Class CMPS 261
Section 001
Problem Programming Assignment #3
Name McKelvy, James Markus
CLID Jmm0468
Due Date 12:30pm November 3, 2005

II. Design Documentation

II.1 System Architecture Description

The object(s) within the program are:
UnionFind: UnionFind.h
MazeGenerator: MazeGenerator.h UnionFind.h Cell.h

The main objective of the application is to utilize the implementation of the UnionFind class within the MazeGenerator class to create automatically a random maze. The application's main driver will be used to prompt the user for the dimensions of the random maze to be created and to create it as well as output it to the user. The main driver will create any size maze that the user desires so long as the dimensions are 3x3 or greater and there is enough memory to allocate it.

II.2 Information about the Objects

II.2.1 Class Information
Name: MazeGenerator
Description: A class representing a maze. It will generate a random maze immediately after construction.
Base Class: UnionFind

II.2.2 Class Attributes
Name: rows
Description: Represents the numbers of rows in the maze
Type: int
Range of acceptable values: Any value greater than 0

Name: columns
Description: Represents the numbers of columns in the maze
Type: int
Range of acceptable values: Any value greater than 0

Name: cells
Description: Represents the cells of the maze
Type: Cell *
Range of acceptable values: boolean

Name: maze
Description: Represents the paths formed by cells of the maze

Type: UnionFind *
Range of acceptable values: Any value of type "int"

II.2.3 Class Operations
Prototype: MazeGenerator(int x, int y);
Description: Default Constructor
Preconditions:
1. The value of "x" is assumed to be > 2
2. The value of "y" is assumed to be > 2
3. There is enough memory for the object to be allocated
Postcondition:
1. If the value of "x" is < 1, then the number of rows will be 4. Otherwise the number of rows will be "x".
2. If the value of "y" is < 1, then the number of columns will be 4. Otherwise the number of columns will be "y".
3. "cell" is allocated to be the size of "rows" * "columns"
4. "maze" is allocated to have the number of partitions: "rows" * "columns"
Cost Analysis: O(n log*(n))
Visibility: public

Prototype: ~MazeGenerator();
Description: Destructor
Precondition:
1. The dynamic cell array "cells" has been allocated
2. The dynamic UnionFind array "maze" has been allocated
Postcondition: Deallocates the memory used by this object
Cost Analysis: O(1)
Visibility: public

Prototype: void print();
Description: Prints the maze to the screen using "cout"
Preconditions: None
Postcondition: Prints the maze to the screen using "cout"
Cost Analysis: O(1)
Visibility: public

Prototype: void generate();
Description: Handles the generation of a random path through the maze
Preconditions:
1. The variables "rows" and "columns" have been set
2. The variables "cells" and "maze" have been allocated
Postcondition: A completely random path is made through the maze
Cost Analysis: O(n log*(n)) ~ O(n)
Visibility: private

Prototype: bool knockDown(int location);
Description: Knocks down the appropriate walls for a given location only if the cell chosen and the random wall chosen are not currently connected by

a path
Preconditions:
1. The value of "location" is assumed to be a valid index in the maze
Postcondition:
1. For the location, the walls which are not allowed to be knocked down are eliminated.
2. If the neighboring cell is not already connected by a path, the walls of both cells are "knocked down" and the value of "true" is returned.
3. If the neighboring cell is already connected by a path, the value of "false" is returned.
Cost Analysis: O(log*(n))
Visibility: private

Prototype: int isEdge(int location)
Description: Finds where in the matrix a particular location is: a corner, a side, or inner
Preconditions:
1. The variable "location" is assumed to be a valid index of the "cells" array
2. The variable "rows" is assumed to be defined and > 0
3. The variable "columns" is assumed to be defined and > 0
Postcondition:
1. An int is returned based upon the location in the maze that a particular index is.
   a. 1 if it is the top left corner
   b. 2 if it is the top and not a corner
   c. 3 if it is the top right corner
   d. 4 if it is the left and not a corner
   e. 5 if it is the right and not a corner
   f. 6 if it is the bottom left corner
   g. 7 if it is the bottom and not a corner
   h. 8 if it is the bottom right corner
   i. 0 if it is none of the above (an inner cell)
Cost Analysis: O(1)
Visibility: private

II.2.1 Class Information
Name: UnionFind
Description: A class representing the Union/Find construct containing partitions. It has member functions allowing it to perform a find on an element, as well as union on members of different partitions.
Base Class: N/A

II.2.2 Class Attributes
Name: array
Description: Represents the Union/Find data structure in an array
Type: int *
Range of acceptable values: Any value of type "int"

Name: max
Description: Represents the maximum number of partitions
Type: int
Range of acceptable values: Any value greater than 0

Name: currentPartitions
Description: Represents the current number of partitions
Type: int
Range of acceptable value: Any value greater than 0 and less than or equal to "max".

