Developing Minesweeper & Intelligent Agent

Sankalp Varshney / ML / 2015272 / 6th Semester

Graphic Era (Deemed to be) University

Note

This report is in accordance with the Python program that I developed. This paper discusses the game, rules, development with Pygames, intuition to construct AI agents, and areas for further research. Please familiarize yourself with the source program, as this report is simply meant to supplement it.

Introduction

Minesweeper is a game wherein mines are concealed in a square grid. Safe squares contain numbers that indicate how many mines touch the tile. You may solve the game by opening all the safe squares using the number clues. You lose the game if you click on a mine.

Windows Minesweeper always protects the first click. You use the left mouse button to open tiles and the right mouse button to pin flags on mines. When you press the right mouse button again, your flag becomes a question mark. When you open a square that does not contain any mines, it will be empty, and the adjacent squares will open in all directions until you reach squares with numbers. A popular method for starting games is to click at random until you obtain a large opening with a lot of numbers.

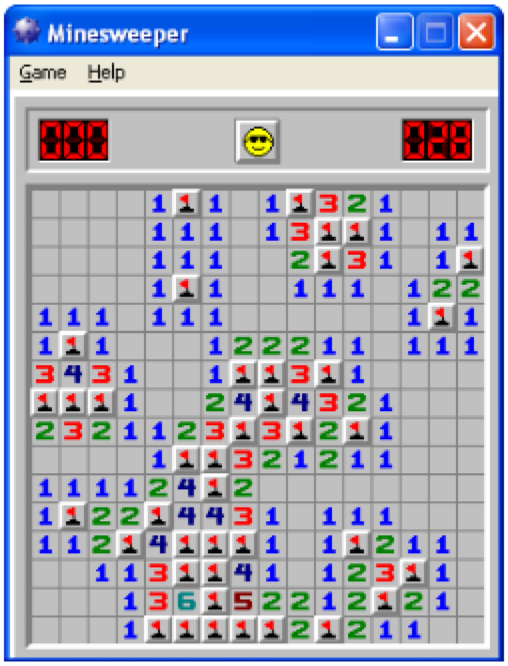
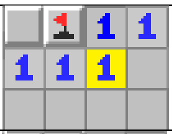
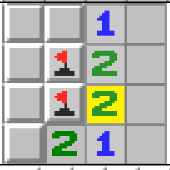
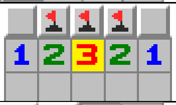
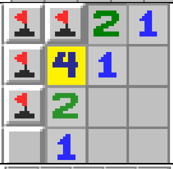
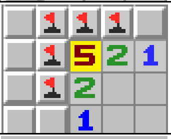
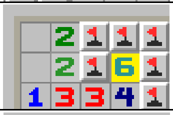
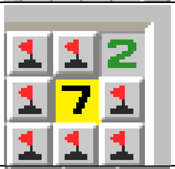
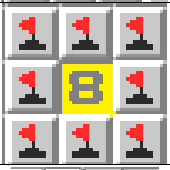
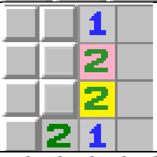
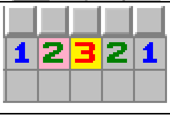


Figure 1 - Minesweeper on Windows XP with 40 mines solved in 28 seconds

Minesweeper fascinates me tremendously. It is a sophisticated game with simple rules that rewards the player for detecting patterns that help solve the grid. Personally, I've been playing minesweeper for around 7-8 years and have developed pattern matching abilities that serve me when I play the game. I generally play on a 32 x 18 grid with 140 mines, and my fastest completion time is 3 Minutes 7 Seconds, which is considered a great score.

There are a few simple patterns that can help you sweep the minefield far more efficiently than trial and error. When a number touches the same number of squares, those squares must be mines. Consider these examples:

Case	Intuition
	The 1 on the corner contacts 1 square thus it must be a mine.
	The 2 touches 2 squares so they must both be mines.
	The 3 touches 3 squares so they must all be mines.
	The 4 touches 4 squares so they must all be mines.
	The 5 touches 5 squares so they must all be mines.
	The 6 touches 6 squares so they must all be mines.
	The 7 touches 7 squares so they must all be mines.
	The 8 touches 8 squares so they must all be mines.
	The pink 2 touches three squares and cannot be solved. The yellow 2 touches two squares and can be solved
	The pink 2 touches three squares and cannot be solved. The yellow 3 touches three squares and can be solved.

