Project 1: System Architecture

EECS 581 Group 6

Introduction

This document serves as an introduction to the codebase for the Group 6 implementation of EECS 581 Project 1: Minesweeper.

This project is implemented in **Python** and is designed to be executed from a terminal. While the project can technically be run from the IDLE kernel, it is not designed for this platform, and console output may be slightly affected. Machines running this project must have the "**os**" and "**random**" libraries available.

To run the project code, simply type:

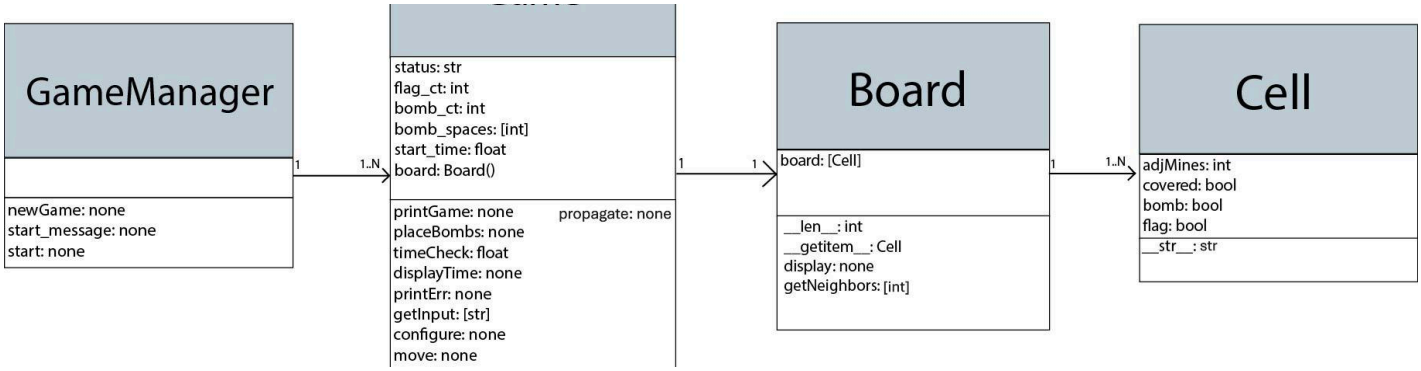
```
python3 minesweeper.py
```

The program will present an initial introductory message. Once a game has begun, game commands follow a standard format described by the regular expression "(m|f)[1-10][a-j]". Mining, or uncovering, a cell is represented by "m", while flag toggling is represented by "f". The second component of the command represents the row number, numbered 1 through 10. The third component represents the column label, ranging from a to j. Error messages are present to assist with any confusion regarding game controls and rules.

System Architecture

Class Diagram

A high-level design of all components is included below:



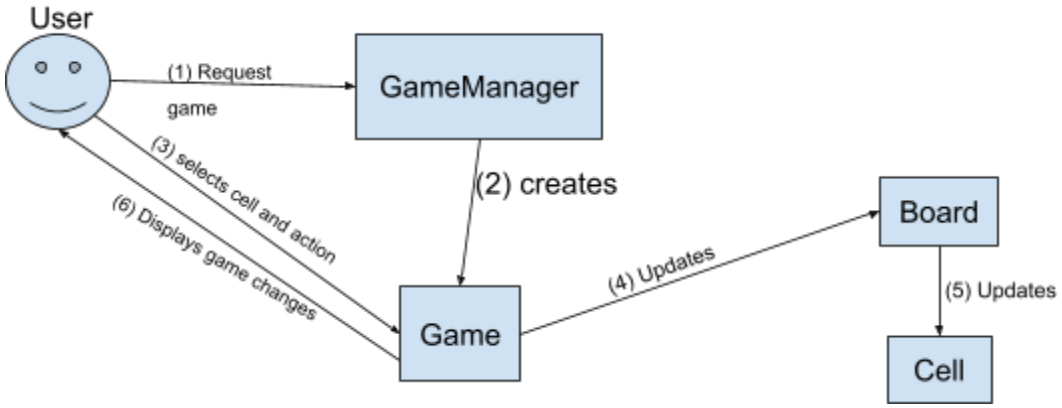
Data Flow

The high-level data-flow of the project can be described as follows:

1. The User initiates a new game through the GameManager.
2. The GameManager creates a new Game instance.

3. The User issues game commands directly to the Game instance, which processes the input.
4. The Game instance processes the command based on game rules and updates its Board instance.
5. In particular, it accesses and affects the Cell instances stored within the Board, effectively passing through the Board instance.
6. The Game instance displays the results of the action to the user.

An illustration of the data flow is included below.



Component Descriptions

Cell

- Description:
    - A data structure that represents a single cell in a game grid. It contains all relevant information about the state of the cell, such as the number of adjacent mines, whether or not it has been uncovered, whether or not it contains a bomb, and whether or not it is currently flagged.
- Attributes:
    - **adjMines**: Integer number of adjacent mines in the grid.
    - **covered**: Boolean which is True if the cell has not been mined.
    - **bomb**: Boolean which is True if the cell contains a bomb.
    - **flagged**: Boolean which is True if the cell is currently flagged.
- Methods:
    - **__init__()**:
        - Initializes cells as covered, unflagged, and with no bomb and 0 adjacent mines.
    - **__str__()**:
        - Returns a string version of the cell, which is either the bomb emoji, flag emoji, number of adjacent mines, or blank space, depending on the state of the cell.

