Giving the Antagonist Strategic Smarts

Simple Concepts and Emergence Creates Complex Strategic Intelligence

By Mark Baldwin

Introduction

In order for a game to provide entertainment, there must be barriers or challenges for the game player to be overcome. This act of succeeding in overcoming these challenges is where much of the entertainment comes in. These barriers can be passive, such as figuring out which gem to choose in a simple puzzle, or active such as that BIG DRAGON sitting on top of the treasure that you so desperately want, glaring at you as he prepares to burn you to cinders. Now that dragon will probably have an artificial intelligence (AI) that responds when you attack it, but for much of the game it just sits there in a vegetative state brooding over it's treasure. Or at most, it's concerned with its own personal goals within the game world independent of your own.

But there can be a third type of barrier, an intelligent antagonist that is reacting to every decision you make and is actively putting barriers in your way. An intelligence that is controlling its own units, resources or characters striving for the same goals that you are, but trying to accomplish them first. This intelligent antagonist is your equal in the game, normally working under the same rules and resources as you are. This might be your opponent in a game of Chess or the coach to the New England Patriots as you coach your own team trying to beat him in the Superbowl. Or as I will be using as an example, it will be an opponent in a simple game of Capture The Flag.

While not all games have active challengers like an opponent in a chess game or an opposing coach, many do. To control these challenges, one needs 'intelligence' either in the form of another human (say in an online Chess game) or in the form of artificial intelligence. Decision making in this form of challenge is typically strategic in nature, and quite complex. The computer as the antagonist must manage and coordinate a complex set of resources towards a goal which is in conflict with the protagonist, i.e. the game player.

Strategic Smarts

Over the years I have developed a number of methodologies to solve this problem. For example, I have spent a great deal of time trying to find practical ways to implement neural networks for an antagonist artificial intelligence. One that I find to be both practical and effective, and I have given a number of lectures on over the years is a methodology I describe at Project AI. The reason for the label will become apparent later in this paper, but I would like to build up to it piece by piece.

First let us look at the problem and some of its components. The most important is the goal. In theory at least, the goal of a good Antagonist AI is to 'win' the game, and in so doing defeat his opponent, i.e. the game player. In practice, this is not quite true. Instead of trying to win the game, the goal of the antagonist is to entertain the game player. Trying to win is a means to that end. Normally winning the game is accomplished through victory conditions, perhaps to destroy all of the enemy or gain the highest score. The player achieves this victory by controlling a large set of resources (units, people, money, etc.) in a coordinated and complex manner.

Although I could select a number of models to describe this process, let us use a simple game. Consider a computer game of capture the flag where each player controls five members on his team. The object of the game is to capture the opponent's flag by moving a player on it. But he may also eliminate opponent players by shooting at them. See Figure 1. In our example the player (be it human or computer) must control units (our resources) in a coordinated manner. There is a decision cycle in which the player issues orders to his resources, the game progresses, the results are observed and new orders are then issued. There is an objective or victory condition that is mutually exclusive for all players of the game, i.e. there can only be one winner. And the decisions can both move the player towards this victory condition and/or prevent the opponents from doing so. Although I will be looking at this capture the flag example, it can be applied to any complex strategic game where there are a large number of strategic decisions being made controlling a large number of unique resources.

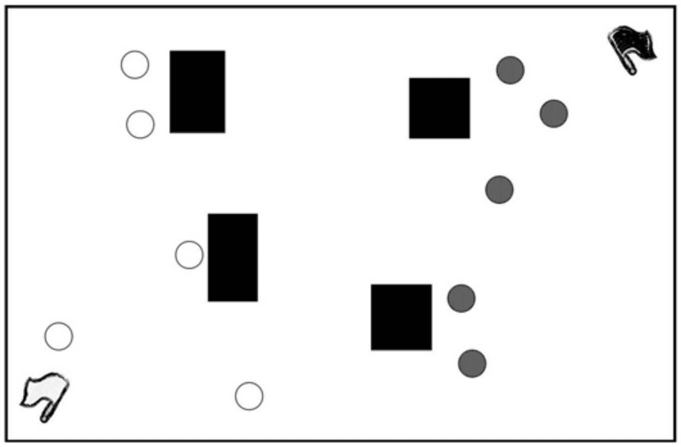
Figure 1: The Game of Capture the Flag

To develop our ideas, we need to start with the simple and build to the complex. I will do this by looking at level's of complexity from which the decisions of the AI will be made. Each level will be explored based on the previous level I will be looking at both how it would be implemented and problems that might occur.

