GSD 6349 Mapping II : Geosimulation
Professor Robert Gerard Pietrusko
<rpietrusko@gsd.harvard.edu>

class Kernel()

Kernel() provides numerous methods for calculating neighborhood functions– for use with cellular automata and agent-based models designed in Processing.

```
      ///////////////
     //           //
    //  METHODS  //
   //            //
  ///////////////

            Kernel()
            Kernel( int hoodType )
void        setNeighborhoodDistance( int num )
void        isTorus()
void        isNotTorus()

//          2D ARRAY GET METHODS
float       getSum(float[][] matrix, int x, int y)
float       getMean(float[][] matrix, int x, int y)
float       getMin(float[][] matrix, int x, int y)
float       getMax(float[][] matrix, int x, int y)
float       getRand(float[][] matrix, int x, int y)
float       getMajority(float[][] matrix, int x, int y)
float       getWeightedSum( float[][] matrix, int x, int y, float[] weights )
float[][]   get( float[][] matrix, int x, int y)

//          LATTICE GET METHODS
float       getSum(Lattice lat, int x, int y)
float       getMean(Lattice lat, int x, int y)
float       getMin(Lattice lat, int x, int y)
float       getMax(Lattice lat, int x, int y)
float       getRand(Lattice lat, int x, int y)
float       getMajority(Lattice lat, int x, int y)
float       getWeightedSum( Lattice lat, int x, int y, float[] weights )
float[][]   get( Lattice lat, int x, int y)

//          PVECTOR GET METHODS (2D ARRAY INPUT)
PVector     getMin( float[][] matrix, PVector loc )
PVector     getMax( float[][] matrix, PVector loc )
PVector     getRand(float[][] matrix, PVector loc )

//          PVECTOR GET METHODS (LATTICE INPUT)
PVector     getMin(  Lattice lat, PVector loc )
PVector     getMax(  Lattice lat, PVector loc )
PVector     getRand( Lattice lat, PVector loc )
```

/*+++*/
Kernel()

Default constructor. Creates a Moore neighborhood with one ring (8 neighbors).

/*+++*/
Kernel(int hoodtype)

INPUTS: int hoodtype // set neighborhood type
1 = MOORE.
2 = VON NEUMANN.
Creates neighborhood with one ring.

/*+++*/
void setNeighborhoodDistance(int num)

INPUTS: int num // set number of neighborhood rings
1 is standard neighborhood.
> 1 extended neighborhood.

/*+++*/
void isTorus()

No inputs. No outputs. Sets the kernel to wrap neighborhood around the edges of the matrix if necessary

/*+++*/
void isNotTorus()

No inputs. No outputs. Does not wrap kernel around the edge of the matrix. If cell has neighbors outside of matrix, it only calculates the based on the cells that are on the matrix. It therefore has a smaller neighborhood.

/*+++*/
float getSum(float[][] matrix, int x, int y)

INPUTS 2D array float[][] matrix of the full environment
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float sum of all of the cells in a kernel centered on cell x,y. This does not include the cell x,y in the sum, only the sum of its neighborhood.

/*+++*/
float getMean(float[][] matrix, int x, int y)

INPUTS 2D array float[][] matrix of the full environment
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float average of all of the cells in a kernel centered on cell x,y. This does not include the cell x,y in the average, only the average of its neighborhood.

/*+++*/
float getMin(float[][] matrix, int x, int y)

INPUTS 2D array float[][] matrix of the full environment
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float minimum value found in kernel centered on cell x,y. This does not include cell x,y, only its neighborhood.

/*+++*/
float getMax(float[][] matrix, int x, int y)

INPUTS 2D array float[][] matrix of the full environment
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float maximum value found in kernel centered on cell x,y. This does not include cell x,y, only its neighborhood.

/*+++*/
float getRand(float[][] matrix, int x, int y)

INPUTS 2D array float[][] matrix of the full environment
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float returns the value of a randomly chosen neighbor in the kernel centered on cell x,y. Cell x,y itself is never chosen as the random value.

/*+++*/
float getMajority(float[][] matrix, int x, int y)

INPUTS 2D array float[][] matrix of the full environment
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float returns the most common value of neighbors in the the kernel centered on cell x,y. If more than one value is equally common, a random selection is made among them.

/*++*/
float getWeightedSum(float[][] matrix, int x, int y, float[] weights)

INPUTS 2D array float[][] matrix of the full environment
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
float[] weights is an array of N values that scale each band in a neighborhood. N > 1, kernel will create a neighborhood of N bands around center cell and scale each value in the band by the corresponding entry in the weights array. For example: weights = {1,0.5}. The standard neighborhood cells will be multiplied by 1. A second ring at a distance of 2 from the center cell will be multiplied by 0.5. The all of the values will be added together.
OUTPUT float returns the summed value of the weighed cells.

/*++*/
float[][] get(float[][] matrix, int x, int y)

