How should I start designing an AI algorithm for an artillery warfare game?

Asked 15 years, 11 months ago Modified 15 years, 6 months ago

Viewed 3k times

19

Here's the background... in my free time I'm designing an artillery warfare game called Staker (inspired by the old BASIC games Tank Wars and Scorched Earth) and I'm programming it in MATLAB. Your first thought might be "Why MATLAB? There are plenty of other languages/software packages that are better for game design." And you would be right. However, I'm a dork and I'm interested in learning the nuts and bolts of how you would design a game from the ground up, so I don't necessarily want to use anything with prefab modules. Also, I've used MATLAB for years and I like the challenge of doing things with it that others haven't really tried to do.

Now to the problem at hand: I want to incorporate AI so that the player can go up against the computer. I've only just started thinking about how to design the algorithm to choose an azimuth angle, elevation angle, and projectile velocity to hit a target, and then adjust them each turn. I feel like maybe I've been overthinking the problem and trying to make the AI too complex at the outset, so I thought I'd pause and ask the community here for ideas about how they would design an algorithm.

Some specific questions:

1. Are there specific references for AI design that you would suggest I check out?
2. Would you design the AI players to vary in difficulty in a continuous manner (a difficulty of 0 (easy) to 1 (hard), all still using the same general algorithm) or would you design specific algorithms for a discrete number of AI players (like an easy enemy that fires in random directions or a hard enemy that is able to account for the effects of wind)?
3. What sorts of mathematical algorithms (pseudocode description) would you start with?

Some additional info: the model I use to simulate projectile motion incorporates fluid drag and the effect of wind. The "fluid" can be air or water. In air, the air density (and thus effect of drag) varies with height above the ground based on some simple atmospheric models. In water, the drag is so great that the projectile usually requires additional thrust. In other words, the projectile can be affected by forces other than just gravity.

language-agnostic artificial-intelligence

edited Jan 25, 2009 at 0:08

Share

Improve this question

Follow

asked Jan 10, 2009 at 19:28

gnovice
126k ● 16 ● 258 ● 363

9 "Don't use matlab for this" -- No, do! The challenge is half the fun. – SCdF Jan 20, 2009 at 11:19

1 It's something of a moot point now since I have the whole game nearly complete! =) Adding the AI players is one of the final steps I'm doing before I post v1.0 on the MathWorks File Exchange. The MUCH simpler beta version is already posted there. – gnovice Jan 20, 2009 at 18:03

I made this game (with Delphi) and used a simple geometrical trick which allow computer to hit the other users in one shot. I assumed the walls reflect the bullet. Making that game was a huge fun for me. – Kamran Bigdely Dec 15, 2011 at 0:36

7 Answers

Sorted by: Highest score (default)

4

In a real artillery situation all these factors would be handled either with formulas or simply brute-force simulation: Fire an electronic shell, apply all relevant forces and see where it lands. Adjust and try again until the electronic shell hits the target. Now you have your numbers to send to the gun.

Given the complexity of the situation I doubt there is any answer better than the brute-force one. While you could precalculate a table of expected drag effects vs velocity I can't see it being worthwhile.

Of course a game where the AI dropped the first shell on your head every time wouldn't be interesting. Once you know the correct values you'll have to make the AI a lousy shot. Apply a random factor to the shot and then walk to towards the target--move it say 30+random(140)% towards the true target each time it shoots.

Edit:

I do agree with BCS's notion of improving it as time goes on. I said that but then changed my mind on how to write a bunch of it and then ended up forgetting to put it back in. The tougher it's supposed to be the smaller the random component should be.

Share Improve this answer Follow

edited Jan 10, 2009 at 20:04

answered Jan 10, 2009 at 19:52

Loren Pechtel
9,073 ● 4 ● 35 ● 46

3

Loren's brute force solution is appealing as because it would allow easy "Intelligence adjustments" by adding more iterations. Also the adjustment factors for the iteration could be part of the intelligence as some value will make it converge faster.

Also for the basic system (no drag, wind, etc) there is a closed form solution that can be derived from a basic physics text. I would make the first guess be that and then do one or more iteration per turn. You might want to try and come up with an empirical correction correlation to improve the first shot (something that will make the first shot distributions average be closer to correct)

Share Improve this answer Follow

answered Jan 10, 2009 at 20:01

BCS
78.3k ● 69 ● 194 ● 298

3

Thanks Loren and BCS, I think you've hit upon an idea I was considering (which prompted question #2 above). The pseudocode for an AIs turn would look something like this:

```
nSims;         % A variable storing the numbers of projectile simulations
               %    done per turn for the AI (i.e. difficulty)
prevParams;    % A variable storing the previous shot parameters
prevResults;   % A variable storing some measure of accuracy of the last shot
newParams = get_new_guess(prevParams,prevResults);
loop for nSims times,
  newResults = simulate_projectile_flight(newParams);
  newParams = get_new_guess(newParams,newResults);
```

```
end
fire_projectile(newParams);
```

In this case, the variable nSims is essentially a measure of "intelligence" for the AI. A "dumb" AI would have nSims=0, and would simply make a new guess each turn (based on results of the previous turn). A "smart" AI would refine its guess nSims times per turn by simulating the projectile flight.