Level 1 – Brownian Motion

For our first level, let's keep things simple. In each decision cycle let us examine each unit, build a list of possible orders that can be given to the unit, and pick one randomly. In other words, look around and do something. See Figure 2.

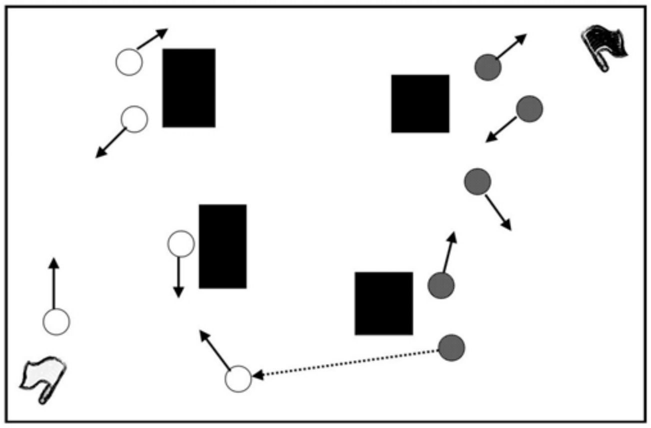
Figure 2: Level 1 – Random Decision Making

The problem with this level of decision making is that the units are not directed in any way towards archiving the computer players goals, i.e. winning the game or preventing the opponent from achieving his goals. It's just Brownian motion (the random motion of atoms), but it might have the advantage that it would confuse the heck out of any opponent.

Level 2– Grab the Win

Let's address the problem of Brownian motion. Instead of random decisions, for each unit let us pick an action that achieves the game's victory conditions. To be specific, look around for each unit, and ask are there any victory goals achievable by the unit? If so, implement it. In Figure 3. I moved a unit near an opponent's flag to show this.

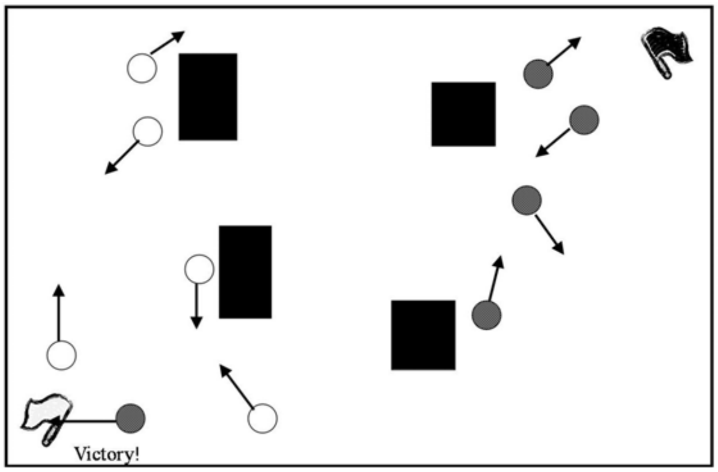

Figure 3: Level 2 – Taking an action that wins a game

When there are alternate actions that achieve the same goal, there is no filter in differentiating equal or nearly equal actions. Also, in most games except for the most simplistic, any one decision or action cannot achieve a victory condition. This puts our decisions back to random decisions like Level 1.

Level 3 – Head towards the Goal

Expanding on the ideas of the Level 2 process, we can evaluate each alternate action by how well it moves us towards the victory conditions. The actions are not evaluated in a true/false analysis like the last level but instead by creating an evaluation function and giving each decision a numerical value. The action with the best value would be the one chosen. For example, a decision that would move a unit to a victory location in 2 turns would be worth more than

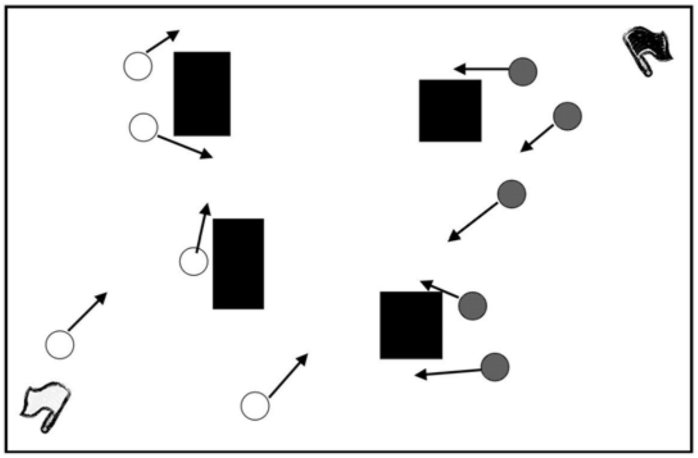

Figure 4: Level 3 – Making decisions that move your resources towards the goal.

a decision that would move the unit to a victory location in 10 turns. Figure 4. shows this. Note that no unit is shooting at any other unit because that doesn't move us towards our victory conditions.

