MATH 496 Final Report

Author:	Mohammed Aaqel Shaik Abdul Mazeed	(mshai4@uic.edu)
Major:	Computer Science	
Term:	Fall 2022	
CRN:	48252	
Credit Hours:	3	
Instructors:	Andrew Shulman	(ashulm2@uic.edu)
	Evangelos Kobotis	(evangelo@uic.edu)
Aim:	Explore machine learning algorithms and apply them to minesweeper.	

Introduction

Minesweeper is a logic puzzle video game generally played on personal computers. It first arrived on the scene in 1992 when it was bundled with Windows 3.1 as *Microsoft Minesweeper*. Since then, it has spawned many different variations, but the classic game has remained iconic amongst Windows games, only being removed as a pre-installed application with the release of Windows 8, and later being published as a free game on the Microsoft Store.

Minesweeper features a grid of tiles, usually of 16x16 size, with 40 mines to uncover. Each tile, when clicked, reveals the number of mines around it. If the player clicks on a mine, they lose the game. The aim is to flag all mines and uncover all tiles that don't contain mines.

Minesweeper is a game with inherent unpredictability. The random position of the mines in each newly-generated game makes it hard to learn effective strategies to win. For this reason, although possible, a traditional approach in reinforcement learning may produce less than desirable results for a beginner. A more fruitful approach would be to hard-code the actions to be taken by the agent, based on what it can learn from the board through previous actions.

Game parameterization

Building on the base provided by *Harvard CS50's Introduction to Artificial Intelligence with Python* , we can parameterize the game and represent our AI's knowledge through 'sentences'.

$$\{(0, 1), (1, 2), (3, 2)\} = 2$$

A sample sentence in our AI's knowledge base.
This lets us know that among (0, 1), (1, 2), and (3, 2), there exist 2 mines.

Every time a move is made, the board adds a new sentence to the knowledge base and existing sentences in the knowledge base are updated accordingly. If the agent is able to discern a tile as being completely safe, it adds the tile to a list of safe moves to make and removes it from every sentence in the knowledge base. If the agent concludes that a tile has a mine underneath, it flags it and removes it from every sentence in the knowledge base.

{(1, 2), (1, 3), (1, 4), (2, 2), (2, 4), (3, 2), (3, 3), (3, 4)} = 3

If a tile at (2, 3) is uncovered and it displayed a '3', the sentence above would be added to the knowledge base and existing sentences would be updated.

Move prioritization

There are three different types of moves that the agent can make at any given moment in the game:

1) Safe move: The agent clicks on a tile it has marked as completely safe.
2) Calculated move: If there are no safe moves to make, find the probability of clicking on a mine for every available move based on the knowledge base, and pick the tile with the lowest probability.
3) Random move: If there are no calculated moves to make, make a random move.

Win/Lose conditions

The agent wins the game when there are no possible moves to make and all mines have been flagged. As expected, the agent loses the game when it clicks on a mine.

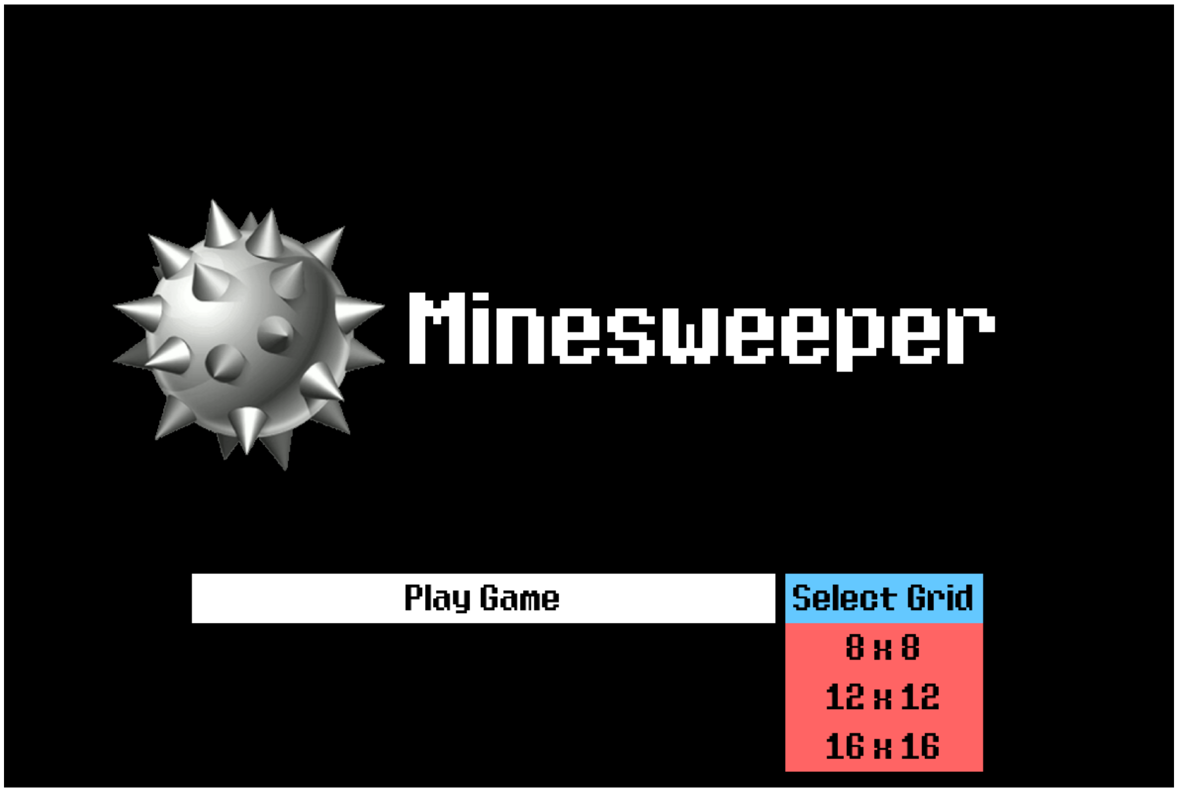
As of now, the agent wins the game roughly 45 - 55 % of the time. The agent was told to play 100 games with three different grid sizes: 8x8, 12x12, and 16x16. This experiment was repeated for each grid size thrice, to get an average win rate of 53% for 8x8, 48% for 12x12, and 46% for 16x16. We can attribute most, if not all these losses to one factor: the random move.

