Project 1

2048 Tile Game

CIS-17a 42448
Hodnett, Victoria
April 30, 2014

Introduction

2048 is an online game that was released in March 2014. It is the product of a bored web-developer, who described it as a clone of a previously released game called 1024.

The object of the game is to get the number 2048 by colliding identical numbered squares, which are then added.

This is really a game of strategy, as you try to move the tiles in such a way that adds tiles, and reduces the number of spaces filled. Once all tiles are filled and no legal moves are allowed, the game is over.

Game Play

At the start of the game you will be prompted to enter your name, to be stored for later recording of your statistics.

You will then be presented with a 4x4 board with two number 2's randomly placed, and all other squares will be filled with zeroes, indicating an empty square. At this point you are prompted on which direction you would like to shift tiles – up, down, left, or right. Once you make your choice, all numbered tiles will move in the direction you chose. If any tiles touch another of the same number, the tiles will "collide" and combine in addition. And then you continue to make another move.

Each collision will result in some power of 2. The object of game is to reach the number 2048. If you successfully obtain the number 2048 on your game board, you will be prompted on whether you would like to continue for more points or stop (points are awarded based on each addition you make). Otherwise, the game continues until there are no more legal moves on the board (there are no more blank spaces and none of the tiles are adjacent to like tiles).

When the game is over you will be shown your score. You will then be prompted on whether you would like to start another game with a new player. If you choose to, you will be prompted for a new name and a new board will appear. Whenever you choose not to start a new game, statistics for each player will be displayed: name, score, time elapsed, and whether they won or lost.

Summary

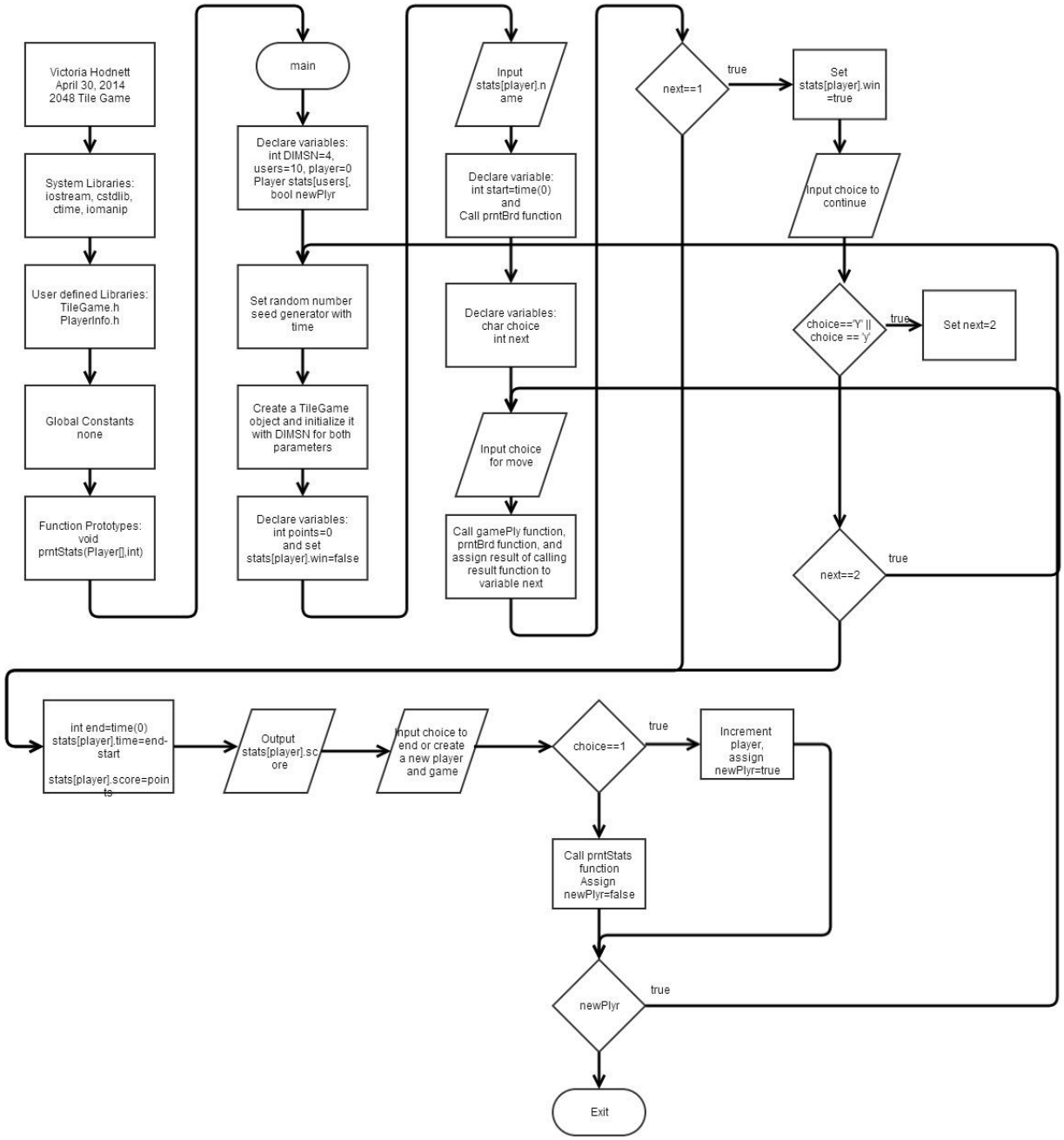
	main.cpp
Lines of Code	72
Comment Lines	59
Blank Lines	9
Total Lines	140
	PlayerInfo.h
Lines of Code	9
Comment Lines	9
Blank Lines	4
Total Lines	22
	TileGame.h
Lines of Code	27
Comment Lines	74
Blank Lines	5
Total Lines	106
	TileGame.cpp
Lines of Code	343
Comment Lines	20
Blank Lines	32
Total Lines	395
LINES FOR ENTIRE PROJECT	663

