Test Report

Overview

- We employed 2 methods of testing in order to fully test our project
    - Backend testing: JUnit: This included testing all methods in the classes that are classified as the model
    - Front end testing: manual user interaction, (We manually tested each and every functionality of of the game through the User interface)

Backend Testing

CoordinatePoint class

Total number of tests = 3

1. testGetRow() - tests to see if the row is returned correctly
2. testGetCol() - tests to see if the column is returned correctly
3. testEquals() - tests to see if the equals method works correctly

GameBoard Class

Total number of tests = 18

1. testFillBoard(): tests fillBoard method. we do this test to see if the board is filled with the correct colors
2. testMakeMove(): tests makeMove method. we do this test to see if the move is made correctly
3. testIsWon(): tests isWon method. we do this test to see if the game is won correctly
4. testIsNotWon(): tests isWon method. we do this test to see if the game is won correctly
5. testNoMovesLeft(): tests noMovesLeft method, getMaxMoves method, getNumMovesLeft method. we do this test to see if the moves mechanism is working correctly
6. testGetNumMovesLeft(): tests getNumMovesLeft method, resetMoves method. we do this test to see if the moves mechanism is working correctly
7. testAddObserver(): tests addObserver method. we do this test to see if the observer is added.
8. testRemoveObserver(): tests removeObserver method. we do this test to see if the observer is removed.
9. testFloodFill(): tests floodFill method, getCellColor method. we do this test to see if the floodFill method is working correctly
10. testGetSelectedColor(): tests getSelectedColor method. we do this test to see if the selected color is correct
11. testGetHint(): tests getHint method. we do this test to see if the hint is correct

12. testGetLevelName(): tests getLevelName method. we do this test to see if the level name is correct
13. testGetRowsColumns(): tests getRows method, getColumns method. we do this test to see if the rows and columns are correct
14. testGetPalette(): tests getPalette method. we do this test to see if the palette is correct
15. testGetCellColor(): tests getCellColor method. we do this test to see if the cell color is correct 16: testResetBoard(): tests resetBoard method. we do this test to see if the board is reset correctly
17. testGetAllPiecesInBoard(): tests getAllPiecesInBoard method. we do this test to see if the pieces are correct
18. testIsEditing(): tests if a board is in editing mode.

GameBoardSquareTest.java:

This file uses the mock object to check the numbers of rows and columns of a board

BoardPointerShapeTest.java:

This files uses mock objects to check if the update() is called am accurate number of times.

Undo Backend Testing

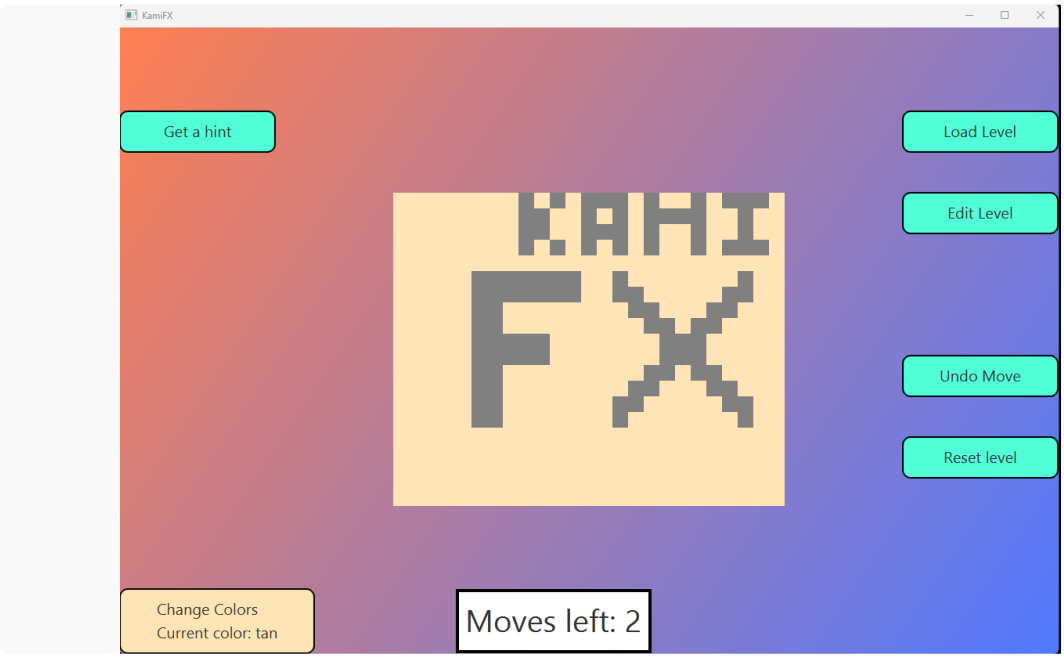
No tests were performed in the backend specifically for undo because, all the existing tests passed even though some of the involved methods were heavily edited and there was extensive Front end testing for undo. Hence backend tests just seem redundant.

```
C:\ManavData\college\Courses\CSC260\csc260-project2-groupd>gradle build

> Configure project :
Project : => no module-info.java found

BUILD SUCCESSFUL in 639ms
8 actionable tasks: 8 up-to-date
```

Front end testing:

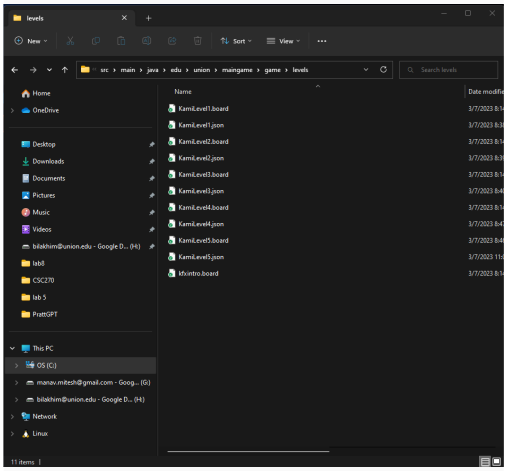
1: Check if edit and saving a level works:

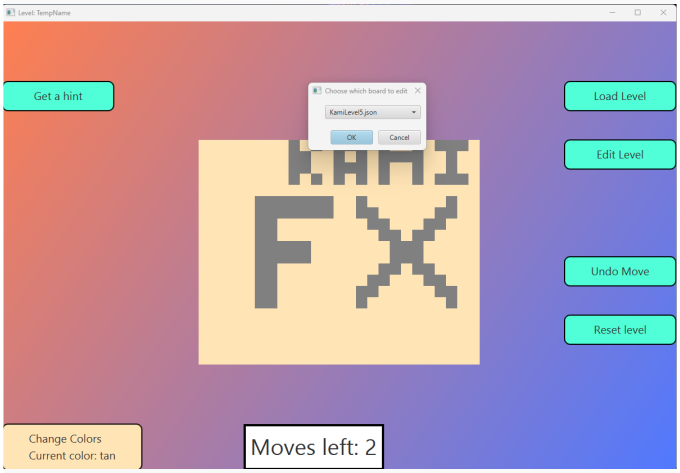
- open a random board

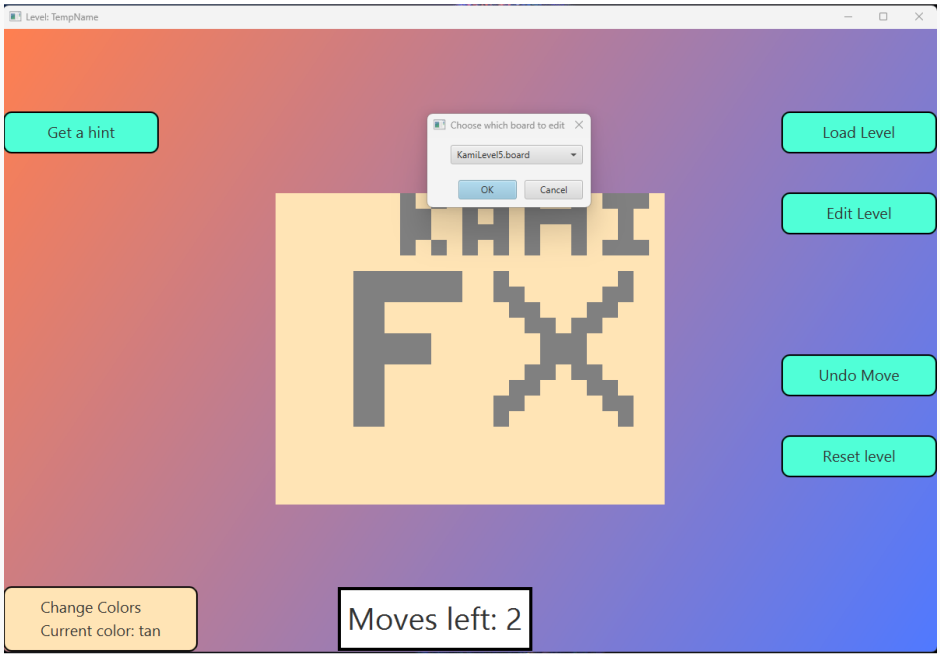


- select edit level

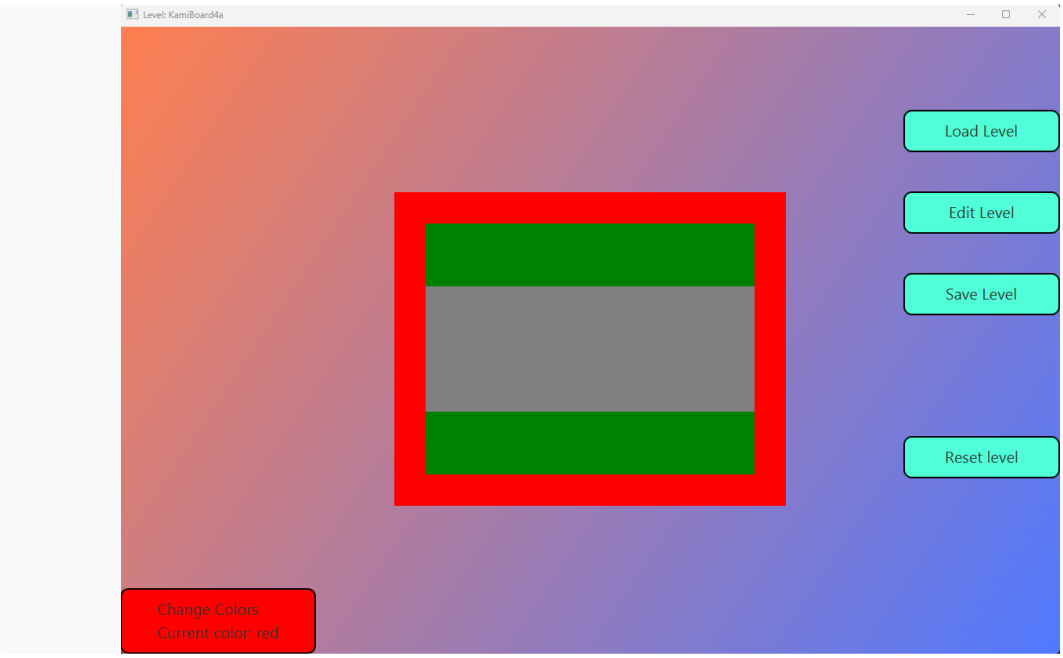
- select the board you want to edit from the dropdown list, as shown the picture two different types of file formats are excepted .board and .json



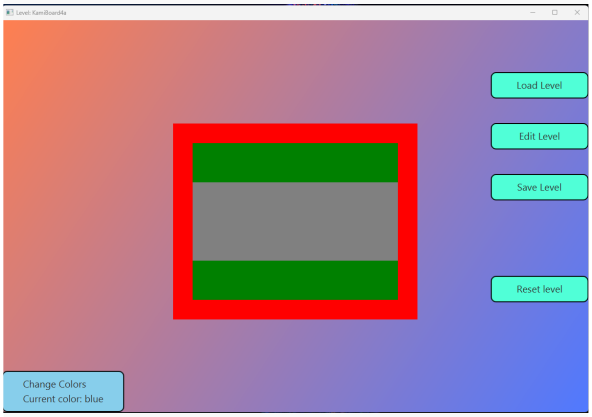


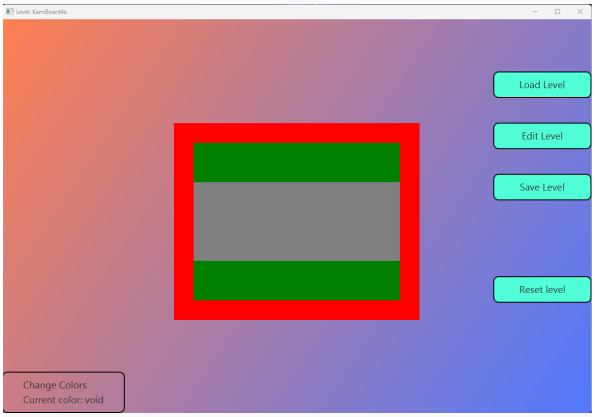


- click ok. note: buttons such as Get a Hint, undo and moves left which do not have functionality are now hidden and a button to save an edited level becomes visible.

