Goal-based AI in RPGs

J. Goldblatt

j.goldblatt@digipen.edu

The RPG genre as a whole lacks variety in its use of AI. Traditionally, enemies follow two patterns: AI Roulette and the ever-present state machine. The former “strategy” requires virtually no work, as the enemies select an action randomly. Players typically encounter randomly-acting agents in standard battles. Because these battles are supposed to finish quickly, any intelligence in the enemies will likely slip past unnoticed by players. Designers instead carefully craft a state machine providing a challenging experience for the player that reacts similarly every instance of the encounter. Combining the two techniques where the list of randomly-selected skills the enemy uses changes based on the current state is also prevalent.

More advanced AI techniques for controlling enemy agents is unnecessary and might even be construed as unfair. In this genre, a killer heuristic already exists! The most optimal strategy targets the healer first and then mop up the rest of the party without too much trouble. Usually the healer's frailty makes this approach even more attractive. Watching players in MMORPGs with a PVP aspect lends even more evidence, as they form an assist train, target the squishy characters (the healers and glass cannons), and lastly focus fire on the tougher opponents. For a single-player experience, creating an AI utilizing this policy likely will not be a fun experience.

But what about the player's allies? An increasing trend in RPGs of late removes direct control of party members. The player instead focuses on their personal avatar and the battle as a whole rather than micromanaging the party. Persona 3 (Atlus 2006), Final Fantasy XIII (Square Enix 2009), and the Mass Effect series (Bioware 2007) strive to make the player's allies autonomous. However, the player did not lose all control, as tactics may be set to prod the agents in the right direction.

Moments of artificial stupidity can be fatal and frustrating to the player. A goal-based system controlling the party provides a unique simple to implement approach, but tuning the system leads to potential nightmares.

The Goal-based Approach

A goal-based architecture revolves around two atomic pieces: Goals and actions. A goal, much like in real life, is simply something that one strives to accomplish. Performing an action effects a change in the world that leads to satisfaction of the goal. In other words, when the AI asks what to do, it follows its goals and chooses how to accomplish the goal via the actions.

Insistence

One of many approaches for managing goals involves the concept of insistence. The insistence for a goal implies how badly the AI needs to satisfy that goal. Traditionally, a higher insistence value reveals a more pressing goal and the agent must act to resolve the situation. A value of zero for insistence represents a completely satisfied goal.

The AI uses actions to reduce the insistence of goals. While determining what to accomplish next, the agent needs to know how much the action will reduce the insistence (i.e. satisfying the goal). This notion of changing the insistence forms the backbone of the algorithm.

Implementation

Both goals and the selection algorithm are trivial to implement. All a goal needs is some insistence value and some way of measuring actions:

```
struct Goal {
    float insistence;

    float delta(const Action& action) = 0;
};
```

The pure virtual function `delta` returns the change in insistence that applying `action` would provide. (The action is not performed yet.) The `Action` class is omitted since it consists entirely of game-dependent details.

So, with a list of goals and actions, the agent needs some way of selecting an action. But to choose an action, the AI must first find a goal to satisfy. Since the goal with the highest insistence represents the most pressing goal, the agent should find actions that accomplish that goal. With that in mind, the algorithm for selecting an action reduces to two max-style loops:

```
Goal* goals[numGoals];
Action actions[numActions];

Goal* mostPressing = goals[0];

for (int i = 1; i < numGoals; ++i)
    if (mostPressing->insistence < goals[i]->insistence)
        mostPressing = goals[i];

Action bestAction = actions[0];
float greatestChange = mostPressing->delta(bestAction);
```

```
for (int i = 1; i < numActions; ++i) {
    float delta = mostPressing->delta(actions[i]);

    if (greatestChange < delta) {
        greatestChange = delta;
        bestAction = actions[i];
    }
}
```

The great thing about this algorithm is the blazing speed. The runtime efficiency is O(g + a), where g and a are the number of goals and actions, respectively. Also, it only requires memory for some temporaries. However, this approach has a few insidious drawbacks.

