Fire Emblem AI

Jazmin Gonzalez-Rivero, Zach Homans, Elizabeth Mahon, Brendan Ritter

Artificial Intelligence - Spring 2013
May 9, 2013

1 Introduction

We are Together We Ride, and our AI project this semester was constructing an enemy AI agent for the game Fire Emblem. Fire Emblem is a classic Japanese turn-based strategy game where the player deploys an army of individualized soldiers to fight against a variety of enemy armies. The main selling point of Fire Emblem is the concept of "permanent death", meaning that any unit that dies stays dead for the remainder of the game. Because of this, players often adopt a "Reset" strategy, meaning that if one of their units ever dies, they completely restart the battle from the beginning. As a result, Fire Emblem players often consider a single unit's death to be a defeat.

Currently, Fire Emblem's AI is spotty at best. Many players believe that the AI follows the following pattern: for each of my units, attack the closest enemy unit that I can do the most damage do. Often, this leads to rather embarrassing situations where the enemy AI could have easily killed a unit, but fails to do so. However, other times this leads to the enemy AI units ganging up on the weakest of the player's units. Combining the player's defeat condition with the current AI behavior, we decided that we wanted to construct an agent that more reliably ganged up on the player's units and achieved victory whenever possible. Our goal in making this AI was to construct an enemy that forced the player to play the game more intelligently, thus forcing them to learn, making them a better player.

2 Structure

Unfortunately, we were unable to find any pre-existing interfaces to use when testing our agent. It's very hard to reprogram the game itself. Therefore, a significant effort was allocated to creating a visual/internal interface for our agent. Our end product was visually mediocre, but it did support the functions we required. Thus, it was adequate for our purposes of creating an AI agent.

Thankfully from the very beginning what was most important to us was creating an agent that acted intelligently. We never intended for the agent to be fully integrated into the actual game, nor were we trying to recreate the game itself. Therefore, most of our efforts were allocated into creating the logic that the agent would follow, not the individual aspects of the game that the agent would need to use. Thus, we decided to simplify the version of the game that our agent would use, while still retaining the features necessary to make the AI intelligent and interesting.

In Fire Emblem, there are entire gamuts of class types, weapon types, terrain types, and more. In our version, there is one unit type and two terrain types, one of which is better to stand on, but costs more movement to travel over. Units don't carry different weapons; they merely have a set attack and defense. Additionally, at the beginning there was no hit chance; every unit always hit. This feature is currently implemented, although units can still not critically hit, which randomly does increased damage. Ranged attacking, which is common in Fire Emblem, is completely disabled, although if implemented it would simply follow the same basic logic that the current agent follows.

3 Basic Agent

As previously stated, the goal of our agent was to reliably gang up on one player unit and do as much damage as possible, in hope to kill it, thus resulting in victory. It's important to note that because the agent prioritizes this victory, it in no way attempts to reduce casualties in its own forces. The agent doesn't care if a given action will kill its own unit, as long as it harms the enemy unit. This behavior also emulates the in-game AI behavior.

Our agent achieves our goal by using the following algorithm:

- First, look at all of the units that the agent owns that have not moved this turn.
- For each unit, calculate the locations where it can move. Do this by considering its movement modifier and terrain modifiers, as well as player units that can obstruct a path.
- For each of these movement locations, determine if there are any adjacent player units. If so, store both the space that the enemy unit can attack from, as well as the average amount of damage that would be done from this location (Recall that hitting is a percentage chance. Average damage is just hit percentage multiplied by the damage of a successful hit).
- Store this information in such a way that each player unit has a list of enemy units that can attack it from each direction.
- For each player unit, determine the top enemy unit that can attack it from each direction. If there is a conflict (i.e. the top units from two directions are the same), then determine the second strongest attacker from each direction and replace the direction where the stronger second attacker is. Repeat this process until there is a maximum of four unique units attacking from different directions. This algorithm ensures that the strongest set of four units will be attacking the player unit.
- Using the above information, calculate the amount of damage that the enemy army can inflict to each player unit.
- Then, determine the player unit that will take the most damage in percentage health if the enemy army were to attack it. Percentage health is used because it's more indicative of injury than pure damage.
- Move the enemy unit that will do the most damage to the designated player unit to the appropriate square and attack. Repeat the algorithm from the beginning.
- If there is less than four enemy units attacking a given player unit, ensure that if there is special terrain adjacent to the player unit that can be filled by an enemy unit, it is filled.
- If there are no more units that can attack, simply do not move.

