COMPUTER SCIENCE AND DATA ANALYTICS

Course: CSCI 6511 Artificial Inteligence

Project2: “Tile Placement”

Student(s): Sokrat Bashirov

Instructor: **Dr. Amrinder Arora**

Washington 2023

Link to the project: https://github.com/sokrat00/AI_Project2_Tile_Placement

The problem involves filling a square landscape with square tiles of different colors and shapes, with the goal of covering the landscape as completely as possible, while also meeting certain target requirements for each color.We must first establish the variables, domains, and constraints because the issue is one of constraint satisfaction. The sections of the overall landscape that are divided into tile size X tile size units can be compared to the variables of the problem. One of the available tiles may be picked by each region. The input file must provide the final number of bushes.

CSP representation of the problem:

Variables - tile_size X tile_size regions.
Domains – EL-SHAPE, FULL_BLOCK, OUTER_BOUNDARY.
Constraints – The count of tiles and final count of colors.

The code that was established in github link, includes several classes and functions that work together to solve the problem.

The **Tile** class (tile_types.py) defines the properties of each type of tile, including its shape and the number of tiles of that type available.
The **Tile_input** class (input.py) reads in the input data from a file, including the landscape, the available tiles, and the target requirements for each color. It converts this input data into a usable format for the solver.
The **Landscape** class (landscape.py) represents the current state of the puzzle, including the current arrangement of tiles on the landscape, the available tiles, and the target requirements for each color. It also includes functions for placing tiles on the landscape and checking if the current arrangement meets the target requirements.
The **solver** (backtracking.py) uses a recursive backtracking algorithm to find a solution to the puzzle. It starts by selecting an available tile and attempting to place it in every possible location on the landscape. For each location, it checks if the placement is valid (i.e. it doesn't overlap with any existing tiles and meets the target requirements) and recursively tries to place the next tile. If it reaches a point where no tile can be placed, it backtracks to the previous step and tries a different option.

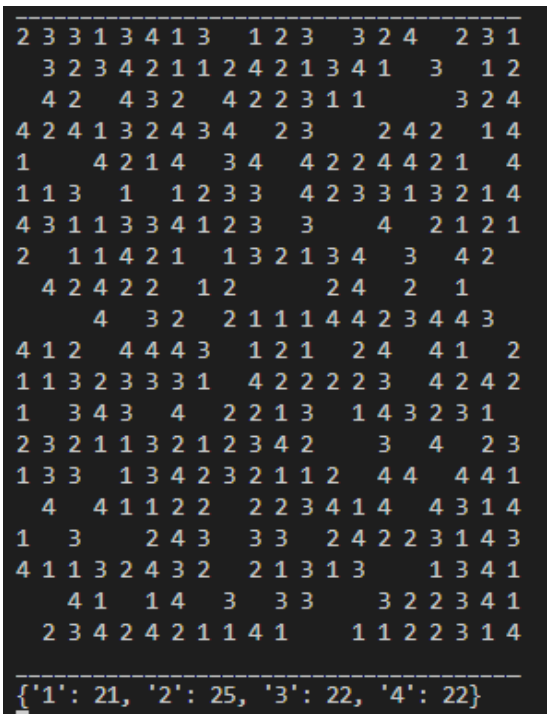
heuristic (landscape.py) – In this function I want to decide which tile to choose next based on the distance
from every color to the final color sets. I get the possible cases after placement of each tile, calculate the distance from current colors to the target colors, and considering every color decide on which tile to choose. The tile that cause most of the color to be minimally distanced from the target is chosen.

Unfortunately, heuristic funtion was not successful on finding the results. In my final version of code, I use the constraint satisfactions of tile counts and final bush count at each step and backtrack if I do not find the result.

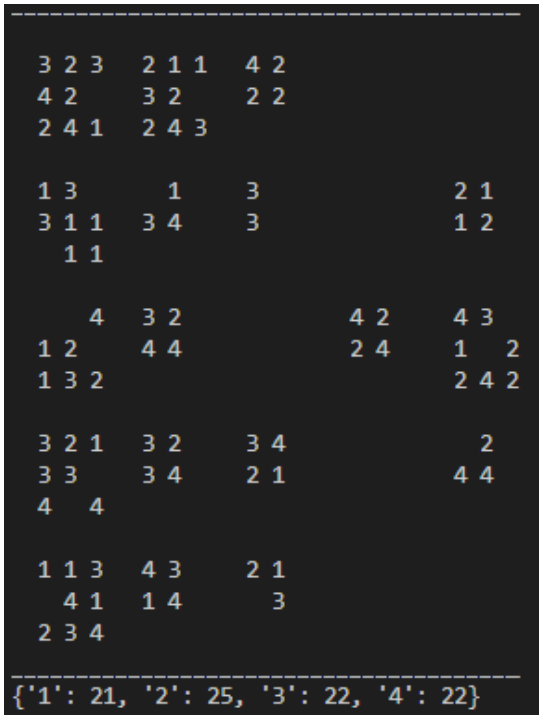
Unit tests.

The correctness of the program at various stages is ensured by the implementation of unit tests. The **test_input.py** file includes tests to make sure the input files are read and parsed correctly. There are tests about landscape operations in the **test_landscape.py** file. It guarantees that the correct color counting is done as well as the placement of various tile kinds on the terrain. Last but not least, the **test_algorithm.py** unit test compares the color count of the obtained result to the color count of its target.
The program outputs the initial version of landscape, its target color counts, the found updated landscape, the current color counts of found landscape and the order in which the tiles are placed on the landscape.

Original landscape:



Final landscape:



Result:

```
# Tiles:
0 4 EL_SHAPE
1 4 EL_SHAPE
2 4 EL_SHAPE
3 4 EL_SHAPE
4 4 EL_SHAPE
5 4 EL_SHAPE
6 4 OUTER_BOUNDARY
7 4 OUTER_BOUNDARY
8 4 OUTER_BOUNDARY
9 4 OUTER_BOUNDARY
10 4 OUTER_BOUNDARY
11 4 OUTER_BOUNDARY
12 4 FULL_BLOCK
13 4 OUTER_BOUNDARY
14 4 OUTER_BOUNDARY
15 4 FULL_BLOCK
16 4 FULL_BLOCK
17 4 OUTER_BOUNDARY
18 4 FULL_BLOCK
19 4 FULL_BLOCK
20 4 FULL_BLOCK
21 4 OUTER_BOUNDARY
22 4 EL_SHAPE
23 4 OUTER_BOUNDARY
24 4 FULL_BLOCK

Done in --- 82.73 seconds ---
```