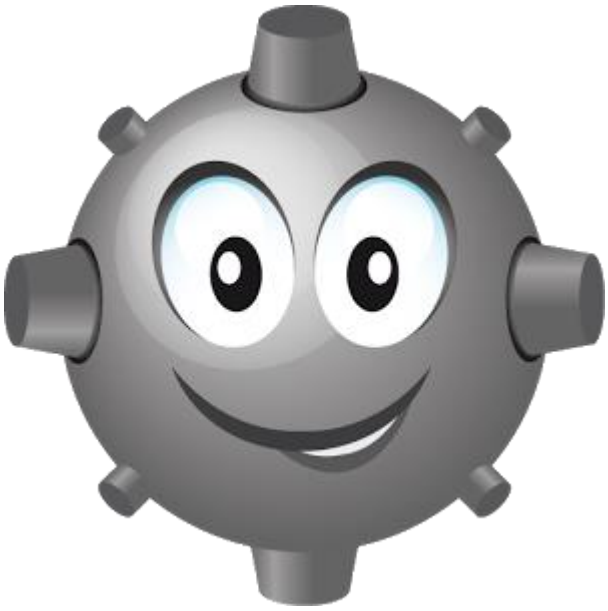
Eduardo Rangel

CSC 17A

Dr. Mark Lehr

Project 2 Minesweeper Game

December 15 2015



Introduction:

Minesweeper is a game that involves a lot of logic for the player and fro the creator (in this case is me). Minesweeper consists of uncovering the coordinates in this case without uncovering the mine which if you uncover the mine you lose the game. The player has to uncover all the coordinates in the game board without uncovering any mine the number of mines in the game depend on the difficulty of the game i set the game so that if the player chooses easy there is only one mine but for the normal and the hard level there are more than one mine in the game board and you have more chances of losing the game. You have the opportunity to choose the size of the game board in this case is from 1 x 1 to 10 x 10. I decided to do this game because the one I did in the first project didn't have a lot of logic and I decided to do this which was challenging for me

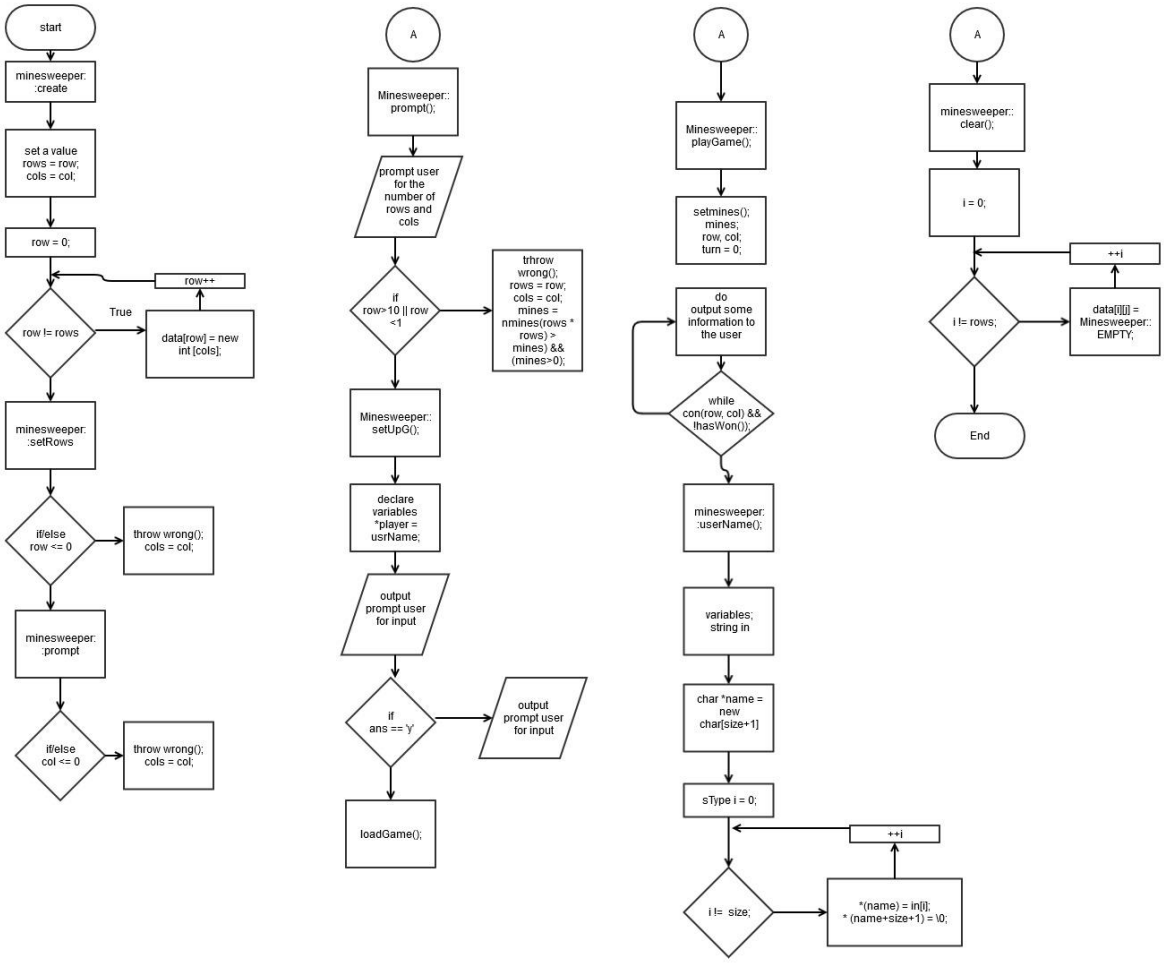
Summary:

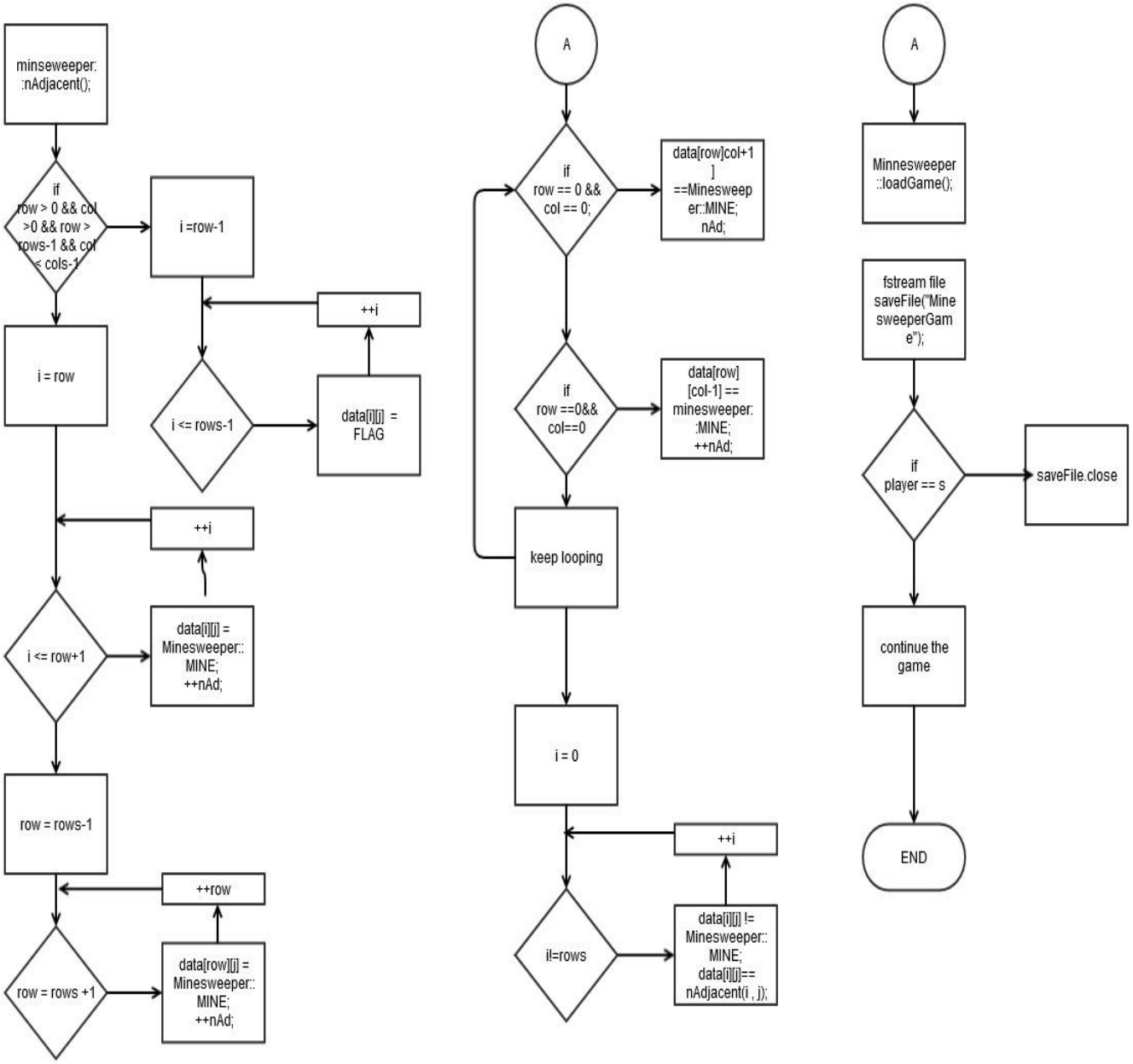
Number of lines: 930

Number of variables: 20

Number of classes: 4

Flowchart


start
minesweeper: :create
set a value rows = row; cols = col;
row = 0;
row != rows
True
data[row] = new int [cols];
row++
minesweeper: :setRows
if/else row <= 0
throw wrong(); cols = col;
minesweeper: :prompt
if/else col <= 0
throw wrong(); cols = col;
A
Minesweeper:: prompt();
prompt user for the number of rows and cols
if row>10 || row <1
trhrow wrong(); rows = row; cols = col; mines = nmines(rows * rows) > mines) && (mines>0);
Minesweeper:: setUpG();
declare variables *player = usrName;
output prompt user for input
if ans == 'y'
output prompt user for input
loadGame();
A
Minesweeper:: playGame();
setmines(); mines; row, col; turn = 0;
do output some information to the user
while con(row, col) && !hasWon());
minesweeper: :userName();
variables; string in
char *name = new char[size+1]
sType i = 0;
i != size;
++i
*(name) = in[i]; * (name+size+1) = \0;
A
minesweeper:: clear();
i = 0;
i != rows;
++i
data[i][j] = Minesweeper:: EMPTY;
End

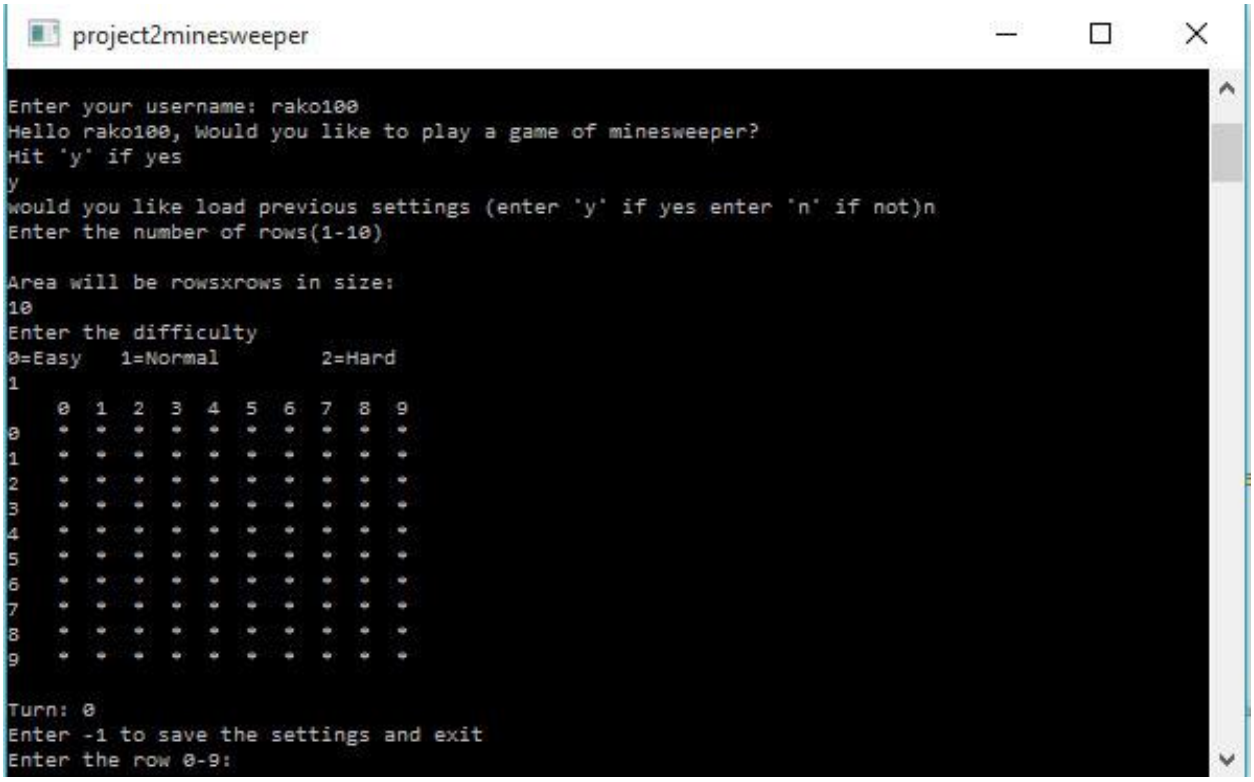

minseweeper::nAdjacent();
if row > 0 && col >0 && row > rows-1 && col < cols-1
i =row-1
++i
i = row
i <= rows-1
data[i][j] = FLAG
++i
i <= row+1
data[i][j] = Minesweeper::MINE; ++nAd;
row = rows-1
++row
row = rows +1
data[row][j] = Minesweeper::MINE; ++nAd;
A
if row == 0 && col == 0;
data[row]col+1] ==Minesweeper::MINE; nAd;
if row ==0&& col==0
data[row][col-1] == minesweeper::MINE; ++nAd;
keep looping
i = 0
++i
il=rows
data[i][j] != Minesweeper::MINE; data[i][j]== nAdjacent(i , j);
A
Minnesweeper::loadGame();
fstream file saveFile("MineSweeperGame");
if player == s
saveFile.close
continue the game
END

Major Variables:

Type	Variable	Description	Location	Library
int	**rows**	This variable sets the number of rows for the game board (0-10)	Line number 20	**GameBoard.h**
	cols	This sets the number of columns for the size of the game board	Line number 21	**GameBoard.h**
	row	This gives a value to the rows variable in the class	Line number 16	**GameBoard.cpp**
	col	This gives a value to the cols variable in the class	Line number 17	**GameBoard.cpp**
	diff	This variable is used to store the difficulty level of the game Easy, normal or Hard when the player enters the choice.	Line number 54	**GameBoard.cpp**
	****data**	This 2D pointer allocates the address of the game board and stores the address of the rows and cols variables	Line number 19	**GameBoard.h**

enum	**difficulty**	This enum holds the difficulty levels of the game.	Line number 21	**Minesweeper.h**
	Squares	This enum holds the Empty. Loser, mines and clear variables.	Line number23	**Minesweeper.h**

Images: this picture shows how the Game Board looks at the beginning of the game



There we can see the three difficulty levels and the size of the game board of 10 x 10 and you have the option to save the game and then load it.

Pseudo Code;

/// dinamically create a Minesweeper

/// Create each column

/// Function returns true if input was valid

/// make sure that the number of mines does not exceed

/// the number of spots available and that mines exist

/// Get the user name

/// ask user if they want to play

/// play if answer is yes

/// Get game information from user

/// Get new data only if user wants to continue

/// Information was invalid

/// Play a game of minesweeper

/// User inputs how many rows and columns and the difficulty

/// Select the row

/// User wants to save the game

/// save the game and exit

/// check bounds

/// Select the column

/// Prepare to print completed Minesweeper

/// Print the complete Minesweeper

/// Function gets the user name as a string converts it to a char array

/// for the 1d dynamic array requirement

/// Function that clears the grid on which game will be played

/// Make sure each square is empty

/// Function return the Minesweeper::Difficulty type from

/// the int variable

/// Function prints the Minesweeper with spaces hidden

/// Print the column index

/// Pad initial output of column indicator

/// KEEP EMPTY spaces and MINEs hidden

/// print out the CLEARED area

/// Function returns the number of mines to set based on Difficulty

/// not on first or last row or first or last column

/// most of the searches take place in this area

/// on the first row, not on first or last column

/// on the last row, not on first or last column

/// on the first column, not on first or last row

/// search to the right

/// on the last column, not on first or last row

/// search to the left

/// top left corner

/// Function returns true if

/// there are 0 landmines adjacent to selected square

/// Clear an area whose values are CLEAR

/// i.e 0 adjacent mines

/// Function shows how many mines are adjacent to selected square

/// for the entire Minesweeper

/// Function reveals what is underneath the square that the user has selected

/// and whether to continue based on what is revealed

/// i.e selecting a mine means you lost, game over

/// Square had adjacent mine

/// reveal the number to the user

/// Function checks whether the player has won

/// if there are no EMPTY spaces left the game is won

/// Function finds the perimeter of the cleared areas

/// Function prints the data variable from the Minesweeper structure

/// writen to a binary file

///print();

/// This is the class that holds the Minesweeper

/// as well as the associated flags that occur when

/// a user selects a square

/// output this if user tries to set negative rows or columns

Code

Abtracts.h

```
class Abstracts {
   protected:
   virtual void setRows(int)=0;
   virtual void setCols(int)=0;
   virtual int getRows() const =0;
   virtual int getCols() const =0;
   virtual void setUpG()=0;
   virtual void print() const = 0;
};
```

GameBoard.h

```
#include "Abstracts.h"

/// Base class for games that require an n*m array such as minesweeper

class GameBoard: public Abstracts{
private:

protected:
   int **data;
   int rows;
   int cols;
   virtual void create(int, int);

public:
```

```
    class wrong{};

    GameBoard(int rows, int cols) {create(rows,cols);clear();}
    virtual ~GameBoard(){destroy();}
    virtual void destroy();
    virtual void setRows(int);
    virtual void setCols(int);
    virtual int getRows() const {return rows;}
    virtual int getCols() const {return cols;}
    virtual void clear();
    virtual void setUpG();//setup the game
    virtual void loadGame();
    virtual void print() const;
    int* operator[](int index) { return data[index];}
    int* operator[](int index) const { return data[index];}
};
```