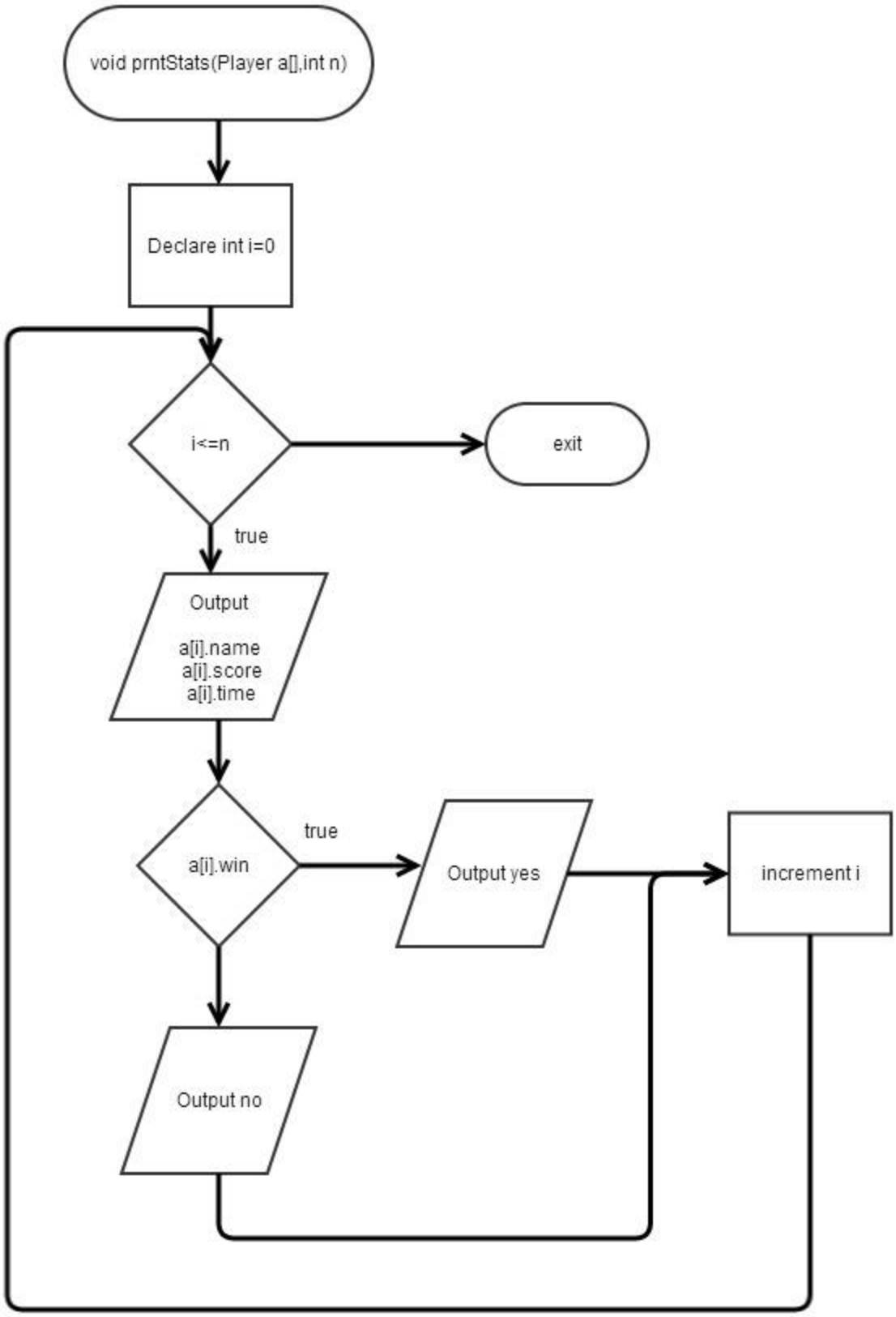
Completing the entire project took about two and a half weeks. The bulk of the program was very easy to implement, but I did face a problem when testing the game as the board filled up. Even when there were legal moves available, once the

board filled up the program crashed. I solved this by simplifying my if-statements in checking for like adjacent tiles.