Methodology of Development

I intended to create an application in the style of the classic Windows minesweeper. The functional requirements are as follows:

- The ability to see and interact with the minesweeper grid is the most fundamental requirement. Initially, all the tiles will be the same color, distinguishing them from the tiles that will be unveiled as the game progresses.
- Safe Initial Click: The first minefield click must not be a mine and must always open an empty tile.
- Left and right clicks: The left and right clicks should be the same as in the legacy game, with the left click revealing a tile and the right click flagging a mine.
- Number of movements: The game's UI must track and display the number of moves made by the user/AI agent.
- SPACEBAR: Hitting the spacebar clears the minefield and restarts the game.
- Win/Loss detection: As the game is played, whenever a mine is clicked or when all mines are flagged, the game ends with the corresponding result displayed on the screen. - Session Statistics: Opening the game initiates a new session that collects the results of the games played in the session and statistics of the entire session are displayed on the screen.
- ENTER Key: Pressing the enter key on the keyboard instructs the pre-programmed AI agent to play the following move.
- L Key: Pressing the L key on the keyboard causes the Agent AI to loop until the key is pressed again. Until manually stopped, the Agent will solve the minefield repeatedly (even after the game has ended, it starts a new game).

I chose to use Python's pygame package to develop an interactive visual program game. Because of the modules in its library, Pygame is a fantastic tool for creating interactive games. The screen in my game, like all other utilities, is a built-in function of pygame. The window is drawn on the screen using the $\boxed{pygame.draw.rect}$ function, which accepts the surface, color, and pygame Rect object as input parameters and creates a rectangle on the screen. Pygame also allows us to use our own fonts, which may be put into the screen using the $font.render$ method.

Pygame additionally accepts mouse input from the functional approach. It maps the exact X&Y coordinates of the game window's pointer, allowing the user to exactly select the tile on the field the It maps the exact X&Y coordinates of the game window's pointer, allowing the user to exactly select the tile on the field the It maps the exact X&Y coordinates of the game window's pointer, allowing the user to exactly select the tile on the field the player desires to.

$\boxed{pygame.mouse.get\ pos()[0]}$ -> x coordinate

$\boxed{pygame.mouse.get\ post()[1]}$ -> y coordinate

The mouseToField() user-defined function takes the cursor's x and y coordinates and compares them to the tile the pointer is pointing at.

```
fieldCoord=mouseToField(pygame.mouse.get_pos()[0],pygame.mouse.get_pos()[1])
```

The keyboard and mouse inputs are registered as events in pygame. The fundamental function of the game is to do all possible actions when an event (keyboard button pushed/mouse pressed, for example) happens. The $pygame.event.get()$ function recognises the pressed keys on the keyboard. Here's an example from the code:

```
for event in pygame.event.get():
            #Space bar to reset the game
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    ms.resetField()
                    ms_ai.knowledge = []
                    waitForNG = False
```

The preceding code excerpt illustrates the action taken after pressing the spacebar key, which resets the field, clears the AI knowledge tuple, and initiates an NG (New Game).

Insight to develop an Artificial Intelligence Agent

An Intelligent Agent (IA) is a logical agent that makes decisions, like a user in a game. Based on numerous percepts, the IA chooses and executes the best action (with previous and present contextual information). A basic reflex agent was the IA that I selected to use in my project. A simple reflex agent is a rational agent that acts on the basis on the present percept rather than the history of percepts. The agent function is built on the condition-action rule. Because the environment is fully observable to the agent in the context of the game, a simple reflex agent is optimal.

The IA accepts input parameters that are based on probability. The likelihood that a tile is a mine, ranging from 0 (Not a Mine) to 1 (A Mine). The IA function first produces probability of a random tile being a mine by dividing the number of covered mines by the total number of covered tiles. This yields a default Probability, which may then be skewed by considering previously discovered mines.