GameBoard.cpp

```
void GameBoard::create(int row, int col) {
    /// dinamically create a Minesweeper
    rows=row;
    cols = col;

    /// Set up the rows
    data = new int *[rows];

    /// Create each column
    for (int row = 0; row != rows; ++row)
```

```
        data[row] = new int [cols];
}

/// Function resets the GameBoard to initial in order to use it again
void GameBoard::clear() {
   for (int i = 0; i != rows; ++i)
        for (int j = 0; j != cols; ++j)
            data[i][j] = 0;
}

/// Function deallocates memory
void GameBoard::destroy() {
   /// delete each dynamically allocated row
   for (int i = 0; i != rows; ++i)
        delete[] data[i];
   /// delete the dynamically allocated structure
   delete data;
}

void GameBoard::setRows(int row) {
   if ( row <= 0 )
        throw wrong();
   rows = row;
}

void GameBoard::setCols(int col) {
   if (col <= 0 )
```

```
        throw wrong();
    cols = col;
}

void GameBoard::print() const {
    for (int i = 0; i != rows; ++i){
        for (int j = 0; j != cols; ++j) {
            std::cout << data[i][j] << " ";
        }
        std::cout << std::endl;
    }
}

void GameBoard::loadGame() {
    cout << "Calling the the base class loadGame. "
          " should be calling the derived class version.\n";
}

void GameBoard::setUpG() {
    cout <<"Calling the wrong function. Use polymorphism to call the derived"
    "class setUp function\n";
}
```

Minesweeper.h

```
#ifndef MINESWEEPER_H
#define          MINESWEEPER_H

#include <string>
```

```
#include "GameBoard.h"

/// This is the class that holds the Minesweeper
/// as well as the associated flags that occur when
/// a user selects a square
class Minesweeper: public GameBoard {
private:

   /// Determines how many mines to set
   enum Difficulty {EASY, NORMAL, HARD};
   /// Flags representing various square possibilities
   enum Squares {EMPTY=10, MINE, CLEAR, LOSER};



   /// number of mines
   int mines;

   void create(int, int);

   //void destroy();
   Minesweeper::Difficulty intToDiff(int);
   bool isValidIn() const;
   int nMines(Minesweeper::Difficulty) const;
   void setMines();
   void setSquares();
   int nAdjacent(int, int, int = Minesweeper::MINE) const;
   bool isClear(int, int) const;
```

```
    void setPerim();
    void showZeros(int, int);
    bool hasWon() const;
    bool cont(int, int);
    void prompt();
    char *userName();

public:
    Minesweeper(int row, int col):GameBoard(row, col)
                        {clear();}

    void setRows(int);
    void setCols(int);
    int getRows() const {return rows;}
    int getCols() const {return cols;}
    void print() const;
    void prntObscr() const;
    void setUpG();//set up the game
    void playGame();
    void clear();
    void saveGame();
    void rules();
    void loadGame();
    int getMines() const { return mines;}

    Minesweeper& operator=(const Minesweeper&);
};
```

```
#endif  /* MINESWEEPER_H */
```

