T1A3 Terminal Application

Matthew Ng

Overview of Application

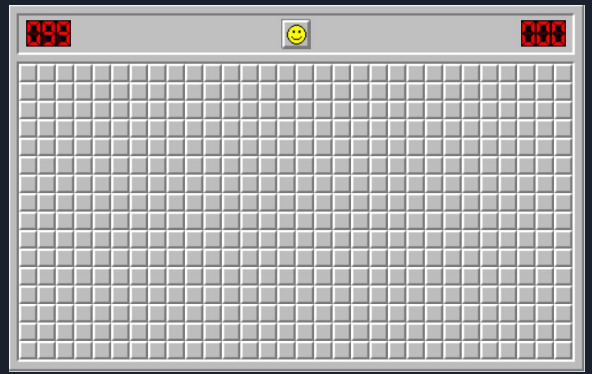
What is it?

- Fully functional version of Minesweeper game playable in terminal

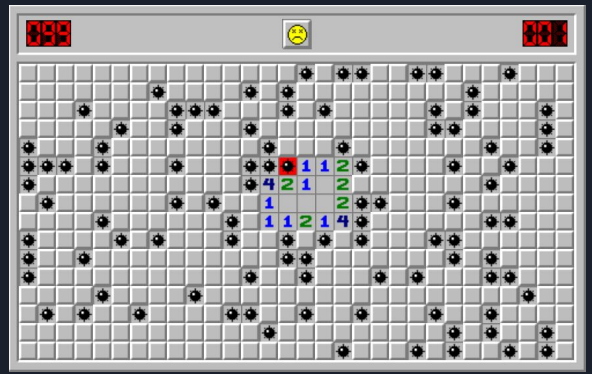
What is Minesweeper?

- Popular puzzle and problem solving video game
- Mines are randomly placed around a grid under tiles
- Tiles can be clicked on to reveal the space
- Player needs to reveal all empty spaces without clicking on any mines to win
- If a mine is revealed, the game ends

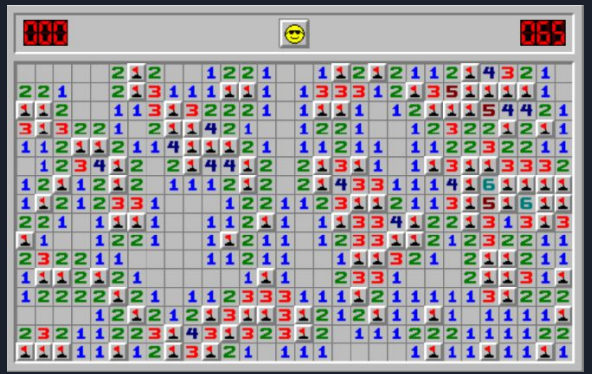
Starting screen



Game over screen



Winning screen



Key Mechanics & Features

Features

1. Reveal spaces
2. Showing adjacent mines
3. Recursive digging
4. Placing flags
5. Timer function
6. Difficulty Setting

Demo of Application

Logic of Application

- When game begins, two boards are created:
  - Hidden board where the mines are placed and each empty space is assigned a value for each adjacent mine
  - Player board which is blank - this is the board that is shown to the player and updates based on user inputs
- Both boards are list of lists that are created using the following:

```
board = [[' ' for i in range(dimensions)] for j in range(dimensions)]
player_board = [[' ' for i in range(dimensions)] for j in range(dimensions)]
```