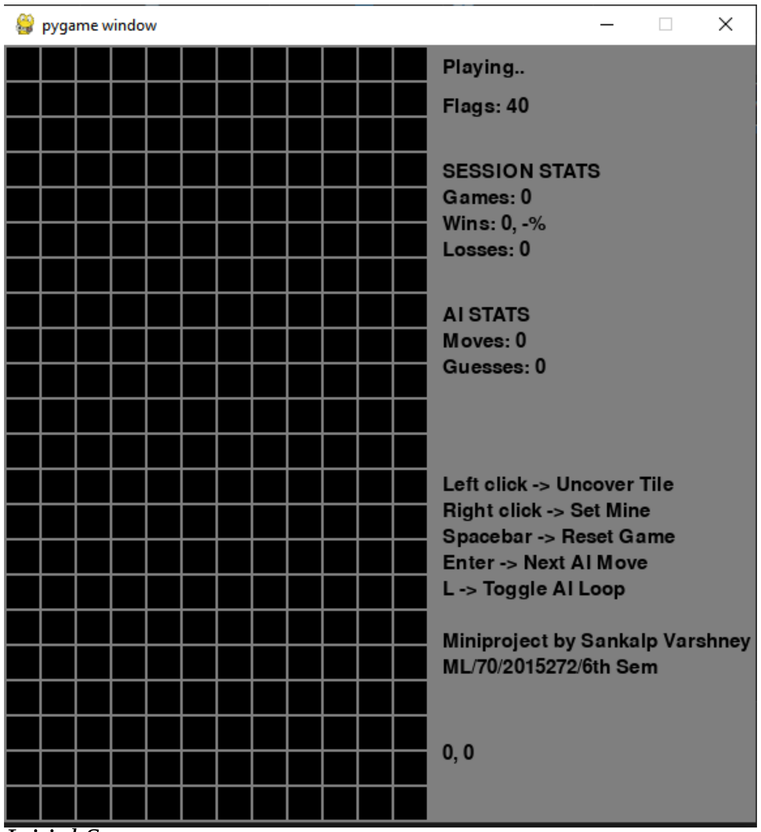
We may utilize the information in the table including mine patterns and their intuition, as previously discussed, to combine our odds of tiles being mines. If a tile has a probability of one, it is classified as a mine. In addition, the IA discovers the tile with the lowest probability in the Knowledge Tuple. The knowledge tuple consists of all adjacent tiles to an uncovered tile, with the probability of the tile being a mine.

Finally, the AI Loop was the last functional need to be implemented. The AI loop is essentially a series of agent movements that are executed repeatedly. For a better understanding, the Intelligent Agent picks the ideal move and either sets a flag or uncovers a tile, and the optimal move is done by the IA in the AI loop until there are no more tiles/cells to execute the optimal move on.

```
pygame.init()
    ms = Minesweeper(width, height, mineCount)
    font = pygame.font.SysFont("comicsansms", 12)
    font_text = pygame.font.SysFont("arialms", 22)
    screen = pygame.display.set_mode((ms.width*(cellPixelWH+borderWH)+borderWH+250,
ms.height*(cellPixelWH+borderWH)+borderWH))

    running = True
    ai_loop = False
    waitForNG = False
    ms_ai = Minesweeper_AI()

    while running:
        fieldCoord = mouseToField(pygame.mouse.get_pos()[0],
pygame.mouse.get_pos()[1])
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
            #Space bar to reset the game
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    ms.resetField()
                    ms_ai.knowledge = []
                    waitForNG = False
            #Enter for IA move
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_RETURN:
                    mv = ms_ai.move(ms)
                    if mv is not None:
                        ms.click(mv[0],mv[1], ms_ai)
                        print("=>"+str(mv))
                if event.key == pygame.K_l:
                    start_t = time.perf_counter()
                    ai_loop = not ai_loop
            if event.type == pygame.MOUSEBUTTONDOWN:
                #Left click to uncover a mine field
                if pygame.mouse.get_pressed()[0]:
                    if (fieldCoord[0] < ms.width) and (fieldCoord[1] < ms.height):
                        ms.click(fieldCoord[0], fieldCoord[1], ms_ai)

                #Right click to flag a spot
                if pygame.mouse.get_pressed()[2]:
                    if (fieldCoord[0] < ms.width) and (fieldCoord[1] < ms.height):
                        ms.setFlag(fieldCoord[0], fieldCoord[1])
        if (ai_loop):
            mv = ms_ai.move(ms)
            if mv is not None:
                ms.click(mv[0],mv[1], ms_ai)
                print("=>"+str(mv))
        screen.fill(grey)
        drawMS(screen, font, ms)
        if (ms.gameOver) or (ms.gameWon):
            if (not waitForNG):
                if (ms.gameOver):
                    lost += 1
                if (ms.gameWon):
                    won += 1
                waitForNG = True
            if (ai_loop):
                end_t = time.perf_counter()
                time_taken = end_t - start_t
                total_t += time_taken
                ms.resetField()
                ms_ai.knowledge = []
                waitForNG = False
                start_t = time.perf_counter()

        printText(ms, ms_ai, font_text, screen)
        pygame.display.update()
```

The driver code for my final version of the minesweeper game is provided below. The first nine lines are used to initialize global variables such as pygame, show the window on the screen, and initialize the Intelligent Agent. Checks for any valid input events, such as button clicks and pushes, in the running function(), which has main control while the game is running. Furthermore, if the AI loop variable is True, the Agent will perform optimum movements continually until the L key is pushed again or the game is terminated.