Board

- Description:
    - Represents the actual grid in a game of Minesweeper. This class can be considered as a list with additional methods. In this program, the grid is represented as a single-dimensional list, where neighboring cells are derived mathematically through the getNeighbors method.
- Attributes:
    - **_board**: Private single-dimensional list of Cell instances.
- Methods:
    - **__init__()**:
        - Populates the board attribute with one hundred Cell instances.
    - **__getitem__(i:int)**:
        - Overloads the indexing operation, so that directly indexing a Board instance yields the index of the private _board attribute.
    - **__len__()**:
        - Overloads the len() function. It returns the length of the private _board attribute.
    - **display()**:
        - Prints out the entire board to the terminal, including row and column labels. Utilizes the string overloading in the Cell instance for simpler printing.
    - **getNeighbors(num:int)**:
        - Takes in an integer representing a cell. It then returns a list of integers, each representing an adjacent cell.

Game

- Description:
    - An entity that represents a single game of Minesweeper and all of its relevant information, including game status, bomb counts, flagged cell counts, and the game grid. It also contains the functionality for obtaining user commands and handling all game logic.
- Attributes:
    - **status**: String for the state of the game (“Playing”, “Victory!”, “Game Over: Loss”).
    - **flag_ct**: Integer representing the number of placed flags.
    - **bomb_ct**: Integer representing the number of bombs in the game.
    - **bomb_spaces**: List of all cell indices chosen as bomb spaces.
    - **board**: Board instance representing the game board.
    - **start_time:** Float representing the starting timestamp of a Game.
- Methods:

- **__init__()**:
    - Initializes the Game with a status of “Playing”. flag_ct, and bomb_ct are set to zero, and bomb_spaces is initialized as empty. Creates a new Board instance. Gains the current timestamp, and sets it as the start_time.
- **timeCheck():**
    - Returns a float, subtracting the **start_time** attribute from the current timestamp in order to return an elapsed time.
- **displayTime():**
    - After obtaining the time elapsed since the game began(using **timeCheck()**), the time is printed in seconds format.
- **printGame()**:
    - Prints the state of the game, including the current state of the board, the game status(status), and the number of mines remaining.
- **placeBombs()**:
    - Using the values placed in bomb_spaces(**must be initialized before using this function**), places bombs in each dictated Cell in board. Then, each Cell in board is processed, updating its adjMines attribute now that bombs have been placed in board.
- **propagate(space:int)**:
    - Function that recursively uncovers a Cell. This function should only be used on a Cell that has zero adjacent mines. It then uncovers all neighbors(unless a neighbor is flagged). If a neighboring Cell also has no adjacent mines(and isn’t flagged), the propagate method is called on that Cell.
- **printErr(msg:string)**:
    - Receives a string message and prints it to the terminal. It then requests the user to press the ENTER key before returning. This program clears the terminal when reprinting the game state, so this function is required to prevent error and hint messages from being immediately cleared.
- **getInput()**:
    - Obtains a user’s command of the form “(m|f)[1-10][a-j]”. An error message is displayed explaining the command syntax in the event of a malformed input. The function breaks the input into the command type(m or f), the row number, and the column letter, returning these three components in a list for further processing.
- **configure()**:
    - Obtains the user’s preferred number of bombs. If the number is not between 10 and 20, the user will be prompted again until a valid number is entered. Generates a randomized set of indices, saving them in the

bomb_spaces attribute. **This function should be called before placeBombs().**

- **move(prebomb:bool)**:
    - Main game logic function. First, it obtains a valid command from the user and derives the space from the last two components. Handles the operations necessary to flag a Cell(directly alters a Cell's flagged attribute) and handles all rule checks. If the command is to mine, the Cell is uncovered. **If the Cell has no adjacent mines, propagate() is called, which uncovers the Cell automatically. If not, the Cell is uncovered directly in move().** Behavior is slightly different based on the **prebomb** attribute. **If it is set to True, the selected space is checked for presence in the randomly generated bomb_spaces list. If it is present, then a new bomb_space is generated. Once the bomb_spaces list is adjusted, placeBombs() is called, installing bombs in the board attribute.** Then, the Cell is uncovered as normal. **If a bomb space is selected and prebomb is False, all bomb spaces are uncovered, and the game status is set to "Game Over: Loss".**
- **checkWin()**:
    - Function that checks if the user has already won the game. It counts the number of covered Cells in the board attribute. If that value is equal to the number of bombs(bomb_ct), then the game is won, and the status attribute is set to "Victory!"
- **checkBombPlacement()**:
    - A helper function that checks if bombs have been placed into the board attribute yet. It is a necessary component to ensure that the first uncovered cell is not a bomb. It iterates through the board attribute, counting the number of Cell instances that have a bomb (bomb=True). Returns True if that count is equal to the bomb_ct attribute.
- **play()**:
    - Comprises the main game loop. Initially configures the game with configure(). Then, runs a while loop, which prints the state of the game (printGame()) and requests a command (move()). **The optional parameter for move is set to True in this loop(see move() description).** This loop ends once the bombs have been placed (once the user uncovers their first cell). After that, another loop begins, performing essentially the same function, with two differences. **The optional command in move() is no longer set, and the checkWin() function is called at the end of every iteration.** The loop breaks once the game status is no longer "Playing".

GameManager

- Description:
    - Entity that coordinates initial contact with the user. It is responsible for printing an initial introductory message, as well as enabling multiple games per execution.
- Attributes: None
- Methods:
    - **__init__()**:
        - Performs no actions.
    - **newGame()**:
        - Creates a new Game instance, and calls its play() method. This function manages the completion of a single game.
    - **start_message()**:
        - Prints an introductory message to the screen, highlighting the objective and controls.
    - **start()**:
        - Main interface with the main() function. Calls for the initial introductory message to be printed. Then, it creates an infinite loop, which is broken unless the user chooses to play another Minesweeper game, allowing for infinite replays.