Bomberman Minimax/Expectimax Report

Members:

Hoang Nguyen
Tuomas Pyorre
Dang Tran

Structure of the approach

Variant 1: A*
For variant 1, the only world without a monster, we imagine that any approach we choose to do would work. A* simply creates a path from the character to the exit. With no external factors to consider, this algorithm should work 100 percent of the time.

Variant 2: A* and Expectimax
Since the monster is moving randomly and not in an optimal way where it's searching for the character, the best approach to this would be a probabilistic approach which is why we chose Expectimax. The chance node here was used on the possible moves that the monster can do.

Variant 3-5: A* and Minimax
In these variants we are exposed to more logical monsters, namely the self-preserving monster and the aggressive monster. As such, we need to put more thought into what the best move for the monsters would be. Minimax considers both the character and the monster's best move.

We did however consider that minimax should not be used for all cases:

- When the character was not close to the monsters
- When the character was close to the exit
- When the character had a clear path to the exit, by getting behind the monster
    - If and only if the astar path length of the character was less than that of the monster, we would always make it to the goal, before the monster makes it to us.

In those cases we used A* or forced the best action to be the exit for each case respectively. This could have been used with expectimax for optimization, however it was not considered at the time.

Interesting tidbits

Individual sections for each interesting bit of your approach

Expectimax:

- getLegalActions() - this helper function list the possible moves that the character/monster can do. This is determined by the walls and the bounds. In the character case, this is also determined by whether there's a monster nearby.
- The utility value in this algorithm was determined by the score and the A* distance from the character to the exit. In cases that the character is killed or the character reaches the exit, the utility value is decremented and incremented more accordingly. Another approach for the utility that ultimately failed was using the euclidean distance between the character and the exit. Without consideration for the path created by A*, the character would be stuck in a corner of the map.
- A flat depth of 2 was used for this algorithm with variant 2. A depth of 3 was used originally and that worked but it was too slow so we opted for 2 which still have a perfect success rate.
- The terminal state is set to be whenever the character is killed, when it reached the exit, or when the flat depth is reached.
- The probability for each action was simply 1/(total number of possible moves)

Minimax

- Reward - this provides the utility for each state -- where the score is impacted by the length of the A* path and the distance between the character and the monster. Instead of adding the distance directly, we mapped the distance with some fixed value instead. This is as the monsters have a detection range, we want to stay as far as possible away from this detection range. We should penalize going near it by a very considerable amount, or our character is assured to die.
- Minimax terminal states -- Since we are using minimax just as a means to avoid the monster, as part of the terminal states (exiting or dying from the monster) we made sure it stops after some number of iterations. The main difference between the terminal states of the expectimax and the minimax-A* algorithm is the values assigned.

Variants:

Variant 1:

- This was obviously the easiest, it allowed us to just structure our Astar code into a search.py file, which we could later call instead of adding the Astar into all of our files individually.

Variant 2:

- Only one we used expectimax for eventually, we also tested expectimax later for other variations too, but we mostly ended up using minimax.

Variant 3:

- The program started getting more difficult here due to us wanting to now implement minimax for beating this variant. This variant was not particularly difficult still, and our minimax did not have to be perfect for beating this variant.

Variant 4:

- This one might have been the most difficult and time consuming of all the variants. The minimax for this one became a lot more complicated than it was previously due to the detection range of 2.
- We wanted a large depth for this variant, so we needed to implement alpha beta pruning, which hopefully we implemented successfully.
    - This turned out to be surprisingly complex, as we needed to figure out WHEN we actually want to change the alpha to alpha, and when the beta comes in, and how they play at each node from top to bottom.
- On top of alpha-beta there are a few tricks to make this one faster.
    - When we know we are going to get to the goal faster than the monster is, that means that we must have a path that is shorter by using Astar, and that must mean that the monster will never be able to catch us. In this case, it only makes sense to rush to the goal immediately.
    - We try to immediately minimize the “dumbest moves”. For example, in 99% of the scenarios, if we immediately stumble on the detection range, we will just die. So we avoid that branch and will not be exploring it too deeply.

Variant 5:

- Surprisingly this took far less time than variant 4. This was essentially just a reimplementation of Variant 3, but we just added the stupid monster in there with the self preserving one.
- With this variant, we are implementing some of the same tricks as in variant 4, like looking at if we are faster to the goal, than BOTH of the monsters, then there is no way they can ever kill us before we get to the goal.
- The interesting part of this one, is that we have a max function, that looks into a min function which monster 1 operates with, and then that one looks into another min function with which monster 2 operates with. This increases the computation by a lot.
- We were able to make the depth only to 2, and still reliably run this variant. It has a weaker % of how much it survives due to this, but it still goes well above 50% time of living.
- This variant also ran into a funny bug on random seed 9, where when the monsters sat next to each other, we could no longer get both keys, so we had to make a try except statement just for that specific seed.

Experimental Evaluation

How many times does your character survive in each situation?

Variant 1: 100%, there's no monster so as long as A* works the character survives.

For the remaining variants, we ran the game with 20 different seeds. As such:

Variant 2: 100% survival with expectimax. Was able to lower the depth searched to 2 from 3 to speed up performances.

Variant 3: 90% survival with expectimax. The monster catches the character from time to time but expectimax still performed pretty well. 100% survival with minimax

Variant 4: 80% survival with minimax, ~40% survival with expectimax.

Variant 5: 70%(likely better) survival rate with minimax. 50% with expectimax, if it cleared the aggressive monster, it would survive.