<u>Output</u>



Playing..

Flags: 40

SESSION STATS
Games: 0
Wins: 0, -%
Losses: 0

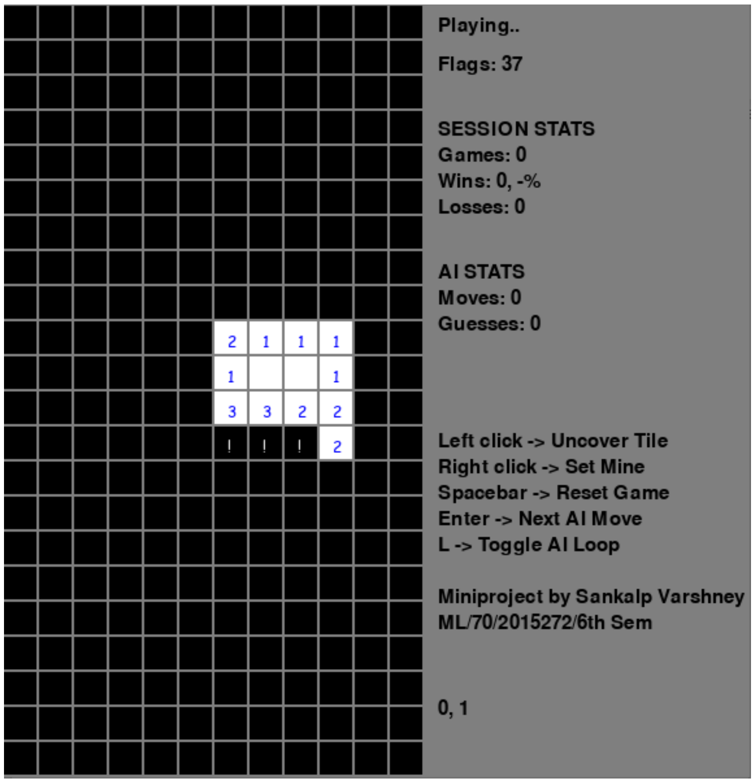
AI STATS
Moves: 0
Guesses: 0

Left click -> Uncover Tile
Right click -> Set Mine
Spacebar -> Reset Game
Enter -> Next AI Move
L -> Toggle AI Loop

Miniproject by Sankalp Varshney
ML/70/2015272/6th Sem

0, 0

Initial Screen



Playing..