Back to RPGs

Players worry about a lot of different statistics while playing an RPG. The most important is hit points (HP); without it, they are toast. Since a player places a high priority on this resource, the AI should follow suit. If at all possible, the agent should not leave itself in a vulnerable position.

The AI needs some goals to satisfy and actions to perform. Actions are easy; they are the skills they use each turn in the battle sequence. Goals remain the biggest piece of the puzzle and the most challenging to tune to achieve sensible results.

Two separate behaviors need to fire appropriately. One is to support both itself and its allies along with attacking any enemies on the battlefield. The most problematic portion of designing an AI in this context involves the switch between these two disparate behaviors. So, how about designing goals for these behaviors specifically? Say, a "Support" goal and an "Attack" goal.

It's a Trap!

To make these goals interact with the action selection algorithm, the insistence values need to be generated during the character's turn. To simplify matters, the "Support" goal for right now just implies healing. (A later extension includes buff or debuff abilities.) When the agent (or anyone else, for that matter) has low HP, the "Support" goal should be more pressing. Therefore, generate a higher insistence in perilous situations. Then the insistence yielded by the goal uses the total damage received in some way.

A quick philosophical side note, as this affects the rest of the design. In some RPGs, values such as current and max HP of an enemy are hidden from the player. Even in these cases, the AI-ally should have access to these values. Making a poor choice should be avoided at all costs, even if it means the AI "cheats." Operate as if a wrong move leads to a game over.

With that out of the way, the AI queries the current and maximum HP of enemies. Why

are these statistics important? To measure relative power levels and ease of destruction (and hopefully satisfy this nebulous “Attack” goal).

The next step boils down to an interesting choice. Consider this situation: The party faces a multitude of enemies of varying ferocity. Say, a boss battle with a few additional mooks thrown in for good measure. Is it better to take out the weaker enemies then focus on the boss, or just burn the boss down quickly? Even to players, the strategy boils down to the individual encounter. Pick a behavior and stick with it or make the behavior swappable. For the rest of this discussion, the AI targets “weaker” enemies first.

Just as a quick guesstimate, in the boss battle example the HP values are pretty disparate. As such, a decent measure of power level can be gathered from comparing the HP of the enemies. For now, the AI-ally will use the enemies current HP to determine insistence in some way. (Exactly how is not relevant quite yet.)

Quick recap: The agent considers the “Support” and “Attack” goals. Both use HP of various characters to determine the insistence. Recall that the agent yearns to switch between two behaviors with these two matching goals. To choose between attacking and healing, the AI selects the goal with the highest insistence.

Finally, a problem. The “Support” goal uses the party's HP in some way, while the “Attack” goal examines the enemies' HP. These values have absolutely nothing to do with each other! Normalizing these values by considering the HP as a percentage still does not work. Take again, for instance, the boss battle. The percentage HP for the player's will fluctuate wildly while the boss just gets whittled down over time. Utilizing the HP statistic in this way is doomed to fail, but this brings up a key point. *Goals are not behaviors*. Goals engender behaviors.

All is not lost, but a new approach is needed. The main issue with this approach involves the viewpoint (on two different accounts). First, no amount of tuning will save these goals. The ideas behind them are important, though. Secondly, the selection algorithm focuses on choosing an action that looks good. But there is another way...

Making Actions Look Bad

So one of the main issues with the previous methodology involves the aspect of choosing the best looking action. The agent should still do that, but what if goals made actions look worse?

A notion slightly related insistence is discontent. In some contexts, they may be one and the same. Discontent is best described as an “unhappiness” metric. As the AI-ally considers actions, it rates the state of all of the goals and chooses the action that makes it happiest. The previous algorithm did not account for side effects of actions. By considering all of the goals for each action, it finds the action that best satisfies all of them.

Implementation

The flavor of the algorithm is pretty similar, but the structure of goals and actions change:

```
struct Action {
    float discontent;
};

struct Goal {
    void getDiscontent(const Action&) = 0;
};
```

Dropping insistence, a goal now calculates a discontent value for the action. Note that it is not strictly necessary for an `Action` to store a discontent value, but proves invaluable for debugging. Enumerating and seeing the discontent values greatly decreases tuning time.

