Project 2

Text-based Minesweeper v2

CSC-5 Section 46024
John Decker
8/1/14

Introduction:

Title: Minesweeper

This is a text-based iteration of the classic OS/2 (then ported to Windows) game Minesweeper. Instead of using a mouse cursor in a graphical user interface to select grid squares, the player will be prompted for X and Y coordinates.

At the beginning of the program the player is prompted with a menu asking whether the player wants to play Minesweeper or view the saved high scores list. If the player chooses to view the high scores, the 10 highest scores are loaded from a text file and displayed. If the player chooses to play Minesweeper, the player is asked for desired board dimensions and, based on that value, the amount of mines to place. A game board is then generated using pseudo-random numbers. Zero through nine values are assigned to all squares. Values of 0 through 8 are assigned to squares based on amount of adjacencies to mines (9s). The player then selects squares one by one, by coordinate. As the value of the selected squares are revealed to the player, the player must narrow down where the mines are located and avoid selecting them. If the player selects a coordinate with a mine (value of 9) the player loses the game. If the player eliminates all non-mine squares, then the player wins. If the player wins, then their score is added up, the high scores list is opened, the players score is bubble sorted amongst the scores from the file, and written to "scores.txt." Win or lose, the player is then asked if he or she would like to play again, and if not, brought back to the main menu.

Summary:

Project size: 480 lines
Number of variables: 16
Number of functions (other than main): 2

This project includes many for loops, many if statements, a few while loops, and one switch.
The portion of the code to display the board was fairly straight forward. The portion used to generate the board was only slightly trickier in figuring out and keeping track of how to check the positions of mines relative to the square being assigned a value (e.g top-right, bottom-left, etc.). But certainly the biggest headache was the generation of the random numbers and keeping the amount of mines limited to the amount chosen. In my early attempts the resulting mine placement often looked something like this:

```
  0 1 2 3 4 5 6 7 8 9
0 |x|_|_|x|x|_|_|_|x|
1 |x|_|_|x|x|_|_|_|x|
2 |x|_|_|x|x|_|_|_|x|
3 |x|_|_|x|x|_|_|_|x|
4 |x|x|x|x|x|x|x|x|x|
5 |x|x|x|x|x|x|x|x|x|
6 |x|_|_|x|x|_|_|_|x|
7 |x|_|_|x|x|_|_|_|x|
8 |x|_|_|x|x|_|_|_|x|
9 |x|_|_|x|x|_|_|_|x|
```

If the (CONSTANT * m) wasn't added to (random / (random * m), board creation would fail. After

looking on a cryptography website and finding a formula to relate I and J and the dimensions of the square to the random number, and fiddling around with the constant in the line “random = random / (random * m) + (CONSTANT * m);” I finally found a number that worked; Pretty much any constant seemed to give a correct output. 1621 was chosen at random.

In Version 2, a main menu and high scores system were added. The high score system was supported by reading and writing to file to save scores after program exit.

