Minesweeper Application

Overview

This application will contain a version of minesweeper, with standard difficulty options of

1. Beginner (10 x 10 grid, 10 mines)
2. Intermediate (16 x 16 grid, 40 mines)
3. Expert (30 x 16 grid, 99 mines)

There will also be an 'AI' functionality, where the user can play against the AI on the same, or different difficulties, with the boards displayed side by side.

The AI can play on beginner, intermediate, or expert boards, and the user will also have the option of setting the skill level of the AI

The user will also be able to watch the AI play the game, without playing themselves.

Statistics about each game and players will be saved to a database, and players will be able to save their current game to a database and load the game (if you use the same name)

A web page will display the high-scores for human players, AI players, and both combined. High scores will be based on board size/difficulty and elapsed time.

Stretch goals:

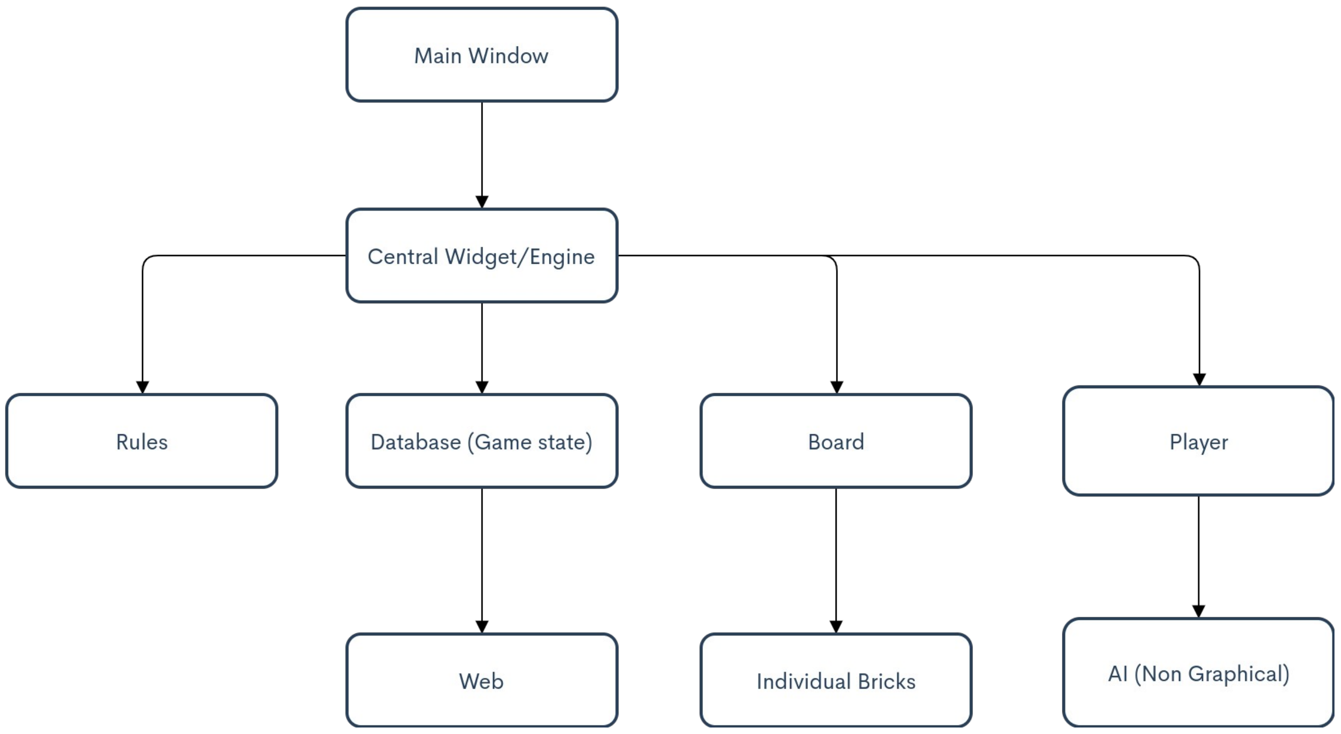
1. Add extra game difficulties where mines are added dynamically as time passes
2. Add extra game difficulties where remaining hidden bricks are revealed momentarily, randomly throughout the game – reorganize mines?

* Maybe through the AI?

3. Optimize AI
4. Real multiplayer – with player to player interaction
5. Upon bomb click, puzzle mini game to diffuse bomb
6. Maximum score, when bombs explode they detract from maximum score, cascading bombs

* check out battleship game strategy

Application Layout



Classes

- Main Window:
  - Windows to begin game (Single/Multiplayer window, Difficulty window)
  - Menus:
    - File: Load, Save, New, Quit
    - Edit: Preferences
    - Help: Rules/Link to website
- Central Widget / Engine:
  - Get moves from player/ receives moves from board
  - Verifies every moves with rules/game state
  - Once move is verified initiates change in game state and board
  - Contains Board class, rules class, game state class, and players class
- Board:
  - A purely graphical representation of the game state
  - Contains grid layout of current game state
  - Each item in grid is a Brick
  - Contains all graphics for altering board and appearance of bricks on board
  - Gets signals from Brick clicks for human, and receives instructions from engine if computer player
- **Game state**:
  - Contains all information about the current state of the game including individual brick information
  - Contains methods to change game state but does not decide when they need to be called (all methods to change the game state are called from the engine)
  - Save capabilities: when a game is saved the current game state will be sent to the database
- Rules:
  - Contains valid actions, and logic to determine validity of actions
    - Only one brick can be clicked at a time
    - If bricks have already been clicked they cannot be clicked again
    - win/loss verification
- Player:
  - Human or AI
    - If player is AI, contains logic for solving game
      - Queue of actions to be performed, which is pulled from by the engine
      - Limits information available to AI
    - If player is Human contains player profile, high scores, other statistics about game play for that player
- Brick:
  - Contains information about square, including visibility, coordinates, bomb/no-bomb, bombs touching, flagged/no flag

Database

Each Player’s information and game information will be stored separately whether the player is the computer or a person.

Save state – Many to one relationship with game info

Save ID (int) serial	Size (var char)	Visible bricks (int array)	Bomb Locations (int array)	Game ID (int)

***** Values in visible bricks and bomb locations correspond to coordinates

Game Info- One to many relationship with Game state | many to one relationship with player info

Game ID (int) - serial	Difficulty (char varying)	Time Elapsed (int)	Mines Left (int)	Game Status (int)

Player Info – One to many relationship with game info

Player ID (int) - serial	Game ID (int)	Player Name (char varying)	Human/AI (char varying)

GUI Layout/ Web Layout