Recall that discontent is an unhappiness metric, but the agent selects an action that leaves it in the best overall state. Therefore the algorithm aims to minimize the discontent value:

```
Action actions[numActions];
Goal* goals[numGoals];

for (int i = 0; i < numActions; ++i) {
    float sum = 0;

    for (int j = 0; j < numGoals; ++j)
        sum += goals[j]->getDiscontent(actions[i]);

    actions[i].discontent = sum;
}

Action bestAction = actions[0];

for (int i = 1; i < numActions; ++i)
    if (actions[i].discontent < bestAction.discontent)
        bestAction = actions[i];
```

Still pretty simple, but loses some efficiency. Runtime is now O(ga), where g and a are goals and actions, respectively. Fortunately, like the previous algorithm, it only requires temporary storage.

The severity of the runtime hit is really application dependent. For the case of RPGs, the number of actions will be pretty limited since the agent only considers what it can accomplish right now. The added power of considering side effects of actions comes at a cost, unfortunately.

An interesting quirk with the discontent approach occurs when writing the `getDiscontent` function: The final value is usually taken to some power. Linear functions do not cause actions to look bad fast enough. Squaring the value usually works, but sometimes getting good results requires a variety of different powers. Be careful with odd powers and the potential for negative values; these can wreak havoc with the selection process.

Actions

Before continuing, actions need to be defined a bit more precisely. Earlier, actions were stated to be the skills the character can use during its turn. This is almost correct, but misses a crucial piece of information. In RPGs, skills are not used in a vacuum, but instead act on some target. Encapsulating the target information aids in the selection process, because the agent no longer discerns how to use the skill.

Adding targeting information to actions reduces the amount of clutter in the goal code and code duplication. Generating the actions is a completely separate process from considering them. Using some higher level system to enumerate all of the potential actions the agent could perform simplifies the entire architecture. Also, only send legal actions to the agent. Why consider an action if the character lacks the resource to perform it anyway? Separating action generation from the goal architecture is an even better idea when the game has a charm mechanic that reverses the alliances of the character temporarily.

Now, since actions contain some information about how to use the skill, the number of actions increased. With a single target attack skill, the skill repeats across multiple actions, but all have a different target. For the rest of the discussion, only single target skills are considered. Extending the architecture to handle multi-target actions is not too difficult.

Goals

Again, the trick for this architecture requires some creative thinking about goals. The main problem is switching between healing and attacking. With the new vantage point, when does healing look bad? In other words, is it ever a bad idea to heal?

The answer is yes. The blatant case occurs when targeting a character with full health. Instead of wasting time and resources to heal characters with most or even all of their health, the agent should switch to doing something else. If it is not quite clear on how to discourage healing, one more case might help. What if the AI should be healing, but which heal should it choose if it has multiple skills that could work?

Say one of the characters lost 100 HP. Further suppose the AI-ally has two different healing skills, one that heals for 75 HP and a really expensive skill that gives 300 HP. The second heal seems like overkill in this case. Well, players coined the term “overheal” (popular in MMORPGs), and just so happens that is exactly what the AI

should attempt to minimize. With a goal that minimizes overhealing, the agent will choose the more appropriate heal in this case, but it also helps with the other issues. Overhealing must look so bad that the attack actions appear much more attractive.

When any player runs out of HP, they can no longer act. Another thing the agent needs is some sort of self-preservation instinct. Once it gets low on HP, actions that heal itself should look much better than attacking. Hence this new "Conserve HP" goal generates a higher discontent when in dangerous situations.

With a self-preservation instinct and some way of managing overhealing, the AI-ally switches between attacking and healing. The main idea in trying to create the goals related to attacking revolves around generating a lower discontent for actions affecting enemies with less maximum HP.

It would be remiss to not consider another very related case. Imagine the player battles three different instances of the same enemy. Their maximum HP matches perfectly, but the battle should proceed in a specific way. Death is the ultimate debuff. A dead enemy deals no more damage to the party, so the more optimal strategy focuses fire on each enemy in turn rather than in a round robin fashion. Leaving a weakened enemy alive should be strongly discouraged. As long as the agent utilizes both maximum and current HP when deciding who to attack, both cases fall under the same "Target" goal.

