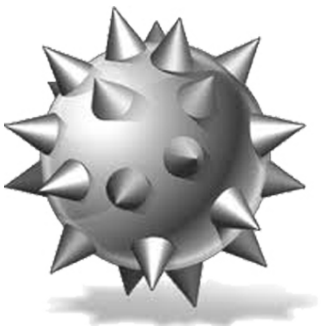
CS6.201: Introduction to Software Systems

Hackathon

Implementing Minesweeper Using PyGame and NumPy

Prakhar Jain

April 19, 2025



Objective

Your task is to implement the classic Minesweeper game using the `PyGame` library for the graphical interface and the `NumPy` library for handling the game grid and random module for mine placement.

Requirements

Libraries and Modules

The following libraries and functions are required for the implementation of the game:

- **PyGame** (Graphical Interface and Event Handling):
  - `pygame.init()` — Initializes all Pygame modules
  - `pygame.display.set_mode()` — Creates a display window
  - `pygame.time.Clock()` — Manages the game's frame rate
  - `pygame.display.set_caption()` — Sets the window title
  - `pygame.image.load()` — Loads image assets
  - `pygame.font.SysFont()` — Creates fonts for rendering text
  - `pygame.event.get()` — Retrieves user input events
  - `pygame.QUIT` — Detects when the game is closed
  - `pygame.KEYDOWN` — Detects key press events
  - `pygame.K_r` — Identifies when the 'R' key is pressed
  - `pygame.MOUSEBUTTONUP` — Detects mouse button release
  - `pygame.Rect()` — Defines rectangular game objects
  - `pygame.display.update()` — Refreshes the display
  - `pygame.quit()` — Exits the game and cleans up resources
- **NumPy** (Data Handling and Randomization):
  - `numpy.empty()` — Creates an empty array for the game grid
  - `numpy.random.randint()` — Generates random integers for mine placement

Game Specifications

- **Background color:** (192, 192, 192)
- **Grid color:** (128, 128, 128)
- **Game dimensions:** 10 × 10 (width × height)
- **Number of mines:** 9
- **Grid size:** 32 pixels
- **Borders:**
  - General border: 16 pixels
  - Top border: 100 pixels
- **Display dimensions:**
  - Width: 352 pixels computed as `Grid size` × `Game width` + 2 × `Border`)
  - Height: 420 pixels computed as `Grid size`×`Game height`+`Border`+`Top border`)
- **Gameplay Mechanics:**
  - Left-click to reveal a cell.
  - Right-click to flag a mine.
  - If a mine is clicked, the game ends, and the player is prompted to restart the game by pressing the **R** key.
  - If all non-mine cells are revealed, the player wins and is prompted to restart the game by pressing the **R** key.
- **User Interface:**
  - Display mine counter.
  - Display timer.
- **Win/Loss Conditions:**
  - The player wins when all non-mine cells are uncovered.
  - The player loses if a mine is clicked.

Implementation Details

Classes and Functions

GridCell Class

This class represents a single cell in the Minesweeper grid.

- `__init__(self, x, y, value)`: Initializes a grid cell with the given coordinates and value.
  - **Variables:**
    - `x`: X-coordinate of the grid cell.
    - `y`: Y-coordinate of the grid cell.
    - `value`: Value of the cell (-1 for mine, 0-8 for number of adjacent mines).
    - `clicked`: Boolean indicating if the cell has been clicked.
    - `mineClicked`: Boolean indicating if a mine was clicked.
    - `mineFalse`: Boolean indicating if a flagged cell was incorrectly flagged.
    - `flag`: Boolean indicating if the cell is flagged.
    - `rect`: Pygame Rect object for the cell's position and size.
- `draw(self)`: Draws the grid cell based on its state (e.g., whether it is clicked, flagged, or contains a mine).
- `reveal(self)`: Reveals the grid cell and, if necessary, its neighbors. If the cell is empty (value 0), it will reveal neighboring cells.
- `reveal_neighbors(self)`: Reveals neighboring cells if the current cell is empty (value 0).
- `reveal_all_mines(self)`: Reveals all mines when a mine is clicked, typically used when the game is over.
- `update_value(self)`: Updates the value of the cell based on the number of neighboring mines.

Minefield Class

This class represents the entire Minesweeper grid and manages the placement of mines.

- `__init__(self)`: Initialize the minefield grid and mine positions.
  - **Variables:**
    - `grid`: Numpy array representing the grid of cells.

    * `mines`: List of mine positions.

- `generate_mines(self)`: Randomly place mines in the grid.
- `generate_grid(self)`: Create the grid and update cell values based on mine positions.

Game Class

- `__init__(self)`: Initialize the game state, mine count, and timer.
  - **Variables:**
    * `state`: Current state of the game ("Playing", "Game Over", "Win").
    * `mine_left`: Number of mines left to be flagged.
    * `time`: Elapsed time in the game.
    * `minefield`: Instance of the Minefield class.
- `reset(self)`: Reset the game state and regenerate the minefield.
- `handle_events(self)`: Handle user inputs and game events.
- `check_win(self)`: Check if the player has won the game.
- `draw_game_state(self)`: Draw the current state of the game.
- `game_loop(self)`: Main game loop to run the game.

Helper Functions

- `draw_text(txt, size, y_offset=0)`: Draw text on the screen. This function is used to display messages when the game is won or over. It also prompts the player to restart the game by displaying a prompt to press the "R" key after the game ends, whether they win or lose.

Game Logic

- Initialize the Pygame display and load sprites.
- Create a grid of cells, each with a value indicating the number of neighboring mines or if it is a mine.
- Implement user interactions to reveal cells and place flags.
- Check for win conditions and handle game over scenarios.
- Display the game state, including the number of mines left and the elapsed time.

Deliverables

- A fully functional Minesweeper game implemented using Pygame and Numpy.
- The game should handle user input, display the game state, and provide feedback for win/loss conditions.
- It should reset correctly and allow for replay.
- The game should also handle errors gracefully.

Submission

- Submit Python script and any required resources
- Ensure the code is well-documented.
- You are not allowed to modify the class structure. You must complete the game using the provided class structure.

Grading Rubric

Category	Criteria	Points	Comments
Game Functionality	Correct Minesweeper grid generation (10x10, 9 mines, correct borders and dimensions).	10	Incorrect grid size, mines, or dimensions lead to deductions.
	Proper mine placement and adjacent mine counting.	10	Errors in mine placement or numbering lead to deductions.
	Correct handling of user interactions (left-click reveal, right-click flag).	10	Incorrect interactions reduce points.
	Correct cascade reveal functionality (reveals adjacent empty tiles correctly and stops at numbered cells).	10	Problems with cascade reveal reduce points.
Game Logic	Detects win condition (all non-mine cells revealed).	10	Incorrect win detection leads to deductions.
	Detects loss condition (mine clicked).	5	Game should end correctly on mine click.
	Timer and mine counter display update correctly.	5	Errors in timer/mine counter reduce points.
	Game resets correctly on pressing ‘R‘.	5	Game should restart properly.
Code Quality & Documentation	Well-commented code using docstrings, with additional single-line comments where logic is too complex.	5	Lack of comments reduces points.
User Interface & Aesthetics	Game visuals match specifications (background, grid color, borders).	5	Incorrect aesthetics result in deductions.
	Display of text elements (win/loss messages, restart prompt, mine counter, timer) is clear. Win/loss messages and the restart prompt may overlap the Minefield, but the clock and mine count should not.	5	Missing, unclear, or incorrectly positioned text elements result in deductions.
Total		**80**	