CS 331 Introduction to Artificial Intelligence

FINAL REPORT

# Solving the Bomberman Maze using several search strategies

## Two and a half Bombermen

GROUP 27


| | |
|---|---|
| **Project Lead Fnu Tulha** | **17100138** |
| **Muhammad Hamza Altaf** | **17100269** |
| **Anas Zahid** | **17100066** |
| **Ahmed Ali** | **17100012** |


Submission Date: 13th December 2016

Assigned TA: Maryam Khalid

Submission Date:

Assigned TA:

# Contents

## Project Summary

In the very beginning phases of the project, it was supposed to be a naïve implementation of the A* search algorithm to the Bomberman maze problem. However, with during the course of the semester it turned into the solution of the Bomberman maze problem with the use of adversarial search and MDPs in addition to A * search. This project explores the relative differences of the approaches that are used to solve the problems, the issues faced in specific approaches and the solutions that were proposed to deal with these problems.

Secondly, the minimax algorithm had to be developed from scratch since the Bomberman game that was built upon was an existing project of a CS course by a group of students. A version of an adversarial search algorithm had to be implemented by considering the context of the game. Since drawing the entire game tree of Bomberman was a task of high time complexity; O($4^n$) to be exact where n is the number of moves taken by player 1 to reach the destination, it became clear we had to use a different approach to deal with this predicament. Therefore, the entire minimax algorithm was divided into n sub intervals where each interval represented a minimax move on part of the player and the enemy. In essence, this had turned into a real time gaming problem where two agents were facing off in a minimax fashion to achieve each of their desired goals.

# Enter results for the different algorithms here and their comparison.

## Introduction

Bomberman was developed by Hudson Sot in 1983. It is a strategic maze based game in which the user has to plant bombs carefully and in efficient manner to destroy obstacles and reach the desired position in the maze. This project will be an implementation of an artificially intelligent agent within the game using A*, adversarial search and MDPs as the base algorithms. The aim is not to make the game but construct the intelligent agent in such a way that this search works in the way it is supposed to reach the desired goal. [1][2] In this report we intend to explain the challenges, pitfalls of applying A*, Minimax, Expectimax and MDPs to this version of the game and how our solutions when compared with each other showed the best approach that could be used for a fixed number of games. In the project description section we intend to explain the details of the algorithms used and the changes that had to be made to the game so that our algorithms could be made compatible with it. It is noteworthy to mention that the structure of the game had to be understood from scratch so as to understand the exact changes that had to be made so that the maximum amount of work could be done by us. The reader of this document should look forward to different versions of strategies being used to evaluate the best move to a particular goal location in the game as the report progresses.

## Background

Pathfinding is a very popular AI problem in the game industry. For years, various pathfinding algorithms were created and implemented. Examples of such algorithms are Dijkstra's algorithm, breadth-First Search and depth first search algorithm which were created to solve the shortest path problem. However, it was proved that the optimal solution to the pathfinding problem is the A* algorithm. A* search algorithm works by exploring the unexplored locations repeatedly. When a location is explored, it checks if the new location is the final goal. The algorithm ends if this condition is met otherwise it further explores the neighboring locations. A* might be a very good search algorithm however it requires a huge amount of memory to perform search on large and complex environments. To avoid this wastage of memory we can allocate a minimum amount of memory before we execute A*. If, during the execution, the memory gets exhausted, we create a new buffer to progress the search. The buffer size will be kept dynamic so minimum memory gets wasted. [3].

## Project Description

[4]

This game was a project made by students of an introductory computer science course BOMBERMAN E177 Spring 2013 Final Project created by Nelson Wang, Tianyu Wang and Peter Kwon. The game uses a grid to represent map locations, hence, we can simplify the movement of our player (Bomberman) as movement of an object in a maze. Although it seems perfect, it is not that simple. The player in this game has the choice to create his own maze by bombing the walls however the player needs to decide on many problems. For example, bombing a wall that is closer to the goal might not be a very good choice every time and after placing a bomb the player immediately needs to run to a safe location as the explosion of the bomb kills the player if the player is closer than 3 grids to the explosion. So a major part of this project involves designing of a heuristic that caters for all these issues. Programming will be done in Python and libraries for GUI may be used but as mentioned earlier the initial goal is to solve the problems associated with incorporating A* search in this particular game. The paper that we used an an inspiration for our project was indeed the stepping stone behind an entire project dedicated to an analysis of various search techniques to evaluate the best move in a particular game. [5][6]
It is important to mention that this is the only code that was referenced from an external source and all other work was original work by the team members of this group.

