

Solution for AI

based on: Kevin Dill, Denis Papp, *A Goal-Based Architecture for Opposing Player AI*,

<http://www.gameai.com/papers.php>

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

20 grudnia 2012

- 1 Overview
- 2 Architecture
- 3 The strategic AI
- 4 The Reactive AI
- 5 Cheating

To create AI players for two simultaneously developed real-time strategy games.

Previous projects that required large amounts of custom-written AI code for each decision made.

Objectives:

- primary: to provide a single, easily extensible source for all high-level decisions;
- provide a solid challenge against a human player on a random map, with no economic or military advantage and no visible cheating;
- support for different AI personalities, user-created AIs, limited teamwork between allied AIs, and the ability for human players to use team commands to influence the actions of allied AIs.

To accomplish these goals, a two-layered architecture was developed.

- The **strategic AI** uses our *goal engine* to make broad decisions which outline the overall strategy to be followed. It is computationally expensive, but uses a time-sliced architecture and only needs to think occasionally.
- The **reactive AI** then fills in the second-to-second decisions involved in implementing that strategy.

The general philosophy throughout this project was that **it is much better to give the AI the ability to make its own decisions about what to do based on the current situation rather than hard coding or scripting the AI's decisions.**

That would make the AI far more effective than anything we might script **but** there are situations in which the designer knows best. Particularly in the early game, there are **certain steps which should be reproduced every time in order to give the AI a strong opening.**

The general philosophy throughout this project was that it is much better to give the AI the ability to make its own decisions about what to do based on the current situation rather than hard coding or scripting the AI's decisions.

That would make the AI far more effective than anything we might script but there are situations in which the designer knows best. Particularly in the early game, there are **certain steps which should be reproduced every time in order to give the AI a strong opening.**

Goals - the basic building blocks of this architecture. Every action that a player might make is described as a goal, which can be assigned a priority which indicates the importance of executing that action given the current situation. A simplified set of goal types might include:

- ATTACK — attack an enemy player;
- DEFEND — defend your own or an allied position against enemy attack;
- RECOVER — retreat a damaged actor to a safe location where it can be repaired;
- EXPLORE — explore the map, either to discover new locations or to scout for new enemy construction in areas you've already seen;
- RECRUIT — build new military or civilian actors;
- CONSTRUCT — build new buildings;
- GIVE — give money or actors to an allied player;
- DESTROY — sell a building or disband an actor.

The vast majority of the logic is placed in the goal itself, rather than in the engine, so that custom goals can be written which generate their priority based on whatever information seems appropriate.

- each goal type can have many instances,
- goal instances can be dynamically generated and destroyed.

You might have one ATTACK goal for each enemy actor, dynamically creating and destroying these goals as targets enter and exit the game world.

Goals are implemented by subclassing from a base class. The subclass must include code to determine whether the

- state (active, selected, finished),
- base priority,
- current priority,
- sorts of resources required for execution.

Examples for state

A goal is active if it is currently reasonable to execute it. Consider, for example, the RECOVER goal. This goal allows an actor to recover when it is damaged, with a priority that depends on the amount of damage the actor has taken.

A goal is finished if, for one reason or another, we can get rid of it entirely. For example an ATTACK goal is finished if we successfully destroy the target actor.


Examples for base priority

The base priority is the **relative priority** of that goal compared to all others given the current game environment but assuming some reasonable allocation of resources. For example, the base priority of the ATTACK and DEFEND goals assumes that the total friendly strength assigned to the goal is 1.2 times the total enemy strength in the region (a tweakable setting defined for each AI personality). Similarly, the CONSTRUCT and RECRUIT goals assume that sufficient resources can be reserved to pay for the actor being created.

Examples for current priority

The current priority is the priority of the goal with the resources **actually assigned**. For example the ATTACK and DEFEND goals will have higher priority if you assign more strength to them, and lower priority if you assign less strength. Likewise, the priority of the EXPLORE goal will vary depending on the location of the scout actor assigned.

High-level strategic decisions are made by periodically running through the goal engine's "think" process. Goals selected during one think will remain active at least until the next time the goal engine thinks. The vast majority of thinks are scheduled approximately 30 seconds apart, but we also support for certain event-driven thinks¹.

¹Generally an event-driven is used only early in the game, when it is essential that no time be wasted if we are to remain competitive with human players. 

The first step of the think cycle is to

- go through all of the goals and find those which are either finished or currently inactive.
- determine the base priority of each candidate goal. To this we add a random fuzzy factor, and if the goal was selected in the previous think cycle we also add a bonus for goal inertia. The goal inertia bonus is generally larger than the fuzzy factor, and is intended to prevent the AI from flip-flopping between two similarly attractive actions.

The strategic AI

The Think Cycle – the second step

The second step – standard resource allocation problem. We need to allocate resources to each goal in such a way as to optimize the total priority of all selected goals. This problem is made more complicated by the fact that **the priority of a goal may vary depending on the resources assigned to that goal.**

The strategic AI

The Think Cycle – the third step

In the third step, the goals are sorted by priority and an initial resource assignment to each one is generated. For each goal we assign just enough resources to have a reasonable chance of success — in general, this is the same as the resource assignment assumed when the base priority is calculated.

