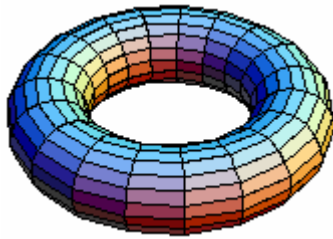


CSCI 135 Software Design and Analysis I

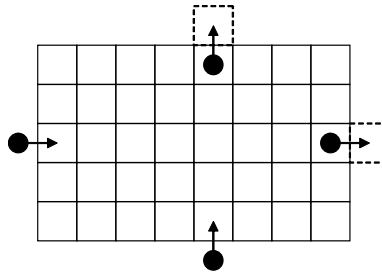
Project B

Living on a random torus

Saad Mneimneh
Visiting Professor
Hunter College of CUNY



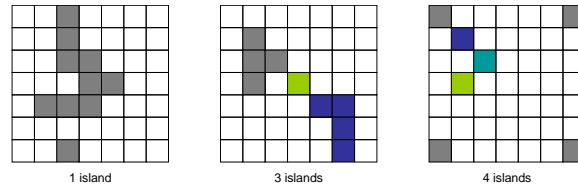
The goal of this project is to imagine life on a random torus and compute some interesting properties. A torus is the surface of a donut shape (shown above). Therefore, it can be represented as a two dimensional array that wraps around in both directions.



Such an array will be created dynamically as an $m \times n$ array of spots (a C++ vector can also be used to avoid the need to declare a destructor, a copy constructor, and an assignment operator in a class). Each spot is chosen randomly to be either land or water. For instance, the following simple class can be used to represent a spot:

```
class Spot {  
public:  
  
    bool land;  
    bool visited; //keep reading  
};
```

The main goal of this project is to compute the average number of islands produced on a random torus. An island is defined as a set of land spots such that we can go between any two spots by traveling only up, down, left, or right, and without crossing any water spots (while keeping in mind that we can wrap around in both directions). Here are some examples:



A suggested layout for the project is the following:

```
class Torus {
    Spot ** tor; //or vector<vector<Spot> > tor;
    int m;
    int n;

public:
    Torus(int m, int n); //constructs an mxn torus
    Torus(const Torus& t); //copy constructor if needed
    ~Torus(); //destructor if needed
    Torus& operator=(const Torus& t); //assignment operator if needed
    void regenerate(); //regenerates the torus randomly
    int islands()const; //returns number of islands
    Spot& at(int i, int j) {
        return tor[(i%m+m)%m][(j%n+n)%n];
    }
    const Spot& at(int i, int j)const {
        return tor[(i%m+m)%m][(j%n+n)%n];
    }

    . . .

};

int main() {
    int m;
    int n;
    cin>>m,n;

    Torus t=Torus(m,n);
    float average=0;
    for (int i=0; i<100; i++) {
        t.regenerate();
        average=average+t.islands()/100.0;
    }
    cout<<average<<' '\n';
}
```

The bulk of the work is the **island()** member function. One possible implementation of this function is the following. Start at any spot (i, j) of land and make that spot visited (by setting the boolean **visited** to true for that spot). Add the spot to an empty vector (e.g. add i and j). Repeatedly, remove the last spot from the vector and check the up, down, left, and right neighbors of that current spot. For each one, if the spot is an unvisited land, make that spot visited and add it to the end of the vector. The process stops when the vector becomes empty, and we've identified one island. This process is repeated until all land spots are visited.

Note: An alternative implementation is to make use of a recursive function. Observe that the vector described in the above process acts like a stack.