## Project Description

**Version 1: A* search**

A star algorithm was implemented by considering the cost function as well as the heuristic function at particular point in the maze. The Manhattan distance was chosen to the the heuristic for this particular problem. The goal was predefined and the player had prior knowledge of it. This was one fundamental assumption about the system. However, this particular heuristic was not saving us time. For example, it chose specific bricks in its path and bombed those obstacles instead of choosing a simpler way to reach the goal. So we came to the conclusion that in our implementation we could not take into account the time factor

**Version 2: MDPs**

Implementation of an MDP came to our mind after we implemented a naive version of value iteration in our assignment. However, our version was on a completely different level because in addition to incorporating bricks in our policy evaluation we had to consider bomb placement along a particular path as well. In the usual cases, the evaluated path completely avoids the bad

$$V_{i+1}(s) := \max_a \left\{ \sum_{s'} P_a(s,s') \left( R_a(s,s') + \gamma V_i(s') \right) \right\},$$

states.

The value iteration algorithm update equation.

**Version 3: Minimax**

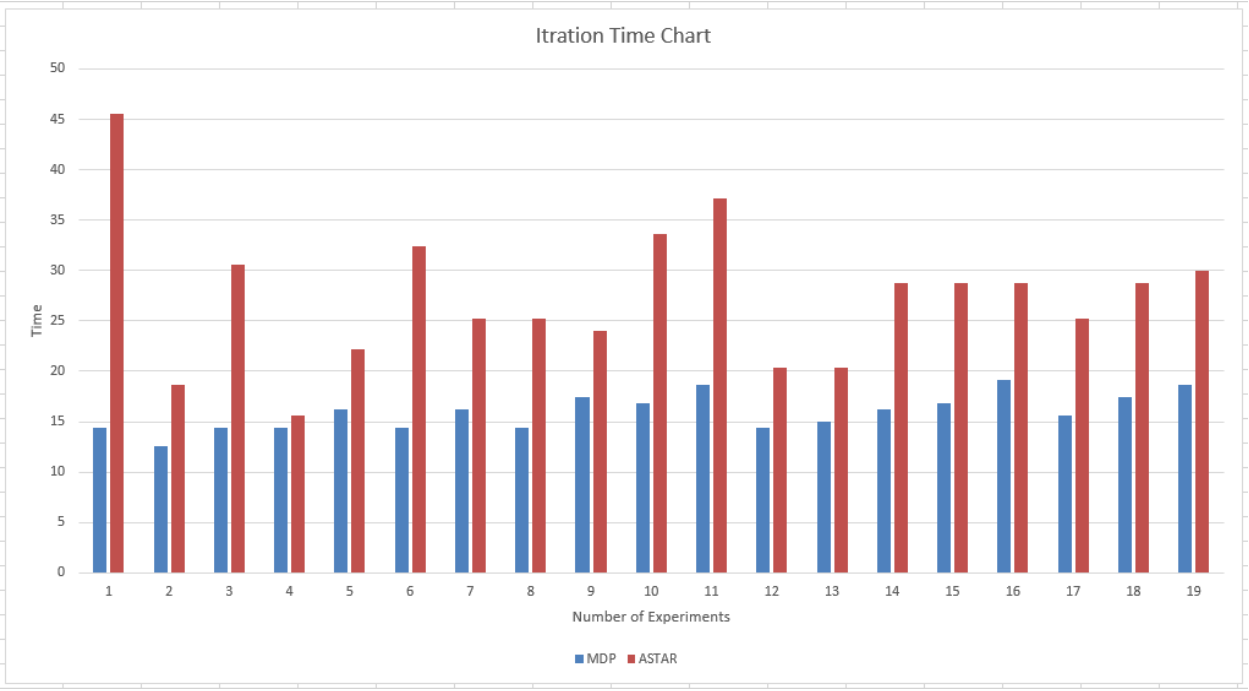
We initially began this part by introducing a different version of the game by introducing a separate enemy from the pacman game which would follow the minimax agent model along with the opponent. The mathematical model that we used here was that since minimax is an implementation of an adversarial game that is being played by two opponents, why not consider a turn by turn game to evaluate all the possible nodes of the game tree and out of those nodes evaluate the best possible pathway that satisfies the utility conditions of the minimax algorithm. So we could visualize the minimax algorithm as a subset of multiple minimax moves that eventually result in the evaluation of an entire search tree form the start position to the final position. The challenges faced in this section will be explained in the contributions section. But the basic mathematical formulation that followed has been explained above.

