Assignment Report: Software Engineering Project: Artificial intelligence implementation

Moira Shooter and Anastasija Belaka

January 19, 2018

Abstract

I will describe the functions, the issues, and the way of thinking of the artificial intelligence (AI), so the people who will take this over will understand and can work on it later on.

1 Functions

1.1 chooseSlot

Most of the functions such as chooseCards, chooseEnergy, learnCards, chooseConditions, and agree functions we took over from the example code RandomAI class. We tweaked the chooseSlot code for when we have an energy card. We made a variable m_energySlot that returns the index of the slot we want to play an energy on. We also made a variable called m_card that stores the card we want to play. We check if the card type is an energy card if it is return m_energySlot. If we want more control of playing cards on certain slots we should have implemented if conditions or switch-case statements.

1.2 playBasicPokemon

We iterate through the hand, we then check if the card is a pokemon card, if it is a pokemon static cast that card into an energy card to check if the card is a basic pokemon (stage = 0). We store that specific index from the hand, check if we can play that card onto the bench, if we can it plays the card. We also set the _indexHand variable (see code) to -1 because otherwise we get issues, the _indexHand otherwise has no value and it crashes the game.

1.3 playEvolutionCard

In the beginning of the code we store variables, so it doesn't get too expensive and the artificial intelligence will be a bit faster. We store the pokemons names that is not a basic pokemon (stage != 0) in a vector called _listOfPokemons. We then iterate through our hand, check if we have a pokemon card, convert that card into a pokemon card. If it is a pokemon card we iterate through the list of not basic pokemons and check if their pre-evolution is

equal to the card in our hand. If it is store the index from your hand into the _indexHand variable. Again in the beginning we store -1 to the _indexHand so if there is no evolution card it has a value -1 and it will not check if it can be played.

1.4 biggestAttack

To attach an energy we made a function biggestAttack otherwise the attachEnergy function would be too big. Our original idea was to have a data type that stores the type of energy and the amount. And every time we attach an energy card to the pokemon we would decrease the amount of that type of energy. We could not accomplish that so we chose to check which is the biggest attack (most requirements). The function returns which attack it is with it's requirements.

1.5 indexHandEnergy and indexBenchEnergy

We made those two functions because we had a problem when it was integrated within the attachEnergy function. Instead of checking the active card if it has enough energy it would check the amount of energy on the card that just has been played. Therefore we made two functions that returns the index of the hand and same for the index for the bench. We store both to -1 explanation is the same as for playBasicPokemon and playEvolutionCard. But as you can see in the code we iterate through the bench, see if the bench has a pokemon on it, if it does check if it needs energy by checking if the amount of energy on that card is less than the required energy needed. If this is true we iterate through our hand if we have an energy in our hand we convert it into an energy card and check if the type of energy card is equal to one of the required ones or if the required energy is of the type colourless. We then store the index of the bench or hand into a variable called _posBench or _posHand. It will return that variable so we can check if we can play that energy card onto the card that needs an energy card. See subsection **attachEnergy**

1.6 attachEnergy

Here we just check if we can play a certain energy card onto the bench if we can play the card.

1.7 playTrainerCards

We made a switch-case statement because trainer cards has more possibilities such as type SUPPORT, ITEM and STADIUM. At the moment we iterate through the hand check if we have one of these type of cards if we do, we store the index into a variable and then check if that index of the hand can be played or not. Most of the support cards in Bright Tide deck just draws cards which is always useful in a game. We implemented for a couple of specific item cards, if for example our active cards health is not 100 percent check if we have an item card that heals, if it does have that card it heals. It also checks if we have basic cards in our bench if we do not and have an item card that searches for basic cards it plays that item card. We wanted to implement more if conditions but unfortunately we could not.

1.8 whichAttack

The whichAttack function returns if the artificial intelligence can attack or not by iterating through its attacks and checking if it has enough energy to attack a certain attack. (first attack has a value of 0 and second attack has a value of 1) The "-1" means it cannot attack at the moment.

1.9 retreatPokemon

We created a retreatPokemon function which allows to retreat the active pokemon card when its current (remaining) hp is less or equal to half of its original hp. The pokemon card with a low hp retreats to the bench and the other pokemon card (from the bench) becomes an active card. If there is no pokemon card on the bench, the active pokemon card cannot retreat.

1.10 setTime

We made a setTime function to make the artificial intelligence think slower, but towards the end the GameHub team made one internally for the game.

1.11 turn

In the turn function we all use the above functions except for the setTime function in a certain order. The order is important, but if you want to make the artificial intelligence unpredictable we would change the order of the functions every time our turn is called. We would have done that if we had time and have implemented the behaviour tree correctly. But the attack function should always be called at the end in the turn functions.

2 Things that need to be done in the future

The attachEnergy function should definitely be re-implemented. So the artificial intelligence would attach the right energy on the card. We think if we had more time or managed our time better we could have thought of more situations that might happen in the game. The artificial intelligence is very basic at the moment and could be smarter.

- Such as implementing when to use an ability
- Such as attaching the right energy to the card and attach an energy card on the pokemon which needs the most energy
- Implement the trainerCards functions
- etc.

Issue with card Herdier, it does not attack.

3 Extra

You can find our code in the folder that Jack Diver has submitted.
If you look at the AIPlayerBT class you will see a lot of functions that are commented. This were the functions we implemented for the behaviour tree but it got too tedious so we gave up on that idea due the lack of time.