So what problems do we see with this level? For one thing, we attribute no value to decisions that support reaching the conditions but in and of them selves do not archive victory. An example might be killing an opposing unit.

Level 4 – Using Sub-goals

Many actions that units can engage in cannot be directly attributed to the victory goals of a game. The solution is to develop sub-goals which assist the artificial intelligence in achieving the victory conditions, but in and of themselves are not victory conditions. These sub-goals are generally easier to accomplish in the short term than the primary game goals may be. When making a decision for a unit, the AI evaluates the possibility of achieving these sub-goals. Such sub-goals might include killing enemy units, protecting the flag, maintaining a defensive line or achieving a strategic position. If the sub-goal creation and evaluation process is done judiciously, each resource will them be taking some action that moves the player towards the final victory goals. Because of this, using sub-goals can actually produce a semi-intelligible game. Figure 5 shows some of this.

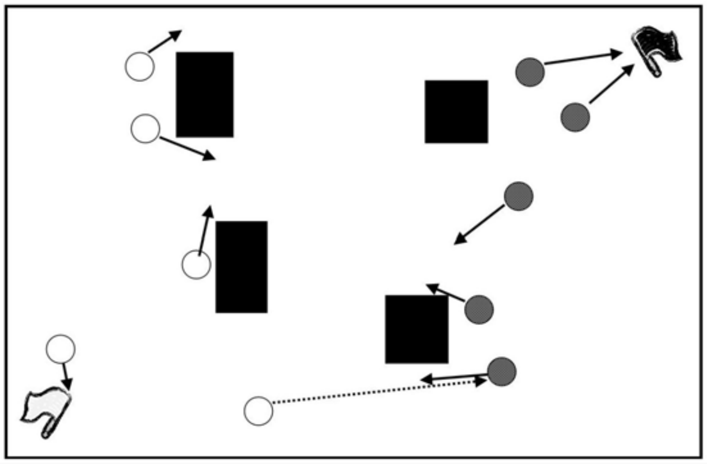
Figure 5: Level 4 – Using sub-goals to make decisions

Even though we are starting to get a playable AI, there are still problems. Each unit makes its decisions independent of all others. But it is playing against a human who does coordinate his resources. It's like warfare pre-Napoleonic. It works well until the opponent starts coordinating their forces.

Level 5- Coordination

So how do we allow for this coordination? Let us allow the AI making a decision for a unit to examine the decisions being made for other friendly units. Weigh the possible outcomes of the other units planned actions, and balance those results with the current unit's action tree. Now we are balancing our resources with the goals that need to be accomplished, and are not engaged in overkill or underkill.

This allows for coordination, but not strategic control of the resources. However, this level is actually beyond what many computer game AI's actually do today. One of the big problems is that it can lead to iterative cycling. The process of changing a units decision based on other unit decisions, then requires the other units to evaluate and their decisions thus having the potential or creating a circular continuous process that is either not resolved or consumes a great deal or resources to resolve.

Level 6 – A Command Hierarchy

We are now getting to some interesting ideas. Let us create a strategic (or grand tactical) decision making structure that can control units in a coordinated manner.

This leads us to the problem of how does one coordinate diverse resources to reach a number of sub-victory goals. This question may be described as strategic level decision making. One solution would be to look at how the problem is solved in reality, i.e. on the battlefield or in business.

The solution on the battlefield is several layers of a hierarchical command and control. For example, squad's 1, 2 and 3 are controlled by Company A that in and of itself is controlled by a higher layer of the hierarchy. Communications of information mostly go up the hierarchy (information about a squad 20 miles away is not relayed down to the local squad), while control mostly goes down the hierarchy. Upon occasion, information and control can cross the hierarchy, and although it's happening more now than 50 years ago, it is still relatively infrequent.

As a result, the lowest level unit must depend on its hierarchical commander to make strategic decisions. It cannot itself because a) it doesn't have as much information to make the decision with as it's commander, and b) it is capable of making a decision based on known data different that others with the same data, causing chaos instead of coordination.

OK, first cut solution. We build our own hierarchical control system, assigning units to theoretical larger units or in the case where the actual command/control system is modeled (V for Victory), the actual larger units. Allow these headquarters to control their commands and work with other headquarters as some type of 'mega-unit'. These in turn could report to and be controlled by some larger unit. See Figure 6.