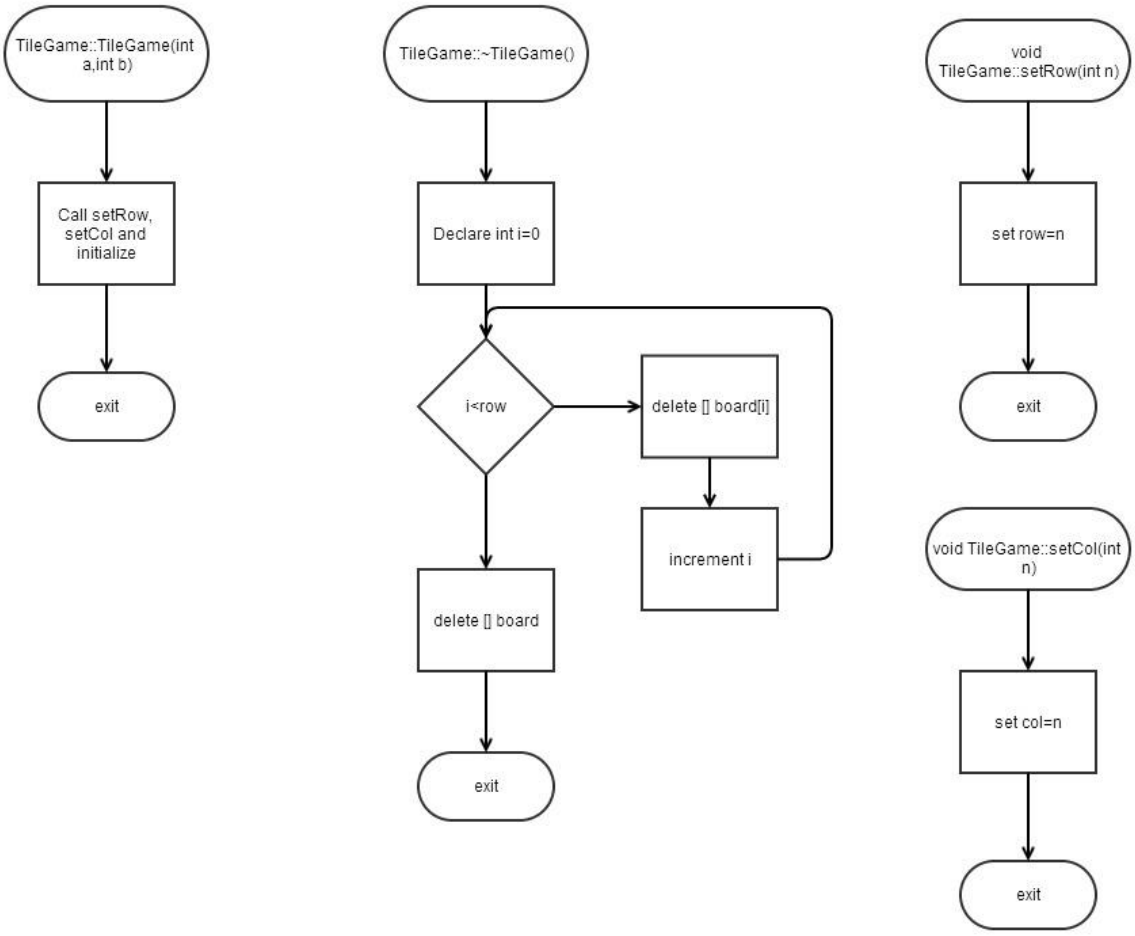
Flowcharts

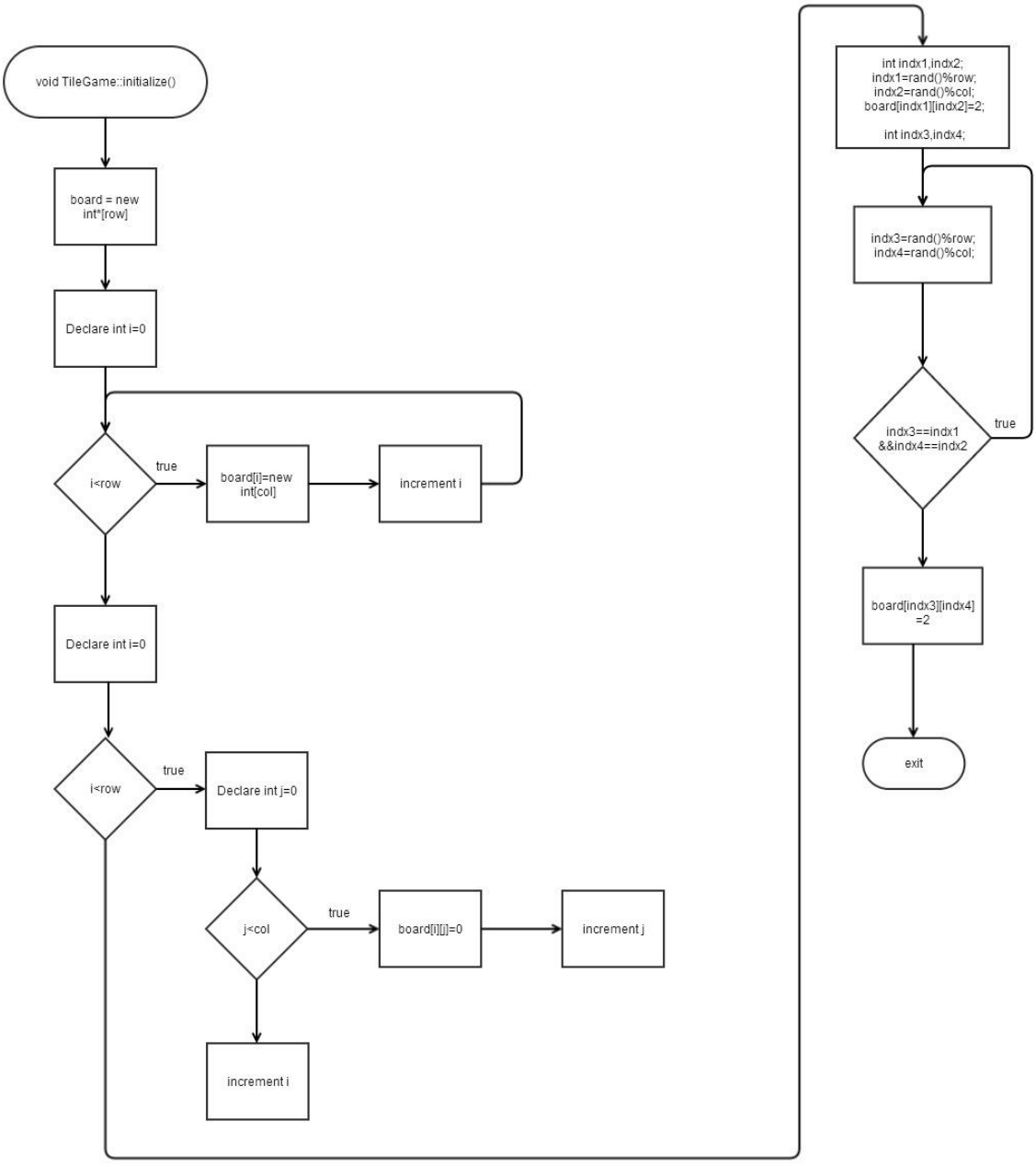
Main.cpp

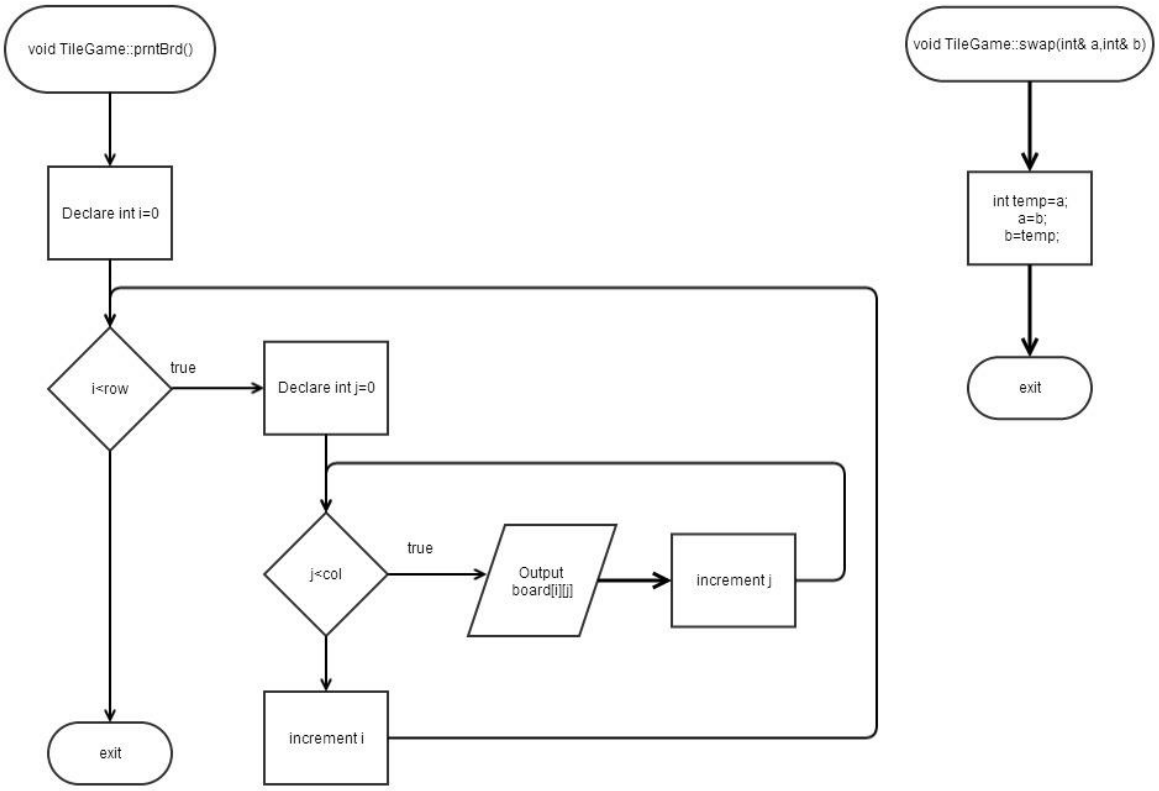

Victoria Hodnett
April 30, 2014
2048 Tile Game
System Libraries:
iostream, cstdlib,
ctime, iomanip
User defined Libraries:
TileGame.h
PlayerInfo.h
Global Constants
none
Function Prototypes:
void
prntStats(Player[],int)
main
Declare variables:
int DIMSN=4,
users=10, player=0
Player stats[users[,
bool newPlyr
Set random number
seed generator with
time
Create a TileGame
object and initialize it
with DIMSN for both
parameters
Declare variables:
int points=0
and set
stats[player].win=false
Input
stats[player].n
ame
Declare variable:
int start=time(0)
and
Call prntBrd function
Declare variables:
char choice
int next
Input choice
for move
Call gamePly function,
prntBrd function, and
assign result of calling
result function to
variable next
next==1
true
Set
stats[player].win
=true
Input choice to
continue
choice=='Y' ||
choice == 'y'
true
Set next=2
next==2
true
int end=time(0)
stats[player].time=end-
start
stats[player].score=poin
ts
Output
stats[player].sc
ore
Input choice to
end or create
a new player
and game
choice==1
true
Increment
player,
assign
newPlyr=true
Call prntStats
function
Assign
newPlyr=false
newPlyr
true
Exit