For goals which require money, such as CONSTRUCT or GIVE, we apply the notion of **goal commitment**. The general idea is that we want to be able to save up sufficient money to execute the more expensive goals as long as that money can be obtained within a reasonable amount of time. To do this all lower priority goals which require monetary resources are marked as inactive for this think cycle in order to prevent them from using resources needed for this goal.

Actors optimization

We go through all the goals which require military actors (such as ATTACK and DEFEND goals) and calculate the change in overall priority if we move actors between them. In certain cases it might be advantageous to launch one extremely strong attack, for example, while in other cases you might have enough forces to launch two or more weaker attacks.

Actors commitment

Similar to goal commitment, we found some cases where it was advantageous to lock actors onto a goal. This is an issue which needs to be approached cautiously, since it violates our basic philosophy that it is better to let the AI make its own decisions rather than hard-coding behavior. In almost all cases you want the AI to retain the flexibility to be able to change its mind when the situation changes, but there are a few specific circumstances where it was necessary to override that.

Actors lock

There are two situations in which we lock actors.

- The second situation is if an actor is assigned to an ATTACK goal and it has already engaged the enemy forces it is targeting. It appears like a mistake for that actor to leave in the middle of the attack to fulfill another goal (unless it is forced to retreat, of course). Even if this reassignment is the correct course of action (which is extremely rare), the player perceives that the AI has made an error.
- The second situation is when an actor is on a RECOVER goal, we lock it until it has reached full strength (assuming there is a safe location available for recovery).

The RAI runs on a one second think cycle, and makes decisions such as selecting formations, coordinating arrival of actors, ordering damaged actors to retreat, and selecting targets for military actors in combat. Unlike the SAI it is lightweight enough that it doesn't require interruption during its decision cycle.

One important requirement of the system is that the designers want to be able to create multiple distinct personalities for the AI.

For example we might have AIs that prefer

- certain types of actors,
- a rushing strategy,
- to build a strong economic base before attacking,
- and so forth,
- or simply allow the players to create their own AIs

The vast majority of the data required to run the AI is contained in the egos. A typical AI profile includes at least three egos (one for the early game, one for when things are going well, and one for emergencies).

Each ego first contains a list of filters which have to be satisfied in order for that ego to be activated.

The following are the most commonly statistics used to filter egos

- current monetary income,
- number of cities controlled,
- number of military actors available.

Constuction templates

Construction templates simply provide a bonus to the construct goal — it is up to the designer to determine whether that bonus should be large (making it a mandate) or small (making it a suggestion).

- The ego can include construction templates, which give it sets of buildings which should be built together in cities to ensure that we have a city which can build strong military actors.
- Another commonly used construction template encourages strong economic development in the remaining cities.

Military templates

Military templates, tell the ego which military actors to use together. For example we might have an ego that prefers to mix infantry and ranged actors, or one which likes to use a mix of light and heavy cavalry. Again, these can be mandates or suggestions, and we can also control what portion of our total actors will be influenced by these templates.

Different values used to tune the priorities for each goal type as well as the RAI.

- Additive and multiplicative bonuses, which can be used to scale any numerical value depending on its importance to the ego.
- Exponents, which can be used to curve the range of values. For example when you have insufficient strength to launch an attack, the goal is still given some priority but that priority drops off exponentially based on the ratio between your strength and the enemy's strength.
- Minimum and maximum values.
- Inversion. In some cases, certain egos want a value to be large while others want it to be small. For example, some egos prefer to explore close to known territory while others prefer to explore distant areas first.

The Reactive AI

Ego specifier: parameters (2/2)

- Repeat penalties. For example, we might have a repeat penalty for recruiting certain actor types. We apply the penalty once for each existing actor of that type. If we have so many actors of that type that the total priority is less than zero, then we won't recruit any more.
- One time bonuses. These are similar to repeat penalties but are only applied if we don't have any actors of the specified type. For example we might have a one time bonus to build a structure which allows us to recruit certain types of actors. By using one time bonuses with different values we can exert a high level of control over the order in which the ego constructs its initial structures.
- Fuzzy factors. In many cases we need a bit of extra randomness.

The original goal was to create an AI that did not cheat at all. In very limited circumstances we did find that it was useful to provide the AI with information that it technically should not possess, but which would often be available to a human player **through intuition and meta-knowledge** of how random map generation works.

Cheating

AI with perfect knowledge of enemy strengths

Our first cheat was to provide the AI with perfect knowledge of enemy strengths (but not actual troop locations). We did not provide the AI with specific knowledge of actors or allow it to attack actors (or buildings) that it could not see. While the information provided is technically more than a human would have, we find that a moderately experienced player actually has a fairly good intuition of enemy strengths.

Our second cheat was in the domain of exploration. Quite simply, we found that occasionally the AI would simply fail to find the enemy (or anything else of interest) until fairly late in the game, and therefore would never get off the ground. Obviously, an AI which doesn't present any challenge in this way isn't a lot of fun to play against. In order to do this, we provided the AI with a small bonus to explore areas with “interesting” items (such as monster lairs, resource points, enemy buildings, etc). Exactly how large the bonus was and which sorts of items the AI would look for depended on the ego. We found that experienced players are able to predict with a fair level of accuracy where these interesting areas are likely to be based on past observation of randomly created maps, so this seemed like reasonable information to give to the AI as well.