**Project 2**

**[Connect Four]**

**CIS17A**

**Name: Prieto, Heriberto**

**Date: 06/04/2017**

**Introduction**

Title: Connect Four

Classic board game consisting of two players.

Choose a column to drop a palette. Attempt to connect 4 palettes in any format before your opponent.

When either player 1 or player 2 have connected four, they have won and the game is over.

**Summary**

Project Size: 245 lines(Including spacing)

Number of Variables: 7

Number of Methods: 19

Conceptually this game isn’t too difficult to build. Main focus for this project was to build an efficient project with respect to time and space. I used ternary operators as much as possible, attempted to steer from if-statements as much as possible. After looking over Bitcoin and Node.js module source code, I was extremely impressed with their style and attempted to replicate their cleanliness for easy reading. Im EXTREMELY comfortable with working with 2D Arrays now and I hope it shows. I implemented template classes, inheritance, polymorphism and overloading an operator.

**Pseudo Code**

-> Declare Variables

-> Allocate Memory for 2D Array with 6 rows and 7 columns.

-> Begin while loop that takes in user input and performs Connect Four Game Logic.

**Program:**

/\*

\* File: main.cpp(Remastered)

\* Author: Heriberto Prieto

\* Created on June 4, 2017, 5:26 PM

\* Included Concepts:

- Line 19: Classes

- Line 118: Inheritance

- Line 140: Operator Overloading

- Line 118: Polymorphism

- Line 18: Templates

\*/

#include <cstdlib>

#include <iostream>

using namespace std;

/\* Template for 2DArray \*/

template <class T>

class Table {

private:

int columns;

int rows;

T \*\*table;

public:

//Class Constructor

Table();

//Class Constructor: Set Columns and Rows

Table(int,int);

//Class Destructor

~Table();

//Create Table

void create(T,int,int);

//Reset Table

void reset(T);

//Mutates char at given index

void setValue(T,int,int);

//Accesses char value at given index

T getValue(int,int);

//Accesses number of rows int table

int getRows()

{ return this->rows; }

//Accesses number of columns in table

int getColumns()

{ return this->columns; }

//Prints table values

void print();

};

/\* Class Constructor \*/

template <class T>

Table<T>::Table() {

this->columns = 0;

this->rows = 0;

}

/\* Class Constructor: Sets Columns and Rows of table \*/

template <class T>

Table<T>::Table(int rows,int columns) {

this->rows = rows;

this->columns = columns;

this->create(this->rows,this->columns);

}

/\* Frees memory of table/2DArray \*/

template <class T>

Table<T>::~Table() {

for (int r = 0; r < this->rows; r++)

delete []this->table[r];

delete []this->table;

}

/\* Creates new table/Allocates memory for table \*/

template <class T>

void Table<T>::create(T space,int rows,int columns) {

//Set Data Members and Allocate Memory

this->rows = rows;

this->columns = columns;

this->table = new T\*[this->rows];

for (int i = 0; i < this->rows; i++)

this->table[i] = new T[this->columns];

this->reset(space);

}

/\* Sets all values in table to blank spaces \*/

template <class T>

void Table<T>::reset(T space) {

for (int r = 0; r < this->rows; r++)

for (int c = 0; c < this->columns; c++)

this->table[r][c] = space;

}

/\* Sets the value in table at index \*/

template <class T>

void Table<T>::setValue(T value,int row,int column) {

if (row <= this->rows && column <= this->columns)

this->table[row][column] = value;

}

/\* Returns the value at given row and column \*/

template <class T>

T Table<T>::getValue(int row,int column) {

if (row <= this->getRows() && column <= this->getColumns())

return table[row][column];

return -1;

}

/\* Prints all values in table \*/

template <class T>

void Table<T>::print() {

for (int r = 0; r < this->rows; r++) {

for (int c = 0; c < this->columns; c++)

cout<<this->table[r][c]<<" ";

cout<<endl;

}

}

/\* Contains required properties for a game of Connect Four \*/

class ConnectFour : public Table<char> {

private:

int turns;

public:

//Class Constructor

ConnectFour();

//Turns accessor

int getTurns()

{ return this->turns; }

//Decrements number of turns

void turnTkn()

{ this->turns -= 1; }

//Increments number of turns

void turnRdo()

{ this->turns += 1; }

//Something happens

void dropChp(int,int);