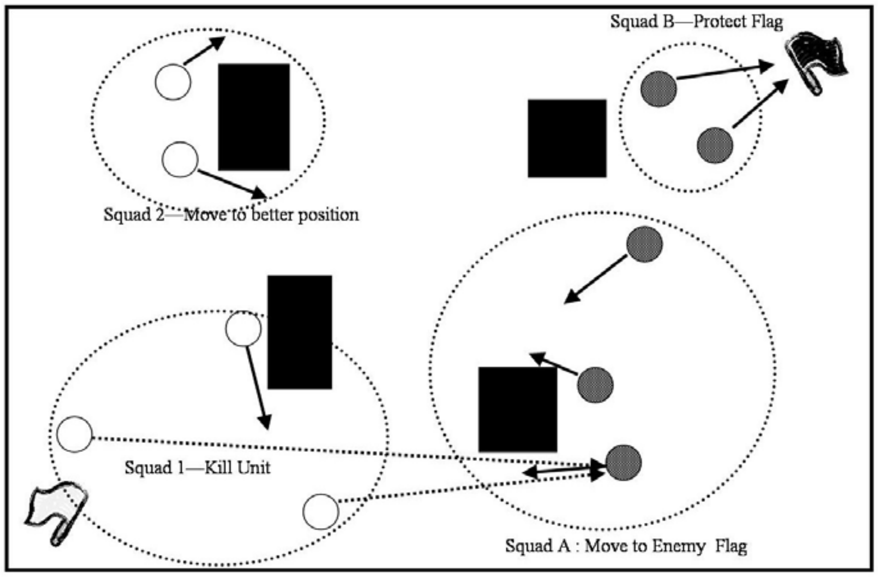

Figure 6: Level 6 – Using a command hierarchy to make decisions.

But there seems to be some problems here. The hierarchical command system modeled on the real world does not make optimal use of the resources. Because of the hierarchical structure, too many resources may be assigned to a specific task, or resources in parallel hierarchies will not cooperate. For example, two units might be able to easily capture a victory location they are near, but because they each belong to a separate command hierarchy (mega unit) they will not coordinate to do so. Note that if by chance they did belong to the same hierarchy, they would be able to accomplish the task. In other words, this artificial structure can be too constraining and might produce sub optimal results.

And a single human opponent does not have these constraints. All of the information exists in one place, the players mind and the barriers of communication are therefore removed.

First, we have to ask ourselves, if the hierarchical command and control structure is not the best solution, why do business and the military use it? The difference is in the realities of the situations. As previously pointed out, in the battlefield, information known at one point in the decision making structure might not be known at another point in the hierarchy. In addition, even if all information was known everywhere, identical decisions might not be made from the same data. However, in game play, there is only one decision-maker (either the human or the AI) and all information known is known by that decision-maker. This gives the decision maker much more flexibility on controlling and coordinating resources than does the military hierarchy.

In other words, the military and business system of strategic decision is not our best model. Its solution exists because of constraints on communication. But those constraints do not usually exist in strategy games (command and control is perfect) and therefore modeling military command and control decision making is not our perfect model to solve the problem in game play AI.

And we want the best technique of decision-making we can construct for our AI. So below is an alternative Sixth Level attack on the problem...

Project AI – A Level 6 Alternative

This leads us to a technique I call Project AI. Project AI is a methodology that extrapolates the military hierarchical control system into something much more flexible.

The basic idea behind Project AI is to create a temporary mega-unit control structure (called a Project) designed to accomplish a specific task. Units (resources) are assigned to the Project on an as needed basis, used to accomplish the project and then released when not required. Projects exist temporarily to accomplish a specific task, and then are released.

Therefore, as we cycle through the decision making process of each unit, we examine the project the unit is assigned to (if it is assigned to one). The project then contains the information needed for the unit to accomplish its specific goal within the project structure.

Note that these goals are not the final victory conditions of the game, but very specific sub-goals that can lead to game victory. Capturing a victory location is an obvious goal here, but placing a unit in a location with a good line of sight could also be a goal, although less valuable. Figure 7 shows an example of this.