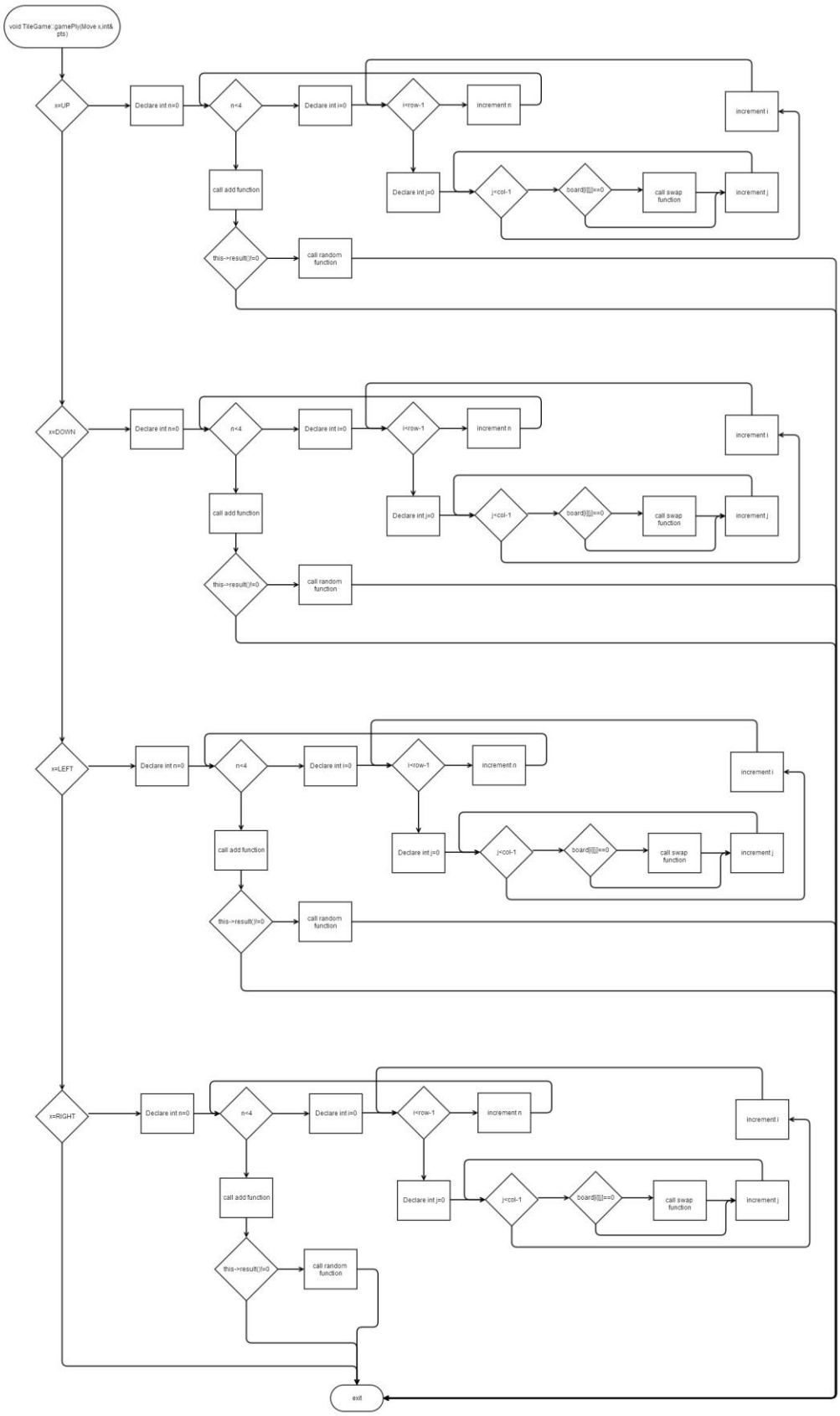

void prntStats(Player a[],int n)
Declare int i=0
i<=n
exit
true
Output
a[i].name
a[i].score
a[i].time
a[i].win
true
Output yes
increment i
Output no

TileGame.cpp



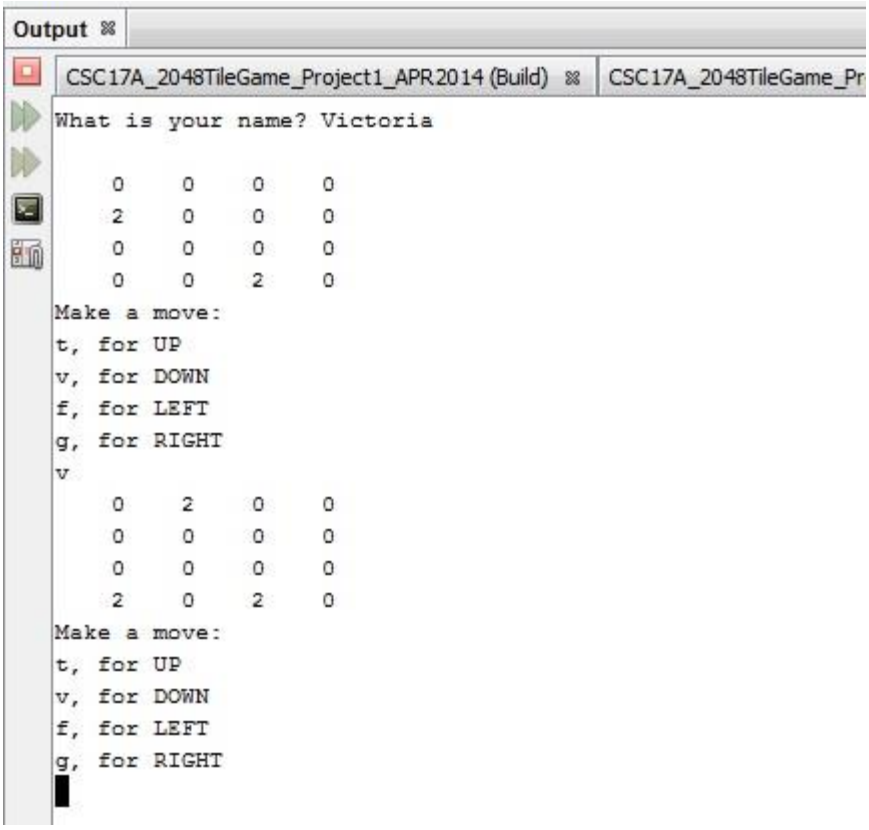

void TileGame::initialize()
board = new int*[row]
Declare int i=0
i<row
true
board[i]=new int[col]
increment i
Declare int i=0
i<row
true
Declare int j=0
j<col
true
board[i][j]=0
increment j
increment i
int indx1,indx2;
indx1=rand()%row;
indx2=rand()%col;
board[indx1][indx2]=2;
int indx3,indx4;
indx3=rand()%row;
indx4=rand()%col;
indx3==indx1 &&indx4==indx2
true
board[indx3][indx4] =2
exit

