees, the animal's action is simply set to the one with the highest utility value using the following function:

Listing 3 Picking the action with the highest score.

```
void setActionToHighest
{
    score = 0.0f;
    for(auto it = scores.begin()...)
    {
        if((*it).score > score)
        {
            current = it.a;
            score = (*it).score;
        }
    }
}
```

6 Conclusion

The BDS based on a score architecture is simple to understand and implement into AI behaviour in games. In situations where other behaviour systems fail, the use of a score architecture shines. For example, situations where AIs in a game have a huge variety of actions they can take, the BDS with score architecture handles it very well, as implementing a new action just needs adding a new tree, and implementing new situations just needs a new node on a tree, while in other structures, new situations or actions may require a whole reorganization of an AI's logic.

BDS with a score architecture can be used in games such as *Sim City, Cities: Skylines* and *Jurassic World*, where decisions are fundamental for the game's inmersion, and can also work very well in turn-based combat games where there are hundreds or thousands of different actions or attacks that AIs may have. Before attempting to go towards the most common decision systems, consider the use of BDS with a score architecture, and you may save head scratching problems in the future.

7 References

7.1 Online Documents:

The third volume in the Game AI Pro book series (published June 2017).Behavior Decision System Dragon Age Inquisition's Utility Scoring Architecture provided by Taylor & Francis http://www.gameaipro.com/GameAIPro3/GameAIPro3_Chapter31_Behavior_Decision_System_Dragon_Age_Inquisition%E2%80%99s_Utility_Scoring_Architecture.pdf