Minesweeper.cpp

```
#include <fstream>
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <string>
#include <iomanip>
#include "Minesweeper.h"

using namespace std;



void Minesweeper::create(int row, int col) {
   /// dinamically create a Minesweeper
   rows=row;
   cols = col;

   /// Set up the rows
   data = new int *[rows];

   /// Create each column
   for (int row = 0; row != rows; ++row)
      data[row] = new int [cols];
}
```

```
void Minesweeper::setRows(int row) {
   if ( row <= 0 )
      throw wrong();
   rows = row;
}

void Minesweeper::setCols(int col) {
   if (col <= 0 )
      throw wrong();
   cols = col;
}

void Minesweeper::prompt() {
   cout << "Enter the number of rows(1-10)\n"<<endl;
   cout<<"Area will be rowsxrows in size: "<<endl;
   int row;
   cin >> row;
   /// invalid sizes
   if (row > 10 || row < 1)
      throw wrong();
   rows = row;
   cols = row;
   int diff;
   cout << "Enter the difficulty\n"
   "0=Easy\t 1=Normal\t 2=Hard\n";
   cin >> diff;
   mines = nMines(intToDiff(diff));
```

```
}

/// Function returns true if input was valid
bool Minesweeper::isValidIn() const{
   /// make sure that the number of mines does not exceed
   /// the number of spots available and that mines exist
   return (((rows * cols) > mines) && (mines>0));
}

void Minesweeper::setUpG() {
   /// Get the user name
   char *player = userName();
   /// ask user if they want to play
   cout << "Hello " << player
        << ", Would you like to play a game of minesweeper?\n"
          "Hit 'y' if yes\n";
   char ans;
   cin >> ans;

   /// play if answer is yes
   if (ans == 'y') {
      cout << "would you like load previous settings (enter 'y' if yes enter 'n' if not)";
      char ans2;
      cin >> ans2;
      if ( ans2 == 'y') {
         loadGame();
      }
```

```
    else
        /// Get game information from user
        prompt();
    if (isValidIn()) {
        while (ans == 'y' && isValidIn()) {
            playGame();
            cout << endl;
            cin.ignore();
            cout << "Would you like to play again " << player << "? ";
            cin >> ans;
            cout << endl;
            /// Get new data only if user wants to continue
            if (ans =='y') {
                prompt();
                clear();
            }
        }
    }
    /// Information was invalid
    else
        throw wrong();
}
cout << "Game is Over.\n";

/// Cleanup
delete player;
```

```
    cout << endl;
    cout << "Goodbye\n";
}

/// Play a game of minesweeper
/// User inputs how many rows and columns and the difficulty
void Minesweeper::playGame() {
    setMines();
    prntObscr();
    char row, col;
    int turn = 0;
    int initialTime = static_cast<unsigned int>(time(0));
    do {
        int begTime = static_cast<unsigned int>(time(0));
        cout << "Turn: " << turn++ << endl;
        /// Select the row
        do {
            cout << "Enter s to save the settings and exit\n";
            cout << "Enter the row " << 0 << "-" << rows-1 << ": ";
            cin >> row;
            /// User wants to save the game
            /// save the game and exit
            if ( row == 's') {
                saveGame();
                return;
            }
            /// check bounds
```

```
    } while (row < 0 || row >= rows);
    /// Select the column
    do {
        cout << "Enter the column " << 0 << "-" << cols-1 << ": ";
        cin >> col;
        /// check bounds
    } while (col < 0 || col >= cols);

    /// endTime
    int endTime = static_cast<unsigned int>(time(0));
    cout << "Turn took: " << endTime - begTime << " seconds.\n";
    cout << endl;
} while (cont(row, col) && !hasWon());

/// Prepare to print completed Minesweeper
if (hasWon()) {
    cout << "You win\n";
    setSquares();
}
else{
    cout << "You have, lost\n";
    setSquares();
    data[row][col]= Minesweeper::LOSER;
}

int finalTime = static_cast<unsigned int>(time(0));
cout << "Game was completed in " << finalTime - initialTime << " seconds.\n";
```

```
    /// Print the complete Minesweeper
    print();
}

/// Function gets the user name as a string converts it to a char array
/// for the 1d dynamic array requirement
char* Minesweeper::userName() {
    cout << "Enter your username: ";
    string in;
    cin >> in;

    typedef string::size_type sType;
    sType size = in.size();
    /// make room for '\0'
    char *name = new char[size+1];
    for (sType i = 0; i != size; ++i) {
        *(name+i) = in[i];
    }
    *(name+size+1) = '\0';

    return name;
}

/// Function that clears the grid on which game will be played
void Minesweeper::clear() {
    /// Make sure each square is empty
```

```
    for (int i = 0; i != rows; ++i)
        for (int j = 0; j != rows; ++j)
            data[i][j] = Minesweeper::EMPTY;
}

/// Function return the Minesweeper::Difficulty type from
/// the int variable
Minesweeper::Difficulty Minesweeper::intToDiff(int choice) {
    switch (choice) {
        case (0):
            return Minesweeper::EASY;
            break;
        case (1):
            return Minesweeper::NORMAL;
            break;
        case (2):
            return Minesweeper::HARD;
        default:
            return Minesweeper::EASY;
            break;
    }