Pseudo-code

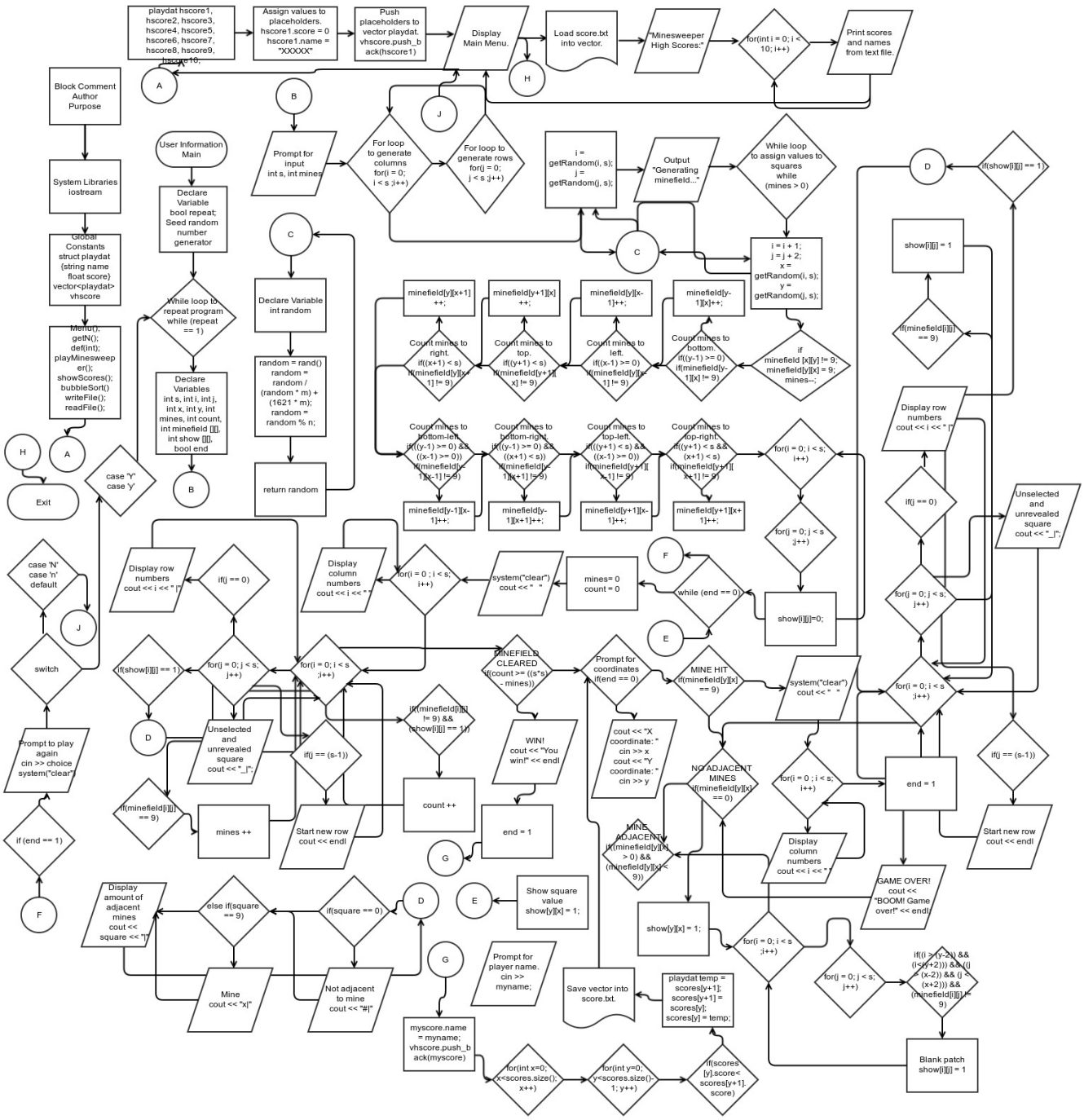
Initialize
Seed random number generation.
Main Menu for Minesweeper, high scores, or exit.
    Switch case display high scores
    Load scores from file.
    Print scores.
    Return to Main Menu.
Switch case play Minesweeper.
Load high scores from file.
Repeat while boolean (repeat) is true.
    Prompt for dimensions (s).
    Prompt for amount of mines (mines).
    Generate board.
        Generate random numbers.
        While integer array (minefield [x][y]) is not 9:
            Check square above generated square for a mine.
            Check square below generated square for a mine.
            Check square to the right of generated square for a mine.
            Check square to the left of generated square for a mine.
            Check square to the top-right of generated square for a mine.
            Check square to the bottom-right of square for a mine.
            Check square to the top-left square for a mine.
            Check square to the bottom-left of square for a mine.
            Assign the square the value equal to the amount of adjacencies.
        For loop to get next random numbers.
    Prompt for X and Y and check squares while loop:
        Display column numbers.
        For loop to display board.
            Display row numbers.
            Show square.
            Print unselected and unrevealed square.
            Start new row.
            Increase count.
        Check if all mines have been discovered (win condition).
            If true display “win.”
            Prompt for name.
            Calculate player score.
            Bubble sort high scores list, including players.

Write sorted high scores to a text file.
Prompt for X and Y coordinates.
Check for mine hit (loss condition).
Display column numbers.
For loop to display board.
Display row numbers.
Show positions of mines.
Show empty squares.
If true display “game over.”
Check for adjacent mines.
Prompt to play again.
Switch statement.
'Y' or 'y' to repeat.
'N' or 'n' to go back to Main Menu.
Main Menu default case: exit.

Major Constructs:

Vectors, Arrays, Struct, Bubble sort, Keyboard input and screen output, File input and output, Data types (boolean, integers, floats, strings, and characters), Referenced variables, If-else statements, For loops, While loops, Switches, Functions, Addition and multiplication, Random number generation.