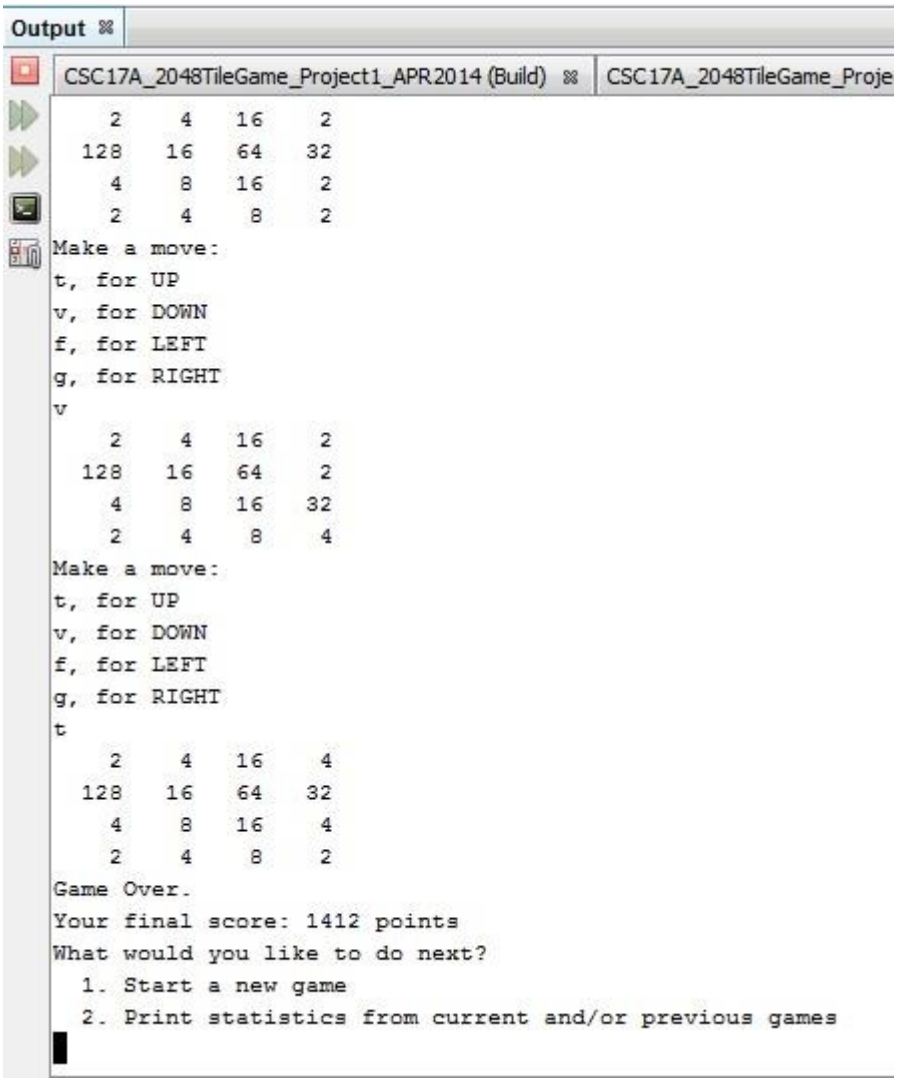

void TileGame::prntBrd()
Declare int i=0
i<row
true
Declare int j=0
j<col
true
Output
board[i][j]
increment j
increment i
exit
void TileGame::swap(int& a,int& b)
int temp=a;
a=b;
b=temp;
exit

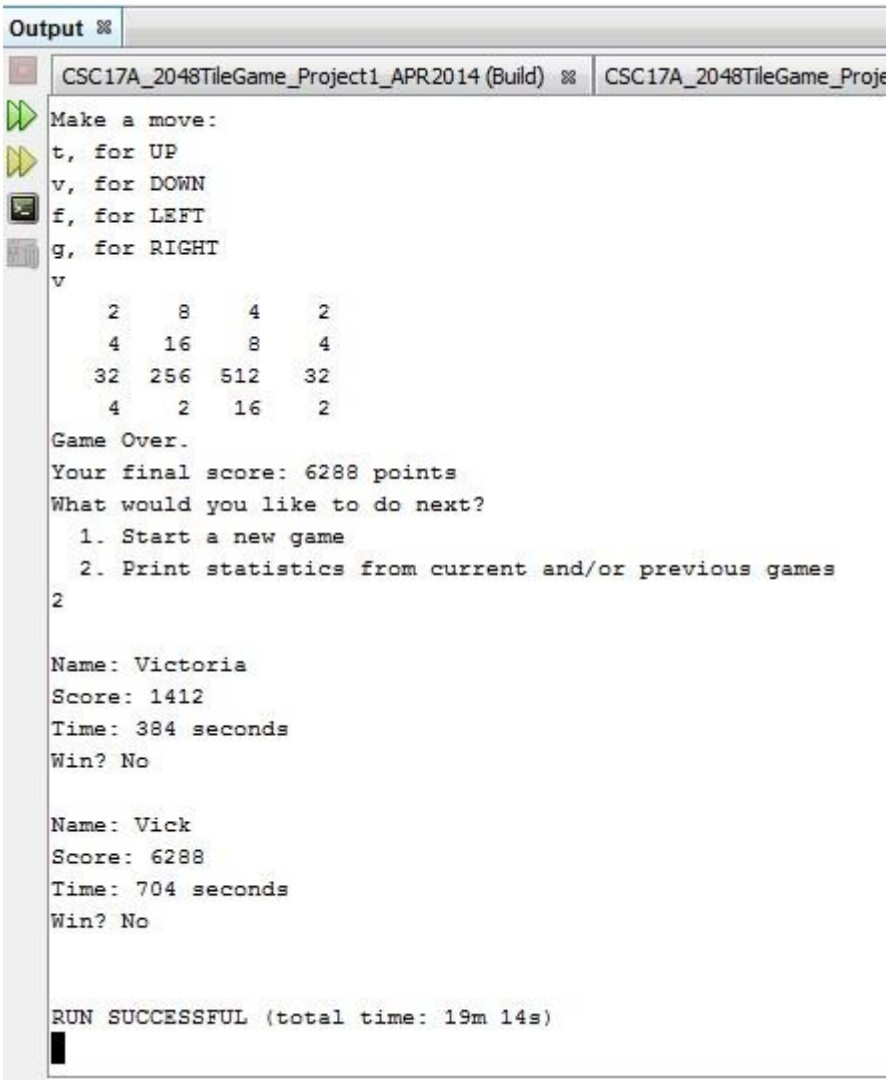

void TileGame::gamePly(Move x,int& pts)
x==UP
Declare int n=0
n<4
Declare int i=0
i<row-1
increment n
increment i
call add function
Declare int j=0
j<col-1
board[i][j]==0
call swap function
increment j
this->result()!=0
call random function
x==DOWN
Declare int n=0
n<4
Declare int i=0
i<row-1
increment n
increment i
call add function
Declare int j=0
j<col-1
board[i][j]==0
call swap function
increment j
this->result()!=0
call random function
x==LEFT
Declare int n=0
n<4
Declare int i=0
i<row-1
increment n
increment i
call add function
Declare int j=0
j<col-1
board[i][j]==0
call swap function
increment j
this->result()!=0
call random function
x==RIGHT
Declare int n=0
n<4
Declare int i=0
i<row-1
increment n
increment i
call add function
Declare int j=0
j<col-1
board[i][j]==0
call swap function
increment j
this->result()!=0
call random function
exit

*I did not flowchart the add, random or result functions of the TileGame class.

