Homework 5: Spatial Games
Programming for Scientists
Submit via Autolab

1. Reading

Read Ch. 8 and Ch. 9 of *An Introduction to Programming in Go* (on pointers and structs).

2. Set up

The set up is the basically the same as for the drawing assignment.

1. Download the template from Piazza, and unzip the `spatial` folder into the `go/src` directory.
2. Add your `gifhelper.go` and `canvas.go` files into the `spatial` directory.
3. Make sure that you already have the `code.google.com` and `gogif` directories under `src`.
4. At the command line, navigate to the location of your `spatial` directory.
5. In this assignment, you will edit the `spatial.go` file provided within this directory. Note that you also have `canvas.go` to facilitate drawing and `gifhelper.go` to generate an animated GIF.

3. Reading data from files

3.1 Opening and closing files

If you want to read the data from a file you must "open" it so that it is available for use. To do that in Go, you use the `os.Open` function, which returns a variable that represents the file and whether there was an error. For example:

```
var filename = "field.txt"
file, err := os.Open(filename)
if err != nil {
    fmt.Println("Error: something went wrong opening the file.")
    fmt.Println("Probably you gave the wrong filename.")
}
```

To do this, you must `import "os"`.

Once you are done with a file, you should close it using the `Close()` function, which is called using the syntax:

```
file.Close()
```

if your file variable was named `file`.

3.2 Reading data from files

Once you have a file open, there are many ways to read data from it. We will see the most common, which is to use something called a `Scanner`. A scanner reads through the file, line by line.

1. You create a `Scanner` using the `bufio.NewScanner` function:

```
scanner := bufio.NewScanner(file)
```

where `file` is a file variable that you have opened (**not** a filename). To do this, you must `import "bufio"`

2. Now you can loop through the lines of a file using a **for** loop of the following form:

```
for scanner.Scan() {
    fmt.Println("The current line is:", scanner.Text())
}
```

The `scanner.Scan()` function tries to read the next line and returns **false** if it could not. Inside the **for** loop, you can get the current line as a string using `scanner.Text()` as above.

3. Once you're done reading the file, it's good practice to check to see if there was an error during the file reading. You do this by checking whether `scanner.Err()` returns something that isn't **nil**:

```
if scanner.Err() != nil {
    fmt.Println("Error: there was a problem reading the file"
    os.Exit(1)
}
```

3.3 Another example reading lines

This code reads the lines in a file and puts them into a slice of strings.

```
func readFile(filename string) []string {
    // open the file and make sure all went well
    in, err := os.Open(filename)
    if err != nil {
        fmt.Println("Error: couldn't open the file")
        os.Exit(1)
    }

    // create the variable to hold the lines
    var lines []string = make([]string, 0)

    // for every line in the file
    scanner := bufio.NewScanner(in)
    for scanner.Scan() {
        // append it to the lines slice
        lines = append(lines, scanner.Text())
    }
```

```
        // check that all went ok
        if scanner.Err() != nil {
            fmt.Println("Sorry: there was some kind of error during the file reading")
            os.Exit(1)
        }

        // close the file and return the lines
        in.Close()
        return lines
    }
```

3.4 Parsing a string that contains data

The code above to read a file reads a file line by line, and each line is a **string**. Often you may have several data items encoded on the same line. For example, suppose the first line of your file contains the width, height, and length of a cube:

```
10.7 30.2 15.8
```

We would like to extract these 3 floating point data items from the line. Again there are several ways to do this. We'll see two.

The First Method: Using Split. The first uses `strings.Split` function, which takes a single string, plus a string that says what substring separates the item. For example, if your string contains

```
var line string = "10.7,30.2,15.8"
```

You could split it into 3 strings using:

```
var items []string = strings.Split(line, ",")
```

Now, `items` will contain the 3 data items:

```
items[0] == "10.7"
items[1] == "30.2"
items[2] == "15.8"
```

The things in `items` are still strings. They are now in a format similar to what you have seen with `os.Args`. You must convert them to **float64**, **int**, etc. as appropriate.