**Version 4: Expectimax**

In this version of the game, the opponent was modelled as an enemy whose moves would be arbitrary based on the random function of Matlab. Out of the four possible locations to move somewhere at any point in time, as pseudorandom number would have to be generated and out of a list of 4, the highest one selected would correspond to the best move that would be selected.

## Results

What results did you obtain?



## Conclusion and Future Work

So basically, an implementation of A*, minimax, Expectimax as well as MDPs to demonstrate how the best path to a goal can be evaluated for this particular system was the scope of this project. Initially the project was only supposed to be an A* implementation of the project but finally

Write future work in paragraph 2. This will include tasks that you had originally planned but could not complete, what more can be done to make the system better and if there are any other directions that can be taken and that add to the capabilities of the current system, sticking to the challenges you were dealing with.

## Activities

What are the tasks that you completed for the project?

-A star algorithm

-MDPS

-Minimax

-Expectimax

-Analysis :

- Representation of A star and MDPs
- Graphical analysis of A star and MDPs
- Representation of minimax and expectimax
- Graphical analysis of minimax and expectimax

## Timeline

| ACTIVITIES | Start Time (weeks) | End Time (weeks) |
|---|---|---|
| Background research | 0 | 1 |
| A star implementation of algorithm | 1 | 3 |
| A star implementation for simple maze | 3 | 4 |
| -Understanding framework of bomberman game | 4<br><br><br>5 | 5<br><br><br>6 |

| | | |
|---|---|---|
| -Incorporating automated movement of agents In bomberman<br><br>-Complete a star implementation<br><br>-Work on minimax begins<br><br>-work on Expectimax begins<br>-Expectimax version two begins<br><br>-MDPS | 6<br><br>7<br><br>9<br><br>11<br><br>13 | 7<br><br>9<br><br>11<br><br>13<br><br>15 |
| | | |

## Contributions

In the early stages of the project, there were mainly problems with the integration of our concepts with the already made game code of Bomberman, which as a group, focusing on accessibility, we managed to utilize very early on.

**Member 1 [Anas Zahid]:**

- A star algorithm implementation
  i. Choosing a heuristic to begin with that satisfied the conditions of this game specifically
  ii. Implementation of the actual algorithm was difficult and challenging
- MDPs implementation
  - Implementation of the optimal policy using matrix representation
  - The algorithm was in itself challenging to debug and run for a matrix that is 15 by 15
  - Setting the reward for the bricks was problematic because the window associated with time consideration for a particular part of the maze was very small. This caused a predicament when we tried to set the reward. This was solved by iteratively updating the reward and not keeping it constant

**Member 2 [Fnu Tulha]:**

- Expectimax implementation

- Minimax implementation:
  The challenges faced here were:
  -Incorporating the tree structure in the current format of the game
  -Modelling of a special enemy using Manhattan distance
  -Going up to a specific depth for the game tree which was challenging to implement recursively
  -Implementing the boundary conditions for the sides of the maze (a little bug)
- Analysis using the minimax and Expectimax graph
  - Games were run for specific values of the grid location where the goal was varied and comparisons were drawn between different approaches

