YOUR PROJECT TITLE AND ACRONYM REPLACE THIS

YOUR NAME REPLACES THIS

Assignment 3

7/2/20

For this assignment, you will describe and implement release 1 of your term project. You will incorporate *sorting* for a particular kind of circumstance, and *multithreading*. Try to build out your existing project—and paste it at the top of this document, heading and all--so that you have something substantial to show others at the end of the term, otherwise start a brand-new project.

If you lack the programming skills to implement multithreading, use pseudocode.

Submit this completed Word document, replacing all parts "Replace this …", observing and retaining the gray text. Your materials—in black 12-point Times New Roman—should not exceed 5 pages excluding references, figures, and appendices. Use the Appendix sections for additional material if you need to. These will be read only on an as-needed basis.

3.1 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project—half-page (12-point Times New Roman) limit. By the end, term projects will incorporate most of the topics discussed in the course—so read ahead get a feel for what's to come. To do this, you may need to alter the direction of your project or introduce an additional project. You may alter this or even replace it as the semester progresses. You will probably find it useful to use your project acronym.

The goal of the project is to create an amalgam of algorithms that can process a series of arbitrary tilings and tiling rule sets. The main algorithm that is run for this term project is an algorithm that I am working on independently of the class, and is called a PenroseAutomata. The PA algorithm is split up into two extremely computationally complex parts. The first part, is a projective algorithm that takes a dim dimensional mother lattice whose hyperplanes are offset by a shift vector shiftVect, and projects it using a function such that it can create any unitary 2D tiling. Once the tiling is constructed, it is evaluated such that neighborhoods are generated for each tiling, along with a series of statistics that are quickly measured. Then the most intensive part of the algorithm takes place which is the Automata portion of the PenroseAutomata. The tiling is automated with an arbitrary set of state change functions, which act as a generalization of Conway's game of life but with states chosen from a state continuum .

This however is not the function of the algorithms to be created in class. The first algorithm completed for Assignment 2, was a quadTree generating algorithm. Which takes a 4 dimensionally projected tiling (a 2D grid) and converts it into a canonicalized quad tree g generations after an initial condition is set.

3.2 PROJECTED I/O EXAMPLE FROM PROJECTED COMPLETED PROJECT

Provide an example of projected *concrete* output for designated input. You will not be held to fulfilling exactly this—it is just explanatory at this point, to indicate where your project is going. We recognize that project direction and details will change as the term progress. This section refers to the project as a whole, not just to what you will produce this week, so we can gain an idea of what you have in mind overall.

https://github.com/jcanedo279/PenroseAutomata

One can look at the above repo to have a rough representation of what the algorithm does. Taking in a series of parameters that represent a tiling and an automata, the algorithm spits out statistics of the tiling automata and a gif of the automata itself. If you are to scroll to the bottom of the readme, there are a series of gifs that represent several methods of creating these and how certain parameters affect the results. Below the gifs is a better explanation of what each of the parameters inputed into the algorithm does.

Because of the extremely large nature of the output, in the form of several individually large frames, and the also large input, I will refrain from putting a direct example. But rather will explain approximately what is inputted and outputted. Furthermore because a genetic algorithm is used to optimize the emergent complexity of the automata, the example input will be even more simplified.

Inputs:

Tiling:
-Dimension and size of mother lattice
-Initial condition of lattice hyperplanes
State Space:
-Number of colors and number of states
Automata:
-Color sequence (numColors length), and color sequence generation method
-State transition matrices (three)
-Generation info. (min, max, fit, print)
-Calculate using minimum set of boundaries
-Use a boundary approximation technique
-OG Conway's Game of Life mode
-Tile outline and display options
-Information needed to create a boundary
-Information needed to create an invalid set of tiles
-Directory paths information
-Genetic algorithm information

Outputs:

Within outputData/ directory and within the poolMultigridData/ directory (for a genetic algorithms) or unfitMultigridData/ directory (for non genetic algorithms), the output is stored. Specifically, the output produces as follows:

-An animation of the Penrose automata

-detailedInfo.json and paramSafetyCheck.json, used for storing the genetic info of the automata and for storing the parameter check system output

-Color composition (magnitude and normalized percentage) vs. generation plot

-Average change vs. generation plot

-Value statistics vs. generation plot

-Tile stability vs. generation plot

-Boundary statistics vs. generation plot (if using boundaries)

-Tile fitness vs. generation plot

More concrete examples of the actual files are displayed in different levels of depth in the main README displayed at the root of the GitHub repository linked above.

In the future, one could hopefully use my term project to analyze more information on the tiling automata. Such as converting the tiling into a quad tree or performing Dijkstra's greedy search on the tiling.

3.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE

Supply functional requirements statement that you accomplished for this assignment, together with input where applicable, and output. Please try to state requirement in declarative form, as illustrated in the examples, because here we want to know the functionality intended (*what*, not *how*). The example material should be deleted. Keep in mind that the implementation of these requirements will incorporate *divide and conquer*, and that will probably influence the requirements you choose to implement in this assignment.

3.3.1 Multigrid redundant sort

Req1: Given four input parameters (xi, xf, yi, yf) (defaulted to the whole grid) representing the boundaries of a sub matrix of a larger given matrix, the sub matrix is copied from the larger given matrix and stored.