Two more questions spring from this:

1) What goes into the function get_new_guess? How should I adjust the three shot parameters to minimize the distance to the target? For example, if a shot falls short of the target, you can try to get it closer by adjusting the elevation angle only, adjusting the projectile velocity only, or adjusting both of them together.

2) Should get_new_guess be the same for all AIs, with the nSims value being the only determiner of "intelligence"? Or should get_new_guess be dependent on another "intelligence" parameter (like guessAccuracy)?

Share Improve this answer Follow

answered Jan 10, 2009 at 21:36
gnovice
126k ● 16 ● 258 ● 363

3 This should be added to the question. – Loren Pechtel Jan 10, 2009 at 23:45

3

A difference between artillery games and real artillery situations is that all sides have 100% information, and that there are typically more than 2 opponents.

As a result, your evaluation function should consider which opponent it would be more urgent to try and eliminate. For example, if I have an easy kill at 90%, but a 50% chance on someone who's trying to kill me and just missed two shots near me, it's more important to deal with that chance.

I think you would need some way of evaluating the risk everyone poses to you in terms of ammunition, location, activity, past history, etc.

Share Improve this answer Follow

answered Jan 10, 2009 at 22:01
Uri
89.6k ● 51 ● 226 ● 322

1 Good point. For now it isn't a problem since the game can only have 2 players (but eventually I want it to have more). – gnovice Jan 10, 2009 at 22:09

3

I'm now addressing the response you posted:

While you have the general idea I don't believe your approach will be workable--it's going to converge way too fast even for a low value of nSims. I doubt you want more than one iteration of get_new_guess between shells and it very well might need some randomizing beyond that.

Even if you can use multiple iterations they wouldn't be good at making a continuously increasing difficulty as they will be big steps. It seems to me that difficulty must be handled by randomness.

First, get_initial_guess:

To start out I would have a table that divides the world up into zones--the higher the difficulty the more zones. The borders between these zones would have precalculated power for 45, 60 & 75 degrees. Do a test plot, if a shell smacks terrain try again at a higher angle--if 75 hits terrain use it anyway.

The initial shell should be fired at a random power between the values given for the low and high bounds.

Now, for get_new_guess:

Did the shell hit terrain? Increase the angle. I think there will be a constant ratio of how much power needs to be increased to maintain the same distance--you'll need to run tests on this.

Assuming it didn't smack a mountain, note if it's short or long. This gives you a bound. The new guess is somewhere between the two bounds (if you're missing a bound, use the value from the table in get_initial_guess in it's place.)

Note what percentage of the way between the low and high bound impact points the target is and choose a power that far between the low and high bound power.

This is probably far too accurate and will likely require some randomizing. I've changed my mind about adding a simple random %. Rather, multiple random numbers should be used to get a bell curve.

Share Improve this answer Follow

answered Jan 11, 2009 at 0:38

Loren Pechtel
9,073 ● 4 ● 35 ● 46

You may be right about the nSims iterations causing faster-than-desired convergence, although it would probably depend on the accuracy of get_new_guess. I'm going to work on an updated pseudocode and I'll put it in the re-edited question sometime soon. – gnovice Jan 12, 2009 at 0:28

The version I'm suggesting converges *VERY* fast.
– Loren Pechtel Jan 12, 2009 at 1:29

1

Another thought: Are we dealing with a system where only one shell is active at once? Long ago I implemented an artillery game where you had 5 barrels, each with a fixed reload time that was above the maximum possible flight time.

With that I found myself using a strategy of firing shells spread across the range between my current low bound and high bound. It's possible that being a mere human I wasn't using an optimal strategy, though--this was realtime, getting a round off as soon as the barrel was ready was more important than ensuring it was aimed as well as possible as it would converge quite fast, anyway. I would generally put a shell on target on the second salvo and the third would generally all be hits. (A kill required killing *ALL* pixels in the target.)

In an AI situation I would model both this and a strategy of holding back some of the barrels to fire more accurate rounds later. I would still fire a spread across the target range, the only question is whether I would use all barrels or not.

Share Improve this answer Follow

answered Jan 11, 2009 at 0:47

Loren Pechtel
9,073 ● 4 ● 35 ● 46

The game I'm designing is turn-based, with 1 shell fired each turn. An additional caveat I will eventually have to factor in is the choice of ammunition, since I currently have 10 different types of shells programmed. I'll add some extra info about this to the question soon. – gnovice Jan 12, 2009 at 0:30

1

I have personally created such a system - for the web-game Zwok, using brute force. I fired lots of shots in random directions and recorded the best result. I wouldn't recommend doing it any other way as the difference between timesteps etc will give you unexpected results.

Share Improve this answer Follow

answered Jun 25, 2009 at 19:01

Iain
9,442 ● 11 ● 50 ● 65