}

/// Functions prints the Minesweeper with all the squares revealed.
/// used mostly after player loses
void Minesweeper::print() const {
    cout << "Here's what the board looked like\n";
```

```
   for (int row = 0; row != rows; ++row){
      for (int col = 0; col != cols; ++col) {
         ///
         if ( *(*(data+row) + col) == Minesweeper::LOSER)
            cout << "t  ";
         else if (*(*(data+row) + col) == Minesweeper::MINE)
            cout << "x  ";
         else if (!isClear(row, col))
            cout << nAdjacent(row, col) << "  ";
         else
            cout << "0  ";
      }
      cout << endl;
   }
   cout << endl;
}

/// Function prints the Minesweeper with spaces hidden
void Minesweeper::prntObscr() const{
   /// Print the column index
   for (int i = 0; i != cols; ++i){
      /// Pad initial output of column indicator
      if (i==0)
         cout << "  ";
      cout << setw(3) << i;
   }
   cout << endl;
```

```
   for (int row = 0; row != rows; ++row){
      for (int col = 0; col != cols; ++col){
         if(col == 0 && row < 10) cout << row << "  ";
         if (col == 0 && row >= 10) cout << row << " ";
         /// KEEP EMPTY spaces and MINEs hidden
         if (data[row][col] == Minesweeper::EMPTY ||
            data[row][col] == Minesweeper::MINE)
            cout << setw(3) << right  << "* ";
         /// print out the CLEARed area
         else if (data[row][col] == Minesweeper::CLEAR)
            cout << setw(2)<< 0 << " ";
         /// Print out the actual value of the square
         else
            cout << setw(2)<< data[row][col] << " ";
      }
      cout << endl;
   }
   cout << endl;
}

/// Function returns the number of mines to set based on Difficulty
int Minesweeper::nMines(Minesweeper::Difficulty d) const {
   if (d==Minesweeper::EASY)
      return (rows*cols)/10;
   else if (d==Minesweeper::NORMAL)
      return (rows*cols)/5;
   else
```

```
    return (rows*cols)/3;
}

/// Function places mines in grid
void Minesweeper::setMines() {
  int minecpy = mines;
  /// keep looping through Minesweeper until all mines are set
  while (minecpy) {
    for (int i = 0; i != rows; ++i) {
      for (int j = 0; j != cols; ++j) {
        /// place mines if result of rand()%15 == 0
        if ((rand() % 100) % 10 == 0){
          ///only place mines if mines are still available
          /// and current space is empty
          if (minecpy && data[i][j] == Minesweeper::EMPTY) {
            /// set the mine
            data[i][j] = Minesweeper::MINE;
            /// decrement number of mines available
            --minecpy;
          }
        }
      }
    }
  }
}

/// Function returns how  many 'flag' elements surround a given square
```

```
int Minesweeper::nAdjacent(int row, int col, int FLAG) const{
   int nAd=0;            /// the number of adjacent mines

   /// not on first or last row or first or last column
   /// most of the searches take place in this area
   if ( row > 0 && col > 0 && row < rows-1 && col < cols-1) {
      /// search the 3x3 grid surrounding a cell
      for (int i = row-1; i <= row+1; ++i) {
         for (int j = col-1; j <= col+1; ++j)
            if (data[i][j] == FLAG)
               ++nAd;
      }
   }
   /// on the first row, not on first or last column
   else if ( row == 0 && col > 0 && col < cols - 1) {
      for (int i = row; i <= row+1; ++i) {
         for (int j = col-1; j <= col+1; ++j)
            if (data[i][j] == Minesweeper::MINE)
               ++nAd;
      }
   }
   /// on the last row, not on first or last column
   else if ( row == rows-1 && col > 0 && col < cols - 1) {
      for (int i = row-1; i <= row; ++i) {
         for (int j = col-1; j <= col+1; ++j)
            if (data[i][j] == Minesweeper::MINE)
               ++nAd;
```

```
    }
}
/// on the first column, not on first or last row
/// search to the right
else if ( col == 0 && row > 0 && row < rows - 1) {
    for (int i = row-1; i <= row+1; ++i) {
        for (int j = col; j <= col+1; ++j)
            if (data[i][j] == Minesweeper::MINE)
                ++nAd;
    }
}
/// on the last column, not on first or last row
/// search to the left
else if ( col == cols-1 && row > 0 && row < rows - 1) {
    for (int i = row-1; i <= row+1; ++i) {
        for (int j = col-1; j <= col; ++j)
            if (data[i][j] == Minesweeper::MINE)
                ++nAd;
    }
}
/// top left corner
else if (row == 0 && col == 0) {
    if (data[row][col+1] == Minesweeper::MINE) ++nAd;
    if (data[row+1][col] == Minesweeper::MINE) ++nAd;
    if (data[row+1][col+1] == Minesweeper::MINE) ++nAd;
}
/// top right corner
```

```
    else if (row == 0 && col == cols-1) {
        if (data[row][col-1] == Minesweeper::MINE) ++nAd;
        if (data[row+1][col] == Minesweeper::MINE) ++nAd;
        if (data[row+1][col-1] == Minesweeper::MINE) ++nAd;
    }
    /// bottom left corner
    else if (row == rows-1 && col == 0) {
        if (data[row-1][col] == Minesweeper::MINE) ++nAd;
        if (data[row-1][col+1] == Minesweeper::MINE) ++nAd;
        if (data[row][col+1] == Minesweeper::MINE) ++nAd;
    }
    /// bottom right corner
    else if (row == rows-1 && col == cols-1) {
        if (data[row-1][col-1] == Minesweeper::MINE) ++nAd;