Flags: 37

SESSION STATS
Games: 0
Wins: 0, -%
Losses: 0

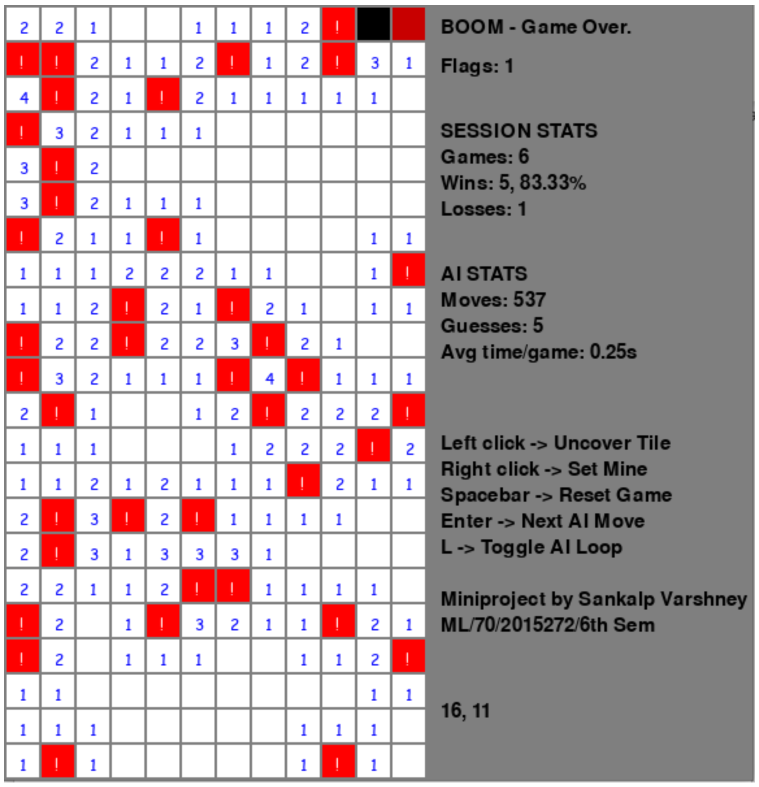
AI STATS
Moves: 0
Guesses: 0

Left click -> Uncover Tile
Right click -> Set Mine
Spacebar -> Reset Game
Enter -> Next AI Move
L -> Toggle AI Loop

Miniproject by Sankalp Varshney
ML/70/2015272/6th Sem

0, 1

Always safe First Click



BOOM - Game Over.

Flags: 1

SESSION STATS
Games: 6
Wins: 5, 83.33%
Losses: 1

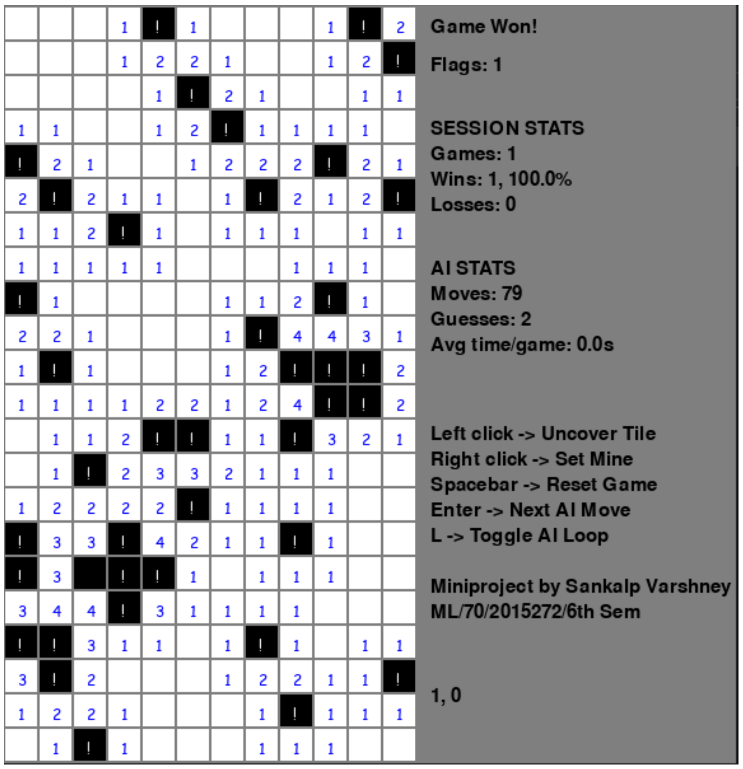
AI STATS
Moves: 537
Guesses: 5
Avg time/game: 0.25s

Left click -> Uncover Tile
Right click -> Set Mine
Spacebar -> Reset Game
Enter -> Next AI Move
L -> Toggle AI Loop

Miniproject by Sankalp Varshney
ML/70/2015272/6th Sem

16, 11

Game over when a Mine is uncovered



Game Won!

Flags: 1

SESSION STATS
Games: 1
Wins: 1, 100.0%
Losses: 0

AI STATS
Moves: 79
Guesses: 2
Avg time/game: 0.0s

Left click -> Uncover Tile
Right click -> Set Mine
Spacebar -> Reset Game
Enter -> Next AI Move
L -> Toggle AI Loop

Miniproject by Sankalp Varshney
ML/70/2015272/6th Sem

1, 0

Game won when all mines are flagged

Citations

How to Play Minesweeper, https://minesweepergame.com/strategy/how-to-play-minesweeper.php.

Jones, Matthew. “Solving Minesweeper with C# and Linq.” *Exception Not Found*, Exception Not Found, 4 Jan. 2021, https://exceptionnotfound.net/solving-minesweeper-with-c-sharp-and-linq/.

Minesweeper Report - University of Utah. https://my.ece.utah.edu/~kstevens/6712/minesweeper-report.pdf.