INPUTS 2D array float[][] matrix of the full environment
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float[][] returns a 2D array of the neighborhood around cell x,y. The rows of the array correspond to each neighborhood band. The columns of the array are the values of that band clock-wise from the top left corner. for example: to access the immediate neighborhood, output[0][i]. To access the second ring, if it exists, output[1][i]. Set the number of neighborhood bands using the **setNeighborhoodDistance()** function, above.

```
                ////////////////////////////
               //  METHODS USING LATTICE   //
              //        INPUTS            //
             ////////////////////////////
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
float    getSum(Lattice lat, int x, int y)

     INPUTS Lattice of the full environment ( see Lattice() class)
            int x the x coordinate of the kernel's center cell
            int y the y coordinate of the kernel's center cell
     OUTPUT float sum of all of the cells in a kernel centered on
            cell x,y. This does not include the cell x,y in the sum,
            only the sum of its neighborhood.

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
float    getMean(Lattice lat, int x, int y)

     INPUTS Lattice of the full environment ( see Lattice() class)
            int x the x coordinate of the kernel's center cell
            int y the y coordinate of the kernel's center cell
     OUTPUT float average of all of the cells in a kernel centered on
            cell x,y. This does not include the cell x,y in the
            average, only the average of its neighborhood.

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
float    getMin(Lattice lat, int x, int y)

     INPUTS Lattice of the full environment ( see Lattice() class)
            int x the x coordinate of the kernel's center cell
            int y the y coordinate of the kernel's center cell
     OUTPUT float minimum value found in kernel centered on cell x,y.
            This does not include cell x,y, only its neighborhood.

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
float    getMax(Lattice lat, int x, int y)

     INPUTS Lattice of the full environment ( see Lattice() class)
            int x the x coordinate of the kernel's center cell
            int y the y coordinate of the kernel's center cell
     OUTPUT float maximum value found in kernel centered on cell x,y.
            This does not include cell x,y, only its neighborhood.

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
float    getRand(Lattice lat, int x, int y)

     INPUTS Lattice of the full environment ( see Lattice() class)
            int x the x coordinate of the kernel's center cell
            int y the y coordinate of the kernel's center cell
     OUTPUT float returns the value of a randomly chosen neighbor in
            the kernel centered on cell x,y.
```

/*+++*/
float getMajority(Lattice lat, int x, int y)

INPUTS Lattice of the full environment (see Lattice() class)
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float returns the most common value of neighbors in the the kernel centered on cell x,y. If more than one value is equally common, a random selection is made among them.

/*+++*/
float getWeightedSum(Lattice lat, int x, int y, float[] weights)

INPUTS Lattice of the full environment (see Lattice() class)
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
float[] weights is an array of N values that scale each band in a neighborhood. N > 1, kernel will create a neighborhood of N bands around center cell and scale each value in the band by the corresponding entry in the weights array. For example: weights = {1,0.5}. The standard neighborhood cells will be multiplied by 1. A second ring at a distance of 2 from the center cell will be multiplied by 0.5. The all of the values will be added together.
OUTPUT float returns the summed value of the weighed cells.

/*+++*/
float[][] get(Lattice lat, int x, int y)

INPUTS Lattice of the full environment (see Lattice() class)
int x the x coordinate of the kernel's center cell
int y the y coordinate of the kernel's center cell
OUTPUT float[][] returns a 2D array of the neighborhood around cell x,y. The rows of the array correspond to each neighborhood band. The columns of the array are the values of that band clock-wise from the top left corner. for example: to access the immediate neighborhood, output[0][i]. To access the second ring, if it exists, output[1][i]. Set the number of neighborhood bands using the **setNeighborhoodDistance()** function, above.

```
    ////////////////////////////
   //                          //
  //  METHODS USING PVECTOR   //
 //      INPUTS AND OUTPUTS  //
//                          //
////////////////////////////
```

/*+++*/
PVector getMin(float[][] matrix, PVector loc)

INPUTS 2D array float[][] matrix of the full environment
PVector loc should contain the x,y coordinates of the kernel's center cell
OUTPUT PVector of minimum value found in kernel centered on cell x,y. The content of the PVector is as follows:
PVector.x : x-coordinate of cell containing min value
PVector.y : y-coordinate of cell containing min value
PVector.z : minimum value.

/*+++*/
PVector getMax(float[][] matrix, PVector loc)

INPUTS 2D array float[][] matrix of the full environment
PVector loc should contain the x,y coordinates of the kernel's center cell
OUTPUT PVector of maximum value found in kernel centered on cell x,y. The content of the PVector is as follows:
PVector.x : x-coordinate of cell containing max value
PVector.y : y-coordinate of cell containing max value
PVector.z : maximum value.

/*+++*/
PVector getRand(float[][] matrix, PVector loc)

INPUTS 2D array float[][] matrix of the full environment
PVector loc should contain the x,y coordinates of the kernel's center cell
OUTPUT PVector returns the value of a randomly chosen neighbor in the kernel centered on cell x,y. y. The content of the PVector is as follows:
PVector.x : x-coordinate of random neighbor cell
PVector.y : y-coordinate of random neighbor cell
PVector.z : value of random neighbor.

/*+++*/
PVector getMin(Lattice lat, PVector loc)

INPUTS Lattice of the full environment (see Lattice() class)
PVector loc should contain the x,y coordinates of the kernel's center cell
OUTPUT PVector of minimum value found in kernel centered on cell x,y. The content of the PVector is as follows:
PVector.x : x-coordinate of cell containing min value
PVector.y : y-coordinate of cell containing min value
PVector.z : minimum value.

/*+++*/
PVector getMax(Lattice lat, PVector loc)

INPUTS Lattice of the full environment (see Lattice() class)
PVector loc should contain the x,y coordinates of the kernel's center cell
OUTPUT PVector of maximum value found in kernel centered on cell x,y. The content of the PVector is as follows:
PVector.x : x-coordinate of cell containing max value
PVector.y : y-coordinate of cell containing max value
PVector.z : maximum value.

/*+++*/
PVector getRand(Lattice lat, PVector loc)

INPUTS Lattice of the full environment (see Lattice() class)
PVector loc should contain the x,y coordinates of the kernel's center cell
OUTPUT PVector returns the value of a randomly chosen neighbor in the kernel centered on cell x,y. y. The content of the PVector is as follows:
PVector.x : x-coordinate of random neighbor cell
PVector.y : y-coordinate of random neighbor cell
PVector.z : value of random neighbor.