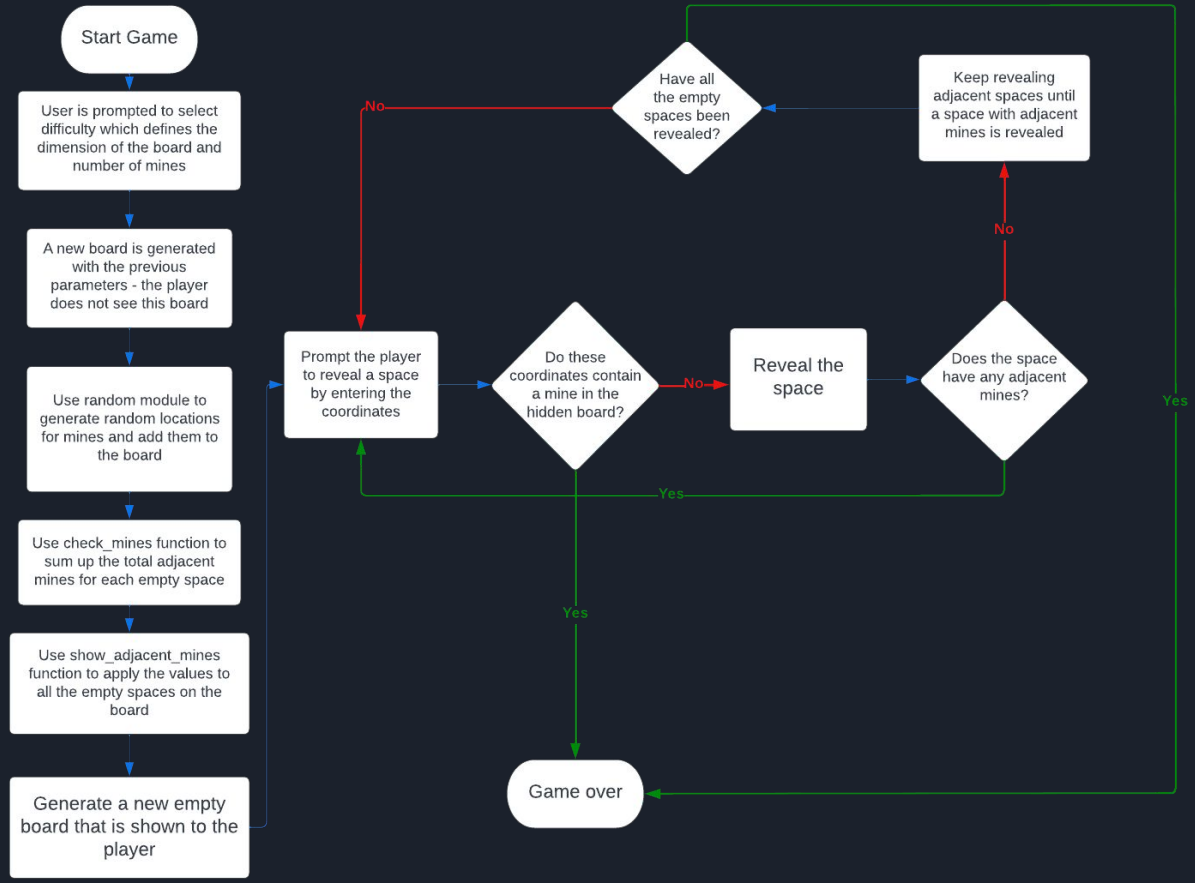
```
Hidden "True" Board:

[1, 1, 0, 0, 0]
[@, 2, 1, 0, 0]
[3, @, 3, 1, 1]
[3, @, 4, @, 1]
[2, 2, @, 2, 1]
```

```
Player Board:

[#, #, #, #, #]
[#, #, #, #, #]
[#, #, #, #, #]
[#, #, #, #, #]
[#, #, #, #, #]
```

Code Snippet

```
def createboard(dimensions, max_mines):
    """Function to create a new board and add mines"""
    board = [[' ' for i in range(dimensions)] for j in range(dimensions)]

    # Randomly generate mine locations and add them to the board
    mines = 0
    while mines < max_mines:
        # Generate random row and column number between 0 and the dimensions of the grid
        random_row = random.randint(0, dimensions -1)
        random_col = random.randint(0, dimensions -1)

        # Don't place a mine if the space already has a mine
        if board[random_row][random_col] == '@':
            continue
        # Otherwise, plant mine
        else:
            board[random_row][random_col] = '@'
        # Increase mine index by one
        mines += 1

    # Show the number of adjacent mines around each empty space
    show_adjacent_mines(dimensions, board)
    return board
```