        if (data[row-1][col] == Minesweeper::MINE) ++nAd;
        if (data[row][col-1] == Minesweeper::MINE) ++nAd;
    }
    /// return number of mines from appropriate if statement
    return nAd;
}

/// Function returns true if
/// there are 0 landmines adjacent to selected square
bool Minesweeper::isClear(int row, int col) const {
    if (nAdjacent(row, col))
        return false;          /// nAdjacent returned 1 or more
    return true;               /// nAdjacent returned 0
```

```
}

/// Clear an area whose values are CLEAR
/// i.e 0 adjacent mines
void Minesweeper::showZeros(int row, int col) {
   /// check bounds
   if ( row >= rows || row < 0 || col >= cols || col < 0)
      return;
   if (isClear(row, col) && data[row][col] != Minesweeper::CLEAR){
      data[row][col] = Minesweeper::CLEAR;
      /// go up one row
      showZeros(row+1, col);
      /// go down one row
      showZeros(row-1, col);
      /// go right one col
      showZeros(row, col+1);
      /// go left one col
      showZeros(row, col-1);
   }
   /// space was not clear or already shown
   else
      return;
}

/// Function shows how many mines are adjacent to selected square
/// for the entire Minesweeper
void Minesweeper::setSquares() {
```

```
    for (int i = 0; i != rows; ++i)
        for (int j = 0; j != cols; ++j)
            /// don't look for adjacent mines in areas where
            /// mine is already located
            if (data[i][j] != Minesweeper::MINE)
                data[i][j] = nAdjacent(i, j);
}

/// Function reveals what is underneath the square that the user has selected
/// and whether to continue based on what is revealed
/// i.e selecting a mine means you lost, game over
bool Minesweeper::cont(int row, int col) {
    /// check if user selected a losing square
    if (data[row][col] == Minesweeper::MINE)
        return false;

    /// Square is a zero, clear the surrounding area if necessary
    else if (isClear(row, col) ){
        showZeros(row, col); /// show cleared area
        setPerim();
        prntObscr();
        return true;
    }
    /// Square had adjacent mine
    /// reveal the number to the user
    else {
        data[row][col] = nAdjacent(row, col);
```

```
        prntObscr();
        return true;
    }
}

/// Function checks whether the player has won
/// if there are no EMPTY spaces left the game is won
bool Minesweeper::hasWon() const {
    for (int i = 0; i != rows; ++i)
        for (int j = 0; j != cols; ++j)
            /// if there are empty spaces player has not won
            if (data[i][j] == Minesweeper::EMPTY)
                return false;
    /// there were no empty spaces left. Player has won
    return true;
}

/// Function finds the perimeter of the cleared areas
void Minesweeper::setPerim() {
    for (int row = 0; row != rows; ++row ) {
        /// avoid searching at left and right edge of array
        for (int col = 0; col != cols; ++col) {
            /// when you're not on the bounds of the array
            if (row > 0 && row < rows-1
                && col > 0 &&  col < cols-1){
                if (data[row][col] == Minesweeper::CLEAR) {
                    /// check that the previous number has mines adjacent
```

```
                if (data[row][col-1] != Minesweeper::CLEAR)
                    data[row][col-1] = nAdjacent(row, col-1);
                /// check if the next number has mines adjacent
                if (data[row][col+1] != Minesweeper::CLEAR)
                    data[row][col+1] = nAdjacent(row, col+1);
                if (data[row-1][col] != Minesweeper::CLEAR)
                    data[row-1][col] = nAdjacent(row-1, col);
                /// check if the next number has mines adjacent
                if (data[row+1][col] != Minesweeper::CLEAR)
                    data[row+1][col] = nAdjacent(row+1, col);
                /// check the adjacent corners
                if (data[row-1][col-1] != Minesweeper::CLEAR)
                    data[row-1][col-1] = nAdjacent(row-1, col-1);
                if (data[row-1][col+1] != Minesweeper::CLEAR)
                    data[row-1][col+1] = nAdjacent(row-1, col+1);
                if (data[row+1][col-1] != Minesweeper::CLEAR)
                    data[row+1][col-1] = nAdjacent(row+1, col-1);
                if (data[row+1][col+1] != Minesweeper::CLEAR)
                    data[row+1][col+1] = nAdjacent(row+1, col+1);
            }
        }
    }
  }
}

void Minesweeper::saveGame() {
  fstream saveFile("gameSave", ios::out | ios::binary);
```

```
    saveFile.write(reinterpret_cast<char*>(this), sizeof(*this));
    saveFile.close();
}

/// Function prints the data variable from the Minesweeper structure
/// writen to a binary file
void Minesweeper::loadGame() {
    fstream saveFile("gameSave", ios::in | ios::binary);
    if (!saveFile.is_open())
        throw "No previous settings found\n";

    saveFile.read(reinterpret_cast<char*>(this), sizeof(*this));
    ///print();
    saveFile.close();



}

Minesweeper& Minesweeper::operator=(const Minesweeper &rhs) {
    create(rhs.getRows(), rhs.getCols());

    for (int i = 0; i != rhs.getRows(); ++i) {
        for (int j = 0; j != rhs.getCols(); ++j)
            data[i][j] = rhs[i][j];
    }
    return *this;
}
```