II.2.3 Class Operations
Prototype: UnionFind(int maxPartitions);
Description: Default Constructor
Preconditions:
1. The value of "maxPartitions" must be > 0
2. There must be enough memory to allocate an array of size "maxPartitions"
Postcondition:
1. If "maxPartitions" is > 0
   a. The value of "max" is set to the value of "maxPartitions"
   b. An array of size "maxPartitions" is allocated in memory
2. If "maxPartitions" is <= 0
   a. The value of "max" is set to the value of "1"
   b. An array of size "1" is allocated in memory
Cost Analysis: O(n)
Visibility: public

Prototype: ~UnionFind();
Description: Destructor
Precondition: The dynamic int array "array" has been allocated
Postcondition: Deallocates the memory used by this object
Cost Analysis: O(1)
Visibility: public

Prototype: int Find(int x);
Description: Finds the partition that an element is in
Precondition: The UnionFind object must be initialized and the item must be of type "int".
Postcondition:
1. If the value of "x" < "0" or >= "max" a "-1" is returned indicating that there is no leader for an invalid partition
2. If the element at the current array position is a negative number, then the current index is returned indicating that the position in the array is a leader of "x"
3. If the element at the current array position is a positive number or zero, then through a series of recursive steps until the partition leader is found, each node not pointing to the partition leader is made

to do so (path compression).
Cost Analysis: O(log*(n)) ~ O(1)
Visibility: public

Prototype: void Union(int x, int y);
Description: Unions (merges) two partitions together into a new partition
Precondition: The UnionFind object must be initialized and the "x" and "y" must be integers
Postcondition:
1. If the values of "x" or "y" are invalid, nothing is done
2. If the partition leaders of "x" and "y" are the same, nothing is done
3. If the partition leaders of "x" and "y" are unique, then merge them by making the smaller partition leader a leader of the larger partition.
4. Change the value of "currentPartitions" if two partitions are merged.
Cost Analysis: O(log*(n)) ~ O(1)
Visibility: public

Prototype: int numberOfPartitions();
Description: Finds the number of partitions in the UnionFind's current state
Precondition: The UnionFind object must be initialized.
Postcondition: The value of "currentPartitions" is returned.
Cost Analysis: O(1)
Visibility: public

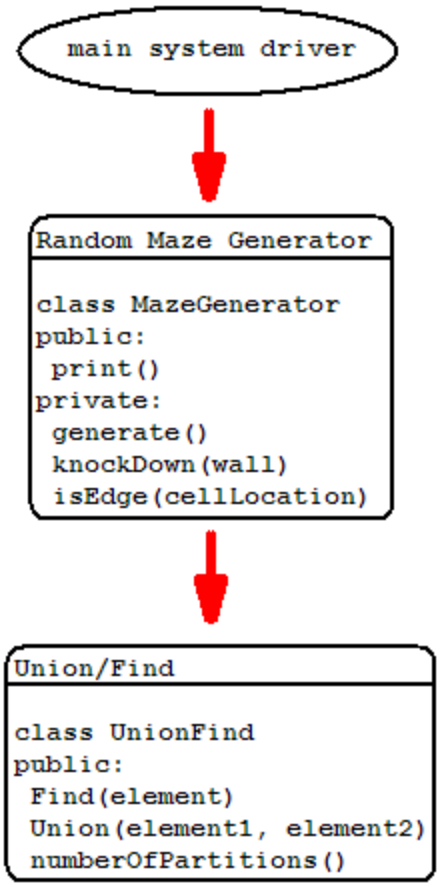
II.3 Information about the Main Application

```
#include<iostream>
#include<cstdlib>
#include "MazeGenerator.h"

using namespace std;

// Prototype: void pause();
// Description: Pauses for the user.
// Precondition: None.
// Postcondition: Continues execution of the program.
// Cost Analysis: O(1)
// Visibility: public
void pause();

int main(){
   bool done = false;
   char choice;

   do{
      cout << endl << "Do you want to create a maze (y/n)? ";
      cin >> choice;
      if(!(choice == 'y' || choice == 'Y'))
        done = true;
```

```
        else{
          int rows;
          int cols;
          MazeGenerator * m;

          cout << endl << "How many rows? ";
          cin >> rows;
          cout << endl << "How many columns? ";
          cin >> cols;
          cout << endl;
          m = new MazeGenerator(rows, cols);
          m->print();
          delete m;
          pause();
        }
    }while(!done);

    return 0;
}

void pause(){
  cout << "Press ENTER.";
  getchar();
  cout << endl << "----------------------------" << endl;
}
```

II.4 Design Diagrams

II.4.1 Object Interaction Diagram



II.4.2 Aggregation Diagram