Flow Chart

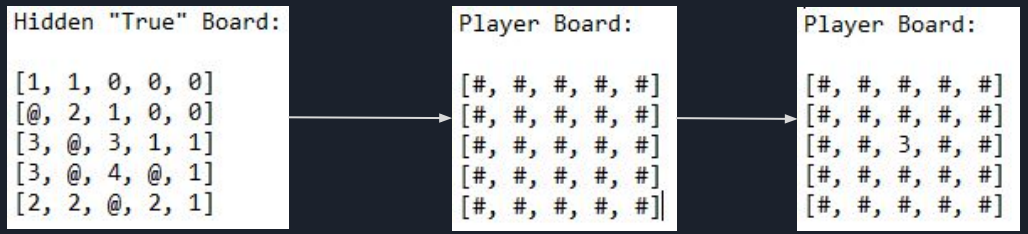

Start Game
User is prompted to select difficulty which defines the dimension of the board and number of mines
A new board is generated with the previous parameters - the player does not see this board
Use random module to generate random locations for mines and add them to the board
Use check_mines function to sum up the total adjacent mines for each empty space
Use show_adjacent_mines function to apply the values to all the empty spaces on the board
Generate a new empty board that is shown to the player
Prompt the player to reveal a space by entering the coordinates
Do these coordinates contain a mine in the hidden board?
No
Yes
Reveal the space
Does the space have any adjacent mines?
No
Yes
Keep revealing adjacent spaces until a space with adjacent mines is revealed
Have all the empty spaces been revealed?
No
Yes
Game over

Feature 1: Revealing Spaces

- User enters coordinates to reveal a space
- If the space is a mine, the game is over
- If the space is not a mine, show the number of adjacent mines around that space

How is this implemented?

- 'While' loop that keeps asking for user to input coordinates until all the empty spaces are revealed, or a mine is revealed
- If the coordinate input is not a mine on the hidden board, assign that value to the player board and return True



Code Snippet (play function)

```
    emptyspace = True
    input_history = set()
    # Create a board that the player sees and updates when spaces are revealed
    player_board = [[' ' for i in range(dimensions)] for j in range(dimensions)]
    while len(input_history) < dimensions ** 2 - max_mines:
        os.system("clear")
        display_board(player_board)
        user_input = re.split(r"[-;,.\s]\s*", input("Please enter a coordinate (row,column).  \nTo place a
                                                    flag, type F after the coordinate (row,column,F)  \n"))
        try:
            row, col, flag = int(user_input[0])-1, int(user_input[1])-1, user_input[-1]
        except ValueError:
            print("That is not a valid coordinate - please try again! ")
            press_to_continue()
            continue
        if len(user_input) == 2:
            if row < 0 or row >= dimensions or col < 0 or col >= dimensions:
                print("That is not a valid coordinate - please try again! ")
                press_to_continue()
                continue
        if len(user_input) == 3:
            if flag == "f":
                place_flag(player_board,row, col)
                continue
            else:
                print("That is not a valid coordinate! To place a flag, type F after the coordinate
(row,column,F).")
                press_to_continue()
                continue
```

Code Snippet (play function)

```
        emptyspace = show_space(player_board, dimensions, newboard, row, col, input_history)
        if not emptyspace:
            break
    # Winning message when all empty spaces are revealed
    if emptyspace:
        os.system("clear")
        print("Congrats, you won!")
        display_board(newboard)
        print(f"Your time was {int(time.time()-start_time)} seconds!")
        if replay() is False:
            return
        else:
            play()

    # Losing message when a mine is revealed
    else:
        os.system("clear")
        print("You stepped on a mine! Game over!")
        display_board(newboard)
        if replay() is False:
            return
        else:
            play()
```

Code Snippet

```
def show_space(player_board, dimensions, newboard, row, col, input_history):
    """Function to reveal the space entered by the player """
    # Add the coordinate entered to the input history to track what spaces have been clicked on
    input_history.add((row,col))
    # If the space has already been revealed, show error message and return True
    if player_board[row][col] == newboard[row][col]:
        print("You have already revealed this spot! Please enter a different coordinate. ")
        press_to_continue()
        return True
    # If the space has a mine, return False
    if newboard[row][col] == '@':
        return False
    # If the space is empty but has adjacent mines, show the number of adjacent mines
    if newboard[row][col] > 0:
        player_board[row][col] = newboard[row][col]
        return True
    # If the space is empty and has no adjacent mines, keep revealing adjacent spaces until
    # an empty space with adjacent mines is revealed
    if newboard[row][col] == 0:
        player_board[row][col] = newboard[row][col]
        for r in range(max(0,row-1), min(dimensions-1,row+1)+1):
            for c in range(max(0,col-1),min(dimensions-1,col+1)+1):
                if (r,c) in input_history:
                    continue
                if player_board[r][c] == 'F':
                    continue
                show_space(player_board, dimensions, newboard, r, c, input_history)
    return True
```