Req2: The now stored sub matrix is then sorted left to right top to bottom such that if one was to read the rows of the matrix as one would a sentence in a book, each subsequent element will be larger or equal to the last. (this is made sure of no need to check)

Req3: The sub matrix is sorted in a way such that it efficiently deals with repeated elements

Req4: The sorted sub matrix is displayed on the console

3.3.2 Display tiling multiprocessing

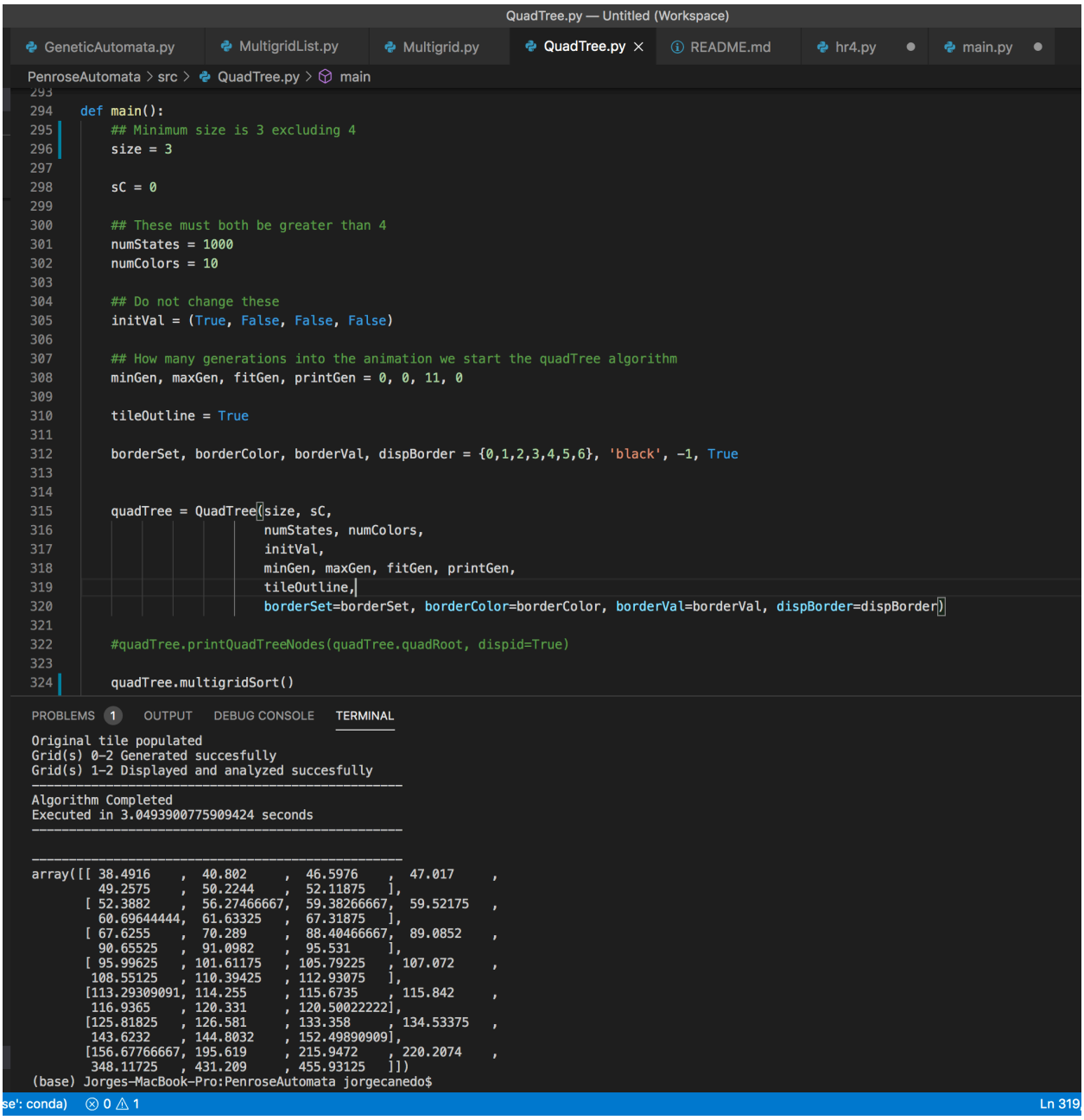
Req1: Given one of three settings, displayTiling() creates and/or returns an axis of tiles by multithreading, multiprocessing, or regular list comprehension.