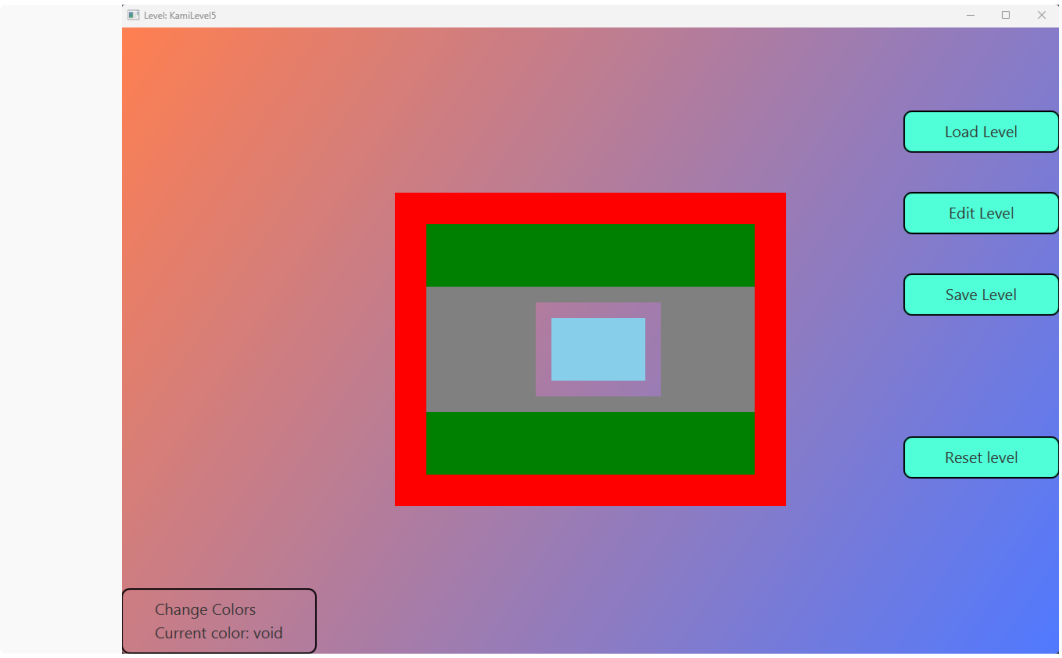


5b: test if all colors are available to choose from on the pallet

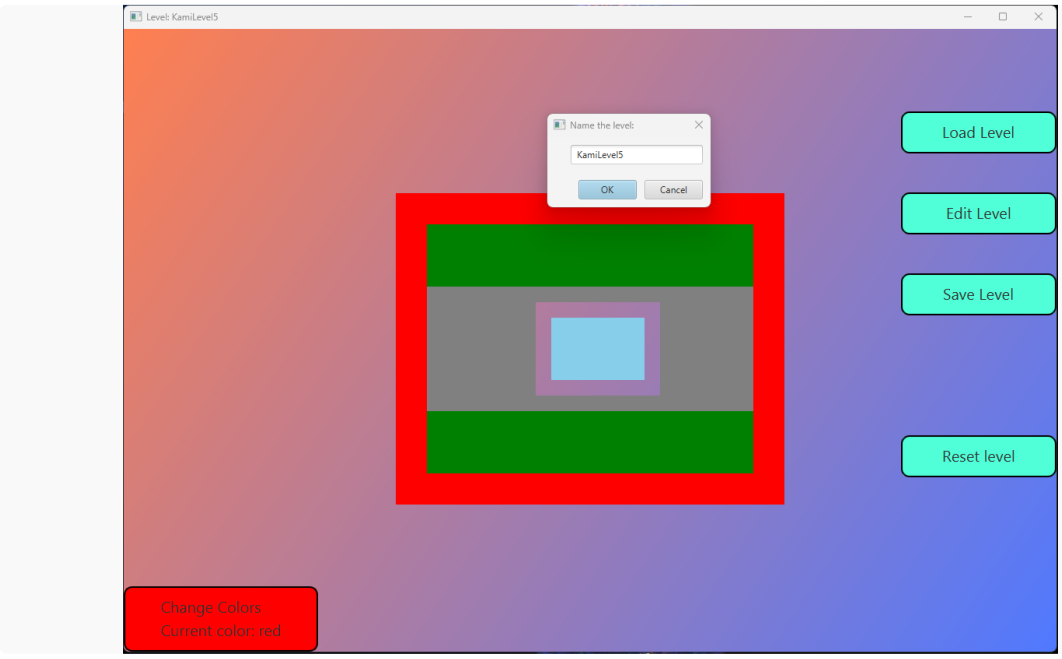




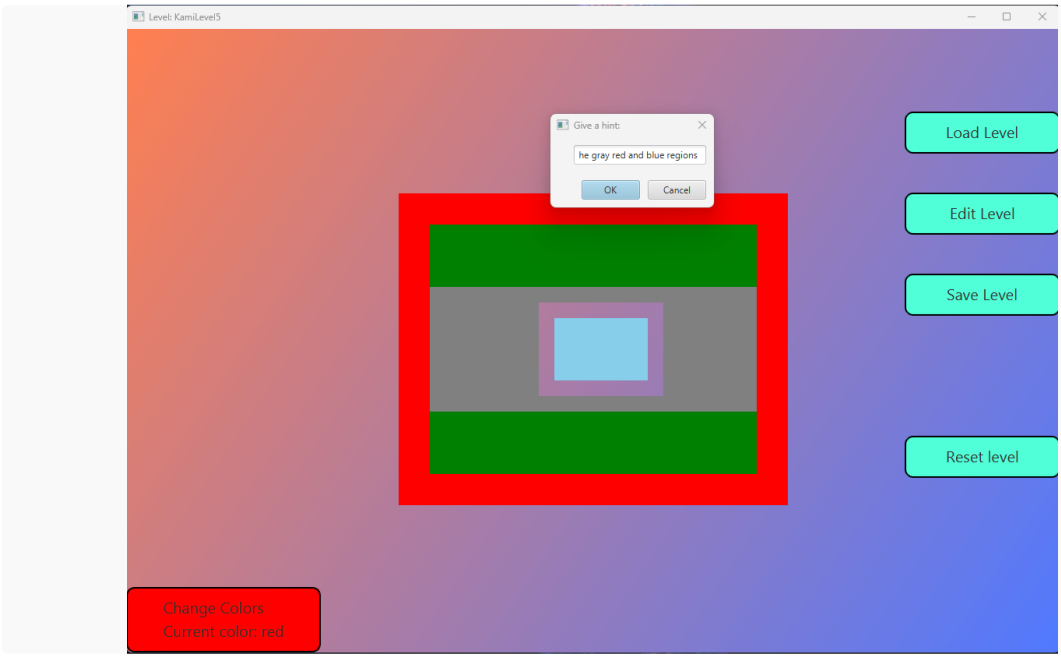
- in order to edit the board, select the color from the palette you like and start filling all the boxes. Here you can click each cell one by one or using the drag and fill feature



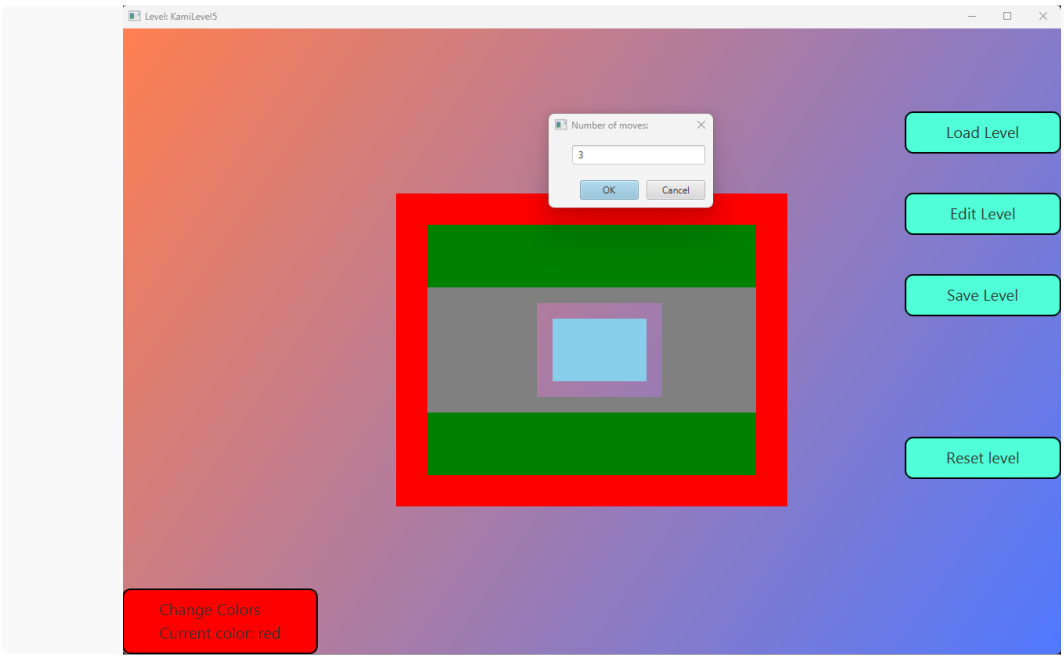
- once the board is complete, click on the save level button. the game will ask you to name the level. I am naming it "KamiBoard5.json"
- click on ok



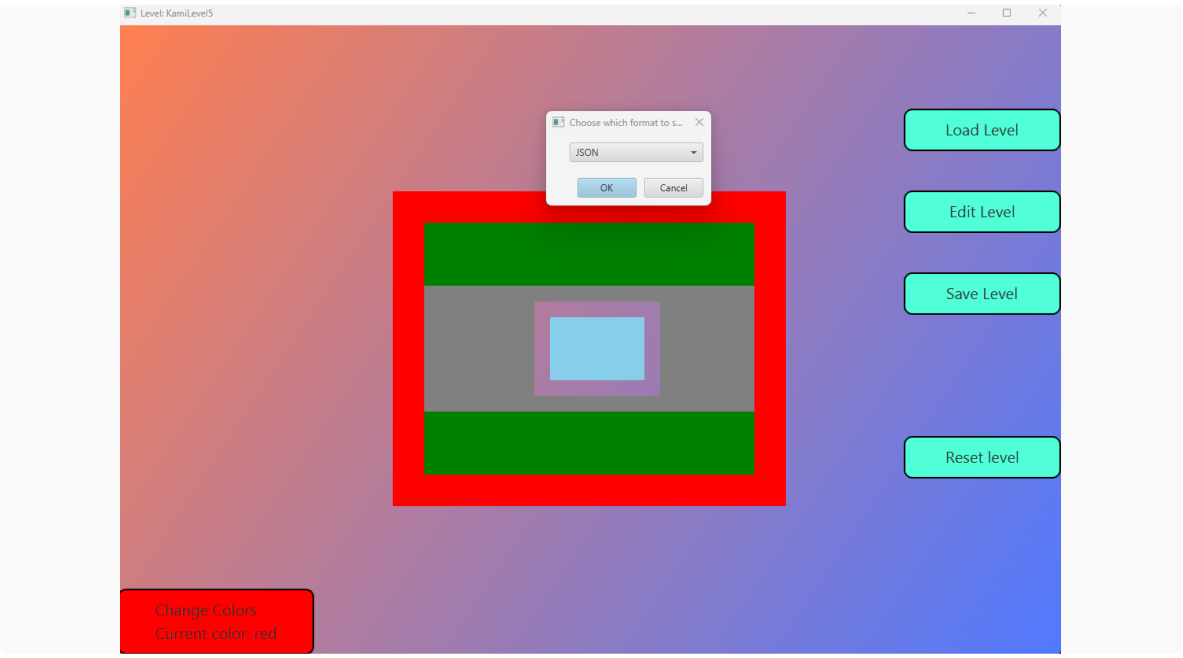
- It will then ask you to give a hint in order to solve the level. For simplicity, i will have the hint as "Choose the green color, select the gray red and blue regions
- click on ok



