CS4341- Introduction to AI

Final Project Report - Bomberman

Alex Hard, Sam Moran, Ethan Schutzman

February 24th, 2019

Introduction

Bomberman is a classic home computer game released in 1983. It has a simple puzzle solving premise making it an ideal candidate for a basic AI to learn. The team sought out to solve the game using A* Search Algorithm, Expectimax, and Approximate Q-learning.

Initial Design

Scenario one variant one can be solved using any graph search algorithm. We discussed possible solutions to the level and decided that it would be most effective to write A* to find the most optimal path because it would be relevant in more complex solutions. From the start we also wanted to try for an ambitious solution. We wanted to implement Q learning to make our artificial player as smart as possible. We also planned an expecti-max implementation as a back up design for if the q-learning was not successful. . From these design decisions we started working on the project. Our first goal was to implement a working version of A* as soon as possible to get our project flowing as A* would then be used when the AI was not using Q-learning.

Algorithms

Our first implemented algorithm was the A* algorithm. Since this was going to be the foundation for our reward functions we wanted to make it as efficient and effective as possible. Initially we had problems with our frontier and our character was not making the optimal move even though that was its desired path. We eventually resolved the issues and had an effective solution that solved scenario one variants one and two with 100% and 85% accuracy respectively. We wanted to see how well the AI would play against the smart monster using just A* and unfortunately it did not succeed with a passing rate. This meant that we needed to design a second algorithm for testing against smarter monsters.

We then used a wall expansion algorithm to effectively treat monsters as a wall and the spaces around the monster as a wall. This allowed us to treat the monster as a moving object which can be pathed around using A*. This expansion when coupled with our A* pathing was able to beat scenario one variant three with enough accuracy to pass the 80% threshold. While it was generally successful against variant three, the AI for that variant was eventually remade using Approximate q-learning.

Next we implemented expectimax as our second algorithm. A basic skeleton of the algorithm was implemented, however expecti-max was never used by the team. By the time the expecti-max algorithm was in a nearly unusable state, the team had already made a working q-learning agent. Because of that success, expecti-max was saved but never used.

Finally we implemented Q learning. The team encountered numerous issues with the algorithm acting almost correctly, but still resulting in failing to solve the problem. Many iterations were attempted fixing, changing, and adding different features to try to correct it. After attending an office hours session, the team learn that they had created a mix between Q-learning

and approximate q-learning. Armed with this new information, the team agreed to switch to strictly approximate q-learning and found greater success.

Variant Analysis

Scenario 1

The scenario 1 pathfinding with few walls proved to be a great testing and development ground for a approximate q-learning algorithm. The team's approach to each variant is below.

Variant 1

Variant 1 was solved using a basic A* algorithm. When tested over 100 randomly seeded games, the AI won 100% of the time.

Variant 2

Variant 2 was solved using a modified A* algorithm. It paths to the exit, but is aware of the monster and will adjust it's path to avoid going near the random monster. the AI was able to win 100% of games played based on 100 randomly seeded games.

Variant 3

Variant 3 is the start of the team's approximate q-learning AI. This version also takes into account the distance to the monster, the distance to the goal and the distance to the corners in the board as the team had found that the agent was being killed by the monster after being backed into a corner. In order to combat this the team added in a corner weight trying to encourage the agent from back tracking into a corner. This added feature was successful, and in our testing of 100 games, the agent was able to win 100%

Variant 4

Variant 4 continued to build off of the approximate q learning from variants 2 and 3. Variant 4, however, was not as successful as its predecessors; the aggressive monster was a bigger challenge. Nonetheless, the agent was able to beat the variant 4 monster 80% of the time based on 100 randomly seeded games.

Variant 5

Variant 5 was the final evolution of the basic approximate q-learning agent. Surprisingly the agent worked rather successfully against the two monster. While not able to win 100% of the times, it was impressive and being able to navigate and 'juke' its way past the monsters to goal. The biggest issue came from identifying the nearest monster. Since the code only looked for the closest monster, the other one could sneak up on the agent and rapidly become the new closet monster, pinning the agent between the two enemies. From the team's testing it was found that the variant 5 agent was about to win 60%-70% of the team against 100 randomly seeded monsters.

Scenario 2

Scenario 2 proved to be extremely difficult for the team and require many hours of working together designing and implement a solution.

Variant 1

Variant 1 saw a return to A* as the solution to the variant. While certainly not the most efficient or the fastest, A* was able to get the variant completed 100% of the time.

Variant 2

Variant 2 is were the challenge began. The team first attempted the variant by using the rather successful approximate q-learning agent from scenario 1 variant 2,3, and 4. This did require modification to incorporate the planting of bombs, something the team did not have in scenario 1. The result was mixed. At times the agent was able to successfully use bombs to create a path to the exit, and also evade or blow up the monster. On the other hand there were also issues where the agent would walk into the monster or the explosion of the bomb, killing itself.

The team also attempted an A* based approach to this problem by creating a state machine with the states: safe, in danger, and bombing. These states were calculated and changed based on the actions of the agent and the monster. In danger, or DANGERZONE as referred to in the code, was based on a 3 block radius around the monster. This was designed to prevent the monster from ever getting close to the agent. Bombing was also meant to prevent the agent from walking into the range of the bomb. This method also worked to some degree of success, and the team saw about 55% of games run resulting in a win.

The team ultimately went with the approximate q-learning mode but struggled with properly preventing the agent from moving into the monster's or the bomb's range, and spent much time troubleshooting small bugs. The idea was to create a list of valid moves, a list of moves that would put the agent into the monster's radius, and a list of move that would put it in the path of the explosion. These lists were then subtracted to find the list of "safe" move. This approach worked at times and failed at other times and proved to be a big choke point for progress. Eventually the team chose to move on to variant 3, where it found great success. The team then chose to use the new variant 3 agent to win variant 2. The win rate was over 85%.