3.4 ILLUSTRATIVE OUTPUT

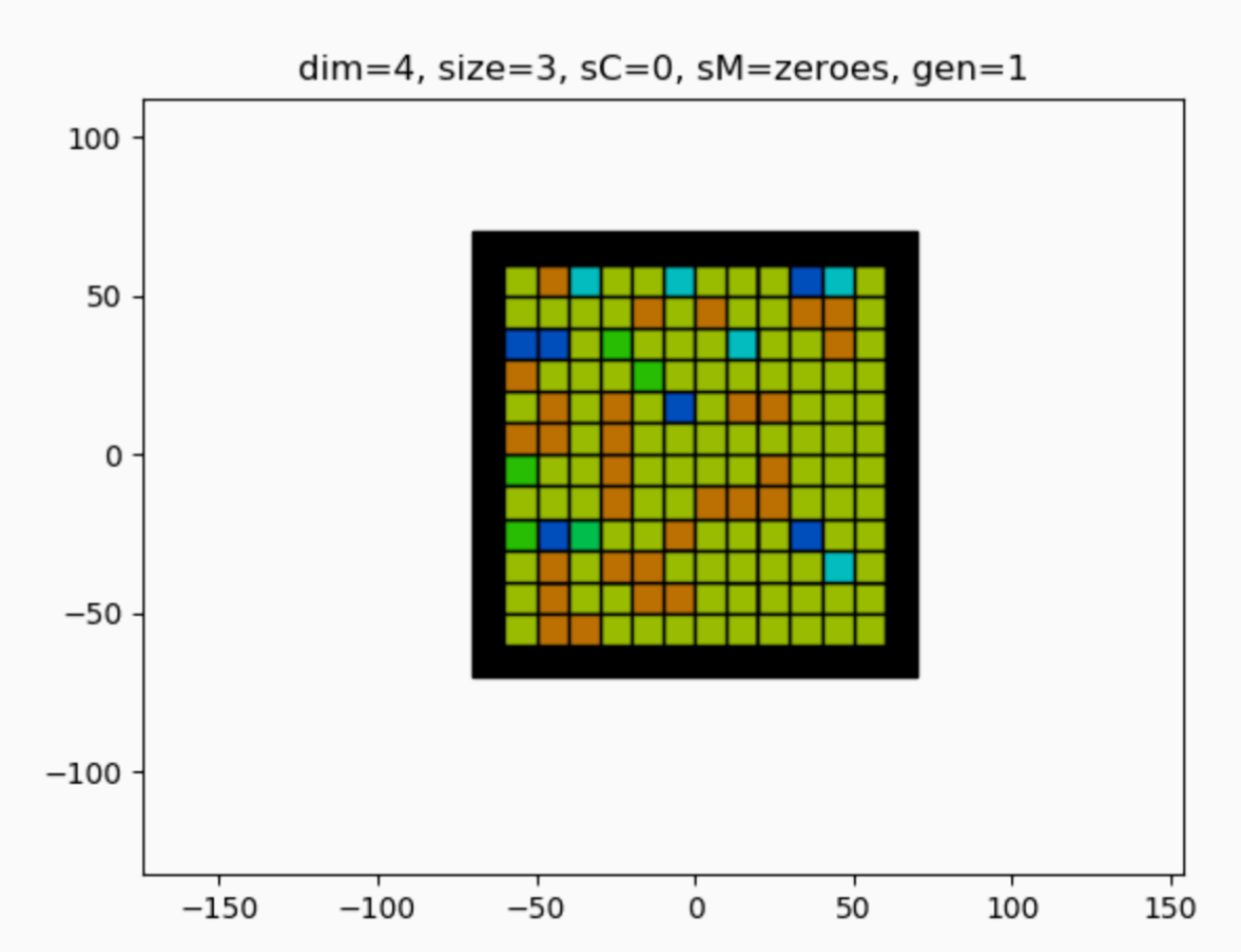
Provide illustrative output from your application showing that the requirements have been met. Explain what class.method(s) produce it.

Req 3.3.1:

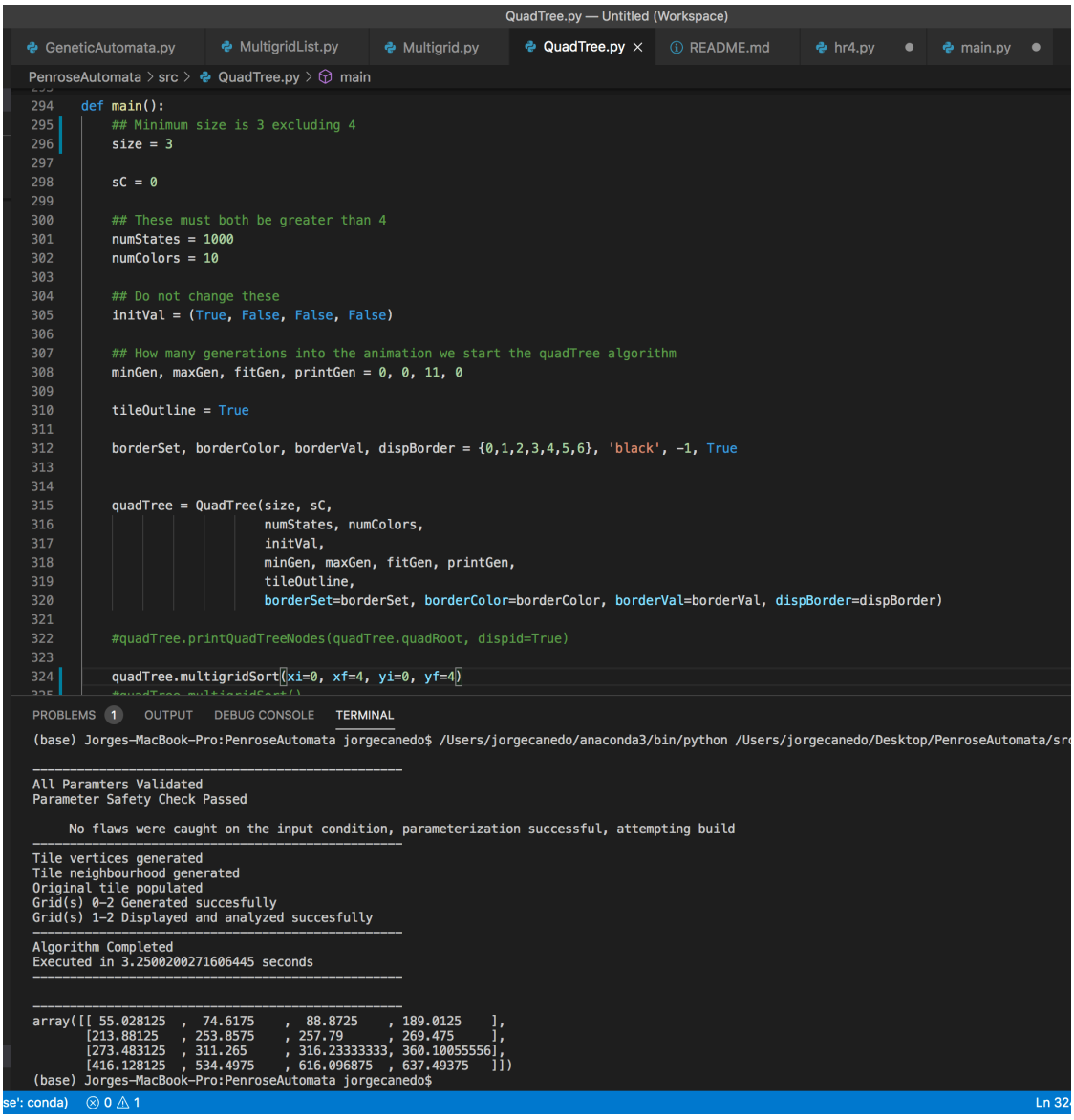
Input non-parameterized multigridSort() one and console output one:



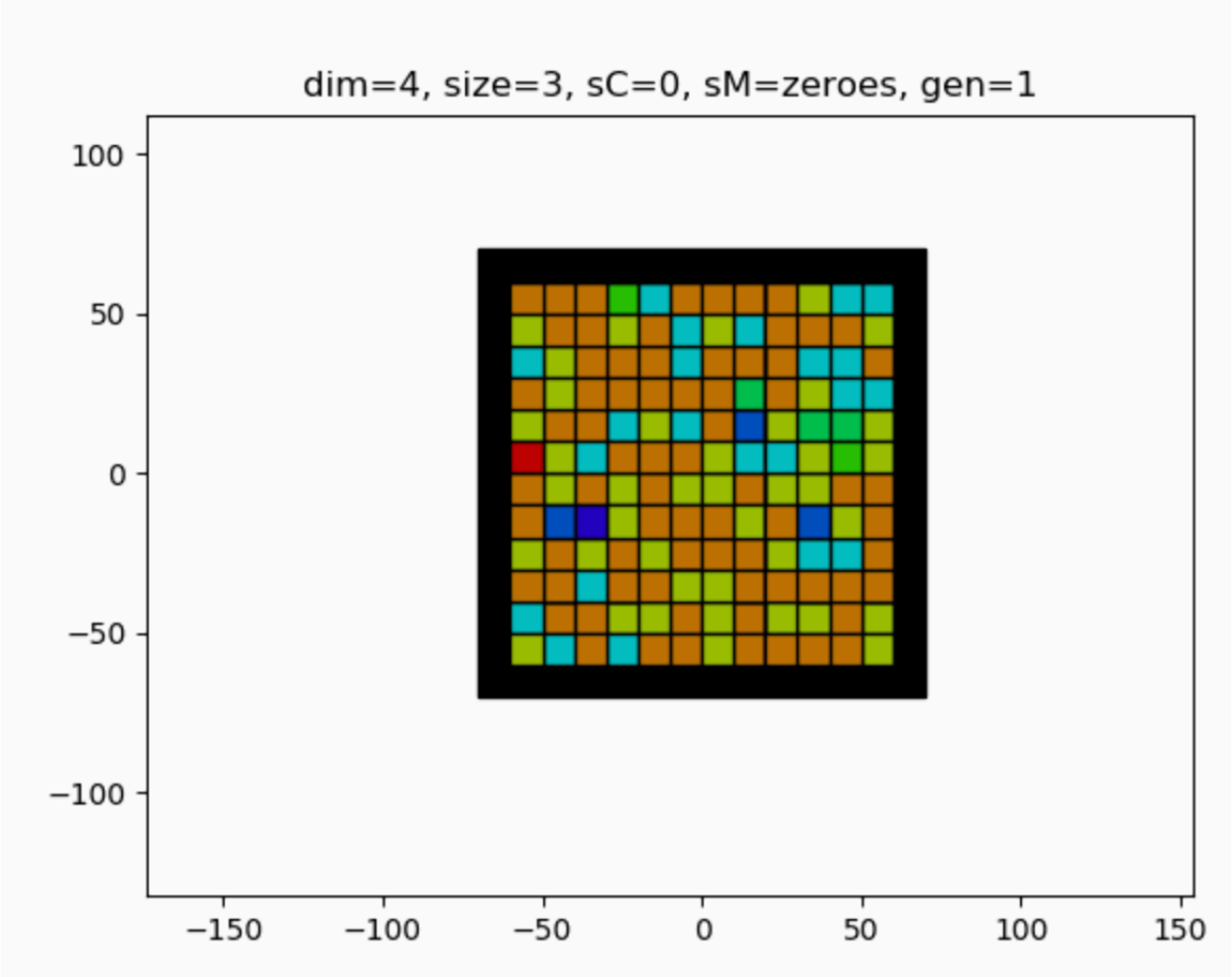
Output displayed tiling one:



Input non-parameterized multigridSort() two and console output two:



Output displayed tiling two:



Req 3.3.2:

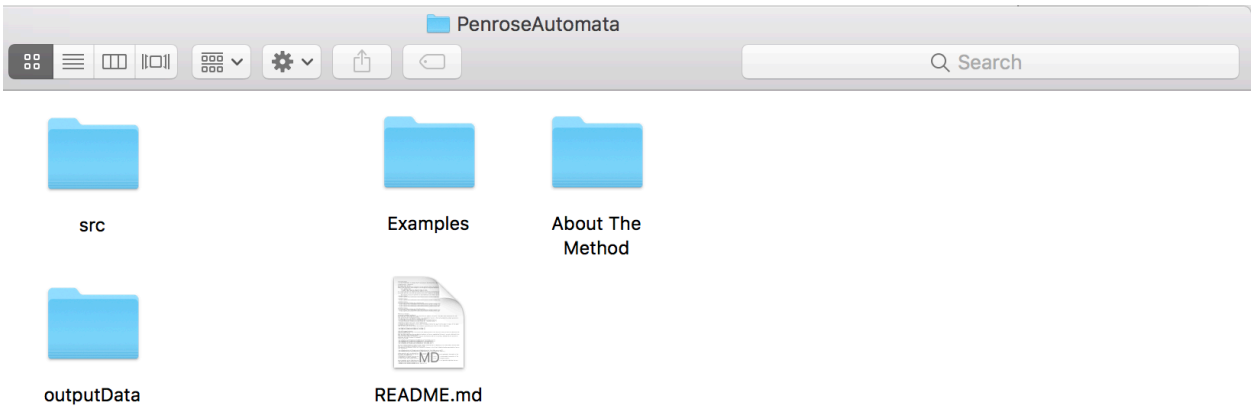
The above displayed tilings one and two are calculated via multithreading and multiprocessing respectively

3.5 YOUR DIRECTORY

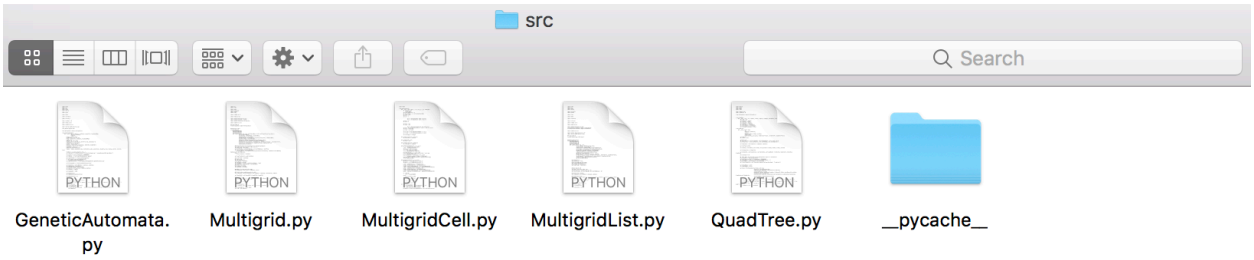
Show a screenshot of your directory, including methods, even if it consists of only one class.

The directory is a tree-like structure where the root folder comprises four sub-folders and a README.

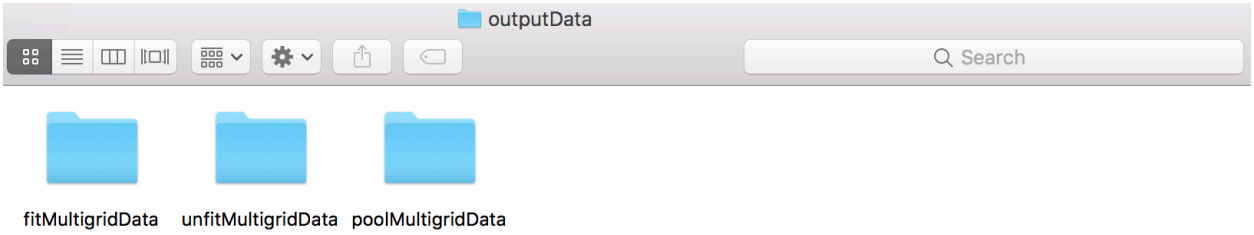
Root folder in the directory ‘PenroseAutomata/‘



The ‘/src/’ sub-directory containing all the executable scripts



The ‘/outputData/‘ sub-directory containing all the output animations/figures/statistics



‘/outputData/poolMultigridData’ comprises ‘fittest.json’ containing the genes of the fittest tilings, ‘/outputData/poolMultigridData/gaStats/’ (containing a figure of the average genetic algorithm population fitness) and folders comprising genetic algorithm outputs which are too nuanced to be discussed here.

Furthermore, ‘/outputData/fitMultigridData’ comprises all the folders of individual animations and their respective statistics where each animation survives to more than fitGen generations

‘/outputData/unfitMultigridData’ comprises all the folders of all remaining animations and their respective statistics

Finally, ’/Examples/‘ and ‘/AboutTheMethod/‘ comprise examples to be used for Github and the mathematical description/ assignment documents (ie assignment photos and submission) respectively.

3.6.1 SPECIALIZED SORTING IMPLEMENTED

Your implementation should include *specialized sorting* in a manner that is useful to your application. Explain where and how you applied these, using the headings below.

The first implementation of this release is a sorting algorithm that is specific to sorting 2D rasterized arrays with lots of repeated elements. This is because, as the automata progresses more generations into the future, it becomes more and more likely for the automata to reach a potential sink, the point at which all tiles on the tiling are in their lowest potential state and unable to react in any way. Furthermore, the sorting method multigridSort() takes in four optional parameters which index a sub matrix of a given larger matrix. One day, it would be fairly easy to sort subsections of the quadTree, ie sorting specific nodes where each node contains a sub matrix.