Sample Input/Output


Output
CSC17A_2048TileGame_Project1_APR2014 (Build)
CSC17A_2048TileGame_Pr
What is your name? Victoria
0 0 0 0
2 0 0 0
0 0 0 0
0 0 2 0
Make a move:
t, for UP
v, for DOWN
f, for LEFT
g, for RIGHT
v
0 2 0 0
0 0 0 0
0 0 0 0
2 0 2 0
Make a move:
t, for UP
v, for DOWN
f, for LEFT
g, for RIGHT


Output
CSC17A_2048TileGame_Project1_APR2014 (Build)
CSC17A_2048TileGame_Proje
2 4 16 2
128 16 64 32
4 8 16 2
2 4 8 2
Make a move:
t, for UP
v, for DOWN
f, for LEFT
g, for RIGHT
v
2 4 16 2
128 16 64 2
4 8 16 32
2 4 8 4
Make a move:
t, for UP
v, for DOWN
f, for LEFT
g, for RIGHT
t
2 4 16 4
128 16 64 32
4 8 16 4
2 4 8 2
Game Over.
Your final score: 1412 points
What would you like to do next?
1. Start a new game
2. Print statistics from current and/or previous games



Data Types Used

Data Type	Example	Description	Location
Int	points	Keep track of points earned	main()
Char	choice	Record user's choice for tile movement	main()
Bool	newPlyr	Determine new game and new player	main()
Player	stats[]	Used to store players	main()

string	name	Store player's name	PlayerInfo.h

C++ Constructs (Gaddis, 7th Edition)

Chapter	Construct	Location
2	Variables	main
	Data types	main
	Boolean expression	main
3	Input/output	main
	String	PlayerInfo.h
	Type casting	main
4	switch	TileGame::gamePly
	if, if-else, if-else-if	TileGame::add
5	Increment	TileGame::result
	Do-while/while	TileGame::initialize
	For loop	TileGame::~TileGame
	Counters	TileGame::random
6	Functions, function calls	TileGame.h,TileGame.cpp
	Call by value,return value	TileGame::setRow, TileGame::result
	Call by reference	TileGame::gamePly
7	Arrays, multidimensional	TileGame::initialize
11	Structures	main
	Array of structures	main
	Enumerated data types	TileGame.h
13	Classes	TileGame.h

References

I simply searched for and played the game a few times before planning out this project:

- http://www.2048tile.co/

I also looked up the description for the game:

- http://en.wikipedia.org/wiki/2048_(video_game)