Feature 2: Showing Adjacent Spaces

- For each empty space, assign a value between 0 - 8 to represent the number of mines in the neighbouring spaces
- To illustrate, for a space with coordinates (0, 0):

```
(-1,-1 ) (-1,0 ) (-1,1 )
( 0,-1 ) ( 0,0 ) ( 0,1 )
( 1,-1 ) ( 1,0 ) ( 1,1 )
```

i.e. 8 directions

How is this implemented?

- ‘For’ loop that, for each coordinate, checks the row before and after, as well as the column before and after, to check all 8 locations
- Set adjacent_mines = 0 and index by 1 for every mine that is checked
- Error handling: need to ensure that the checks stay within the bounds of the board, i.e. for spaces at the edges of the board

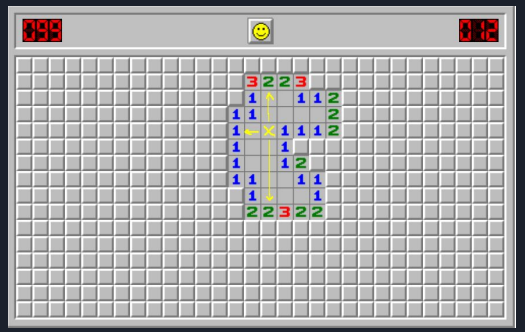
```
Hidden "True" Board:

[1, 1, 0, 0, 0]
[@, 2, 1, 0, 0]
[3, @, 3, 1, 1]
[3, @, 4, @, 1]
[2, 2, @, 2, 1]
```

Code Snippet

```
# Check the number of mines around each space
def check_mines(dimensions, board, row, col):
    """Function to check the number of adjacent mines around each space """

    adjacent_mines = 0

    # Ranges must be in range of the board i.e. accounting for the edges of the board
    for x in range(max(0,row-1), min(dimensions-1, row +1)+1):
        for y in range (max(0,col-1), min(dimensions-1,col+1)+1):
            # Don't check the space itself
            if x == row and y == col:
                continue
            # Increase adjacent_mine index by 1 for every mine around the space
            if board[x][y] == '@':
                adjacent_mines += 1
    # Return the number of adjacent mines around the space
    return adjacent_mines

# After the number of adjacent mines is counted, replace the space on the board with this number
def show_adjacent_mines(dimensions, board):
    """Function that assigns the adjacent mines value to the empty spaces """
    for x in range(dimensions):
        for y in range(dimensions):
            # Don't replace the spaces with a mine
            if board[x][y] == '@':
                continue
            # Replace the space with the returned number from check_mines()
            board[x][y] = check_mines(dimensions, board, x, y)
```

Feature 3: Recursive Digging

- When an empty space with no adjacent mines is revealed, keep digging in the neighbouring tiles until a space with adjacent mines is reached



How is this implemented?

- If a space is revealed and its assigned value i.e. adjacent mines is 0, use a ‘for’ loop to keep repeating the reveal space function in the surrounding tiles until a tile with value > 0 is reached

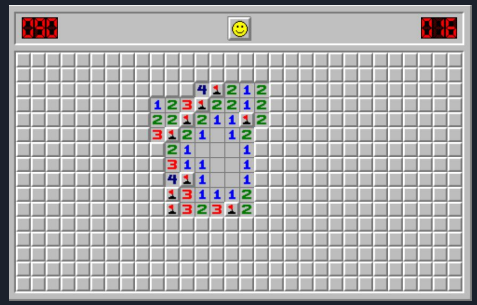
Code Snippet

```
def show_space(player_board, dimensions, newboard, row, col, input_history):
    # Add the coordinate entered to the input history to track what spaces have been clicked on
    input_history.add((row,col))
    # If the space has already been revealed, show error message and return True
    if player_board[row][col] == newboard[row][col]:
        print("You have already revealed this spot! Please enter a different coordinate. ")
        press_to_continue()
        return True
    # If the space has a mine, return False
    if newboard[row][col] == '@':
        return False
    # If the space is empty but has adjacent mines, show the number of adjacent mines
    if newboard[row][col] > 0:
        player_board[row][col] = newboard[row][col]
        return True
```

```
    # If the space is empty and has no adjacent mines, keep revealing adjacent spaces until
    # an empty space with adjacent mines is revealed
    if newboard[row][col] == 0:
        player_board[row][col] = newboard[row][col]
        for r in range(max(0,row-1), min(dimensions-1,row+1)+1):
            for c in range(max(0,col-1),min(dimensions-1,col+1)+1):
                if (r,c) in input_history:
                    continue
                if player_board[r][c] == 'F':
                    continue
                show_space(player_board, dimensions, newboard, r, c, input_history)
    return True
```