The grid is to be sorted left to right top to bottom as if reading the rows of the matrix as one would lines in a book would be reading in strictly increasing order

A matrix:	can be sorted into	An output matrix:
[[9, 8, 7]		[[0, 0, 0]
[6, 0, 0]		[0, 0, 6]
[0, 0, 0]]		[7, 8, 9]]

To do this we can create a dictionary of the unique values in the grid along with their respective number of occurrences, this can be done in Theta(n**2) time complexity where n=2^k is the side length of the square sub matrix. We then sort the keys natively by quick sort in O(u**2) time where u is the number of unique elements in the grid. Finally the sorted values are re-placed into the grid in Theta(n**2) time. Overall this algorithm performs sorting a matrix in Theta(n**2)+O(u**2).

One day, it would be fairly easy to sort subsections of the quadTree, ie sorting specific nodes where each node contains a sub matrix.

3.6.2 SORTING AND MULTITHREADING IMPLEMENTED

Your implementation should include *concurrency* in a manner that is useful to your application. Explain where and how you applied these, using the headings below.

Using cProfiler to profile the code, we can see how much time is spent in each of the methods in order to find out which methods are most worth multithreading. We must also add some restrictions to this profiling method, that being that whatever we multithread cannot be sequential. That is, multithreading cannot be applied to an output that builds off of itself, ie I cannot multithread popIter (the method that calculates the state of the grid based off of the previous state) since the next state of the tiling is dependent on the previous state.

However one method does stick out, a particularly costly method. This method is displayTiling() (or the analogous displayBoundaries() which displays the tiling of a boundaried tiling). displayTiling() is an ideal method because it does not build upon previous work and because it is so time consumptive.

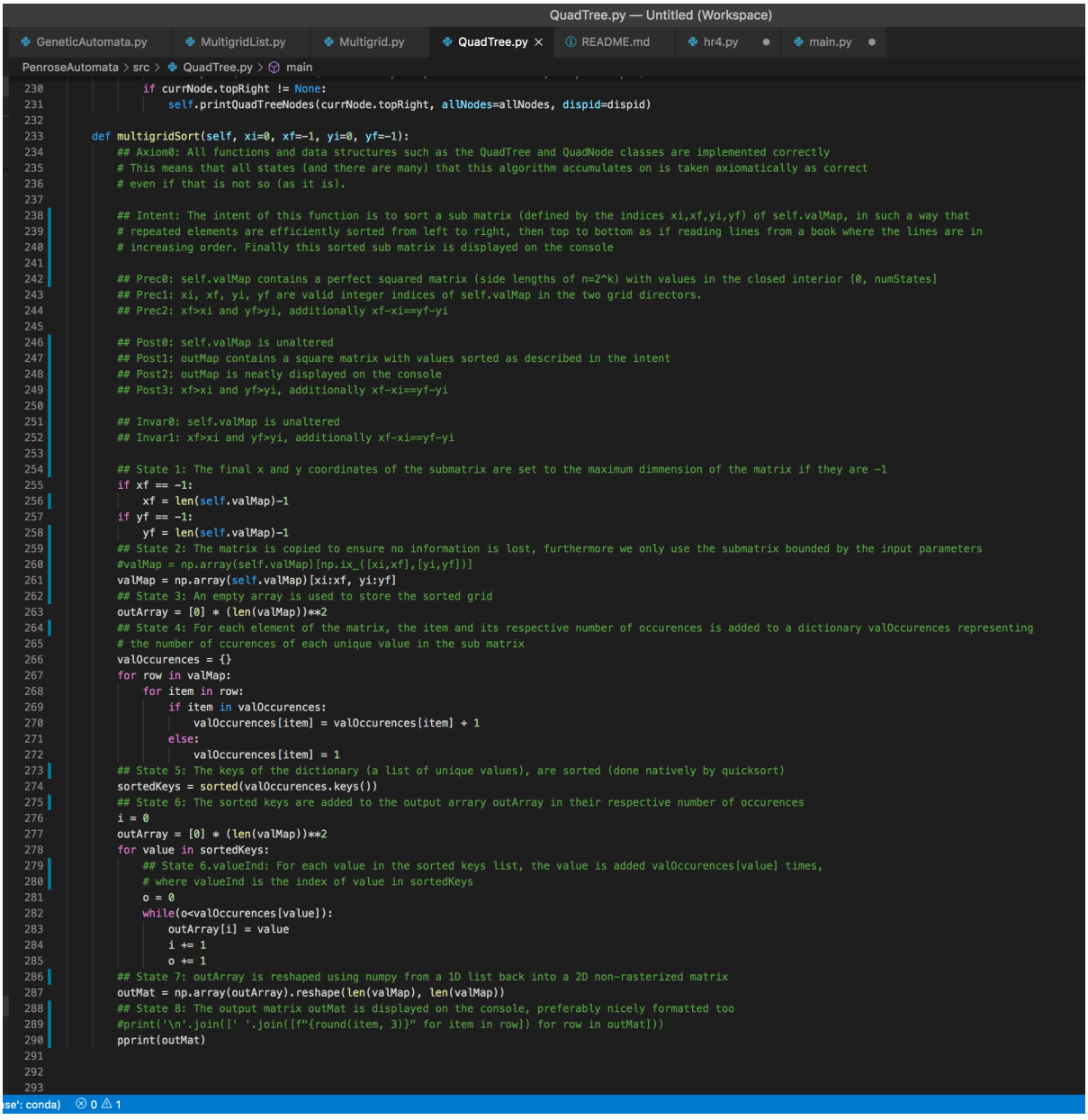
displayTiling() works by iterating over each tiling and creating and properly coloring a patch object (containing the vertices of a polygon) which it adds to a list of patch objects. Ideally, several threads could compile this faster than a single thread, but with Python's Global Interpreter Lock which limits global python interpretation to one per process, Python is unable to take advantage of parallel threading in any way that I have been able to find. After several attempted implementations, two have stuck out. Using multithreading pools and multiprocessing pools. While multiprocessing pools do execute in parallel, the time complexity due to overhead of transferring the pools around is slightly larger than the time complexity of calculating the result, making it ineffective. And while multithreading cannot execute concurrently in python, this approach to multithreading would work in many other languages (ie Java or Cpp), yet multithreading will still execute in the same amount of time as not multithreading so no harm no fowl.