Flowchart:



Major Variables:

Global:

struct playdat – Used to link a string and a float.

string name – The player's name.

float score – The player's end of game score = ((time of game)/(area of board))*(amount of mines)*10000.

vector <playdat> vhscore – Vector to hold struct playdat.

main():

playdat hscore1, hscore2, hscore3, hscore4, hscore5, hscore6, hscore7, hscore8, hscore9, hscore10 –

Placeholders in case the score.txt file has become deleted or is not yet created.
Int inN – Number of player's main menu selection.

playMinesweeper():
playdat myscore; - Wector position to hold name and score.
bool repeat – Repeat the game.
int s – Dimensions of board.
int minefield [][] - Two dimensional array containing the values of the game board, 0 – 9.
int show[][] - Two dimensional array used to display the game board.
int mines – Amount of mines.
int count – Spaces successfully cleared.
int i – Used for the first dimension of the array in the for loops.
int j - Used for the second dimension of the array in the for loops.
int x – X coordinate.
int y – Y coordinate.
bool end – Used to indicate the end of the game.
int time1- Used to get the game start time.
int time2 - Used to get the game end time.
float timediff - Used for duration of game.
string myname – Name input.
char choice – Y or N to use in repeat switch.

getRandom():
int m – Used to relate the random number to ‘i’ or ‘j’.
int n – Used to relate the random number to ‘s’.
int random – Random number returned to main.

showBoard():
int square – Value assigned to square being shown.

writeFile():
ofstream outputFile ("scores.txt") – High scores output file.
ifstream inputFile ("scores.txt") – High scores input file.