**Member 3 [Muhammad Hamza Altaf]:**

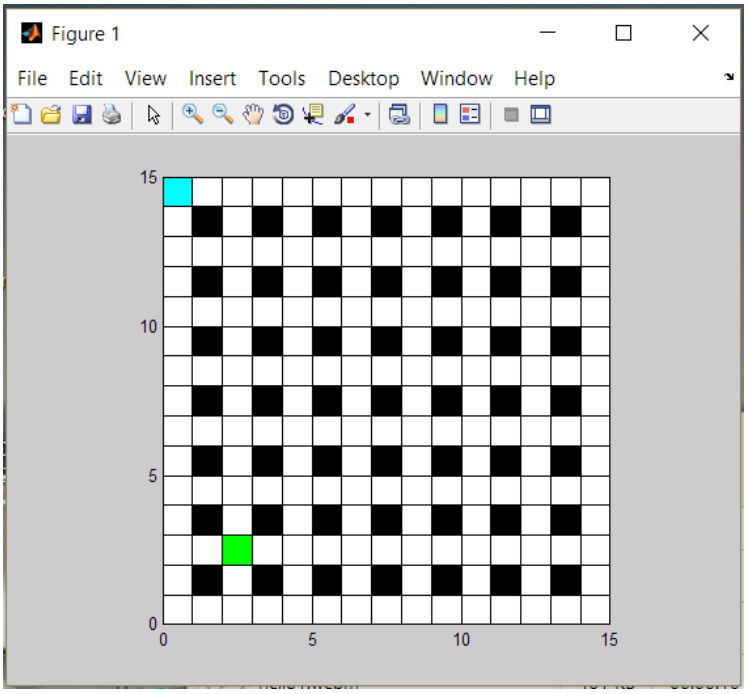
- Implementation of a version of pacman based bomberman from the bottom up
- Implementation involved the opponent being modelled as a minimiaing agent anf the player being modelled with free will (use of keyboard keys)
- Heuristic of Euclidean distance used in the search strategy
- Due to issues merging this version of minimax the idea for the minimax algoriuthm was inspired from this version in an attempt to make it mnore effeicient

**Member 4 [Ahmed Ali]:**

-My part towards the end of the project revolved around aestheticism and presentation, primarily being a state space representation.
-I worked on the graphical representation of A* search strategy. A goal state is given in the code, to be the top left corner of the graph. For the end state, you need to choose one yourself. A* search implementation is represented in the form of blue blocks covering the screen, and the red path being the best path, or path taken by agent.

-An initial approach to representation was to make a complete search tree for the traversal of agent. This proved to be slower, with the tree having 4^**N** nodes. **N** being the height, which went up to 18.

**Blue: Start**
**Green: Goal**



# REFERENCES


1. *McFerran, Damien (2008). "Hudson Profile - Part 1 (RG)" (PDF). Issue 66. Retro Gamer Magazine. pp. 68–73. Retrieved 2011-01-19.*
2. *McFerran, Damien (2009). "Hudson Profile - Part 2 (RG)" (PDF). Issue 67. Retro Gamer Magazine. pp. 44–49. Retrieved 2011-01-19.*
3. Cui, X. and Shi, H. (2011) A*-Based Pathfinding in Modern Computer Games. International Journal of Computer Science and Network Security, 11, 125-130.
4. *Image* http://bomberman.wikia.com/wiki/Bomberman_(series)
5. Delling, D.; Sanders, P.; Schultes, D.; Wagner, D. (2009). "Engineering route planning algorithms". *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*. Springer. pp. 117–139. doi:10.1007/978-3-642-02094-0_7.
6. Zeng, W.; Church, R. L. (2009). "Finding shortest paths on real road networks: the case for A*". *International Journal of Geographical Information Science*. **23** (4): 531–543.doi:10.1080/13658810801949850.
7. GUI Programming in Python using TkInter *https://wiki.python.org/moin/GuiProgramming*