The Second Method: Using Sscanf. The second method works if you have a small number of data items on the line. It uses a function with a strange name: `fmt.Sscanf`. This stands for "scan" a "S"tring using a "f" format. You use it as follows:

```
var line string = "10.7   30.2   15.8"
var f1, f2, f3 float64
fmt.Sscanf(line, "%v %v %v", &f1, &f2, &f3)
```

This call uses some strange syntax. We can break it down:

- `line` is the string that contains the data you want to parse.
- `"%v %v %v"` is the format string that says how the data in the first parameter is formatted. Each item `%v` means "there will be a some data item here", so this format string means that there will be 3 pieces of data separated by spaces. `Sscanf` will figure out what type of data the value is based on the next parameters (described below).
- the `&f1, &f2, &f3` parameters say where to store each of the floating point numbers in the format string. This `&` syntax is new. Its purpose here is to allow the function `Sscanf` to change the values of the `f1,f2,f3` variables. We will see this more in the future. Since `f1,f2,f3` are **float64**s, `Sscanf` will read floats for each of them.

The format string can be very complex and can parse things besides floats. Some examples:

1. This will read two integers separated by a comma followed by a float separated by spaces:

   ```
   var c, d int
   var f float64
   var line string = "101,31 2.7"
   fmt.Sscanf(line, "%v,%v %v", &c, &d, &f)
   ```

2. This will scan two string separated by spaces:

   ```
   line := "Dave Susan"
   var name1, name2 string
   fmt.Sscanf(line, "%v %v", &name1, &name2)
   ```

The nice thing about the `Sscanf` method is that it handles both the parsing and conversion for you. The downside is that the number of data items must be small and known ahead of time.

`Sscanf` is part of a family of functions that read and print data: `fmt.Scanf`, `fmt.Sprintf`, `fmt.Printf` and others that all work the same way. You can read more about them here: http://golang.org/pkg/fmt/. This is the reason for the name `fmt` that we've seen for a long time now: most of the functions in this package do "formatted" input and output in the style of `Sscanf`.

4. Assignment

4.1 The Prisoner's Dilemma

The Prisoner's Dilemma refers to the following situation: Two criminals who committed a crime together are being interrogated by the police in separate rooms. Each prisoner has a choice to either *cooperate* with their partner in crime and stay silent, or to be a *defector* and rat on their partner to gain favor with the police. The outcome of the interrogation depends on the choices made by the two prisoners:

- If both prisoners cooperate with each other and stay silent, then each prisoner only goes to prison for a short time (because there is little evidence against them).
- If both prisoners defect (i.e., rat on each other), then they both get long sentences.
- If one prisoner stays silent in an attempt to cooperate, but the other defects, then the one who testifies gets a big reward: immunity from prosecution. On the other hand, the person who stayed silent gets a long sentence.

We can model these outcomes in the following way: Each prisoner either adopts a strategy of cooperation ("C") with their partner or defection ("D") from their partner.

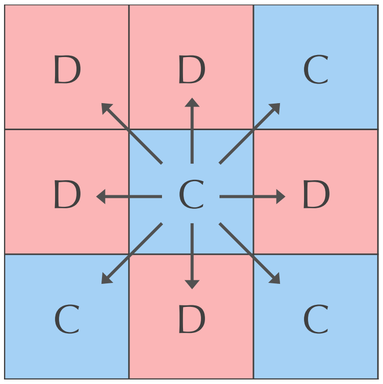
Prisoner 1	**Prisoner 2**	**Outcome**
C	C	both get 1 point
D	D	both get 0 points
D	C	prisoner 1 gets b points and prisoner 2 gets 0 points
C	D	prisoner 1 gets 0 points and prisoner 2 gets b points

That is: 0 points for a long sentence, 1 point for a short sentence, and b points for immunity. Here $b > 1$ is a parameter that says how much reward a prisoner gets for being the only defector.[1]
z The Prisoner's Dilemma is widely studied as a model for real policy situations. Pollution is a good example: if no one pollutes (everyone adopts a "C" strategy), everyone does well. If everyone pollutes (everyone adopts a "D" strategy), we all lose. But if you live in the only country that pollutes (the only "D" country), then you get the economic benefit without the global cost (a big reward for you).

