Bomberman Approximate Q-Learning Report

Members:

Hoang Nguyen
Tuomas Pyorre
Dang Tran

Structure of the approach

For all the variants we choose to use our state machine system from project 1 with additional use of the approximate q learning algorithm to adjust the weights.

We also explored q-learning to no successful degree. We talk about the issues that came from our implementation of q learning (not approximate q learning).

Bombing State Machine:
For a large portion of the decision process we incorporate a state machines:

1. **Exploring** - general exploration, avoiding the monster and heading to the exit
2. **Bombing** - place a bomb
3. **Running** - move diagonally away from the bomb after placing
4. **Waiting** - stay in place after moving (move(0, 0)) since there were some issues with the character randomly moving after running when it wasn't told to, resulting in it going into the explosion sometimes.
5. **Keep Waiting** - last state before the explosion is gone, essentially the same as exploring.

Our implementation of approximate q learning occurs in the *exploring* state.

Interesting tidbits

Approximate Q-Learning:

The features that we arrived at were:

- The euclidean distance between the character and the monster
- The distance between the character and the exit
    - Use the length of a A* path if one was available
    - Otherwise use the euclidean distance

Using those features, the approximate Q-learning algorithm is shown below:

alpha: 0.8
decay: 0.9

Features-based representations:
fe = $1/(1 + d_e)^2$ where d_e is the euclidean distance to the exit
fm = $1/(1 + d_m)^3$ where d_m is the euclidean distance to the monster

Calculating the Q Value
Q = wm * fe + wm * fm

Updating the weights
Δ = (reward + decay * max Q') - Q

we += alpha * Δ * fe
wm += alpha * Δ * fm

With we and wm are the weights to the features fe and fm, respectively.

Tuning the algorithm, the weights were augmented to be 0.3/0.4 what it is since the high numbers were giving issues. If we take a look at fm, a power of 3 was used to reduce bomberman's avoidance of the monster. At power 2, bomberman was too scared and remained in the corner. We have a minus wm since we want to avoid the monster.

One thing that was noticeable was the commonality in the deaths of bomberman. It seems like once the bomb has exploded, the bomberman was trapped in the corner. With aggressive monsters, this would place the character between the explosion and monsters. We could in the future try to seek the behavior of killing monsters before

progressing, or choose to place the bomb in between the character and the monster. This is somewhat connected to choosing a better cell when running away from the bomb.

Bombing State Machine Future Improvements

Looking at our state machine there are places we can improve on for next time:

By default the character is in the *exploring* state and it'll go into the *bombing* state whenever there's a wall next to it. This is less than ideal if we wanted our implementation to learn how to place bombs. On a separate note, it would be good if bomberman transitions to the *bombing* state when it's near the monster.

The *running* state could also be improved, right now it just chooses the first possible diagonal move when it needs to leave the bomb range. Instead of choosing the first diagonal move, we choose the diagonal move furthest away from the monsters.

Experimental Evaluation

Modifications to scenario file:

For each variant file, we added a **for** loop that runs the game 20 times with random seeds from 1-200. We added a 5 second wait to allow for data collection.

We also recorded the final weights of our bomberman character, and for the survival rates shown, bomberman was initialized with these weights:

we = 3.2240646069681054
wm = -0.28043526385758355

Variant 1: 100% Survival Rate
Variant 2: 85% Survival Rate
Variant 3: 75% survival Rate
Variant 4: 60% survival Rate
Variant 5: 50% survival Rate