Templates.h

```
#ifndef TEMPLATES_H
#define	TEMPLATES_H

template<class T>
class Game {
private:
   T* p;
public:
   Game():p(0){}
   Game(T* t):p(t){}
   ~Game() {delete p;}
   Game<T>& operator=(const T&);
   operator bool(){return p;}
   T* operator->() const;
   T& operator*() const;
};

template<class T>
T* Game<T>::operator->() const {
   /// only return p if it points to something
   if (p)
      return p;
   return 0;
}

template<class T>
```

```
T& Game<T>::operator*() const{
   if (p)
      return *p;
}

template<class T>
Game<T>& Game<T>::operator=(const T& rhs) {
   p = &rhs;
}



#endif
```

Main.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <iomanip>

using namespace std;
//user libraries
#include "Minesweeper.h"
#include "Templates.h"

//Global constants

//function prototypes
```

```
void GameRules();
//execution begins here
int main(int argc, const char * argv[]) {
    ///call function here to show the game rules
    GameRules();

    ///Cast for random time seed generator
    srand(static_cast<unsigned int>(time(0)));

    Game<GameBoard> m(new Minesweeper(10,10));
    //throw the exeption
    try {
        m->setUpG();
    }
    ///error output
    catch (Minesweeper::wrong) {
        cout << "Size was invalid\n";
    }
    catch (const char* s) {
        cout << s << endl;
    }

    return 0;

    ///Exit stage right
}
```