4.2 Spatial Games

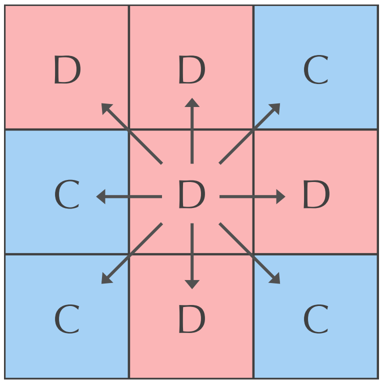
This assignment adds a twist: we assume that there are n^2 prisoners arranged in cells in a prison in an n-by-n grid. Every prisoner is either a C-prisoner (cooperator with their fellow prisoners) or a D-prisoner (defector who snitches to the police). The D-prisoners testify against the prisoners in the cells that neighbor theirs, and the C-prisoners always stay silent. The reward to a prisoner is the sum of the points according to the scheme above. For example, consider this C-cell and its neighbors:

[1] In practice, the outcomes should be negative (a prison sentence is rarely seen as a reward), but we have positive outcomes here in order to simplify the problem.



The center cell plays 8 prisoner dilemma games, and gets a total of 3 points: 0 points for interacting with their D neighbors, and 3 points for interacting with their C neighbors.[2]

Another example:



The center cell gets $4b$ points: 0 points for interacting with each of its D neighbors, and b points for interacting with each of its C neighbors.

After calculating the score, each prisoner gets to change strategies: it adopts the strategy (C or D) of the prisoner in its neighboring cells, *including itself*, that scored the most points. (Ties may be broken arbitrarily.) After each round, the points are reset to 0.

In summary, the overall procedure is:

1. prisoners are either C or D prisoners.
2. prisoners get points according to their neighbors using the point system in the table above.
3. after all the scores have been calculated, the prisoners adopt new strategies (either C or D) according to the best strategy in their neighborhood (which includes themselves) in the previous round.
4. this procedure is repeated for a given number of steps.

These "spatial games" were first explored in:

Nowak and May, Evolutionary games and spatial chaos, *Nature* **359**:826–829 (1992).

[2]The original paper on these spatial games includes interactions with oneself. This doesn't affect the qualitative behavior of the system but is less intuitive.

4.3 What to do

In this assignment, you will write a program to simulate this group of prisoners arranged in a grid. You will write a program that can be run with the following command line:

```
./spatial FILENAME b STEPS
```

where FILENAME gives the file that contains the initial assignments of C or D strategies (in the format described below), STEPS is an integer that indicates how many steps you should run your program for, and b is the reward that a D cell gets against a C cell as described above.

The main tasks are to write functions to (a) read in the initial field, and (b) update the scores and strategies according to the rules above. As a result, you should produce two things:

1. A PNG image ("Prisoners.png") corresponding to the final version of the board after `STEPS` steps.
2. An animated GIF displaying all `STEPS` steps of running the program.

Important Note: The Autolab autograder has not yet been updated to include the animated GIF component, so please comment out the code rendering the GIF when submitting to Autolab.

4.4 How to draw an animated GIF

We can generate a single image from a canvas object `c` by calling the field `c.img`, which is an `image.Image` type. By placing each image generated from a series of Prisoner's Dilemma boards into a `[]image.Image` slice, we can then generate an animated GIF.

In the code template, you will find a file called `gifhelper.go`. This file contains the function `Process` that takes a slice `[]image.Image` and a file name (such as `"prisoners"`) as a string, and generates a GIF with this file name from the image slice. (It needs the functions in the `gogif` folder in order to run.)

4.5 Format of the initial field file

The FILENAME command line parameter will be the name of a file that contains the field dimensions and the initial type for each of the cells. The first line of the file will contain the number of rows and columns, e.g.:

```
10 15
```

means there are 10 rows, each having 15 columns.

