Project Proposal

TP0

Project Description
Name: Blendoku
Description: This project is a recreation of the online game Blendoku. The player is given a randomly shaped grid of squares, some of which are filled with colors and others are empty. A set of additional color squares is also provided in the waiting zone. To clear a level, the player must fill the empty squares in the board using the color squares in the waiting zone in order to create smooth color sequences (gradients) both horizontally and vertically.

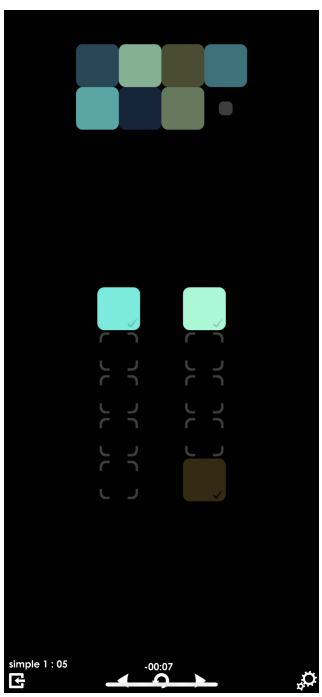
Similar projects
A similar project is the existing game Blendoku. My project will be a simplified version of that game. The existing Blendoku contains more than 500 levels, each with a predefined shape of grid. My project will contain fewer levels. Existing Blendoku also has the ability to compare the player score (the time it takes to clear the level) with the average score of the world's players. My project will only keep track of a single player's time. My project will also not include additional features such as achievements. Other than that, my project will have all the necessary features that Blendoku has, such as a hint system, a locking block mechanism, etc. This project also has some similarities with the game Sudoku. They are both played on a grid and have a single solution. The goal is to find that solution by following the game's rules.

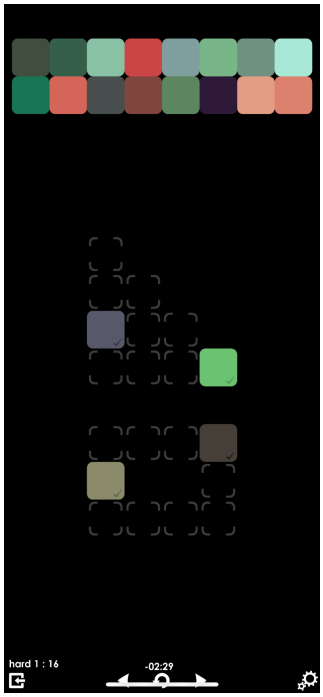
Structural Plan
I will have a Blendoku Class, and each level of the game will be an instance of this class.
The functions I will divide my code into are:

- A function that generates a board shape
- A function that generates color blocks
    - Randomly generate starting and ending colors
    - Then generate the in-between colors based on transition rules, or step size
- A function that finds the solved version of the board
- A function that generates some pre-filled cells
- A function that deals with user input
    - onKeyPress
        - To move the color squares between the waiting zone and the board
    - Double click
        - Lock a cell
- A function that checks for completion
- A function of the hint system
- A function that handles the main loop to play the game

Algorithmic Plan

The trickiest parts are randomly generating dynamic board sizes and determining and implementing the color blending rules.
When generating the shape of the board, I plan to use a randomized approach. However, I also need to consider that each square generated must have an adjacent square on at least one of its 4 sides. If a square has neighboring squares on its corners, then it will be invalid. Therefore, I need to take this rule into account when I randomly generate the board shape.





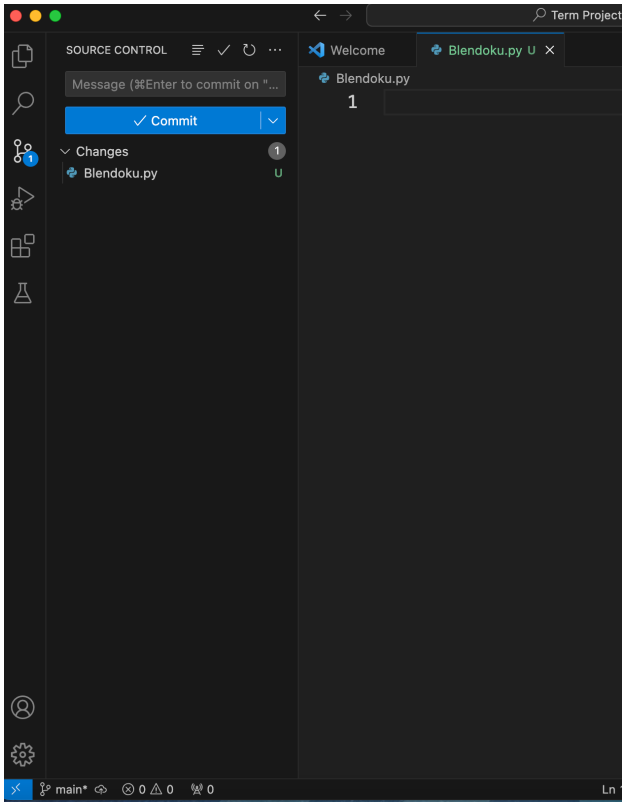
(Examples of different board shapes)
For the color blending rule, I need to define the exact color blending rule. I plan to use RGB values and work with the difference between the RGB values of two colors (for example, adjacent cells should only have one of the RGB values differ by 10 (a small value)). I plan to use backtracking and recursion to generate a solution board for each level first, so that I can check if each move of the player results in a solved board, and this solution board also allows me to implement a hint system (fill a cell for the player with the correct color).