Program:

```
/*
 * File: main.cpp
 * Author: John Decker
 * Purpose: Project 2
 * Created on July 26, 2014, 1:35 PM
 */
//System Libraries
#include <cstdlib>
#include <iostream>
#include <ctime>
#include <string>
#include <vector>
```

```
#include <fstream>
using namespace std;

//Global Constants
struct playdat {
   string name;
   float score;
   };
vector<playdat> vhscore;

//Function Prototypes
void Menu();
int getN();
void def(int);
void playMinesweeper();
void showScores();
void bubbleSort(vector<playdat> &scores);
void writeFile();
void readFile();
int getRandom(int m, int n); //Function to get a random number based on s, i, j.
void showBoard(int square); //Display the type of the square (Mine or no mine).

//Execution begins here
int main(int argc, char** argv)
{
   srand(static_cast<unsigned int>(time(0))); //Seed random number.
   playdat hscore1, hscore2, hscore3, hscore4, hscore5, hscore6,
         hscore7, hscore8, hscore9, hscore10; //High score placeholders.
   hscore1.score = 0; //Assign values to placeholders.
   hscore1.name = "XXXXX";
   hscore2.score = 0;
   hscore2.name = "XXXXX";
   hscore3.score = 0;
   hscore3.name = "XXXXX";
   hscore4.score = 0;
   hscore4.name = "XXXXX";
   hscore5.score = 0;
   hscore5.name = "XXXXX";
   hscore6.score = 0;
   hscore6.name = "XXXXX";
   hscore7.score = 0;
   hscore7.name = "XXXXX";
   hscore8.score = 0;
   hscore8.name = "XXXXX";
   hscore9.score = 0;
   hscore9.name = "XXXXX";
   hscore10.score = 0;
   hscore10.name = "XXXXX";
   vhscore.push_back(hscore1); //Put placeholders into vector.
```

```
    vhscore.push_back(hscore2);
    vhscore.push_back(hscore3);
    vhscore.push_back(hscore4);
    vhscore.push_back(hscore5);
    vhscore.push_back(hscore6);
    vhscore.push_back(hscore7);
    vhscore.push_back(hscore8);
    vhscore.push_back(hscore9);
    vhscore.push_back(hscore10);
    int inN = 0;
    cout << "Welcome to Minesweeper!" << endl;
    while(inN < 3)
    {
        Menu();
        inN = getN();
        switch(inN)
        {
            case 1: playMinesweeper();break;
            case 2: showScores();break;
            default: def(inN);
        }
    }
    return 0;
}

void Menu()
{
    cout<<"\n";
    cout<<"Type 1 to play Minesweeper."<<endl;
    cout<<"Type 2 to view high scores."<<endl;
    cout<<"Type 3 to exit.\n"<<endl;
}

int getN()
{
    int inN;
    cin >> inN;
    cin.clear();
    return inN;
}

void playMinesweeper()
{
    readFile();
    playdat myscore;
    bool repeat;
    repeat = 1;
    while (repeat == 1)
    {
```

```
system("clear");
cout << "Welcome to Minesweeper!" << endl << endl;
int s = 0;
while(s < 2 || s > 10)
{
   cout << "Enter number of squares per side (2 - 10):" << endl;
   cin >> s;
   cin.clear();
}
int minefield[s][s]; //2D array. 0 through 8 are # of adjacent mines, 9 is a mine.
int show[s][s]; //2D array. Square shown by dimensions s.
int mines = 0; //Number of mines.
int count;
int i = 0; //Index for x-axis.
int j = 0; //Index for y-axis.
int x = 0; //X coordinate.
int y = 0; //Y coordinate.
bool end = 0; //Game won or lost.
int time1 = 0; //Used to get the game start time.
int time2 = 0; //Used to get the game end time.
float timediff = 0.0; //Used for duration of game.
string myname;
while(mines < 1 || mines > ((s*s)-1))
{
   cout << "Enter the number of mines (1 - " << ((s*s)-1) << "):" << endl;
   cin >> mines;
   cin.clear();
}
// Create the minefield:
for(i = 0; i < s ;i++)
{
   for(j = 0; j < s ;j++)
   {
      minefield[i][j] = 0;
   }
}
i = getRandom(i, s);
j = getRandom(j, s);
cout << "Generating minefield..." << endl;
while(mines > 0)
{
   i = i + 1;
   j = j + 2;
   x = getRandom(i, s);
   y = getRandom(j, s);
   if(minefield[y][x] != 9)
   {
      minefield[y][x] = 9;
      mines--;
```

```
if((y-1) >= 0) //Count mines to bottom.
{
   if(minefield[y-1][x] != 9)
   {
      minefield[y-1][x]++;
   }
}
if((x-1) >= 0) //Count mines to left.
{
   if(minefield[y][x-1] != 9)
   {
      minefield[y][x-1]++;
   }
}
if((y+1) < s) //Count mines to top.
{
   if(minefield[y+1][x] != 9)
   {
      minefield[y+1][x]++;
   }
}
if((x+1) < s) //Count mines to right.
{
   if(minefield[y][x+1] != 9)
   {
      minefield[y][x+1]++;
   }
}
if(((y-1) >= 0) && ((x-1) >= 0)) //Count mines to bottom-left.
{
   if(minefield[y-1][x-1] != 9)
   {
      minefield[y-1][x-1]++;
   }
}
if(((y-1) >= 0) && ((x+1) < s)) //Count mines to bottom-right.
{
   if(minefield[y-1][x+1] != 9)
   {
      minefield[y-1][x+1]++;
   }
}
if(((y+1) < s) && ((x-1) >= 0)) //Count mines to top-left.
{
   if(minefield[y+1][x-1] != 9)
   {
      minefield[y+1][x-1]++;
   }
}
```

```
        if((y+1) < s && (x+1) < s) //Count mines to top-right.
        {
            if(minefield[y+1][x+1] != 9)
            {
                minefield[y+1][x+1]++;
            }
        }
    }
};
for(i = 0; i < s; i++)
{
    for(j = 0; j < s ;j++)
    {
        show[i][j]=0;
    }
}
//Prompt for X and Y then check squares loop:
time1 = time(0);
while(end == 0)
{
    system("clear");
    count = 0;
    mines = 0;
    cout << "   ";
    for(i = 0 ; i < s; i++)
    {
        cout << i << " "; //Display column numbers.
    }
    cout << endl;
    for(i = 0; i < s ;i++) //Loop to display the board.
    {
        for(j = 0; j < s; j++)
        {
            if(j == 0)
            {
                cout << i << " |"; //Display row numbers.
            }
            if(show[i][j] == 1)
            {
                showBoard(minefield[i][j]); //Show square.
            }
            else
            {
                cout << "_|"; // Unselected and unrevealed square.
