Shapeshifting Coloring Problems

Model AI Assignments, EAAI 2025

Ashwin Bharadwaj* Anio Zhang† Rajagopal Venkatesaramani‡

Khoury College of Computer Sciences

Northeastern University

In this assignment, you are given a PyGame environment (in the `gridgame.py` file) that renders a randomly initialized $n \times n$ grid, with some cells pre-filled with one of four colors. Your goal is to build an AI agent that solves a coloring problem over this grid (see pages 2 and 3 for constraints), such that no two cells that share an edge have the same color.

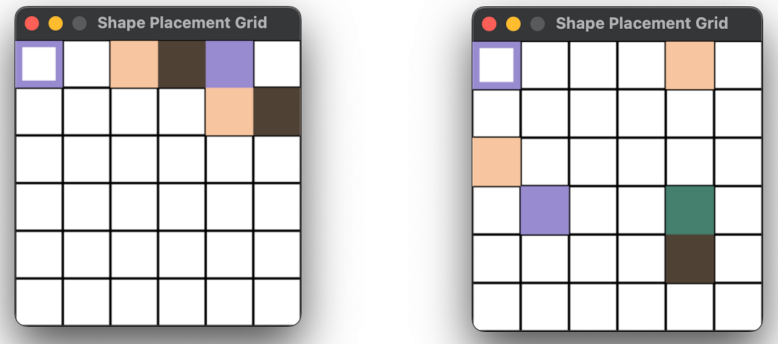
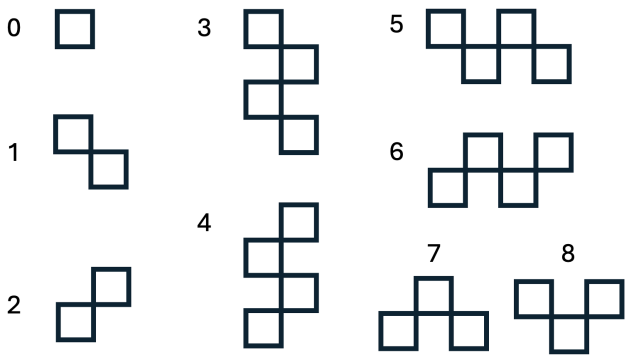


Figure 1: Examples of Initial Configurations

Your agent will attempt to fill the environment by moving a virtual 'brush' over this grid and placing colored shapes, where the shape of the brush can be cycled through the following choices (numbered 0-8):



*bharadwaj.ash@northeastern.edu

†zhang.shengn@northeastern.edu

‡r.venkatesaramani@northeastern.edu

Additionally, each brush can be cycled through one of four colors[1] (numbered 0-3).

0. Indigo 1. Taupe 2. Viridian 3. Peach

Your agent **must** interact with the environment using the `execute()` function called from within the `hw1.py` file, with the following argument options (passed as strings):

- `export`: returns the current state of the grid, a list of shapes with positions and colors currently placed on the grid, and a Boolean indicating whether the coloring constraints have been satisfied.
- `up/down/left/right`: move the brush in the specified direction by one cell. The brush starts in the top left corner of the grid when the program is executed.
- `place`: place a shape on the grid, i.e. color the cells covered by the brush in the currently selected brush color.
- `switchshape`: cycle to the next brush shape option.
- `switchcolor`: cycle to the next brush color option.
- `undo`: undo the last placed shape.

Running the `execute()` function with any argument returns six items:

Variable	Data	Description
`shapePos`	**Brush Position**	Current location of the brush, (X, Y coordinates) `list` of size 2
`currentShapeIndex`	**Current Shape Index**	Index of the currently selected shape, `int`
`currentColorIndex`	**Current Color Index**	Index of the currently selected color, `int`
`grid`	**Grid State**	Updated state of the ($n \times n$ grid), `np.array` (shape $n \times n$)
`placedShapes`	**Placed Shapes**	List of shapes already placed on the grid, `list(int)`
`done`	**Coloring Constraints Satisfied?**	Boolean indicating if coloring constraints are met, `bool`

Table 1: Summary of Data in the Shapeshifting Coloring Environment

[1] If, despite our best efforts, the color choices here pose accessibility concerns, they may be edited in the `gridgame.py` file.