3.6.1 Class model and Sequence Diagram

Identify where you included *sorting* and *multithreading*. To do this use tools (e.g., Visio), PowerPoint, or a combine models as in this example (which you are free to cut and paste from). Insert indications in red (as in this example) to show where the three features below apply.

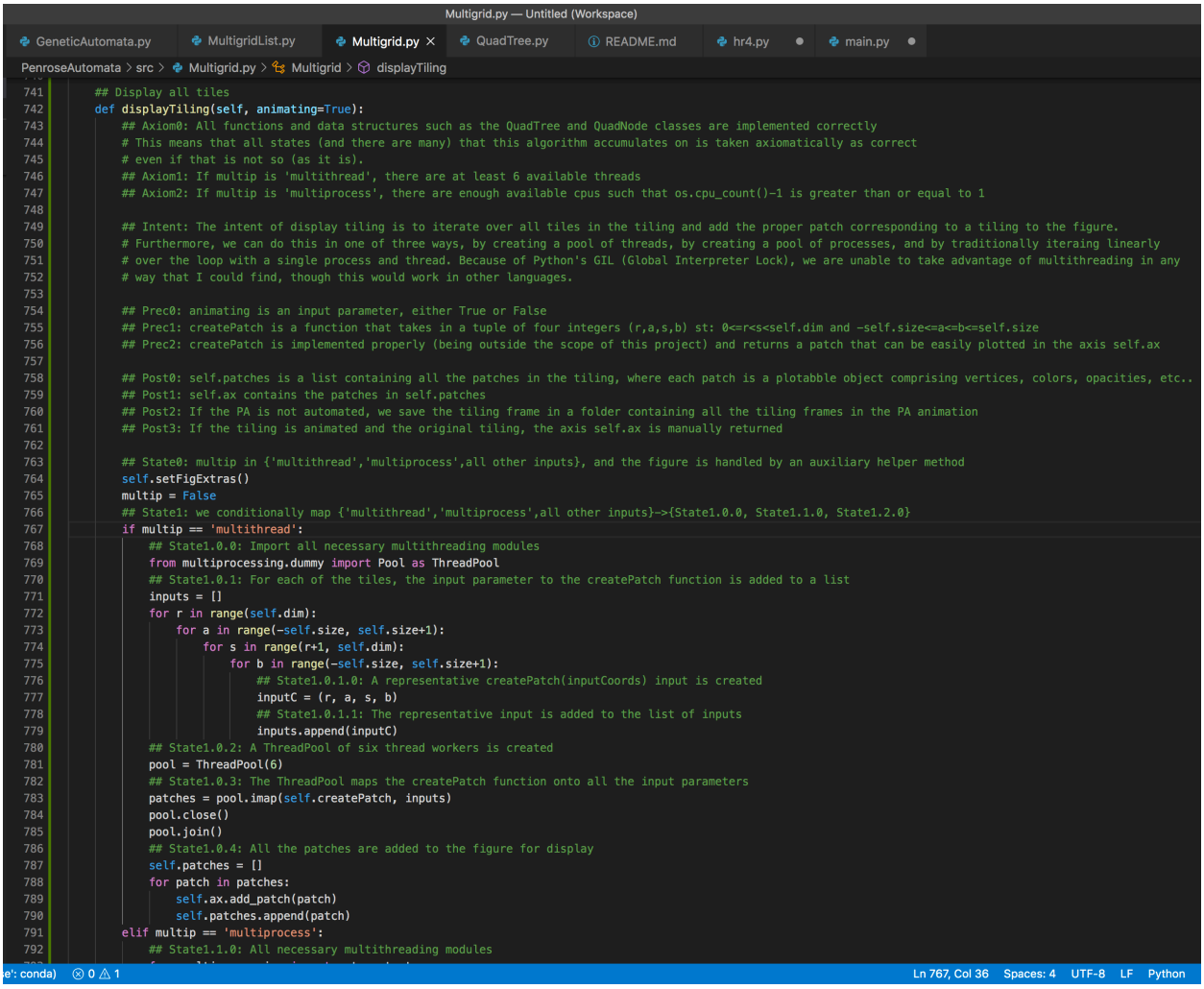
3.6.2 Code showing sorting.