            }
            if(j == (s-1))
            {
                cout << endl; //Start new row.
            }
```

```
            if((minefield[i][j] != 9) && (show[i][j] == 1))
            {
               count++;
            }
            if(minefield[i][j] == 9)
            {
               mines++;
            }
         }
      }
      if(count >= ((s*s) - mines)) //Mines cleared successfully.
      {
         time2 = time(0);
         timediff = time2 - time1;
         //Score = (Total Time / Board Area) * Number of Mines * 10000
         myscore.score = (timediff / (s*s)) * mines * 10000;
         cout << "You win!" << endl;
         cout << "Enter your name (NO SPACES!!!): " << endl;
         cin >> myname;
         cin.clear();
         myscore.name = myname;
         vhscore.push_back(myscore); //Put player score into vector.
         bubbleSort(vhscore); //Bubble sort high score vector.
         writeFile(); //Write new high score list to file.
         end = 1;
      }
      if(end == 0) //Prompt for X and Y coordinates.
      {
         cout << "X coordinate: ";
         cin >> x;
         cin.clear();
         cout << "Y coordinate: ";
         cin >> y;
         cin.clear();
      }
      if(minefield[y][x] == 9) //Mine hit.
      {
         system("clear");
         cout << "   ";
         for(i = 0; i < s; i++)
         {
            cout << i << " "; //Display column numbers.
         }
         cout << endl;
         end = 1;
         for(i = 0; i < s; i++) //Loop to display the board.
         {
            for(j = 0; j < s; j++)
            {
```

```
            if(j == 0)
            {
               cout << i << " |"; //Display row numbers.
            }
            if(minefield[i][j] == 9)
            {
               show[i][j] = 1;
            }
            if(show[i][j] == 1)
            {
               showBoard(minefield[i][j]); //Show positions of mines.
            }
            else
            {
               cout << "_|"; // Show empty square.
            }
            if(j == (s-1))
            {
               cout << endl; // Start new row.
            }
         }
      }
   cout << "BOOM! Game over!" << endl;
   }
   if(minefield[y][x] == 0) //No adjacent mines.
   {
      show[y][x] = 1;
      for(i = 0; i < s; i++)
      {
         for(j = 0; j < s; j++)
         {
            if((i > (y-2)) && (i<(y+2)))
            {
               if((j > (x-2)) && (j < (x+2)))
               {
                  if(minefield[i][j] != 9)
                  {
                     show[i][j] = 1; //Show blank patch.
                  }
               }
            }
         }
      }
   }
   if((minefield[y][x] > 0) && (minefield[y][x] < 9)) //Mine adjacent but not hit.
   {
      show[y][x] = 1;
   }
}
```

```
        if (end == 1) //Prompt for new game.
        {
            char choice;
            cout << "Would you like to play again? (Y or N) (Press Enter)\n";
            cin >> choice;
            cin.clear();
            system("clear");
            switch(choice)
            {
                case 'Y':
                {
                    repeat = 1;
                    break;
                }
                case 'y':
                {
                    repeat = 1;
                    break;
                }
                case 'N':
                {
                    repeat = 0;
                    break;
                }
                case 'n':
                {
                    repeat = 0;
                    break;
                }
                default:
                {
                    repeat = 0;
                    break;
                }
            }
        }
    }
}

void showBoard(int square)
{
    if(square == 0)
    {
        cout << "#|"; //Square not adjacent to mine.
    }
    else if(square == 9)
    {
        cout << "x|"; //Mine.
    }
```

```
   else
   {
      cout << square << "|"; //Display amount of mines adjacent.
   }
}

int getRandom(int m, int n)
{
   int random;
   random = rand();
   random = random / (random * m) + (1621 * m); //Relate random number to m.
   random = random % n; //Relate random number to n.
   return random;
}

void bubbleSort(vector<playdat> &scores) //Bubble sort high scores.
{
   for(int x=0; x<scores.size(); x++)
   {
      for(int y=0; y<scores.size()-1; y++)
      {
         if(scores[y].score < scores[y+1].score)
         {
               playdat temp = scores[y+1];
               scores[y+1] = scores[y];
               scores[y] = temp;
         }
      }
   }
}

void showScores() //Display the high scores list.
{
   readFile();
   system("clear");
   cout << "\tMinesweeper High Scores:\n";
   cout << "_____\n";
   for(int i = 0; i < 10; i++)
   {
      cout <<"#"<< i+1 <<"\t"<< vhscore[i].name <<"\t\t\t"<<
            vhscore[i].score << endl;
   }
}

void writeFile() //Write high scores to text file.
{
   ofstream outputFile ("scores.txt");
   if (outputFile.is_open())
   {
```

```
        for (int i = 0; i < vhscore.size(); i++)
        {
           outputFile << vhscore[i].name <<" "<<
                 vhscore[i].score <<" ";
        }
        outputFile.close();
    }
}

void readFile() //Read high scores from text file.
{
    ifstream inputFile ("scores.txt");
    if (inputFile.is_open())
    {
        for (int i = 0; i < vhscore.size(); i++)
        {
           inputFile >> vhscore[i].name >> vhscore[i].score;
        }
        inputFile.close();
    }
}

void def(int inN)
{
    system("clear");
    cout<<"You typed "<<inN<<" to exit the program."<<endl;
}
```