Brea Koenes script file

```
bjk47@gold25:~/Desktop$ script
Script started, file is typescript
bjk47@gold25:~/Desktop$ cd firestarter/
bjk47@gold25:~/Desktop/firestarter$ make
mpicc -c -I/usr/X11R6/include -Wall firestarter.c
mpicc firestarter.o X-graph.o display.o -o Fire
-L/usr/X11R6/lib -lX11 -lm
bjk47@gold25:~/Desktop/firestarter$ make
make: 'Fire' is up to date.
bjk47@gold25:~/Desktop/firestarter$ cat firestarter.c
/* firestarter.c
 * David Joiner
 * Adjusted by Brea Koenes on 9/23/21
 * Usage: Fire [forestSize(20)] [numTrials(5000)] *
[numProbabilities(101)] [showGraph(1)]
 */
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include "X-graph.h"

#define UNBURNT 0
#define SMOLDERING 1
#define BURNING 2
#define BURNT 3

#define true 1
#define false 0

typedef int boolean;

extern void seed_by_time(int);
extern int ** allocate_forest(int);
extern void initialize_forest(int, int **);
extern double get_percent_burned(int, int **);
extern void delete_forest(int, int **);
extern void light_tree(int, int **,int,int);
extern boolean forest_is_burning(int, int **);
extern void forest_burns(int, int **,double);
extern int burn_until_out(int,int **,double,int,int);
extern void print_forest(int, int **);

int main(int argc, char ** argv) {
   // initial conditions and variable definitions
   int forest_size=80;
   double * prob_spread;
   double prob_min=0.0;
   double prob_max=1.0;
   double prob_step;
   double start_time = 0, end_time = 0;
   int **forest;
   double * percent_burned;
   double * iterations;
   double * avg_iterations;
   double * avg_burned;
   int i_trial;
   int n_trials=5000;
   int i_prob;
   int n_probs=101;
   int do_display=1;
   //xgraph thegraph;

   // check command line arguments
   if (argc > 1) {
      sscanf(argv[1],"%d",&forest_size);
   }
   if (argc > 2) {
      sscanf(argv[2],"%d",&n_trials);
   }
   if (argc > 3) {
      sscanf(argv[3],"%d",&n_probs);
   }
   if (argc > 4) {
      sscanf(argv[4],"%d",&do_display);
   }
   if (do_display!=0) do_display=1;

   // setup problem
   seed_by_time(0);
   forest=allocate_forest(forest_size);

   // create arrays of size 101 doubles to store the
   prob_spread = (double *) calloc (n_probs*sizeof(double),
n_probs*sizeof(double));
   percent_burned = (double *) calloc
(n_probs*sizeof(double), n_probs*sizeof(double));
   iterations = (double *) calloc (n_probs*sizeof(double),
n_probs*sizeof(double));
   avg_iterations = (double *) calloc (n_probs*sizeof(double),
n_probs*sizeof(double));
   avg_burned = (double *) calloc (n_probs*sizeof(double),
n_probs*sizeof(double));

   // MPI resources
   int id=-1, numProcesses=-1;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &id);
   MPI_Comm_size(MPI_COMM_WORLD,
&numProcesses);

   start_time = MPI_Wtime();

   // for a number of probabilities, calculate
   // average burn and output
   prob_step = (prob_max-prob_min)/(double)(n_probs-1);

   for (i_trial = id ; i_trial < n_trials; i_trial += numProcesses) {
      for (i_prob=0; i_prob < n_probs; i_prob++) {
         prob_spread[i_prob] = prob_min + (double)i_prob *
prob_step;


iterations[i_prob]+=burn_until_out(forest_size,forest,prob_sp
read[i_prob],
               forest_size/2,forest_size/2);
```

```
percent_burned[i_prob]+=get_percent_burned(forest_size,fo
rest);
      }
   }

   // MPI reduction pattern
   MPI_Barrier(MPI_COMM_WORLD);
   MPI_Reduce(percent_burned, avg_burned, n_probs,
MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
   MPI_Reduce(iterations, avg_iterations, n_probs,
MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
   end_time = MPI_Wtime();

   // print to console
   if (id==0) {
      for(i_prob=0; i_prob < n_probs; i_prob++) {
               avg_iterations[i_prob]/=n_trials;
         avg_burned[i_prob]/=n_trials;

         printf("%lf, %lf, %lf\n", prob_spread[i_prob],
avg_burned[i_prob], avg_iterations[i_prob]);
            }
      printf("Total time elapsed: %f\n", end_time - start_time);
   }