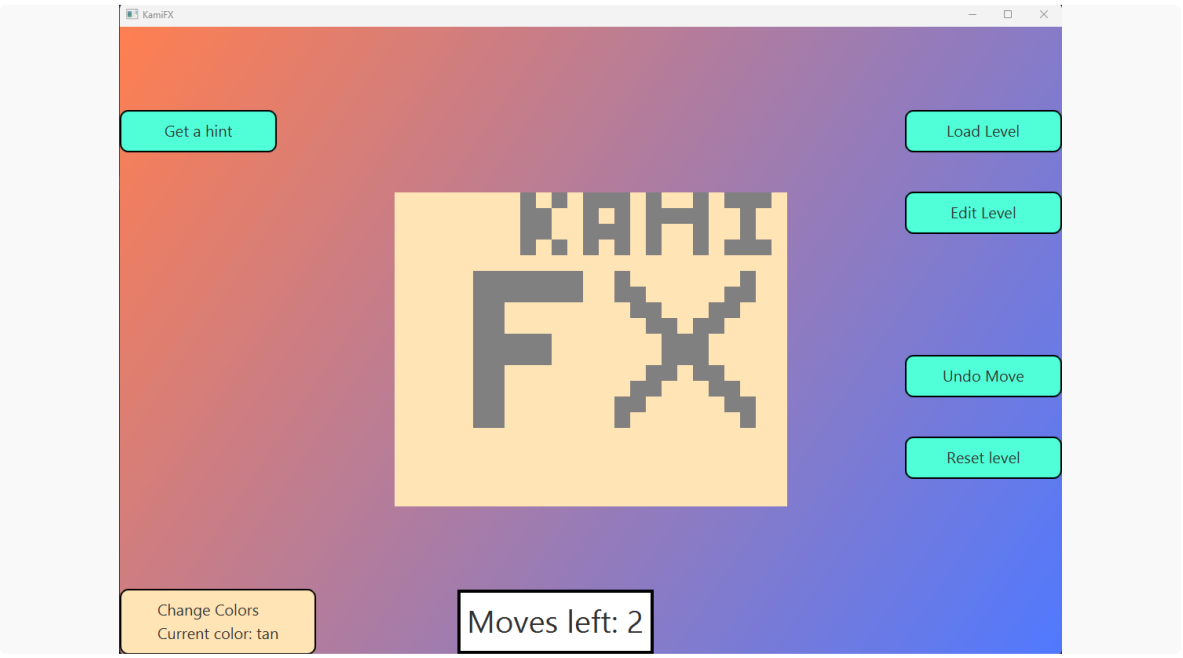
- it will now ask you to give the number of moves allowed to complete this board. For simplicity, I will have the number of moves to be 3.



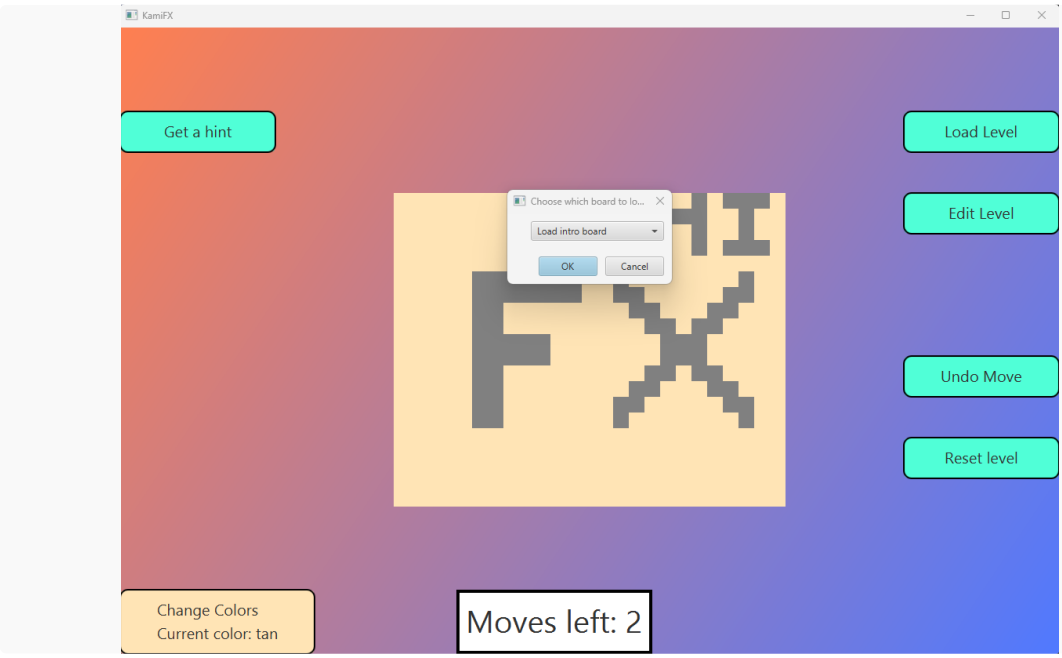
- It will now ask you to choose a format: we will use the JSON format to save the board.



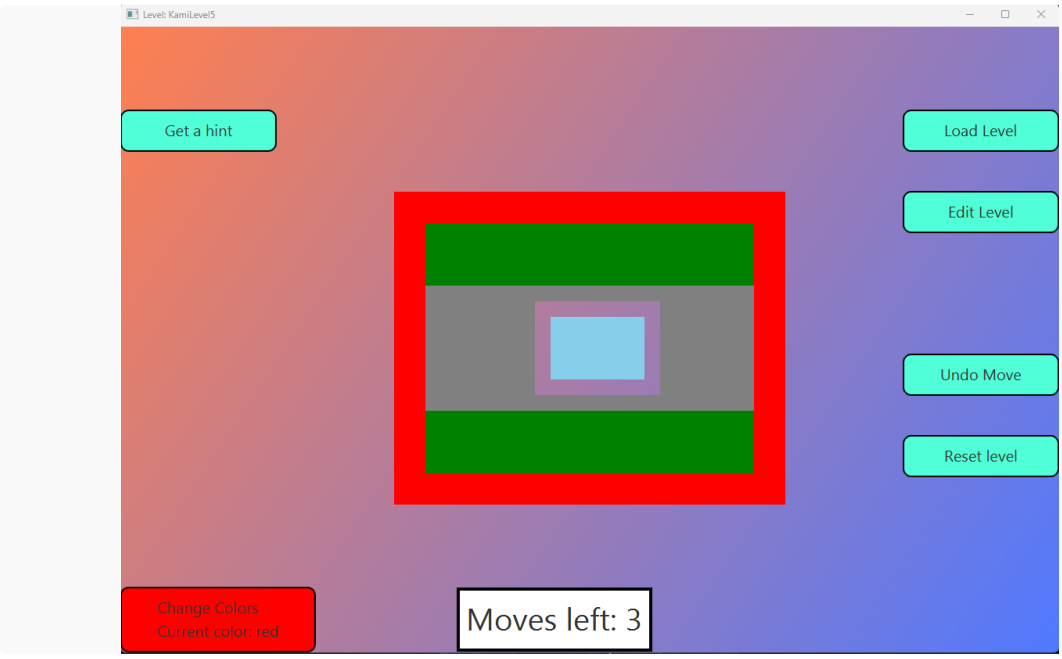
1. Loading an already existing board (we will test the board we just created above to see if all the functions work):
    - starting point