Source Code

```
/*! \mainpage 2048 Tile Game
 *
 *  \section Introduction
 *   2048 is an online game that was released in March 2014. It is the product
 *   of a bored web-developer, who described it as a clone of a previously
 *   released game called 1024.
     The object of the game is to get the number 2048 by colliding identical
 *   numbered squares, which are then added.
 *   This is really a game of strategy, as you try to move the tiles in such a
 *   way that adds tiles, and reduces the number of spaces filled. Once all
 *   tiles are filled and no legal moves are allowed, the game is over.
 *
 *  \section Game Play
 *   At the start of the game you will be prompted to enter your name, to be
 *   stored for later recording of your statistics.
 *   You will then be presented with a 4x4 board with two number 2's randomly
 *   placed, and all other squares will be filled with zeroes, indicating an
 *   empty square. At this point you are prompted on which direction you would
 *   like to shift tiles – up, down, left, or right. Once you make your choice,
 *   all numbered tiles will move in the direction you chose. If any tiles touch
 *   another of the same number, the tiles will "collide" and combine in addition.
 *   And then you continue to make another move.
 *   Each collision will result in some power of 2. The object of game is to reach
 *   the number 2048. If you successfully obtain the number 2048 on your game board,
 *   you will be prompted on whether you would like to continue for more points
 *   or stop (points are awarded based on each addition you make). Otherwise, the
 *   game continues until there are no more legal moves on the board (there are no
 *   more blank spaces and none of the tiles are adjacent to like tiles).
 *   When the game is over you will be shown your score. You will then be prompted
 *   on whether you would like to start another game with a new player. If you
 *   choose to, you will be prompted for a new name and a new board will appear.
 *   Whenever you choose not to start a new game, statistics for each player will
 *   be displayed: name, score, time elapsed, and whether they won or lost.
 */

/*! \file main.cpp
 * File:   main.cpp
 * Author: Victoria Hodnett
 * Created on April 17, 2014, 9:51 PM
 * CSC17 A Project 1
 * 2048 Tile Game
 */

//System Libraries
#include <iostream>
#include <cstdlib>
#include <ctime>
```

```
using namespace std;

//User-defined Libraries
#include "TileGame.h"
#include "PlayerInfo.h"

//Global Constants

//Function Prototypes
void prntStats(Player[],int);

//Execution Begins Here
int main(int argc, char** argv) {
    //Declare variables
    int DIMSN=4; //Dimension length for both rows and columns of board
    int users=10; //Number of users available to store for current game play
    Player stats[users]; //Array to store multiple player statistics
    int player=0; //Counter to determine current player
    bool newPlyr; //validate creation of new player or not

    do{
        srand(static_cast<unsigned int>(time(0)));
        TileGame game(DIMSN,DIMSN);
        int points=0; //Counter to store points
        stats[player].win=false;

        //Start game
        cout << "What is your name? ";
        cin >> stats[player].name;
        cout << endl;
        int start=time(0);
        //Print board
        game.prntBrd();
        //Play game
        char choice;
        int next;
        do{
            cout << "Make a move:\n"
                    "t, for UP\n"
                    "v, for DOWN\n"
                    "f, for LEFT\n"
                    "g, for RIGHT\n";
            cin >> choice;
            game.gamePly(static_cast<Move>(choice),points);
            game.prntBrd();
            next=game.result();
            if(next==1){
                stats[player].win=true;
                cout << "Would you like to continue for more points? Y/N: ";
                cin >> choice;
                if(choice=='Y'||choice=='y')next=2;
            }
        }while(next==2);
        //Display Statistics
        int end=time(0);
        stats[player].time=end-start;
        stats[player].score=points;
        cout << "Your final score: " << stats[player].score << " points " << endl;
```

```
        //New Player?
        cout << "What would you like to do next? \n"
                "  1. Start a new game\n"
                "  2. Print statistics from current and/or previous games\n";
        cin >> choice;
        if(choice=='1'){
            ++player;
            newPlyr=true;
        }else{
            prntStats(stats,player);
            newPlyr=false;
        }
    }while(newPlyr);
    //Exit
    return 0;
}

/*! \fn prntStats
         *  \brief This function will print final statistics for all players.
 *
 *   \param a[] Player array of statistics
 *   \param n integer variable of how many total players created
*/
void prntStats(Player a[],int n){
    for(int i=0;i<=n;i++){
        cout << endl;
        cout << "Name: " << a[i].name << endl;
        cout << "Score: " << a[i].score << endl;
        cout << "Time: " << a[i].time << " seconds" << endl;
        cout << "Win? ";
        if(a[i].win)cout << "Yes " << endl;
        else cout << "No " << endl;
    }
    cout << endl;
}



/*
 * File:   PlayerInfo.h
 * Author: Victoria
 *
 * Created on April 18, 2014, 1:55 PM
 */

#ifndef PLAYERINFO_H
#define      PLAYERINFO_H

/*! \struct Player
    \brief This struct stores the statistics for each player.
*/

struct Player{
    string name; //!< Player name
    int score; //!< Player's final score
    int time; //!< Total time to complete game
    bool win; //!< Result of winning or losing the game
};
```

```
#endif /* PLAYERINFO_H */


/*
 * File:   TileGame.h
 * Author: Victoria
 *
 * Created on April 17, 2014, 9:51 PM
 */

#ifndef TILEGAME_H
#define       TILEGAME_H

/*! \class TileGame
    \brief Contains all fields functions
 *         necessary for game play
*/

/** Enumerated type for making a move
*/
enum Move{
    UP='t',   //!< Shift tiles up
    DOWN='v', //!< Shift tiles down
    LEFT='f', //!< Shift tiles left
    RIGHT='g' //!< Shift tiles right
};

class TileGame {
    private:
        int row; //!< Rows for board
        int col; //!< Columns for board
        int **board; //!< Two-dimensional array board for game play
        /*! \fn setRow
         *  \brief Mutator function to set the rows
         */
        void setRow(int);
        /*! \fn setCol
         *  \brief Mutator function to set the columns
         */
        void setCol(int);
        /*! \fn initialize
         *  \brief Initialize the board for game play
         *
         *  Board will be dynamically allocated, initialized with 0's
         *  and have two 2's randomly placed
         */
        void initialize();
    public:
        /*! \fn TileGame
         *  \brief Class Constructor
         *
         *  Set rows and columns, and initialize board
         *  \param a integer argument for rows
         *  \param b integer argument for columnns
         *  \sa setRow, setCol, initialize
         */
```

```
        TileGame(int a,int b);
        /*! \fn ~TileGame
         *  \brief Class Destructor
         *
         *  Delete allocated memory
         */
        ~TileGame();
        /*! \fn prntBrd
         *  \brief Print current state of the game board
         */
        void prntBrd();
        /*! \fn gamePly
         *  \brief The bulk of the game
         *
         *  Depending on the direction chosen to move tiles, this function
         *  will check for empty tiles, swap tiles, add identical adjacent
         *  tiles, and randomly place a 2 somewhere on the board if there is
         *  space.
         *  \param x enumerated argument type Move for player's move
         *  \param pts reference integer argument to store points
         *  \sa add, result, random
         */
        void gamePly(Move x,int& pts);
        /*! \fn add
         *  \brief Add touching tiles of same value after shifting tiles
         *
         *  \param x enumerated argument type Move for player's move
         *  \param pts reference integer argument to store points
         *  \sa swap
         */
        void add(Move x,int& pts);
        /*! \fn random
         *  \brief Randomly place a number 2 on the board in any available space
         */
        void random();
        /*! \fn swap
         *  \brief Swap tiles in the direction chosen in order to shift tiles
         *
         *  \param a integer argument for one tile's value
         *  \param b integer argument for another tile's value
         */
        void swap(int& a,int& b);
        /*! \fn result
         *  \brief Determine current state of game
         *
         *  This function checks for available moves and if the number 2048
         *  was obtained. Either you win, lose or continue with available moves.
         *  \return integer value indicating state of game
         */
        int result();
};

#endif /* TILEGAME_H */


/*
 * File:   TileGame.cpp
```

```
 * Author: Victoria
 *
 * Created on April 17, 2014, 9:51 PM
 */

//System libraries
#include <iostream>
#include <cstdlib>
#include <iomanip>
using namespace std;

//User-defined libraries
#include "TileGame.h"
#include "PlayerInfo.h"

TileGame::TileGame(int a,int b) {
    //Set dimensions
    setRow(a);
    setCol(b);
    //Initialize
    initialize();
}

TileGame::~TileGame() {
    for(int i=0;i<row;i++){
        delete [] board[i];
    }
    delete [] board;
}

void TileGame::setRow(int n){
    row=n;
}

void TileGame::setCol(int n){
    col=n;
}

void TileGame::initialize(){
    //Allocate memory
    board = new int*[row];
    for(int i=0;i<row;i++)
        board[i]=new int[col];

    //Initialize
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            board[i][j]=0;
        }
    }
    //Randomly choose starting pieces
    int indx1,indx2;
    indx1=rand()%row;
    indx2=rand()%col;
    board[indx1][indx2]=2;

    int indx3,indx4;
    do{
```

```
        indx3=rand()%row;
        indx4=rand()%col;
    }while(indx3==indx1&&indx4==indx2);
    board[indx3][indx4]=2;
}

void TileGame::prntBrd(){
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            cout << setw(5) << board[i][j];
        }
        cout << endl;
    }
}

void TileGame::gamePly(Move x,int& pts){
    switch(x){
        case UP:
            for(int n=0;n<4;n++){
