1. Set up

1. Inside of your src directory, create a directory called sandpile.
2. Copy the canvas.go file from HW4 and the gifhelper.go file from HW5 into the sandpile directory. Also make sure that you still have the code.google.com and gogif directories in your src directory.
3. Set your GOPATH environment variable to the location of your go directory that you made above. On a Mac, use the command

   ```bash
   export GOPATH=/Users/pcompeau/Desktop/go
   ```

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

   On Windows, use the command

   ```bash
   set GOPATH=C:\Users\pcompeau\Desktop\go
   ```

2. Assignment

2.1 Sandpiles

Imagine a huge 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:

\[
\text{topple}(r, c): \text{ if square } (r, c) \text{ has } \geq 4 \text{ coins on it, move 1 coin from } (r, c) \text{ to each of the 4 neighbors of } (r, c) \text{ (diagonal neighbors don’t count, only north, south, east, and west).}
\]

If square \((r, c)\) has < 4 coins, do nothing.

(Important Note: if \((r, c)\) lies on the boundary of the board, model the topple operation as if the coins fall off the edge of the board. In other words, you do not have to keep track of the coins that fall off, but you should remove four coins from \((r, c)\) even if \((r, c)\) lies on the boundary of the board. This is important, because if you don’t account for this assumption, there will be some inputs that cause your program to run forever.)

A configuration of coins is said to be stable if all squares have < 4 coins on them (i.e., there are no more topple operations to perform). If we repeatedly topple until we can’t topple any more, then we will eventually end up at a stable configuration. There is a (surprising!) theorem stating that for a given initial configuration, the order of the topples that you perform won’t affect the stable configuration that you reach.

For example, say that you start with a pile of 1000 coins on a 200 \times 200 board. Then you will obtain the following stable board (zoom in to see the individual pixels):
If you instead start with 100,000 coins on a single square, and no coins elsewhere, then you will end up with the following configuration for a significantly large board:

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 sandpiles 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 \( \text{SIZE} \times \text{SIZE} \). PILE gives the number of coins that are to be placed on the middle square at position \( (\lfloor \text{SIZE}/2 \rfloor, \lfloor \text{SIZE}/2 \rfloor) \) at the start.
This program should find the stable configuration associated with the given initial configuration. It should then create an animated GIF where each frame of the GIF represents one iteration of the board, and 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 single pixel (i.e., a 1 × 1 square).

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

- **Topple(r, c int)**: topples `(r, c)` until it can’t be toppled any more.
- **Contains(r, c int) bool**: returns `true` if `(r, c)` is within the field and `false` otherwise.
- **Set(r, c, value int)**: sets the value of cell `(r, c)`.
- **Cell(r, c int) int**: returns the value of the cell `(r, c)`.
- **IsConverged() bool**: returns `true` if there are no cells with ≥ 4 coins on them and `false` otherwise.
- **NumRows() int**: returns the number of rows on the board.
- **NumCols() int**: 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 2000 100000` in at most 15 minutes (give or take a factor of 2 depending on your computer). However, please keep the following quotation in mind:

> “About 97% of the time: premature optimization is the root of all evil.” — D. Knuth

In other words, it is not a bad idea to start with a working program that works on smaller inputs and then work on ways to speed it up on larger inputs. To test the speed of a function, you may find the benchmarking provided by the `testing` package useful (see [http://tinyurl.com/pcacykp](http://tinyurl.com/pcacykp) for details).

### 3. Learning outcomes

After completing this assignment, you should have

- earned more experience with an “object-oriented” way of thinking
- learned about sandpiles
- worked on optimizing a program faster if needed