This basic agent is fairly successful. It manages to make optimal moves and proved difficult to defeat on our test map.

4 More Advanced Agents

With the basic behavior down, we wanted to try to create more interesting agents. Because we thought our attacking algorithm was fairly solid, we moved our attention to unit movement. In the basic agent, a unit that can't attack anything forfeits its turn. While this actually emulates the in-game AI behavior, we felt that this enabled the player to somewhat abuse the game such that they only triggered the movement of one unit at a time. The actual game solves this issue by linking groups of units together to make a squadron, such that if one unit in the squadron can attack, the whole squadron moves. Additionally, many Fire Emblem maps feature optional objective that discourage this behavior.

We decided on a different approach, due to our simplified structure. We constructed two advanced agents, each with a different movement pattern. The first, our “Aggressive” agent, was simple. Instead of waiting, units that can’t attack find the Manhattan distance to each player unit and move as much as possible towards the unit that is closest. The second, our “Tricky” agent, was far more interesting. Enemy units that can’t attack player units move towards the closest player unit, like the Aggressive agent. However, instead of moving as much as possible, the Tricky agent calculates the attack range of each player unit. It then moves as close as possible to the player unit, without going into its attack range. Therefore, the player unit cannot attack it directly on his next turn. This strategy is devious because if the player waits too long, they will be surrounded by a large group of enemies that they cannot handle all at once.

5 Final Test

To test our advanced agents, we decided to pit them against one another. For our final test, we used the Tricky agent and the Aggressive agent. The Tricky agent won twenty-eight out of the thirty trials. We believe that these results are not surprising, as the strategy employed by the tricky agent is superior, as it doesn’t needlessly put units into harm’s way. Our hypothesis is that the element of chance (each unit has a percentage chance to miss) allowed the Aggressive agent to win on occasion.

6 Current Issues

Currently, there is one major issue in our implementation. Sometimes (but not always), the agent’s units will move on top of the same space. Although our code specifically checks for other units before moving to a new space, we still can’t seem to locate the origin of the error. Despite this, we stand by the remainder of our implementation. We firmly believe that this issue in no way affects the other behaviors of the agent.

7 Conclusions

For such a complex and challenging game, an enemy AI for Fire Emblem is relatively easy to implement. This is in large part because the enemy has little concern for self-preservation. Instead, it has a single consideration to think about: doing the most damage possible. It does not need to balance multiple, possibly conflicting, goals. Implementing an AI for the player, which does need to worry about self-preservation, would likely be much more challenging as the game itself, and thus a strong candidate for further exploration in this area. Other areas of further exploration would most likely involve incorporating additional elements of Fire Emblem into the game and agent, such as healing, rescuing, multiple weapons, ranged attacking, and more. However, even without these features, we believe our agent stands strong as far more reliable than the game’s current agent.

No agent can ever be perfect. Despite flawless logic and behavior, there is always the possibility of failure. The variable result of the battles between the two AI agents emphasizes the importance of chance when playing a game like Fire Emblem. Without chance, the same game would be played every time, especially when using our agents, as the rules the AI employs are completely deterministic.

The time spent on this project has allowed each to us to more fully explore what’s important when making a strong game AI agent. What are its goals? How will it go about achieving them? What does it do when it makes mistakes? How can we make it better? These are all questions we continuously asked during this process. By doing so, we believe we all gained a new understanding and appreciate of what it truly means to make an AI.