02-201 / 02-601 Homework 7: Sandpiles

For 02-601 students: Due: 11:59pm on Tuesday, November 4
For 02-201 students: Extra Credit Due: 11:59pm on Tuesday, November 4 (no extensions for extra credit)

1. Set up

The set up is the basically the same as for homeworks 4 and 5.

1. Create a directory called "`go`" someplace (different than where you have installed Go) [If you've already done this, you don't have to do it again.]
2. Inside of that directory create a directory called `src` [If you've already done this for a previous assignment, you don't have to do it again.]
3. Inside of the `src` directory, create a directory called `sandpile`.
4. Download the template `canvas.go` from BlackBoard (or copy it from a previous assignment) and put it into the `sandpile` directory.
5. Set your GOPATH environment variable to the location of your `go` directory that you made above. On a Mac:

   ```
   export GOPATH=/Users/carlk/Desktop/go
   ```

   where you replace the directory name after the `=` with the location of the `go` directory you just made.

   On Windows use

   ```
   set GOPATH=C:\Users\carlk\Desktop\go
   ```

2. Assignment

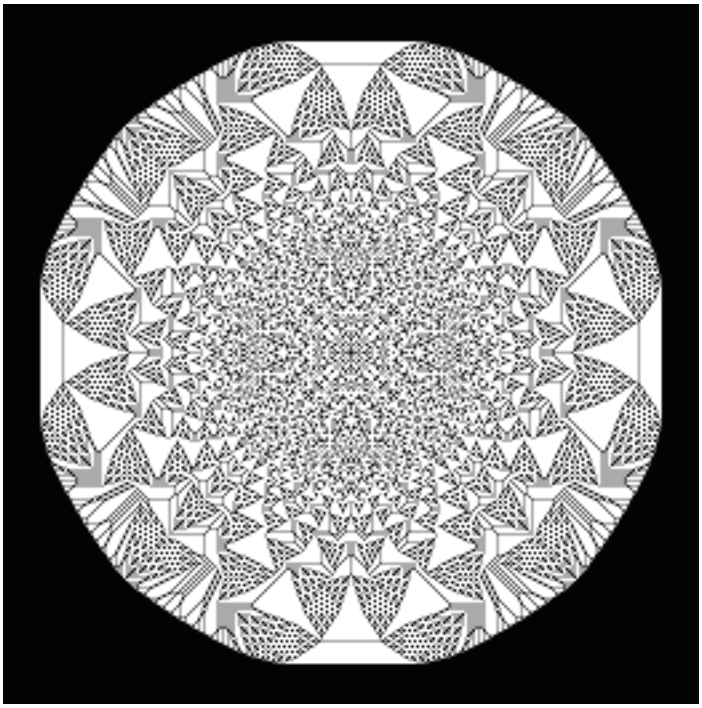
2.1 Sandpiles

Imagine an infinite 2-D checkerboard on which you place piles of coins of various heights on some of the squares (at most 1 pile per square). Consider the following toppling operation:

> topple(r, c): if square (r, c) has ≥ 4 coins on it, move 1 coin from (r, c) to each of the 4 neighbors of (r, c) (diagonal neighbors don't count, only north, south, east, and west). If square (r, c) has < 4 coins, do nothing.

A configuration of coins is said to be *stable* if all squares have < 4 coins on them. If we repeatedly topple until we can't topple any more, we'll end up at a stable configuration. Somewhat surprisingly, you can show than the order of the topples that you do won't affect the final stable configuration that you end up in, given a particular starting point.

For example, if you start with a pile of 10000 coins on a single square, and no coins elsewhere, you will end up with the following configuration:



where the color indicates the number of coins (0=black, 3=white, and 1 and 2 are intermediate shades of gray).

You can read more about these here:

`http://www.cmu.edu/homepage/computing/2014/fall/lifes-a-beach.shtml`

2.2 What you should do

Write a program that can be run with the following command line:

```
sandpile SIZE PILE
```

where `SIZE` and `PILE` are both positive integers. `SIZE` gives the size of checkerboard which will be `SIZE` $\times$ `SIZE`. `PILE` gives the number of coins that are to be placed on the middle square at position ($\lfloor$`SIZE`$/2\rfloor, \lfloor$`SIZE`$/2\rfloor$) at the start.

This program should find the stable configuration associated with the given initial configuration. It should then draw the final board in a PNG file called `board.png`. The colors corresponding to each number of coins should be, given in (Red, Green, Blue) values:

0	(0,0,0)
1	(85,85,85)
2	(170, 170, 170)
3	(255, 255, 255)

Each board square should be drawn as a 1×1 square.

Your program must create a new type called `Board`, with the following methods:

- `Topple(r, c int)` that topples (r, c) until it can't be toppled any more.

- `Contains(r,c int) bool` that returns true if (r, c) is within the field.
- `Set(r,c, value int)` that sets the value of cell (r, c).
- `Cell(r, c int) int` that returns the value of the cell (r, c).
- `IsConverged() bool` that returns true if there are no cells with ≥ 4 coins on them.
- `NumRows() int` that returns the number of rows on the board.
- `NumCols() int` that returns the number of columns on the board.

Speed. Your program should be fast enough to run `./sandpile 200 10000` in at most a few seconds. It should be able to run `./sandpile 200 100000` in about 15 minutes (give or take a factor of 2 depending on your computer speed).

2.3 Tips on how to start

First, write the code for the `Board` functions.

Then, write the code to parse the command line, and a function `CreateBoard` that returns a new board with the right dimensions and the initial configuration.

Next, write `ComputeSteadyState(b *Board) int` that topples squares until the board has converged to a stable configuration.

Finally, write a `DrawBoard` function that draws the board to a PNG.

Between each of those steps, you should make sure your program complies.

3. Learning outcomes

After completing this homework, you should have

- gotten more experience with an "object-oriented" way of thinking
- learned about sandpiles
- worked on making a program faster if needed