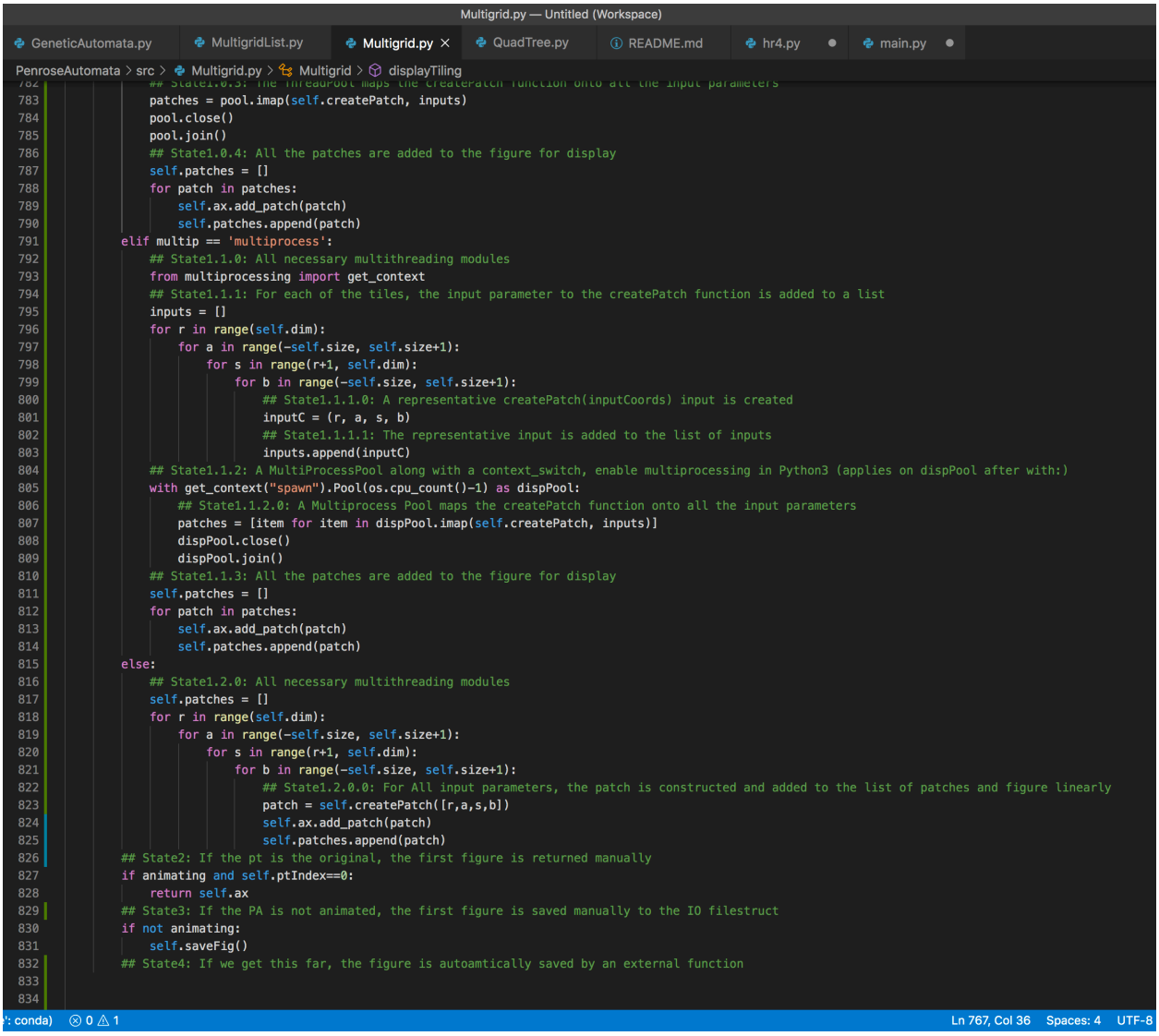
Show the relevant code (only) and explain why specialized sorting is appropriate here. It should be clear where the code is located (class and method).



3.6.3 Code showing multithreading

Show the relevant code (only) and explain why multithreading is appropriate here. It should be clear where the code is located (class and method).





3.7 YOUR CODE

Unless your facilitator requests another method, copy your Eclipse project to your file system, zip it, and attach it. Please contact your facilitator in advance if you want to request an alternative means.

https://github.com/jcanedo279/PenroseAutomata is a repository comprising a master branch and a branch for each assignment, for assignment three, see the branch of assignment three to see the code representative of the images and descriptions above.

3.8 Instructor's Evaluation

Criterion	D	C	B	A	Letter Grade	%
Technical Correctness	No justification of correctness	Technically mostly correct; tested	Implementation correctness well justified; well specified and tested	Implementation correctness thoroughly justified throughout by precise block- and line-specifications and tests.		0.0
Clarity in Presentation	Unclear	Somewhat clear; some comments	Clear with a few exceptions. Well commented.	Entirely clear throughout; very well commented at high and low levels.		0.0
Depth and Thoroughness of Coverage	Shallow or superficial coverage of most topics	Satisfactory depth and thoroughness	Evidence of depth and thoroughness in covering most topics	Evidence of depth and thoroughness in coverring all topics		0.0
				Assignment Grade:		0.0
The resulting grade is the average of these, using A+=97, A=95, A-=90, B+=87, B=85, B-=80 etc. The maximum is 100.						
To obtain an A grade for the course, your weighted average should be >93. A-:>=90. B+:>=87. B:>83. B-:>=80 etc.						

Appendix 3.1 (if needed; should be referenced above, and will be read as-needed only)

Appendix 3.2 (if needed; should be referenced above, and will be read as-needed only)