Feature 4: Placing Flags

- Player is able to place flags on unrevealed spaces where they think a mine is located
- Quality of life feature that helps the player clear the game
- Flagged spaces cannot be revealed until the flag is removed
    - Particularly useful in terminal



How is this implemented?

- To flag a space, the player includes an ‘F’ after the coordinate, i.e. (3, 4, f)
- Use ‘if’ statements for appropriate control flow in the following cases:
    - If the space on the player board is empty, place a flag
    - If the space on the player board is revealed, show error message
    - If the space on the player board is flagged, remove the flag

Code Snippet

```
# Place a flag on a space that may be a mine
def place_flag(player_board, row, col):
    """Function to place a flag"""
    # If the space has already been revealed or already has a flag, prevent a flag
from being placed
    if player_board[row][col] != ' ' and player_board[row][col] != 'F':
        print("You can't put a flag here!")
    # If the space already has a flag, remove the flag
    elif player_board[row][col] == 'F':
        player_board[row][col] = ' '
    # If the space does not have a flag, place a flag
    else:
        player_board[row][col] = 'F'
```

Feature 5: Timer Function

- In-built game timer that tracks the time it takes for the player to finish the game and shows it to the player after the game is completed successfully

How is this implemented?

- Imported the time module and use the time.time( ) method
- time.time(): returns time in seconds since the epoch (January 1, 1970, 00:00:00 (UTC))
- To calculate the time it took for the player to complete the game:
    - Assign start_time = time.time( ) when the game begins i.e. after the board is generated
        - This must be outside the 'while' loop or else the start_time will be repeatedly reassigned on every loop while the game is played
    - When the game is completed, time.time( ) - start_time will return the time in seconds that has elapsed since the game started

Code Snippet

```
    # After the game board is generated, assign start_time
    start_time = time.time()
    # Winning message when all empty spaces are revealed
    if emptyspace:
        os.system("clear")
        print("Congrats, you won!")
        display_board(newboard)
        print(f"Your time was {int(time.time()-start_time)} seconds!")
        if replay() is False:
            return
        else:
            play()
```

Feature 6: Difficulty Options

- Allow the player to select a difficulty option, either ‘easy’ or ‘normal’
- Easy: 5x5 board, 4 mines
- Normal: 10x10 board, 10 mines
- Selecting the difficulty will define the respective parameters and generate the board

How is this implemented?

- An input function within a ‘while’ loop will continue to prompt the player to enter a difficulty until a valid input is entered
- ‘If’ statements used to define the parameters based on what the user has input

Code Snippet

```
    while True:
        difficulty = input("Please enter the difficulty you want to play on. \n"
        'Easy mode: 5x5, 4 mines \n'
        'Normal mode: 10 x 10, 10 mines \n').lower()
        if difficulty == "easy":
            dimensions = 5
            max_mines = 4
            break
        elif difficulty == "normal":
            dimensions = 10
            max_mines = 10
            break
        else:
            os.system('clear')
            print("Invalid difficulty - please enter 'easy' or 'normal'")
            press_to_continue()
            continue
```

Review of Development & Build Process

Challenges

- Understanding how certain key mechanics work, and how they can be replicated in Python
- Learning how to format the player grid in a visually appropriate way
- Defining the show_space( ) function was complicated and took the longest to complete

Highlights & Key Takeaways

- Enjoyable problem solving exercise throughout the project
- Gained deeper understanding over Python concepts
- Should make use of classes in the future