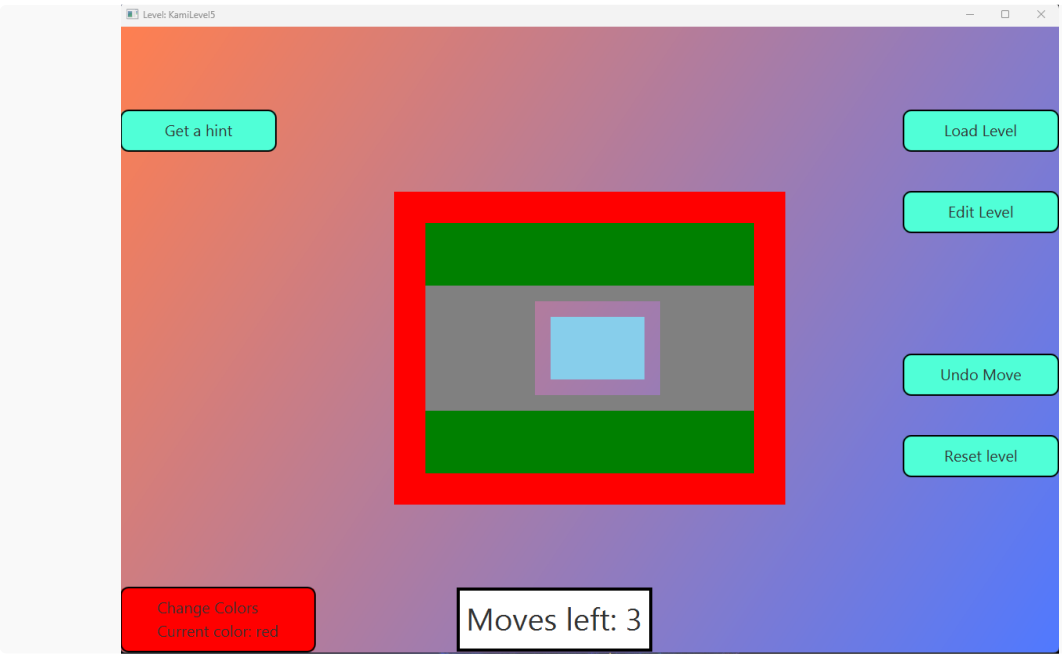
    - click on load level



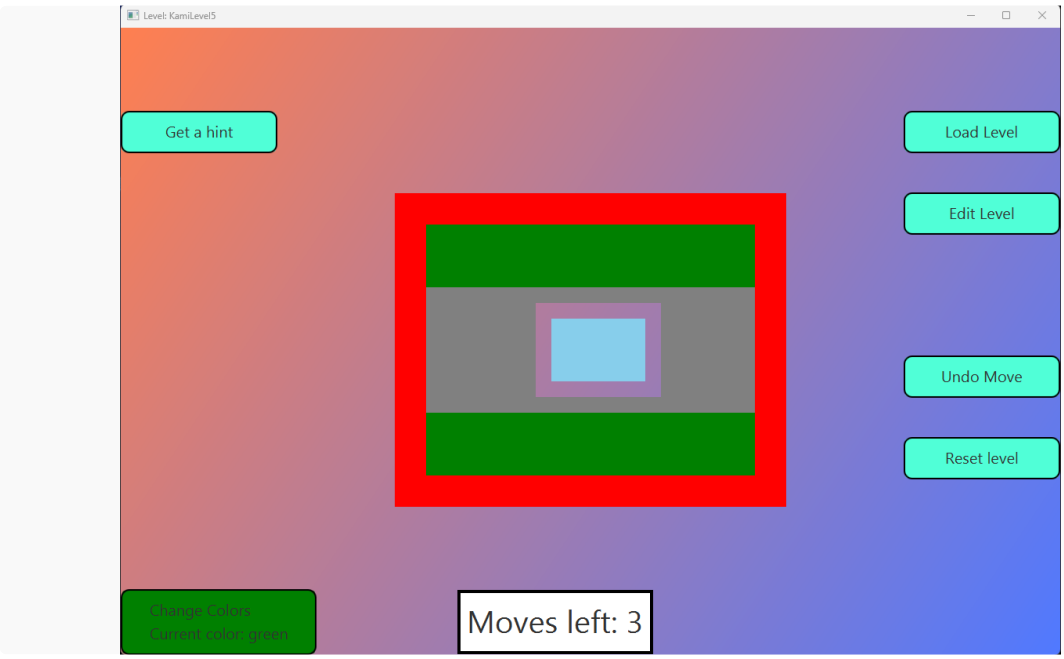
- Select appropriate level from dropdown
- click okay to see the new board loaded and the color pellet changed

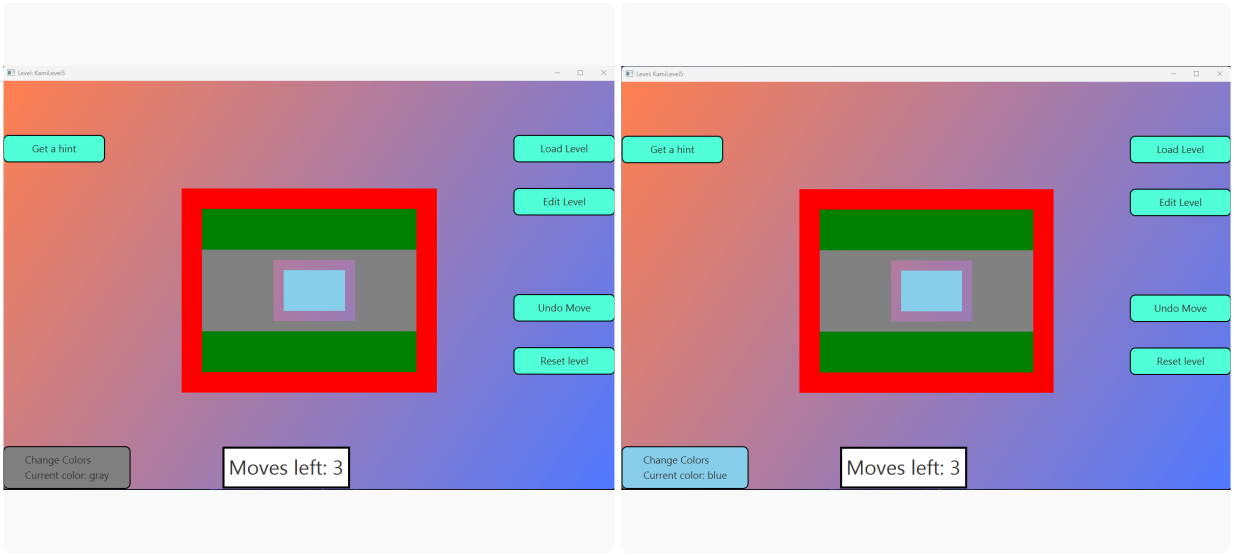


2: Check the color palette for a given board:

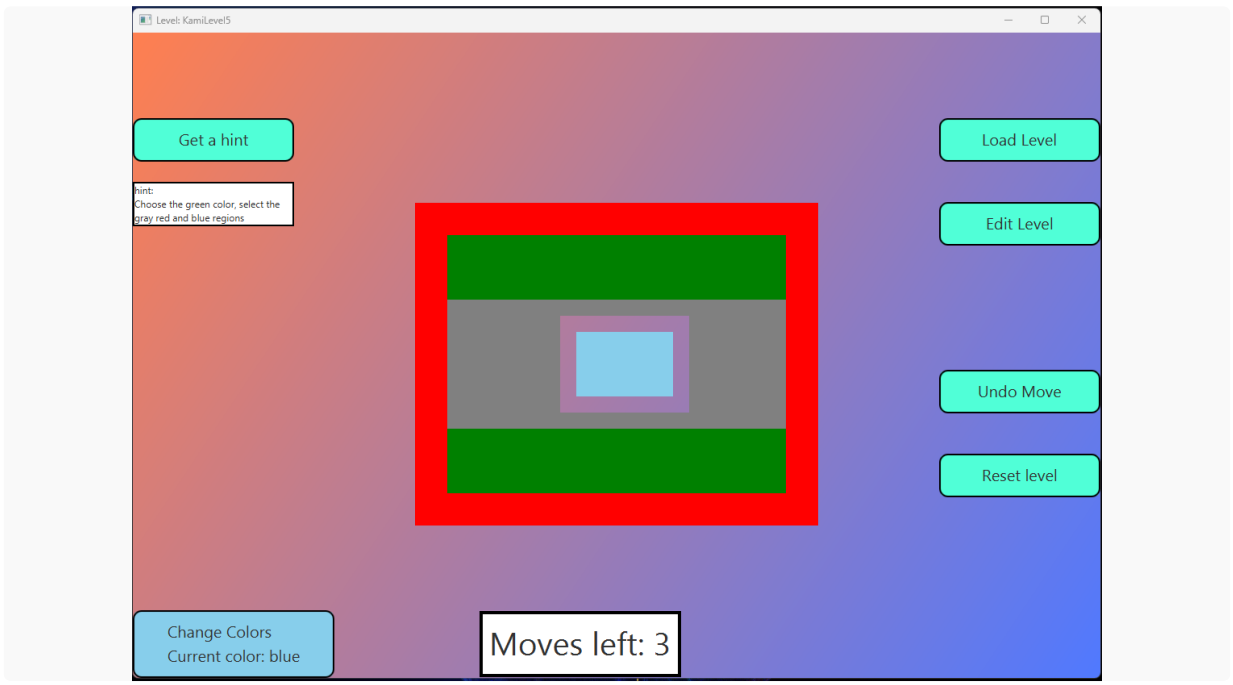


- click on the color palette to check if all relevant colors are there:



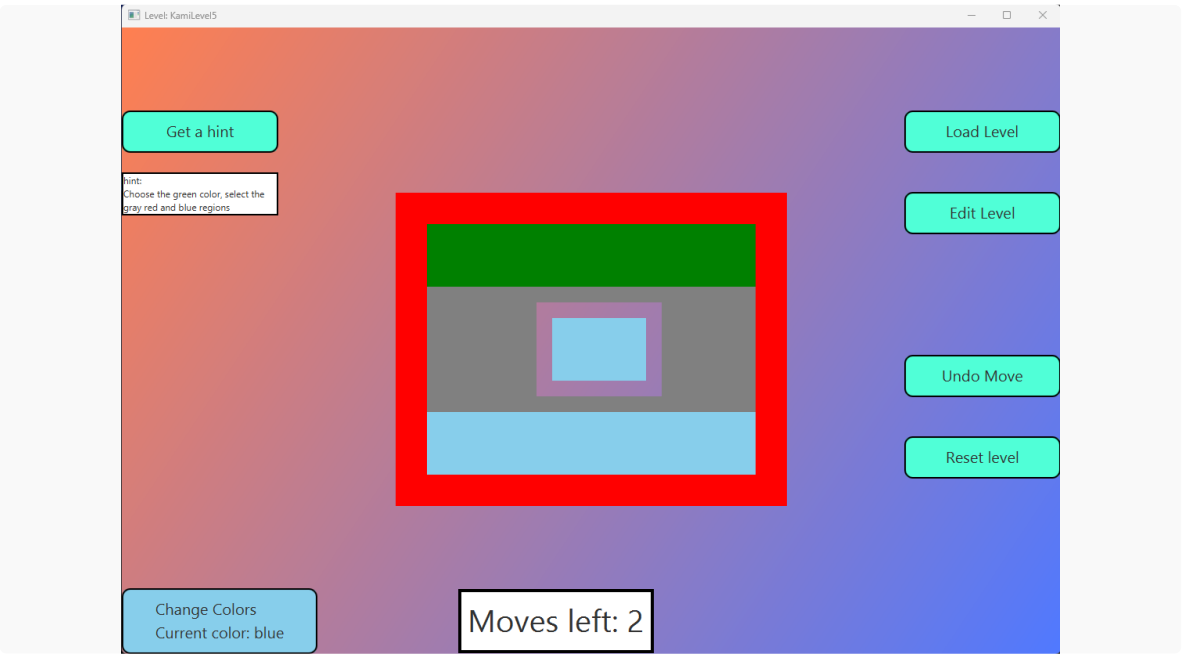


3 Check the get hint by clicking on the hint: it should display some text in order to solve the level

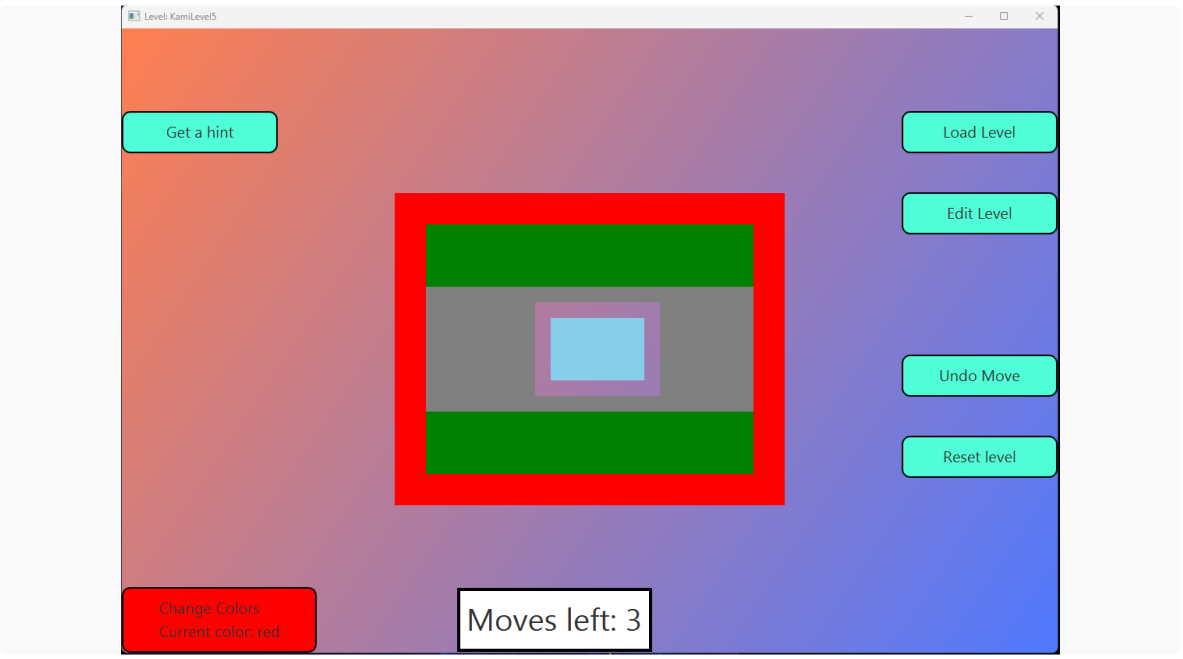


4: change the board and then reset it to go back to the original board:

- board changed by selecting the blue area using the color purple:

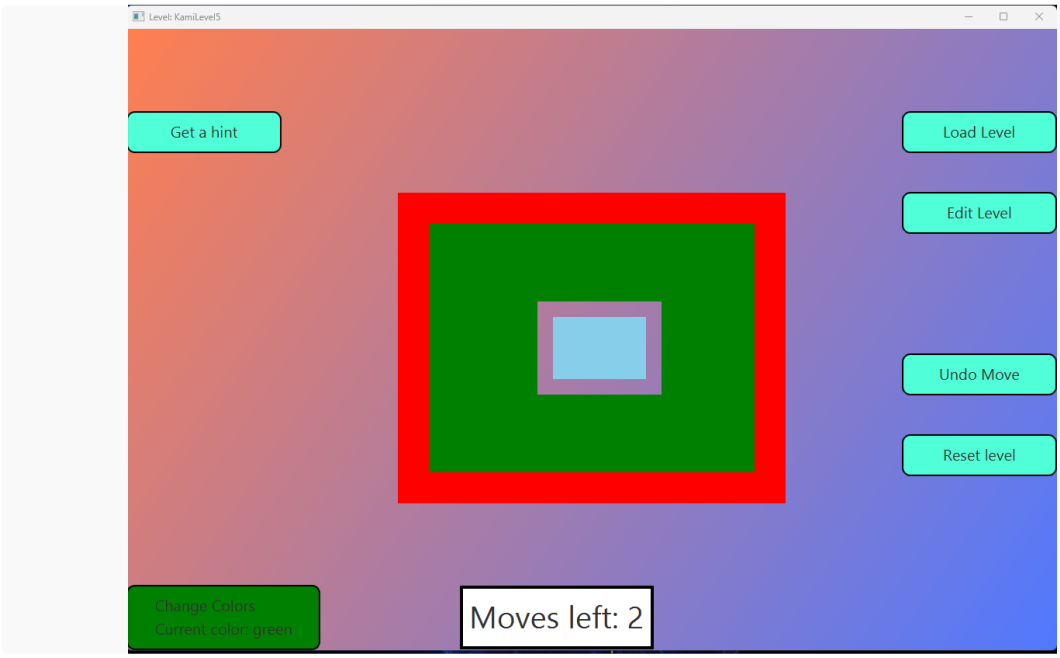


- board is reset using the reset level button

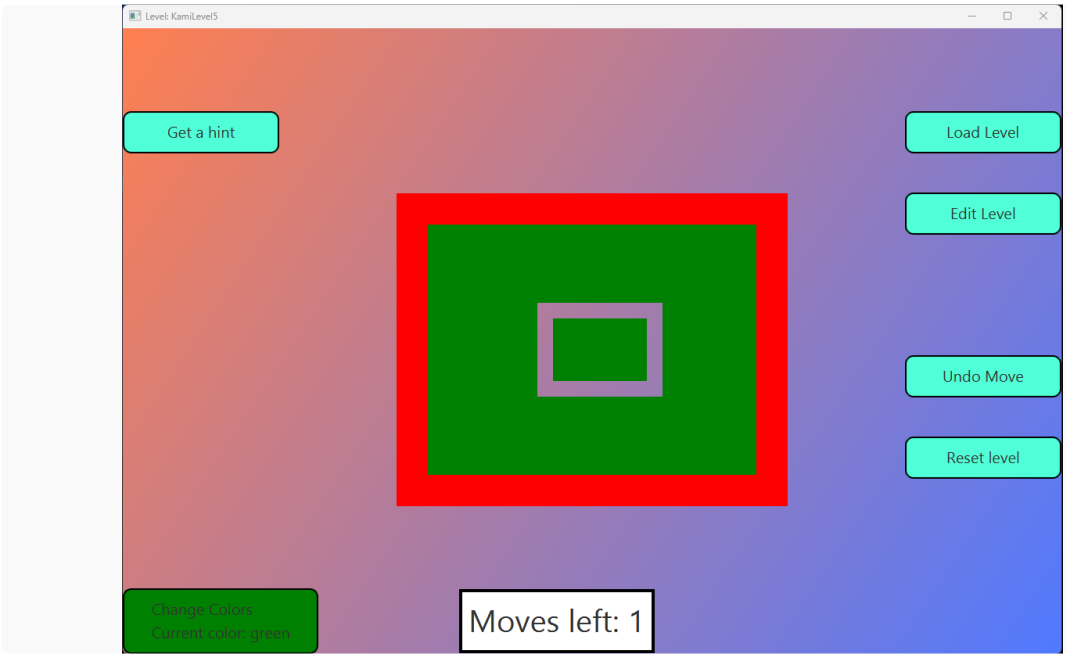


5: testing the Undo button:

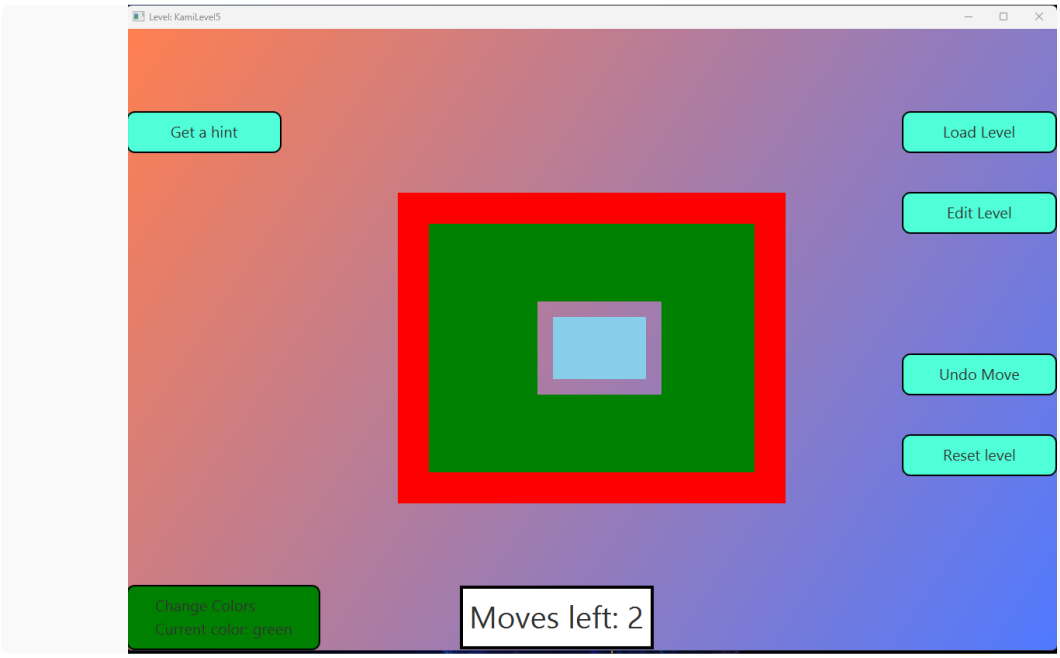
- Move 1:



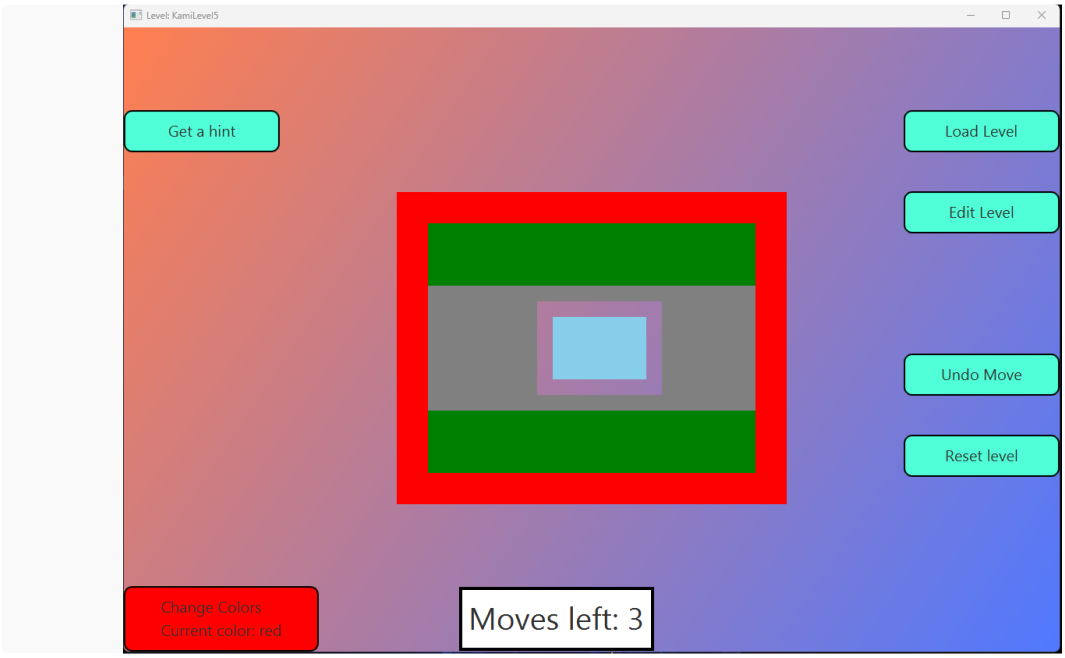
- move 2: (make sure void tiles behave as walls and do not change color)