Each subsequent line is a string of Cs and Ds of length equal to the number of columns. There will be a line for each row. Together, these rows give the initial strategies for each cell. For example:

```
CCCCCCCCCCCCCCC
CCCCCCCCCCCDCCC
CCCCCCCCCCCCCCC
CCCCDCCCDDCCCCC
CCCCCCCCDDCCCCC
```

```
CCCCCCCCCCCCCCCC
CCCCCCCCCCDCCCCC
CCCCDCCCCCCCCCCC
CCCCCDCCCCDCCCCC
CCCCCDCCCCCCCCCC
```

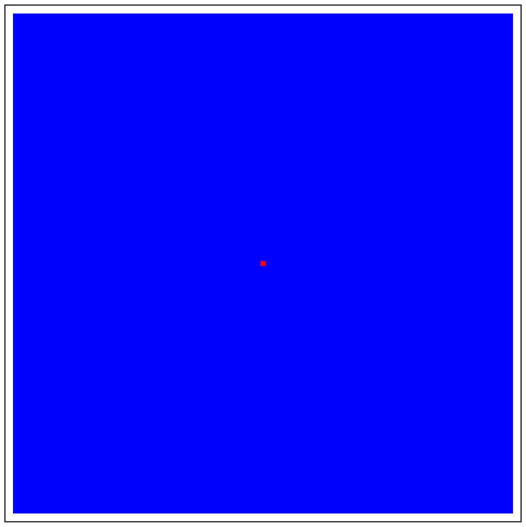
The assignment contains several example files: `f99.txt`, `f100.txt`, `rand200-10.txt`, `rand200-50.txt`, and `smallfield.txt`.

4.6 Tips on how to start

First, install the template from Piazza. Before you write any code to make the field evolve, compile and test your program by reading the starting field `f100.txt` using 0 steps of evolution using the command:

```
./spatial f99.txt 1.65 0
```

This will ensure that you are reading the field and writing the picture correctly. The field should look like this:

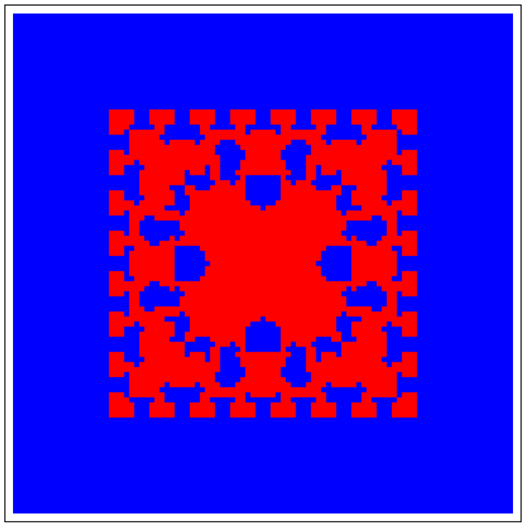


before you do any evolution steps.

Next, write functions for the field to evolve. Now test your program for more steps. If you run the command

```
./spatial f99.txt 1.65 30
```

You should get the picture:

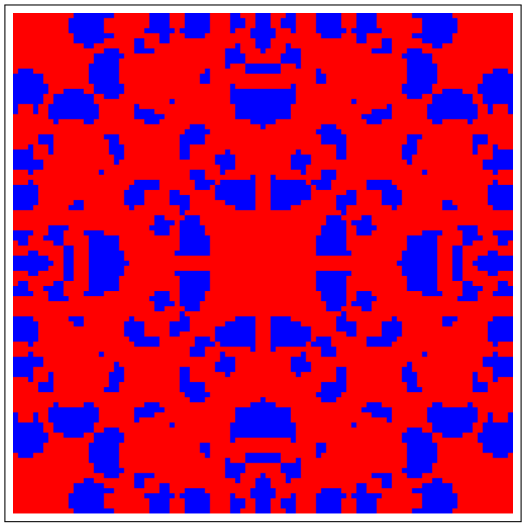


4.7 Explore your program

1. If you execute the command:

```
./spatial f99.txt 1.65 80
```

You should get a picture that looks like:



Try lots of other numbers of steps and see how the pattern changes.