Wait! This last goal is needed for healing, too. The AI targets characters for healing that have lost a disproportionate amount of their HP. Killing two birds with one stone, the agent now differentiates which allied character is in more danger.

A remaining question involves how the "Target" goal interacts with the "Conserve HP" goal. "Conserve HP" will make an action look more attractive if the AI-ally heals itself so that it pushes itself out of danger, while the "Target" goal only considers the current target for that action. The focus of the skill may, in fact, be itself. Then when the agent is low on health, it looks doubly worse to not target itself. The discontent generated from the interaction between these two goals leads to correct behavior.

There is no kill like overkill. Using flashy, expensive techniques to annihilate enemies might be fun, but they are silly to use when the enemy is at low health. The idea relates to the notion of overhealing, but instead, the agent strives to minimize the amount of overkill. An interesting problem rears its head because of this goal, but it will have to wait. The "Minimize Overkill" goal might be omitted entirely if overkill happens to be a feature of the game, like in Valkyrie Profile: Covenant of the Plume (Square-Enix).

Putting it all together

Using only four, carefully crafted, goals, an AI-controlled party member demonstrates the ability to fight, and win, various battles. Furthermore, they even follow the prescribed tactics, such as focusing fire on weaker enemies first!

Four Goals

Many details were omitted when discussing the four goals. Slogging through numbers would have muddied the design process. Decide first on the behaviors, goals that make it possible, then, and only then, crunch the numbers. For now, ignore the power for the discontent calculation. That merits its own section.

Minimize Overhealing

To force overhealing to look terrible, this goal generates more discontent proportional to the amount of healing done that exceeds the maximum HP of the target characters. Also, an action that deals damage to the target should not generate any discontent, so some sort of `if` check will be involved. Then the calculation just involves the current and maximum HP of the target, along with the amount healed:

$$HP_{cur} + amtHealed - HP_{max}$$

However, this value will be negative when the character under heals. While the agent should prefer under healing, the negative value causes an unintended side effect. Calculating discontent involves summing across all of the goals. Then a negative value from any goal makes the action look more attractive.

But what does this actually mean in context? If an agent ponders between two heals of different magnitudes, neither of which overheal, the character prefers using the smaller heal because it generates less discontent by virtue of being more negative. However, the AI-ally should heal for as much as possible, while still avoiding overhealing. Using some sort of even power on the equation might lead to correct behavior, but causes really strange results with odd powers. Making sure the goals produce only positive values saves a lot of headache while tuning. Fixing this issue is pretty simple:

$$\max(0, HP_{cur} + amtHealed - HP_{max})$$

Minimize Overkill

The minimize goals appear similar. Instead of examining how much of the heal goes over the target's maximum HP, instead the agent asks how far below zero the action's damage pushed the target's health. Actions that deal no damage (or heal) should not generate any discontent.

$$damage - HP_{cur}$$

Same problem as before: negative values. Fortunately, the same trick still works:

$$\max(0, damage - HP_{cur})$$

Conserve HP

All of the goals interact with each other in some way, but forcing the behavior switch between attacking and healing relies mostly on this goal. “Conserve HP” actually has two non-trivial cases, requiring the agent to discern whether this skill heals the target.

Should the action heal the target, the AI should consider how effective that skill heals the target. A quick calculation involves the target's maximum health along with the amount healed:

$$\max(0, HP_{max} - amtHealed)$$

The “Conserve HP” goal cares not about overhealing, another goal covers that. The last case involves selecting attack skills. The more damage the agent receives, the goal generates more discontent. So, just consider the current amount of damage for the caster:

$$HP_{max} - HP_{cur}$$

Negatives in this case are a non-issue; at zero or less the character is dead and the algorithm stops running.

