WORCESTER POLYTECHNIC INSTITUTE

ROBOTICS ENGINEERING PROGRAM

Project 1 Report

SUBMITTED BY

Max Berman

Robert Gunduz

Aliaa Hussein

Date Submitted: 2/13/2025

Course and Group: RBE4701 Group 1

Course Instructor: Professor Carlo Pinciroli

Scenario:

Our goal for project 1 is to implement behavior into our character in Bomberman which will allow us to clear five predetermined scenarios. The program is considered a success if it can clear each scenario with a success rate higher than 50 percent over 10 trials each. The maps are the same in all scenarios, and for scenario two and onwards there are varying combinations of monsters to avoid. There are two variations of said monster, one being driven entirely by chance, and the other having self preservation and the ability to pursue our character.

Program structure:

For future flexibility we have decided to implement the framework for a state machine. In all variants provided for this project, the topography of the map remains constant. The variance in each scenario comes with the monster entities present on the map. While there is a clear, wide path from start to finish now, there may not be later. This approach will allow us to easily add new unique behaviors to react to certain environments. The example that comes to mind is using the place bomb action to open a path or eliminate a monster. For the time being, we have utilized two states to achieve the desired results.

By default, our character is programmed to find the most optimal path from start to finish by using the A* Algorithm. Given the environment and available alternatives, the algorithm is complete, efficient, and optimal. The world is represented as a two-dimensional grid, i.e. a graph, with additional “flags” to indicate obstacles, entities, and goals. So long as the heuristic (h(n)) we choose is consistent, A* will be optimal. For this we have experimented with Manhattan distance, Chebyshev distance, and Pythagorean distance. All three of which are consistent and generate slight variations in path planning. The final consideration with A* is its exponential memory growth, meaning we will store an exponential amount of information. However, with the size of the map and resources available, this drawback is negligible. The whole algorithm is repeated on every turn.

In the event a monster is detected within a certain distance of the character, the state will switch to prioritize evading the hostile entity. This is done to self-preserve the life of the character as that matters most of all when trying to get to the exit cell. To avoid the monster, another A* implementation is used to prioritize cells that are farther away from the monster but also adds weighting towards the exit cell. Furthermore, if the monster is detected behind the character, we have programed the character to ignore the nearby monster and run towards the exit as fast as possible.

Important Methods:

next_move (A-Star):

The core of our character's decision making is driven by a slightly modified version of a generalized A-Star algorithm. First and foremost, we want the character to find the quickest possible route to the exit square when it is safe. The method was further modified to incentivize path planning around hazards, in other words, monster entities. When a cell is being added to the frontier dictionary, the cell is checked to see if it falls within a defined radius around the monster. If this check returns true, an additional cost to moving into that cell is added scaling with the distance from the monster. Once the exit square is found, the method traces the path back using a stack. The stack is popped twice to obtain the coordinates of the move target. The method returns a tuple containing X coordinate, Y coordinate, and path length.

avoid_monster:

If a monster entity is detected nearby and in front of the goal, the character defaults to this movement method. The character senses the world for all monster locations on the map, as well as the exit position. All valid neighboring cells are then evaluated by a safety metric based on these positions. First, the reciprocal of the distance from each cell to each monster is summed. Then the reciprocal of the number of moves from each cell to the exit is subtracted. The cell with the minimum score is chosen in this case and returned from this function.

monster_near:

The first part of threat sensing for our character, this method detects if a monster entity is dangerously close to the character. The method compiles a list of all monster locations, then runs each one through the same heuristic function used in 'next_move' to the current position. If this distance, for any monster, falls below the threshold argument, the method returns a tuple of True and the distance of the imminent threat. Otherwise, it returns a tuple of False and –1.

book_it:

In conjunction with 'monster_near', this method is meant to control when the character enters an evade-at-all-costs response. To elaborate, the method uses the heuristic function to obtain the distance between the nearest monster to the exit and the character to the exit. If the former is greater than the latter, the method returns True. This means the character will proceed to move, as quickly as possible, to the exit if it finds it has already passed the monster that is close by.

monster_heuristic:

This method is used as a part of the monster avoidance to calculate what cells the monster can move based on a lookahead input. A lookahead input of 1 is just the neighbors of 8 of where the monster can move to. When using a lookahead greater than 1, it uses the depth of 'neighbors^lookahead' for where the monster can get to based on the number of moves. This method also returns a heuristic number for each monster cell which can be used in A* if desired.

avoid_safe_cell:

As a part of the monster avoidance strategy, this method is used to get a safe cell to move into based on a threshold input. It uses the 'monster_heuristic' method to get all the cells that the monster could move into based on the threshold input and compares it to the character's valid moves. If there is a valid move, it takes the move that is closest to the exit based on the heuristic function. If there are no valid moves then it returns 0.

Experimental evaluation:

Overall, this code is used for all variants as it shows some robustness and gets over 50% completion rate on each variant. Some of the bias values could be set to beat certain variants better than others but our test character can clear each variant with varying success rates. For variants 2-5, 50 trials were conducted to test the validity of the AI's behavior. For variant 1, only 20 trials were completed as there are no monsters in that scenario. The results of the trials are displayed in table 1 below.

Scenario	Trials	Survived	Survival Rate (%)
Variant 1: Alone in the world	20	20	100
Variant 2: Stupid monster	50	46	92
Variant 3: Self-preserving monster	50	41	82
Variant 4: Aggressive monster	50	29	58
Variant 5: Stupid & aggressive monsters together	50	29	58

Table 1: *Survival rate of AI character for variants 1-5*

Variant1.py:

The Test character can clear this stage cleanly with a 100% survival rate. The absence of monsters, and by extension random variables, makes each run identical and ideal. The character remains in normal movement behavior throughout run time.

Variant2.py:

The character clears this stage with a nominally high success rate of 92%. The monster entity in this stage moves randomly and does not react to outside stimuli. This allows our character to 'dance' around the monster or wait until the monster moves into a non-threatening position on the grid. The character may die to the monster when backed into a corner, however this occurrence is rare without the monster having hunting behavior.

Variant3.py

Like the previous scenario, there is one monster present. However, now the monster will prioritize attacking adjacent character entities. In most tests, the character can wait for an opportune moment to sneak past the monster. But certain high-risk states where the character had to move adjacent to the monster are now non-viable. The character gets trapped in corners more frequently.

Variant4.py

Building off the pervious scenario, this variant has a monster which chases player entities within 2 tiles. In addition, the monster starts right next to the goal. These two factors make it far more likely the monster will not only encounter but also eliminate our character. Through testing, we found the success rate to be around 58% of all trials done, with success being highly predicated on the monster's random movements. In reflection, this was the hardest variant for our character to complete.

Variant5.py

Finally, the fifth variant contains two monsters, each from second and third variants respectively. The character can complete the variant with a moderate rate of success, around 58%. Failure comes down to getting cornered by one or both of the monsters. More often than not, the two monsters will stay farther apart, allowing the character to move past them like in variants two and three.

Conclusion

The implemented AI character behaves effectively for the given five variants of the scenario. The character evaluates a scenario using heuristic-based avoidance strategies and incorporates a modified A* pathfinding algorithm. The AI was able to clear the scenarios with over a 50% survival rate for each variant. The variants with the 'aggressive' monster and two monsters notably decreased the survival rate. This implementation is modular allowing for modifications and additions to maximize the survival rates for these

variants. Further improvements can include path prediction based on the monster's movement, placing bombs strategically, including different 'danger levels' as a part of the avoidance strategy.