2. Compare these four runs

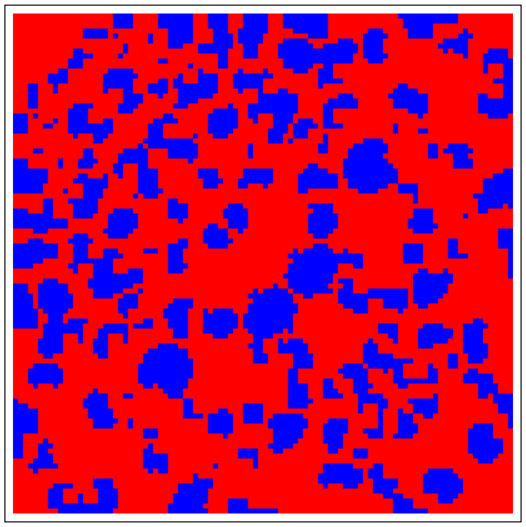
```
./spatial f99.txt  1.59 80
./spatial f99.txt  1.6 80
./spatial f99.txt  1.65 80
./spatial f99.txt  1.7 80
```

What is going on here?

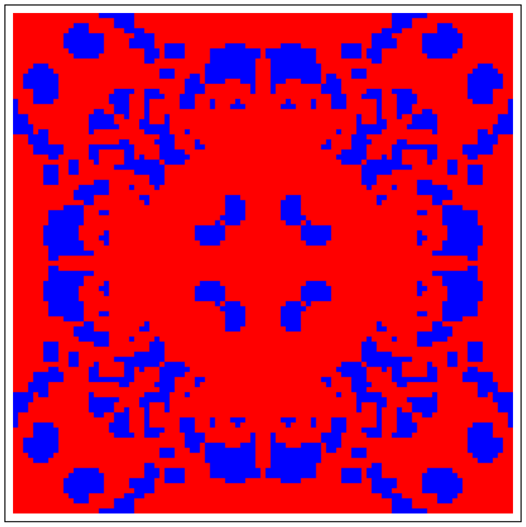
3. The initial field f99.txt is a single D cell in the center of a 99-by-99 field of C cells. The initial field f100.txt is the same, except the field dimensions are 100 by 100. If you run

```
./spatial f100.txt 1.65 120
```

You get:



Notice the result is not symmetric, unlike what you get with f99.txt:

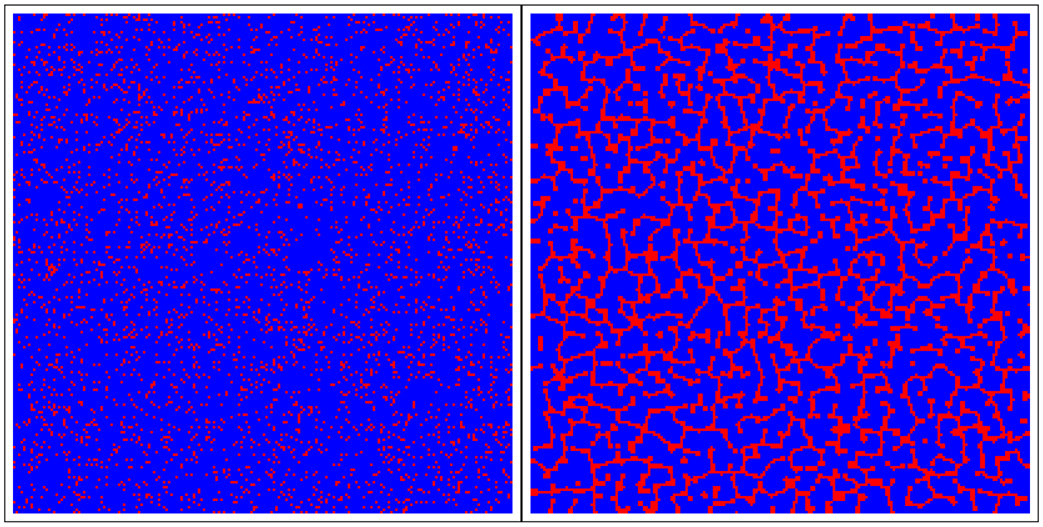


Why do you think that this is the case?

4. Compare these two runs:

```
./spatial rand200-10.txt 1.55 0
./spatial rand200-10.txt 1.55 200
```

You should get:



4.8 Learning outcomes

After completing this assignment, you should have:

- learned how to read data from a file,
- understood how to use struct types,
- learned about the Prisoner's dilemma and spatial games,
- gained additional practice drawing images.

4.9 If you're interested...

Implement the color scheme for drawing the field that is described in the Nowak and May paper cited above (where there are four colors depending on the current and *previous* state of a cell).