Test Report

ColoredGraphSolverTests:

testSolver_MonochromaticGrid(): The rationale for this test was to check if the solver would be an empty list if the graph was already one color. When running this test, the solver did have a list that was empty of hints.

testSolver_1-3(): In these tests, the goal was to make sure that the solver would return a winning outcome for multiple levels. In each of the three different levels that were tested, the solver was able to successfully win the game, meaning that the solver was working correctly.

testSolver_ManyChanges(): The rationale for this test was to see if an exception would be thrown if there were too many hints to produce with the solver, as it could take a very long time if there is no minimum amount of steps for the solver to stop at. With this test, it was evident that the solver would produce an exception.

ColoredGraphTests:

testColoredGraph_Default(): The rationale for this test was to test the default constructor for the ColoredGraph class, and so the vertex set should be empty and so should the colorIds. The outcome of this test was that the ColoredGraph default constructor produced an empty graph.

testColoredGraph(): The rationale for this test was to test if constructing a graph that has already had vertexes added to it would have all of the vortexes in the VertexSext. Every vertice that was supposed to be in the VertexSet was tested, and every one was present in the set.

testGetNeighbors(): The rationale for this test was to check if the neighbors of a vertex would be as expected based on the logic of the game. Two different vertices were tested using an already filled in graph, and the list of neighbors contained the appropriate RectangleGridCells.

testAddVertex(): The rationale for this test is to check that, when adding a vertex to a graph, it appears in the vertex set. The outcome of this test showed that they were being added to the set.

testAddVertex_AlreadyExists(): The rationale for this test was to check if adding a vertex when it already exists in the graph should throw an error. The program did return an error, as expected.

testAddVertex_InvalidColor(): The rationale for this test was to check if adding a vertex with a color that does not appear in the colorIds should let that behavior go through. The program did let it go through as expected.

testPruneGraph(): The rationale for this test was to make sure that pruning the graph condensed adjacent cells with the same colors into one vertex. After running this test, the vertex set only contained three vertices, as expected, since there were three groups of color.

testAddEdge(): The rationale for this test was to make sure that when adding an edge to a graph, the neighbors of a vertex will contain the vertex that is being added to it. The outcome of the test was that this was true.

testRemoveEdge(): The rationale for this test was to make sure that when removing an edge from a graph, it no longer belongs in the vertex set. When running this test, we found that the vertex was no longer in the set.

testGetVertexColor(): The purpose of this test was to make sure that getting the vertex color would return the proper colorId. This test resulted in the proper colorId, meaning that the function was working.

testSetVertexColor(): The purpose of this test was to make sure that setting the vertex color to a new color would be successful and result in a new color for the vertex. The test returned true, so the method was successful in setting the color.

testGetNumberOfVertices(): The purpose of this test was to ensure that the number of vertices was correct when calling this method. The outcome of the test was true, meaning that the method was functioning as expected.

testBuildGraphWithAdjacency(): The purpose of this test was to ensure that when calling the method buildGraphWithAdjacency(), the graph would have edges between any cell that is next to each other based on the row and column. By testing all of the cells in a graph, we found that the outcome was true, meaning that the method functioned properly.

testColorFloodFill(): The purpose of this test was to ensure that the color flood fill for colored graphs was successful. By testing all the cells that should be changed when this is called on a particular vertex in a graph, it showed that the method was working properly and the color was spreading to neighboring cells with the same color.

testGetColorIds(): The purpose of this test was to ensure that getting the colorId would be contained in the set of integers representing the colorIds and that colorIds that were not in the graph should not be in the list of integers. The outcome of this test was true, meaning that the method was providing the correct list of colorIds.

ColorRepositoryTests

testConstruct(): the ColorRepository should be initiated with 4 default colors in there. The purpose is to check if the object is initiated correctly.

testListColors_Default(): after adding a new color, that color should show up in the repository. The purpose of this is to test if listing colors is working correctly.

testListColors_Ids(): test if listing color by a list of Ids return the correct colors. The purpose of this is to test if this API behaves correctly.

testAddColor(): test if adding a color behaves correctly. The color should then exist in color repository

testAddColor_AlreadyExists(): test if adding a color behaves correctly. Since the added color is already within the repository, it should throw an error as expected.

testGetColor(): test if retrieving a color by its id works.