   // plot graph
   //if (do_display==1) {
   //    xgraphSetup(&thegraph,300,300);
   //
xgraphDraw(&thegraph,n_probs,0,0,1,1,prob_spread,percen
t_burned);
   //    pause();
   //}

   // clean up
   delete_forest(forest_size,forest);
   free(prob_spread);
   free(percent_burned);
   free(iterations);
   return 0;
}

#include <time.h>

void seed_by_time(int offset) {
   time_t the_time;
   time(&the_time);
   srand((int)the_time+offset);
}

int burn_until_out(int forest_size,int ** forest, double
prob_spread,
   int start_i, int start_j) {
   int count;

   initialize_forest(forest_size,forest);
   light_tree(forest_size,forest,start_i,start_j);

   // burn until fire is gone
   count = 0;
   while(forest_is_burning(forest_size,forest)) {
      forest_burns(forest_size,forest,prob_spread);
      count++;
   }

   return count;
}

double get_percent_burned(int forest_size,int ** forest) {
   int i,j;
   int total = forest_size*forest_size-1;
   int sum=0;

   // calculate pecrent burned
   for (i=0;i<forest_size;i++) {
      for (j=0;j<forest_size;j++) {
         if (forest[i][j]==BURNT) {
            sum++;
         }
      }
   }

   // return percent burned;
   return ((double)(sum-1)/(double)total);
}


int ** allocate_forest(int forest_size) {
   int i;
   int ** forest;

   forest = (int **) malloc (sizeof(int*)*forest_size);
   for (i=0;i<forest_size;i++) {
      forest[i] = (int *) malloc (sizeof(int)*forest_size);
   }

   return forest;
}

void initialize_forest(int forest_size, int ** forest) {
   int i,j;

   for (i=0;i<forest_size;i++) {
      for (j=0;j<forest_size;j++) {
         forest[i][j]=UNBURNT;
      }
   }
}

void delete_forest(int forest_size, int ** forest) {
   int i;

   for (i=0;i<forest_size;i++) {
      free(forest[i]);
   }
   free(forest);
}

void light_tree(int forest_size, int ** forest, int i, int j) {
   forest[i][j]=SMOLDERING;
}
```

```
boolean fire_spreads(double prob_spread) {
   if ((double)rand()/(double)RAND_MAX < prob_spread)
      return true;
   else
      return false;
}

void forest_burns(int forest_size, int **forest,double
prob_spread) {
   int i,j;
   extern boolean fire_spreads(double);

   //burning trees burn down, smoldering trees ignite
   for (i=0; i<forest_size; i++) {
      for (j=0;j<forest_size;j++) {
         if (forest[i][j]==BURNING) forest[i][j]=BURNT;
         if (forest[i][j]==SMOLDERING) forest[i][j]=BURNING;
      }
   }

   //unburnt trees catch fire
   for (i=0; i<forest_size; i++) {
      for (j=0;j<forest_size;j++) {
         if (forest[i][j]==BURNING) {
            if (i!=0) { // North
               if
(fire_spreads(prob_spread)&&forest[i-1][j]==UNBURNT) {
                  forest[i-1][j]=SMOLDERING;
               }
            }
            if (i!=forest_size-1) { //South
               if
(fire_spreads(prob_spread)&&forest[i+1][j]==UNBURNT) {
                  forest[i+1][j]=SMOLDERING;
               }
            }
            if (j!=0) { // West
               if
(fire_spreads(prob_spread)&&forest[i][j-1]==UNBURNT) {
                  forest[i][j-1]=SMOLDERING;
               }
            }
            if (j!=forest_size-1) { // East
               if
(fire_spreads(prob_spread)&&forest[i][j+1]==UNBURNT) {
                  forest[i][j+1]=SMOLDERING;
               }
            }
         }
      }
   }
}

boolean forest_is_burning(int forest_size, int ** forest) {
   int i,j;

   for (i=0; i<forest_size; i++) {
      for (j=0; j<forest_size; j++) {
         if
(forest[i][j]==SMOLDERING||forest[i][j]==BURNING) {
            return true;
         }
      }
   }
   return false;
}

void print_forest(int forest_size,int ** forest) {
   int i,j;

   for (i=0;i<forest_size;i++) {
      for (j=0;j<forest_size;j++) {
         if (forest[i][j]==BURNT) {
            printf(".");
         } else {
            printf("X");
         }
      }
      printf("\n");
   }
}
bjk47@gold25:~/Desktop/firestarter$ exit
Script done, file is typescript
```