                for(int i=0;i<row-1;i++){
                    for(int j=0;j<col;j++){
                        if(board[i][j]==0)
                            swap(board[i][j],board[i+1][j]);
                    }
                }
            }
            add(x,pts);
            if(this->result()!=0)
                random();
            break;
        case DOWN:
            for(int n=0;n<4;n++){
                for(int i=row-1;i>0;i--){
                    for(int j=0;j<col;j++){
                        if(board[i][j]==0)
                            swap(board[i][j],board[i-1][j]);
                    }
                }
            }
            add(x,pts);
            if(this->result()!=0)
                random();
            break;
        case LEFT:
            for(int n=0;n<4;n++){
                for(int i=0;i<row;i++){
                    for(int j=0;j<col-1;j++){
                        if(board[i][j]==0)
                            swap(board[i][j],board[i][j+1]);
                    }
                }
            }
            add(x,pts);
            if(this->result()!=0)
                random();
            break;
        case RIGHT:
            for(int n=0;n<4;n++){
```

```
                for(int i=0;i<row;i++){
                    for(int j=col-1;j>0;j--){
                        if(board[i][j]==0)
                            swap(board[i][j],board[i][j-1]);
                    }
                }
            }
            add(x,pts);
            if(this->result()!=0)
                random();
            break;
    }
}

void TileGame::swap(int& a,int& b){
    int temp=a;
    a=b;
    b=temp;
}

void TileGame::random(){
    int num;
    int two=0,zero=0;
    for(int i=0;i<row;i++)
        for(int j=0;j<col;j++){
            if(board[i][j]==2)two++;
            if(board[i][j]==0)zero++;
        }
    if(two>0)num=2;
    else if(zero>2)num=2;
    else num=4;
    //Choose next pieces
    int indx1,indx2;
    //Check availability
    int avail=0;
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            if(board[i][j]==0)avail++;
        }
    }
    if(avail>0){
        do{
            indx1=rand()%row;
            indx2=rand()%col;
        }while(board[indx1][indx2]!=0);
        board[indx1][indx2]=num;
    }
}

void TileGame::add(Move x,int& pts){
    switch(x){
        case UP:
            for(int j=0;j<col;j++){
                if(board[0][j]==board[1][j]){
                    board[0][j]+=board[1][j];
                    board[1][j]=0;
                    pts+=board[0][j];//Award points
                    swap(board[1][j],board[2][j]);
```

```
                swap(board[2][j],board[3][j]);

                if(board[1][j]==board[2][j]){
                    board[1][j]+=board[2][j];
                    board[2][j]=0;
                    pts+=board[1][j];//Award points
                    swap(board[2][j],board[3][j]);

                    if(board[2][j]==board[3][j]){
                        board[2][j]+=board[3][j];
                        board[3][j]=0;
                        pts+=board[2][j];//Award points
                    }
                }else if(board[2][j]==board[3][j]){
                    board[2][j]+=board[3][j];
                    board[3][j]=0;
                    pts+=board[2][j];//Award points
                }
            }else if(board[1][j]==board[2][j]){
                board[1][j]+=board[2][j];
                board[2][j]=0;
                pts+=board[1][j];//Award points
                swap(board[2][j],board[3][j]);

                if(board[2][j]==board[3][j]){
                    board[2][j]+=board[3][j];
                    board[3][j]=0;
                    pts+=board[2][j];//Award points
                }

            }else if(board[2][j]==board[3][j]){
                board[2][j]+=board[3][j];
                board[3][j]=0;
                pts+=board[2][j];//Award points;
            }
        }
        break;
    case DOWN:
        for(int j=0;j<col;j++){
            if(board[3][j]==board[2][j]){
                board[3][j]+=board[2][j];
                board[2][j]=0;
                pts+=board[3][j];//Award points;
                swap(board[2][j],board[1][j]);
                swap(board[1][j],board[0][j]);

                if(board[2][j]==board[1][j]){
                    board[2][j]+=board[1][j];
                    board[1][j]=0;
                    pts+=board[2][j];//Award points;
                    swap(board[1][j],board[0][j]);

                    if(board[1][j]==board[0][j]){
                        board[1][j]+=board[0][j];
                        board[0][j]=0;
                        pts+=board[1][j];//Award points;
                    }
                }else if(board[1][j]==board[0][j]){
```

```
                board[1][j]+=board[0][j];
                board[0][j]=0;
                pts+=board[1][j];//Award points;
            }
        }else if(board[2][j]==board[1][j]){
            board[2][j]+=board[1][j];
            board[1][j]=0;
            pts+=board[2][j];//Award points;
            swap(board[1][j],board[0][j]);

            if(board[1][j]==board[0][j]){
                board[1][j]+=board[0][j];
                board[0][j]=0;
                pts+=board[1][j];//Award points;
            }

        }else if(board[1][j]==board[0][j]){
            board[1][j]+=board[0][j];
            board[0][j]=0;
            pts+=board[1][j];//Award points;
        }
    }
    break;
case LEFT:
    for(int i=0;i<row;i++){
        if(board[i][0]==board[i][1]){
            board[i][0]+=board[i][1];
            board[i][1]=0;
            pts+=board[i][0];//Award points;
            swap(board[i][1],board[i][2]);
            swap(board[i][2],board[i][3]);

            if(board[i][1]==board[i][2]){
                board[i][1]+=board[i][2];
                board[i][2]=0;
                pts+=board[i][1];//Award points;
                swap(board[i][2],board[i][3]);

                if(board[i][2]==board[i][3]){
                    board[i][2]+=board[i][3];
                    board[i][3]=0;
                    pts+=board[i][2];//Award points;
                }
            }else if(board[i][2]==board[i][3]){
                board[i][2]+=board[i][3];
                board[i][3]=0;
                pts+=board[i][2];//Award points;
            }
        }else if(board[i][1]==board[i][2]){
            board[i][1]+=board[i][2];
            board[i][2]=0;
            pts+=board[i][1];//Award points;
            swap(board[i][2],board[i][3]);

            if(board[i][2]==board[i][3]){
                board[i][2]+=board[i][3];
                board[i][3]=0;
                pts+=board[i][2];//Award points;
```

```
                    }

                }else if(board[i][2]==board[i][3]){
                    board[i][2]+=board[i][3];
                    board[i][3]=0;
                    pts+=board[i][2];//Award points;
                }
            }
            break;
        case RIGHT:
            for(int i=0;i<row;i++){
                if(board[i][3]==board[i][2]){
                    board[i][3]+=board[i][2];
                    board[i][2]=0;
                    pts+=board[i][3];//Award points;
                    swap(board[i][2],board[i][1]);
                    swap(board[i][1],board[i][0]);

                    if(board[i][2]==board[i][1]){
                        board[i][2]+=board[i][1];
                        board[i][1]=0;
                        pts+=board[i][2];//Award points;
                        swap(board[i][1],board[i][0]);

                        if(board[i][1]==board[i][0]){
                            board[i][1]+=board[i][0];
                            board[i][0]=0;
                            pts+=board[i][1];//Award points;
                        }
                    }else if(board[i][1]==board[i][0]){
                        board[i][1]+=board[i][0];
                        board[i][0]=0;
                        pts+=board[i][1];//Award points;
                    }
                }else if(board[i][2]==board[i][1]){
                    board[i][2]+=board[i][1];
                    board[i][1]=0;
                    pts+=board[i][2];//Award points;
                    swap(board[i][1],board[i][0]);

                    if(board[i][1]==board[i][0]){
                        board[i][1]+=board[i][0];
                        board[i][0]=0;
                        pts+=board[i][1];//Award points;
                    }

                }else if(board[i][1]==board[i][0]){
                    board[i][1]+=board[i][0];
                    board[i][0]=0;
                    pts+=board[i][1];//Award points;
                }
            }
            break;
    }
}

int TileGame::result(){
    //Count blanks
```

```
    int blanks=0;
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            if(board[i][j]==0)blanks++;
        }
    }
    //Check if any are the winning number
    int winNum=0;
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            if(board[i][j]==2048)winNum++;
        }
    }
    //If found 2048, WON
    if(winNum>0){
        cout << "Congratulations, you win!" << endl;
        return 1;
    }else{
        //Check for available moves left
        int availMvs=0;
        if(blanks>0)return 2;//Continue game
        else{

            if(board[0][0]==board[1][0]||board[0][0]==board[0][1]
               ||board[0][1]==board[0][2]||board[0][1]==board[1][1]
                ||board[0][2]==board[0][3]||board[0][2]==board[1][2]
                 ||board[1][0]==board[1][1]||board[1][0]==board[2][0]
                  ||board[1][1]==board[1][2]||board[1][1]==board[2][1]
                   ||board[1][2]==board[1][3]||board[1][2]==board[2][2]
                    ||board[1][3]==board[0][3]||board[1][3]==board[2][3]
                     ||board[2][0]==board[2][1]||board[2][0]==board[3][0]
                      ||board[2][1]==board[2][2]||board[2][1]==board[3][1]
                       ||board[2][2]==board[2][3]||board[2][2]==board[3][2]
                        ||board[3][1]==board[3][0]||board[3][1]==board[3][2]
                         ||board[3][3]==board[2][3]||board[3][3]==board[3][2])
                availMvs++;
            }
            //Available moves?
            if(availMvs>0)return 2;//Continue game
            else{
                cout << "Game Over." << endl;
                return 0;
            }
        }
}
```