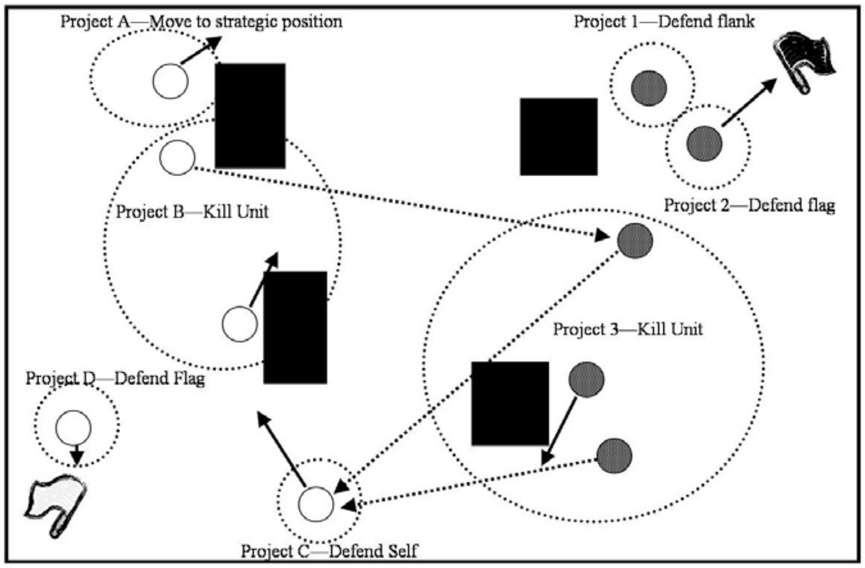

Figure 7: Project AI – Using the project structure to make decisions.

Let's get a little more into the nitty gritty of the structure of such projects, and how they would interact.

What are some possible characteristics of a project?

- Type of project -- What is the project trying to accomplish. This is basically our goal for the project.

Project Type Examples:

- Kill an enemy unit.
- Capture a location.
- Protect a location.
- Protect another unit.
- Invade a region.

- Specifics of the project -- Exactly what are the specifics for the project? Examples are "kill enemy unit 2", "capture location 4", etc.

- Priority of the project -- How important is the project compared to other ongoing projects toward the final victory. This priority is used in general prioritizing and killing off low priority projects should there be memory constraints.

- Formula for calculating the incremental value of assigning a unit to a project -- In other words, given a unit and a large number of projects, how do we discern what project to assign the unit to? This formula might take into account many different factors including how effective the unit might be on this project, how quickly the unit can be brought in to support the project, what other resources have already been allocated to the project, what is the value of the project, etc. In practice, I normally associate a different formula with each project type and then each project carries specific constants that are plugged into the formula. Such constants might include enemy forces opposing the project, minimum forces required to accomplish the project, and probability of success.

- A list of units assigned to the project.

- Other secondary data.

OK, now how do we actually use these 'projects'? Here is one approach...

1) For every turn, examine the domain for possible projects, updating the data on current projects, deleting old projects that no longer apply or have too low a priority to be of value, and initializing new projects that present themselves. For example, we have just spotted a unit threatening our flag, we create a new project which is to destroy the unit, or if the project already existed, we might have to reevaluate its value and resources required considering the new threat.

2) Walk through all units one at a time, assigning each unit to that project that gives the best incremental value for the unit. Note that this actually may take an iterative process since assigning/releasing units to a project can actually change the value of assigning other units to a project. That means that we may have to reassess this multiple times. Also, some projects may not receive enough resources to accomplish their goal, and may then release those resources previously assigned.

3) Reprocess all units, designing their specific move orders taking into account what Project they are assigned to, and what other units also assigned to the project are planning on doing. Again, this may be an iterative process.

The result of this Project structure is a very flexible floating structure that allows units to coordinate between themselves to meet specific goals. Once the goals have been met, the resources can be reconfigured to meet other goals as they appear during the game.

One of the implementation problems that Project AI can generate is that of oscillations of units between projects. In other words, a unit gets assigned to one project in one turn, thus making a competing project more important, grabbing the unit the next turn. This can result in a unit wandering between two goals and never going to either because as it changes its location, the decision making algorithms would then reassign it. The designer needs to be aware of this possibility and protect for it. Although there can be several specific solutions to the problem, there is at least one generic solution. Previously, we mentioned a formula for calculating the incremental value of adding a unit to a project. The solution lies in this formula. To be specific, a weight should be added to the formula if a unit is looking at a project it is already assigned to (i.e., a preference is given to remaining with a project instead of jumping projects). The key problem here is assigning a weight large enough that it stops the oscillation problem, but small enough that it doesn't prevent necessary jumps in projects. So one may have to massage the weights several times before a satisfactory value is achieved. This is a trial and error type of process, and can almost be an art in developing the proper values.

And onward…

One can extrapolate past the "Project" structure just as we built up to it. One extrapolation might be a multilayer level of projects and sub-projects. Much could be done in developing methods to resolve the iteration cycles to improve performance. There are other possibilities as well to explore.

This methodology has been used and improved in a number of my games over the years including Empire, The Perfect General and Metal Fatigue. It is an effective technique for giving the computer opponent strategic smarts.