- Undo 2

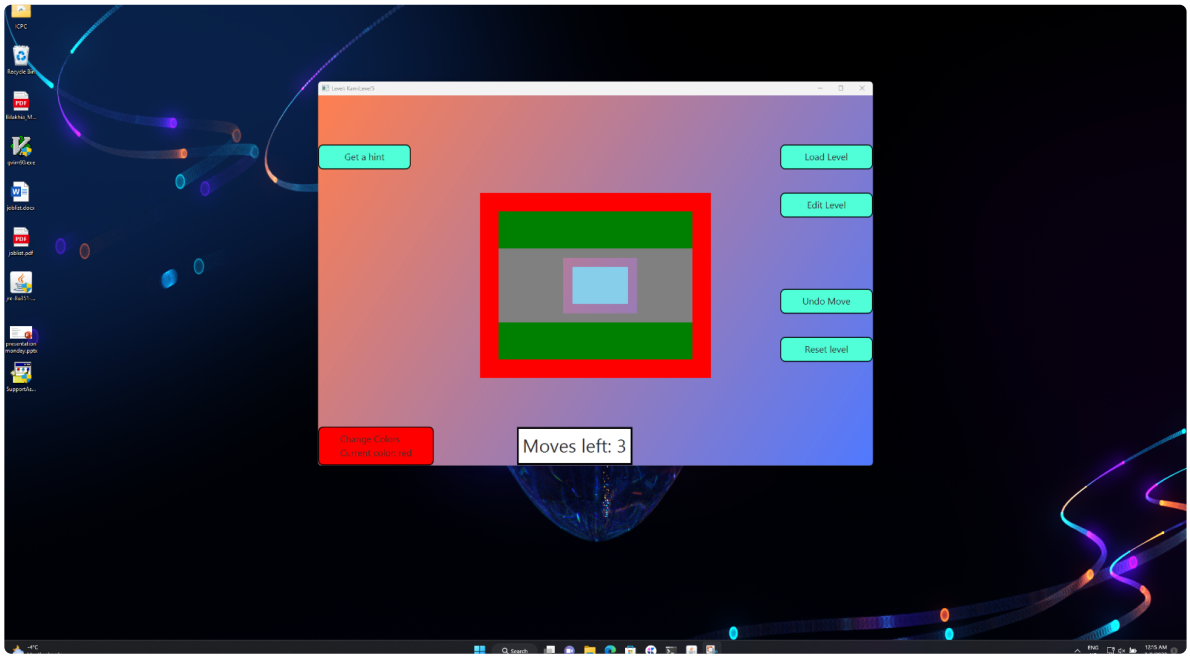


- Undo 1

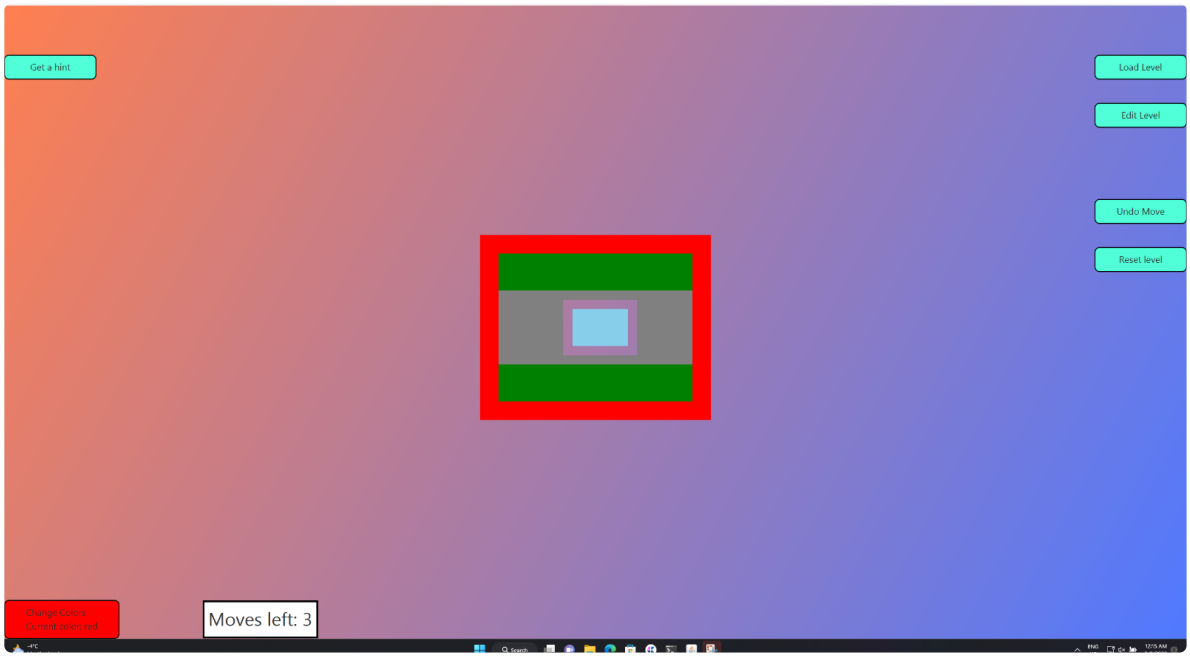


6) Full screen mode

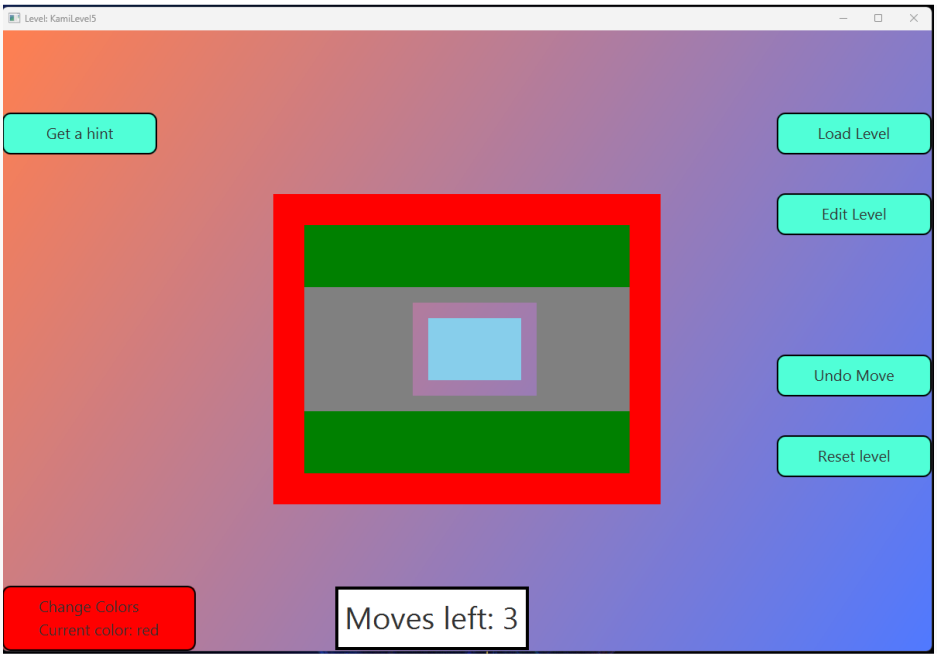
Window at original size:



Window in full screen:



7) Title of the window is now set to the current level name.


Level: KamiLevel5
Get a hint
Load Level
Edit Level
Undo Move
Reset level
Change Colors
Current color: red
Moves left: 3