Target

The targeting calculation looks nasty. The agent is concerned with using skills on things with lower maximum and current HP. Like in the “Conserve HP” goal, the AI tries to cause the greatest change to the target. The other goals counterbalance overdoing the amount of damage/healing.

$$(HP_{cur} + delta) * (HP_{cur} - delta)/HP_{max} * (1 + delta/HP_{max})$$

The `delta` values are positive for healing, but negative for damage. The equation handles both damaging and healing skills. The first term represents the result of the action exactly, while the third term involves the damage dealt as a percent. Since delta is negative for attack skills, the agent prefers to use more damaging skills because of the one. The middle term just helps give a preference for healing.

This goal has the possibility of spitting out a negative number, but appears to work fine. It causes low HP targets to look better; exactly the desired behavior.

Tuning

Using these goals as is will not likely yield good results. The linear functions do not cause actions to look unattractive fast enough in most cases. The actual discontent values are meaningless, but still important to the selection algorithm. Only the relative ordering of the actions matters in the long run.

An easy solution involves taking these calculations to some power. Since most of the goals do not output negatives, any power can be used. (Odd powers on a negative result drastically affects the relative ordering of the actions.) Squaring the function is pretty standard. Unfortunately, there are no other real rules for choosing powers.

The main drawback for a goal-based architecture lies here. Even with sensible goals, a "wrong" choice of powers changes the order of the actions in subtle ways. Ultimately, the only action that matters is the one with the lowest discontent. It is extremely likely that similar actions are only slightly less optimal.

When the agent makes a poor choice, such as attacking rather than healing, focus on making that action look worse in that case. This is generally easier than making the desired action have a lower discontent. Therefore, increase powers on goals that affect actions of that type.

Quirks

After finalizing the magic numbers, the agent battles reasonably well in some scenarios. The most interesting cases occur when the characters have different skill sets. Say, the classical setup where one character utilizes strong attack skills and another healing. The fighter proceeds as expected, taking out the enemies one at a time from weakest to strongest. Since the fighter lacks healing magic, the character relies on its partner to survive.

The healer switches between attacking and healing! While attacking, the healer provides extra damage to the same target. The character displays an unusual preference towards the highest magnitude heals. The agent seems to wait until a target nearly receives that amount of damage before healing. Looking back at the "Conserve HP" goal, it makes perfect sense. A lower magnitude heal always generates more discontent. This might run into problems when the heal's magnitude approaches some character's maximum HP.

Table 1. Example of "Minimize Overkill" Issue

Damage	**Discontent**
20	3.24
40	34.22

The intent behind "Minimize Overkill" is sound, but causes a major issue. Table 1 shows the discontent from two different attack actions on the same target. The target has only 25 HP left. Unfortunately, the agent chooses to deal 20 damage, leaving the enemy at 5 HP. Because of this goal, the agents reveal a preference to leaving enemies alive rather than killing them outright. Creating another goal might counteract this problem.

A very contrived scenario causes a lot of problems for the AI. Suppose the character barely out heals the damage caused by two enemies. The AI must take at least three turns

to kill one enemy. The HP of the character allows for two full rounds of damage, but will fall unless it starts healing. On the third turn, if only one enemy remains, it can survive one last hit.

A solution for this case exists. Burn down one of the enemies, heal, then take out the other enemy without too much trouble. However, the AI chooses a different approach. Once its HP hits a certain threshold, it starts to heal. Since it barely out heals the damage caused by both enemies, it continues healing until it runs out of mana, ultimately failing to defeat both targets.

None of the calculations ascertain the futility of healing. One possible way that may solve this problem tracks the damage the agent receives every round. Adding some clever new goal using this information may help the AI out of this situation.

Conclusion

A goal-based architecture leads to rich behavior, yet still not too difficult to implement. The design requires lots of actions and creative thinking while designing the goals. With only four goals, an AI-controlled party member in an RPG battle makes intelligent decisions.

Goals are not behaviors. Behaviors arise from the interactions between the goals in the discontent-based approach. The most significant drawback for this approach relates to tuning, which is more art than science. Changing the powers on the various goal functions drastically affects the relative ordering of the actions. Drawbacks aside, a goal-based architecture remains a powerful way of expressing behaviors in an RPG-like setting.

References

Millington, Ian and Funge, John, *Artifical Intelligence for Games*, Morgan Kaufmann Publishers, 2009