CS4100
artificial intelligence

Khoury College, Northeastern University

Programming Assignment 1

Instructions

- If you discuss this problem set with one or more classmates, all parties must declare collaborators in their individual submissions within a code comment. Such discussions must be kept at a conceptual level, and no sharing of actual code is permitted.

- You may use any generative AI tool available to you, as long as it is appropriately cited in a code comment. I recommend using AI tools for debugging or conceptual understanding rather than to produce actual functions.

Deadlines

- Submissions should be uploaded to Gradescope by 6:00 PM on **Sep 28**.

- Gradescope will show a 'late' deadline of **Oct 1**. This is intended solely for any students who may wish to invoke the freebie, outlined in the course policies.

- Any submissions received after 6:00 PM on **Sep 28** will be considered late, and will automatically invoke the use of your freebie. If you have used your freebie on a previous assignment, your submission will not be accepted for credit.

- Regrade requests must be submitted on Gradescope within 1 week of receiving your grade, after which no further requests will be entertained.

Reach Out!

If at any point you feel stuck with the assignment, please reach out to the TAs or the instructor, and do so early on! This lets us guide you in the right direction in a timely fashion and will help you make the most of your assignment.

Shapeshifting Coloring Problems

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 next page for constraints), such that no two cells that share an edge have the same color.
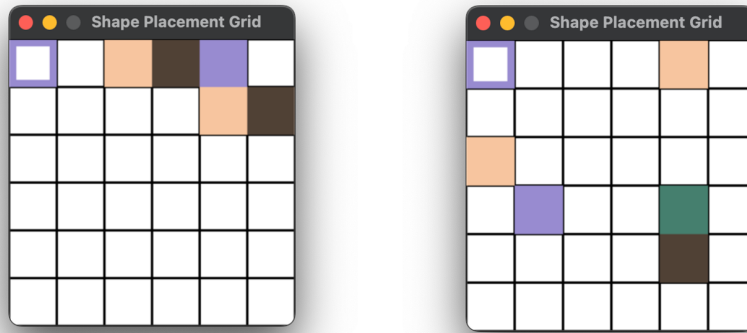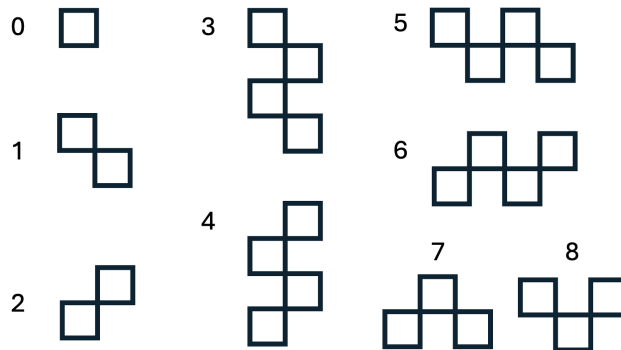

Figure 1: Examples of Initial Configurations

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):

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

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

---

[1]If, despite our best efforts, the color choices here pose accessibility concerns, they may be edited in the `gridgame.py` file. Alternatively, please reach out to the instructors/TAs, and we will be happy to implement a different palette for you.

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

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.

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: Submit a zip file containing your completed `hw1.py` file, the provided `gridgame.py` file, and the `grid.txt, shapes.txt` and `time.txt` files produced in your working directory upon executing the homework script. Upload this zip file to Gradescope, and ensure the autograder runs as intended.