Variant 3

Variant 3 saw a dedicated return to approximate q-learning. The team added more features to the approximate q-learning equation. The team began to weight not only the distance to monster, the distance to goal, and corners of the board, but also the distance to a bomb and being in the spaces where an explosion will occur. The team also weighted placing a bomb near walls to be a good option. This introduced an interesting behavior the team dubbed the "crazy bomber" or more simply "the lunatic". The agent began placing bombs any time it got near walls, even if it had already made hole to progress through. This behavior did have some advantages as the extra bombs happened to kill the monster at times. The team evolved this behavior by adding a 66% chance of placing a bomb when it detected it was near a wall. This

behavior helped prevent the agent from accidentally killing itself as much (as the weights for explosion positions did not act 100% consistently) but also increased how often the agent was killed by a monster. This was because at times the agent only had a moment to place a bomb before being forced to retreat from a monster, but at times the randomness would not allow it to place the bomb, wasting a turn, so this feature was then removed. The team then gave a larger negative reward for being in spaces that were about to be an explosion, making the agent much more effective. This agent was able to win 85% of games run in our tests.

Variant 4

Variant 4 was basically identical to that of variant 3. The same features were used as in variant 3 as the core problem did not change, rather just the aggressive range of the monster. The agent saw moderate success against the aggressive monster, winning about 55% of the time. The team found that the larger range of the monster caused the agent to not be able to react quick enough to its attack or that it began to move into the way of the bomb again. The team attempted to adjust the weights of the features and the learning depth, but to no avail.

Variant 5

Variant 5 was also the same base as variant 3 and variant 4. The agent works surprisingly well against the two monsters, with it often killing the random monster, but falling prey to that of the aggressive monster such as in variant 4. If the team had been able to track both monsters at the same time, or been able to differentiate between a smart and an aggressive monster, the approximate q-learning would have been much more successful. The approximate q-learning agent was able to win 40% of the time in our testing.

Approximate Q-Learning Features

The approximate Q-learning function started with two main features and grew as the needs of the function grew. Listed below are the features the team had implemented and the reasoning for their implementation.

1) Distance to monster. This was one of the first features the team implemented due to its necessity. The feature calculates the A* distance to the nearest monster. This is then used to weight the agent to stay away from the monster. This feature generally has a large negative value over -100.

2) Distance to goal. This was also one of the initial features and was used to entice the agent to move towards the goal. This feature also calculates the A* distanceto the end goal and generally has a large positive weight around 100

3) Distance to corner. Distance to the nearest corner was added next. This was done due to the team seeing the agent attempt to evade a monster but get caught in a corner and eaten. This feature calculates the A* distance to the nearest corner. This generally returns a negative number not as large as the monster, but still rather negative; something around -50.

4) Distance to bomb. This feature calculates the A* distance to the nearest bomb and was implemented for scenario 2 variants. This was used to prevent the agent from standing close to the bomb and blowing itself up. The team found difficulty weighting this feature correctly as they found negative values began to prevent the agent from placing bombs due to an immediate low score.

5) Weight of explosions. This feature was designed to prevent the agent from standing in the path of an explosion. The team found difficult with its implementation and suspects that the world may calculate the path of an explosion in an odd fashion. Regardless this weight was added to be negative; to discourage being near or in an explosion. After much tweaking a lower negative value was found to be effective in preventing self detonation.

Shortcomings

Due to time constraints we were unable to fully realize and implement a 100% accurate Q learning system. We were able to make something that learned and made good decisions but we wanted to spend more time adjusting weights and configuring learning rates to see what would have yielded the best results. The choice to move to approximate q-learning saved the team stress and time but was not as thorough as full q-learning would have been. The team worked ambitiously to complete all 10 scenarios but after approximately 70 hours together working on the project through daily multi-hour work sessions, the team was forced to focus on other classes and exams and instead aimed to make the best possible agents with their remaining time. While disappointed at time with the stagnation of progress, the team never stopped collaborating on a better solution and is proud of what they were able to accomplish.

Final Design

Our final design varies per variant in each scenario. In both scenarios, variant one is a simple exercise in pathing. All future variants utilized approximate q-learning to solve the game. Each variant grows upon the previous as more complex challenges are presented. These added weights and features lead to a better and more advanced AI.

The team was able to learn a lot about approximate q-learning from this exercise and, although it was frustrating at many times, gained true practical experience its creating an implementation.

Recommended Improvements

Looking back at the project the team wished that it was able to discover the issue with the mixed Q-learning and approximate q-learning sooner. Thi would have allowed the team make progress earlier and eventually better the program more. The team would have also wished to have had time to make more complex features for approximate q-learning such as time until bomb explodes and distance relative to bomb and monster. these added features would have allowed the agent to be more mobile and flexible around bombs and evade monsters better while not getting caught in an explosion.

Bugs

Below are a list of potential bugs the team found during the project. Unfortunately the team did not have time to allocate to confirming and providing a fix for them, but the team hopes this list hay help the project code be improved for future use.

- Monster movements in relation to board UI. this bug was **found and fixed** during an office hours session with the professor. It was discovered the board was updating with the monster still shown in its previous position.

- Exiting and losing simultaneously. The team found that at times the character could move to the exit to win, but have the monster also move into the exit and eat the character. This would result in a loss even though the agent made it to the exit. This was not fixed.

- Explosion calculation may be off. The team was unable to spend much time investigating this, and it may not be a bug. However we found at times the character would be killed by an explosion even when not close to it. There may have also been an issue in calculating explosion distance, but once again this may have been the team's misdoings.