Timeline Plan
Nov 20 - Nov 26 (Thanksgiving):
- Core game logic (generate board, generate color squares, implement color blending rules, hint system, different levels)
- User input and interactions

Nov 27 - Dec 1:
- User Interface Design
- Testing and Debugging

Dec 2 - Dec 6:
- Additional Features and Refinement

Version Control Plan



I have downloaded Git, and I will use the source control in VS Code to back up my code. I will publish it to Github so that it is stored online.

Module List

None

TP1 Update

Structural design changes:

I originally designed a function that generates board shape. Now, I realize that this is not necessary, because the original game Blendoku has a defined board shape for each level. So I shouldn't be generating random board shapes. Instead, I now choose to write predefined board shapes for 8 levels, just like we predefined the Tetris pieces in Tetris.

I also combined the two functions - one that generates color blocks and one that finds the solved version of the board. I will use one function (fillSolutionBoard()) to fill the solution board when generating colors.

Previously, I designed “a function that handles the main loop to play the game”. Now, I will be using the screens feature in cmu_graphics, and the function runAppWithScreens(). I will create three screens, one for starting the game (the “play” page), one for the home page (choosing a level), and one for the game.

Algorithmic design changes:

Previously I planned on using backtracking and recursion to generate a solution board for each level. However, now I realize that backtracking only works if I have the options first and then test the composition of the options to find a solution. In my game, I cannot generate the options first.

To generate a single-line gradient, I could simply find the number of colors needed, and randomly generate the RGB values for the first and last color, and then fill in the middle colors with incremental step sizes. In this case, I could use the algorithm of backtracking since I could generate the options beforehand. However, due to the nature of the game where a board consists of various lines (horizontal or vertical), each gradient must seamlessly blend with adjacent ones (I attached an image for reference). Therefore, I cannot generate all the colors (options) first before finding the solution to fill in the board (which is the logistics of Backtracking).

I decided to replace the backtracking algorithm with an algorithm to generate the solution board by filling in the first line with randomly generated gradient, and based on that filling in the rest of the board line by line, checking constantly that the junction results in a coherent gradient as well. Once the solution board is generated, I will have a complete inventory of colors (options) that users can manipulate.

For the color blending rules, I initially considered setting a fixed value for the difference between the RGB values. I have now changed to calculating different step sizes (RGB value differences) based on the starting and ending RGB values for each line of the board, as I realize that the starting and ending RGB value differences will be different for each line.

I also need to write a function to check for double click, so that I can implement my locking feature.

TP2 Update

Algorithm complexity:

I added the features of letting the players create their own board shape as level 9. I also implemented a 2-player mode. Originally, I designed the score system to be based on time. Now I revised it to be based on both time and the number of moves. The player should be aiming for the least number of moves, and after they achieve that, they should aim for the shortest time. I also revised the design for hint generator, now offering two types. One allows the player to choose the cell they want to know, and the other is a smart hint generator that automatically fills a cell that is the most useful (it has the most incorrect or empty cells around it).

Structure

I added another screen “playerlevel” for level 9, which allows the players to draw their own board shape. I also added a new class “twoPlayerClass” for the multiplayer mode. I also organized my code into multiple files, one for the main Blendoku game, one for the levelClass, and one for the multiplayer codes.

TP3 Update

- Let the player determine the number of prefilled cells for Level 9 (creating own board shape level)
- Limit the number of cells that players can fill in to create their own board shape in Level 9
- Save the best score even after closing the app
- Update instruction page