Your Task

Your goal is to build an AI agent that colors this grid using as few colors as possible, such that no two adjacent cells share the same color. Further, your goal is to achieve this coloring using as few shapes and colors as possible - a larger brush may cover multiple cells, but counts as one shape. You may use any of the concepts we have discussed in class so far to implement your agent, but we highly recommend using a local search approach such as hill climbing, simulated annealing or genetic algorithms to get the most out of this assignment. Refer to `hw1.py` for implementation-specific instructions.

Recall that a local search may not always converge to the 'optimal' coloring; so this assignment is graded on correctness, rather than optimality. Any implementation which follows all the rules specified below and on average, leads to a valid coloring of the grid within the autograder's time limits will receive full points.

Environment Rules: Adjacent cells are defined as cells that share an edge between them (i.e., diagonally neighboring cells may share the same color, since they only share a vertex). If a brush partially or fully overlaps with an area of the grid that is already colored, the `execute` function with the `place` argument will fail, i.e. the colors in those cells will not be overwritten.

The environment is built as a Python class. You are only allowed to interact with the environment with the .execute() method. Do not use any other method from the class to modify any variables in the class. Moreover, you can use some of the utility functions to help you write your solution. Feel free to change the arguments for the constructor to make the exercise harder or easier.

Assignment Rules: Any hardcoded solutions, or attempts to leverage the autograder's design to maximize points scored will result in an automatic zero on the assignment. Your agent **must** only use the `execute()` function to interact with the environment using its different arguments, and use its returned values to implement your objective function, or any validity check you may wish to use. Do not directly manipulate the `gridgame` variables - doing so will result in an automatic zero. Treating the environment as a black box (even when you know its internals) is a very important concept, and will serve you well in future assignments, as well as in your AI careers. Implementations must be optimized for good runtime - Gradescope has an autograder timeout of 10 minutes. Any submissions that do not finish executing in 10 minutes will be treated as incomplete, and will be evaluated for partial credit based on correctness.

Submission: Upload your completed `hw1.py` file, and the provided `gridgame.py` file to Gradescope, and ensure the autograder testcases run as intended with no errors. Please make sure you do not rename the files, or zip a directory containing the files; upload the two files directly to Gradescope instead.

Beyond the Assignment - Broader Applications

Graph coloring is a versatile problem in graph theory with numerous practical applications across various fields. Here are some example problems from the real world that are mathematically equivalent to graph coloring.

1. Scheduling

- In exam scheduling, vertices may represent exams and edges connect exams with common students. Colors represent time slots. A proper coloring ensures no student has conflicting exams. Ideas similar to the different brush types in this assignment may be used to optimize room usage, time between exams, etc.
- Graph coloring can be used to schedule classes, ensuring no conflicts in room assignments or student schedules. Faculty room assignments for teaching courses often follow a similar process.
- Graph coloring can optimize aircraft maintenance schedules, minimizing downtime and maximizing efficiency.

2. Resource Allocation

- In radio and mobile networks, graph coloring can be used to assign specific frequencies to a set of transmitters, while the objective function tries to minimize interference. Vertices represent transmitters, edges may represent potential interference as a result of a given frequency allocation, and colors represent frequencies.
- Compilers use graph coloring to allocate registers efficiently. Variables form vertices, edges connect variables that are simultaneously active. Colors represent registers.
- Sudoku puzzles are a resource allocation problem that can be modeled as graph coloring problems. Each cell is a vertex, with edges connecting cells that cannot have the same number. Colors represent the numbers 1-9.

3. Network Analysis

- In various network scenarios, graph coloring can be used to identify and resolve conflicts, such as in wireless sensor networks or traffic management systems.
- Graph coloring may be used in cybersecurity and network security applications.