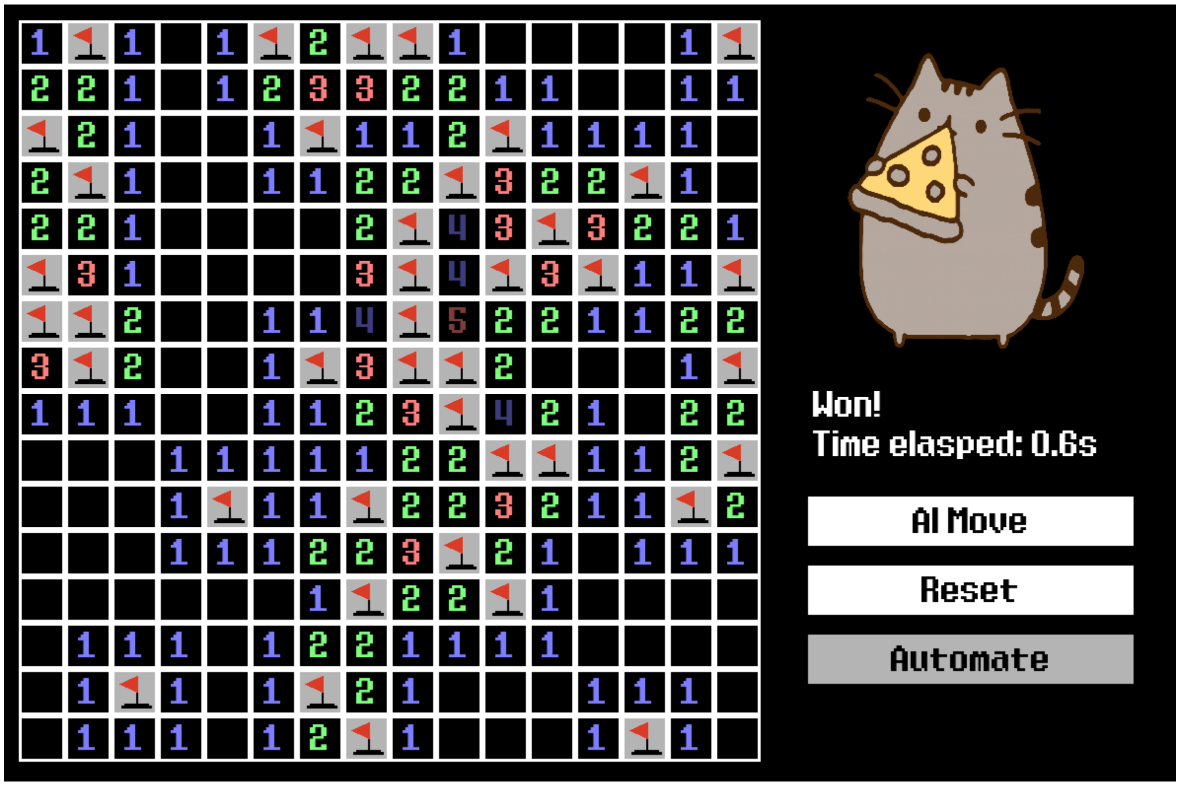
The random move is taken when the agent runs out of safe moves and the calculated moves all have the same probability. Unfortunately, this is a consequence of the nature of the game itself. Even with the inherent randomness, the agent performs very well considering the information at its disposal.

Bells and whistles

Besides the AI, the application also allows the user to select preferred grid size from 8x8, 12x12 and 16x16. Inside the main game, the user can click on 'AI Move' to have the AI agent perform one move, 'Reset' to reset the game, and 'Automate' to let the AI agent play the entire game.

The application also creates a 'log.txt' when executed, and this allows the user to go back and check the moves made by the agent during the games played.





Looking forward

Researching for this project introduced me to different approaches to creating an AI for minesweeper. One fairly common approach was to use Deep Q-Learning, and it might be interesting to explore this avenue in the future.

It might be useful to research common minesweeper strategies and implement them into the agent for better early-game moves. I would also like to add more inference rules to the knowledge base so conclusions can be made quicker and more precisely.