//Prints contents of Connect Four Table/Board

void prntBrd();

//Checks for winner

int gameWon();

//++ operator overloading

void operator++()

{ ++turns; }

//-- operator overloaded

void operator--()

{ --turns; }

};

/\* Constructor: Allocates memory for 2DArray/Connect Four Game Board \*/

ConnectFour::ConnectFour() {

cout<<"Connect Four Constructor"<<endl;

this->turns = 42;

this->create(' ',6,7);

}

/\* Places value at given column, lowest value row without value \*/

void ConnectFour::dropChp(int place,int player) {

char x = 'X',o = '0';

for (int r = this->getRows() - 1; r >= 0; r--)

if (this->getValue(r,place - 1) == ' ') {

(player == 1) ? this->setValue(x,r,place - 1):

this->setValue(o,r,place - 1);

break;

}

}

/\* Prints Connect Four Board/2D Array \*/

void ConnectFour::prntBrd() {

cout<<endl;

for (int r = 0; r < this->getRows(); r++) {

for (int c = 0; c < this->getColumns(); c++)

(c == 0) ? cout<<"|| "<<this->getValue(r,c)<<" || ":

cout<<this->getValue(r,c)<<" || ";

cout<<endl;

}

cout<<"------------------------------------"<<endl;

}

/\* Game Winning Logic, returns player that won. \*/

int ConnectFour::gameWon() {

for (int r = 0; r < this->getRows(); r++)

for (int c = 0; c < this->getColumns(); c++)

if (this->getValue(r,c) != ' ') {

//Bottom Half Vertical Win

if (r > 2)

if (this->getValue(r,c) == this->getValue(r - 1,c))

if (this->getValue(r - 1,c) == this->getValue(r - 2,c))

if (this->getValue(r - 2,c) == this->getValue(r - 3,c))

return this->getTurns() % 2 == 0;

//Top Half Vertical Win

if (r < 3)

if (this->getValue(r,c) == this->getValue(r + 1,c))

if (this->getValue(r + 1,c) == this->getValue(r + 2,c))

if (this->getValue(r + 2,c) == this->getValue(r + 3,c))

return this->getTurns() % 2 == 0;

//Reverse Diagonal Win

if (r < 3 && c < 4)

if (this->getValue(r,c) == this->getValue(r + 1,c + 1))

if (this->getValue(r + 1,c + 1) == this->getValue(r + 2,c + 2))

if (this->getValue(r + 2,c + 2) == this->getValue(r + 3,c + 3))

return this->getTurns() % 2 == 0;

//Diagonal Win

if (r < 3 && c < 4)

if (this->getValue(r,c) == this->getValue(r + 1,c - 1))

if (this->getValue(r + 1,c - 1) == this->getValue(r + 2,c - 2))

if (this->getValue(r + 2,c - 2) == this->getValue(r + 3,c - 3))

return this->getTurns() % 2 == 0;

//Horizontal Win

if (c < 4)

if (this->getValue(r,c) == this->getValue(r,c + 1))

if (this->getValue(r,c + 1) == this->getValue(r,c + 2))

if (this->getValue(r,c + 2) == this->getValue(r,c + 3))

return this->getTurns() % 2 == 0;

}

return 0;

}

int main(int argc,char\*\* argv) {

//Seed

srand(static\_cast<unsigned int>(time(0)));

//Declare variables

int won = 0;//False while four have not been connected

int place; //Position where chip will be dropped

int player; //Depending on turn, keeps track of player turns

ConnectFour \*game = new ConnectFour();

while (game->getTurns() >= 0 && !won) {

//Output to player(s)

cout<<"Turns left: "<<game->getTurns()<<endl;

cout<<"Choose a column from 1 - 7 and drop a chip."<<endl;

game->prntBrd();

//Output to Specified Player

(game->getTurns() % 2 == 0) ?

cout<<"Player 1's turn:"<<endl:

cout<<"Player 2's turn:"<<endl;

cin.clear();

cin>>place;

player = (game->getTurns()%2==0) ? 1 : 2;

//"Drop Chip" into array based on players decision

if (place > 0 && place < 8) game->dropChp(place,player); else ++\*game;

//Check if player won

won = game->gameWon();

//Announce winner

if (won != 0) cout<<"Player: "<<won<<" wins!"<<endl; else --\*game;

}

//Print Game Board

game->prntBrd();

}