ColorTests

testGetRValue(): The purpose of this test was to ensure that the method would return the correct r-value for the color. The outcome of this test was true, meaning that the method was working properly and returning the correct value.

testGetBValue(): The purpose of this test was to ensure that the method would return the correct b-value for the color. The outcome of this test was true, meaning that the method was working properly and returning the correct value.

testGetGValue(): The purpose of this test was to ensure that the method would return the correct g-value for the color. The outcome of this test was true, meaning that the method was working properly and returning the correct value.

testEquals(): This test was to make sure that the equals method for colors was working, as this is very important for the entirety of the program. By running this test, we found that the method was functioning properly and colors that were supposed to be equal were, while colors that should not be equal were not equal to one another.

LevelBuilderFactoryTests

testRegister(): The purpose of the test was to make sure that the register method would properly create a graph if the LevelType existed in the LevelType class. This test is testing both the register and createLevelBuilder, as these work together in the LevelBuilderFactory. The outcome of this test was that there was no error thrown, meaning that the register and createLevelBuilder were working properly in this case.

testCreateLevelBuilder_NoKey(): The purpose of this test was to see if, when no LevelType is registered, an error would be produced since the BuildFactory should not be able to build a LevelBuilder of a type it does not know. In this case, an exception was produced, meaning that the method was functioning properly.

LevelRepositoryManagerTests

testLoad(): The purpose of this test was to ensure that loading a level would work properly when it is in the level repository. To test this, the level was loaded and all key components were checked, with the graph having a correct amount of vertices, number of maximum moves, current color, and that the level would be solved when using the hints. The outcome of this test was true, meaning that loading the level was successful.

testLoad_UnknownFile(): The purpose of this test was to ensure that an exception was produced if the LevelInfo did not already exist in the repository. The outcome of this test was that an error was produced, meaning that the method was functioning as expected.

testSave(): The purpose of this test was to ensure that, when saving a level, it is added to the LevelInfo, meaning that it is now in the repository of saved levels. This test showed that it was in the LevelInfo list, meaning that save was working properly.

MoveTests

testGetColor(): The purpose of this test was to ensure that getting the color of a move was working properly given a specific move. The outcome of this test was true, meaning that the method was behaving properly.

testGetRow(): The purpose of this test was to ensure that getting the row of a specific move would produce the correct integer. In the outcome of this test, we saw that the method was behaving properly.

testGetCol(): The purpose of this test was to ensure that getting the column of a specific move would produce the correct integers. In the outcome of this test, we saw that the method was behaving properly.

RectangleGridCellTests

testAdjacentTo(): test if the implemented adjacentTo() of RectangleGridCell behaves correctly. 2 cells next to each other in terms of their positions in the grid should be adjacentTo each other.

RectangleGridLevelBuilderTests

testLevelBuilder(): The purpose of this test was to ensure that the constructor for RectangleLevelBuilder was working properly and that the number of rows and columns was correct, as well as the colors in the colorIds was also functioning. This test proved to be true.

testGetCurrentColor(): The purpose of this test was to ensure that the current color of the builder was initially red, and that when switching the color, the current should then be the color that had been switched to. The outcome of this test was true, meaning that both cases were working.

testSetColor(): The purpose of this test was to ensure that setting the color of a given cell in the LevelBuilder would change the color of the cell. The test proved that setting the color of the cell was successful.

testChangeGridSize(): The purpose of this test was to ensure that the method to change the grid size was working and altering the size of a LevelBuilder's ColoredGraph, which proved to work correctly.

testRestart(): The purpose of this test was to ensure that restarting the LevelBuilder would return all cells to the default color red. The method performed this correctly for all cells that had been changed.

RectangleGridLevelTests

testConstruct(): test if the loaded level is of correct size and have the right number of moves. The rationale is to test check if the level is properly loaded in. Further testing is conducted in other tests

testGetColorAt(): test if getting the color of a cell(vertex) function properly.

testSwitchColor(): test if switching color is functioning properly. The level should let the user switch to a different color.

testSwitchColor_Invalid(): test if switching color is functioning properly. The level should let the user switch to a different color even if it is not inside the level (will be a bad move on the user side, but it should still works)

testGetCurrentColor(): test if getting current color is behaving correctly. Should return the current color of the level

testPlay(): test if play() is behaving correctly. The floodfill should happen from the target vertex. The rationale is that we have to test if playing API (core of the game) is working as expected.

testPlay_noMovesLeft(): When there is no more left, the level should throw an IllegalArgumentException saying the user can't play anymore because they ran out of moves.

testGetColors(): test if getting the colors that are currently in the level behave correctly. The code passed the test by returning the expected colors.

testGetHints(): Test if getHints() return the right hints to the level. The code gives the right hints as expected

testLevelState_Win(): Test the getLevelState() API. It returns WIN when the level only has 1 color left as expected.