Project 1

CONSOLE TETRIS

Jonathan Holt

48130

10/26/14

INTRODUCTION:

I am currently working on a console based Tetris game. I felt this game would be a great start to a building block of the kind of programs I want to start making when I get my degree eventually. The reason I wanted to build this game is because I have future ideas on possibly creating programs or games that will help my work (UPS). I started with a console style Tetris game because I thought of it has like loading a trailer or truck at UPS. Basically, the point of the game is try and get as many points as the player user can. The user is constantly prompted with the table to be played on and the object or block that they need to place on the table. To do this, the player needs to get a single or multiple rows of numbers lined up horizontally to then be erased and rewarded with points. If I could then take this and possibly turn it into a 3D Tetris game, I would feel incorporating these kinds of games would keep taking me to the goals I have in mind of programming things for UPS. These are my current goals going into my computer science major.

SUMMARY:

-Size - the project stands at about 746 lines of code with comments but without comments

probably at about 600 or a little less.

-Criteria - it meets the requirements for this project because I utilize everything that was asked

of us for the project. I constantly create and return dynamic 2D arrays/pointers in the program particularly lines 180-199, 264-392 and of course the delete function for these at the end of use. The use of structures comes with lines 406-465 is a user created Tetris block that uses a structure to build and 684-761 file structure system. Strings are utilized especially with user input validation lines 477-499, 418-425 and 428-435.

-Variables - 26 variables created in the program.

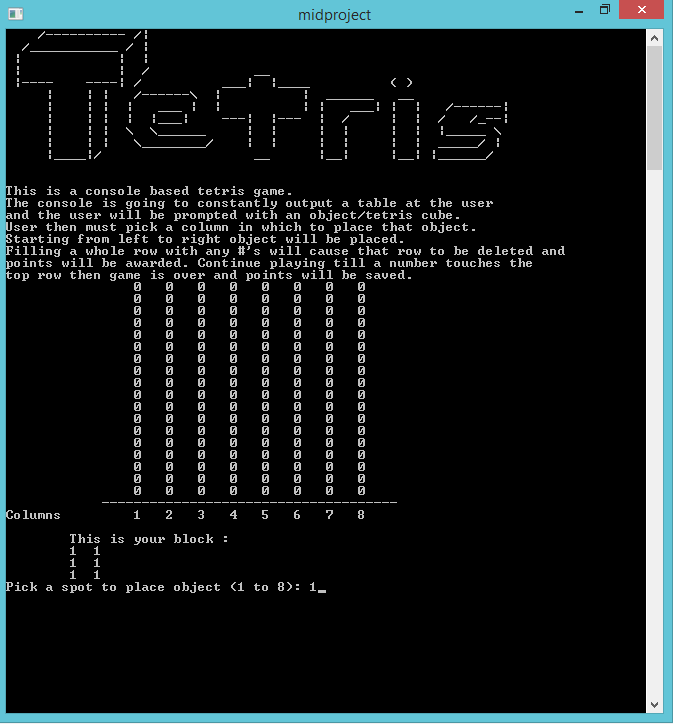
-Production - Started working on program about 4 weeks ago, and have been constantly

updating the program or stepping away when walls were hit to think mentally about overcoming those walls. I believe I spent about 30 hours of actual coding, but a lot more of thinking of ways to incorporate things. Wasn’t too challenging but somewhat difficult, I would’ve definitely been screwed if I waited till the last minute to try and write this program. One thing I was upset about was not being able to destroy multiple rows at once. Program only deletes one row and at a time when numbers fill a row. I also do utilize a break to end the game which I couldn’t get any other way to work without bounds issue screwing the game when blocks were stacked at the top of table. Might expand this project for my final project which will be incorporating classes.

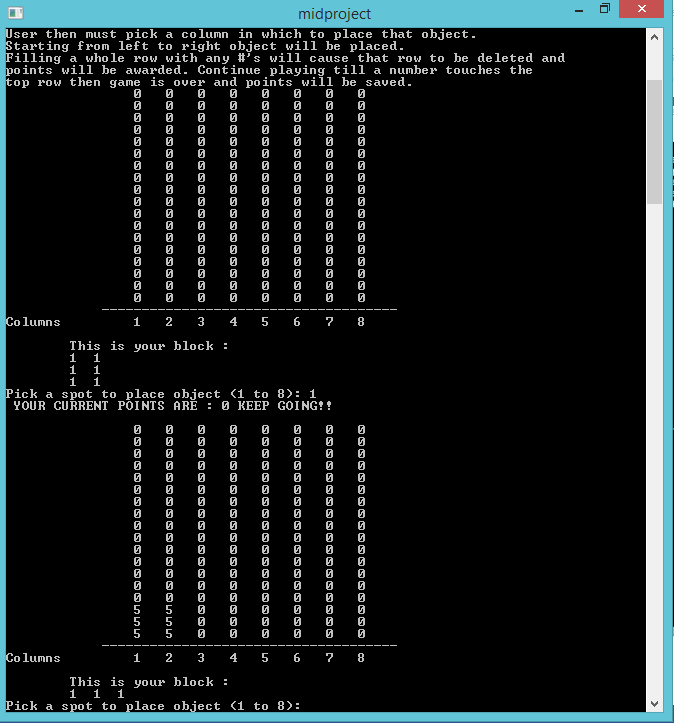
DESCRIPTION:

Started off creating a 2D pointer/array table in which user would try and place blocks given to them randomly. Problem was getting the objects to stack and playing without getting bounds issues. Reward points when a row was filled and shifting the table down 1 row. Create some kind of high score system.

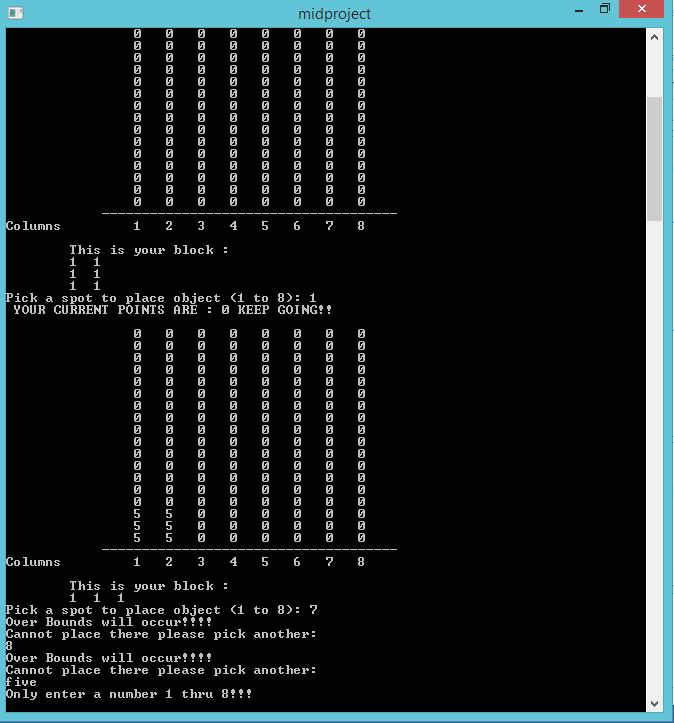
This is beginning of game where user is prompted with placement of the block. User picks the gets placed. In this example ‘1’ is picked.



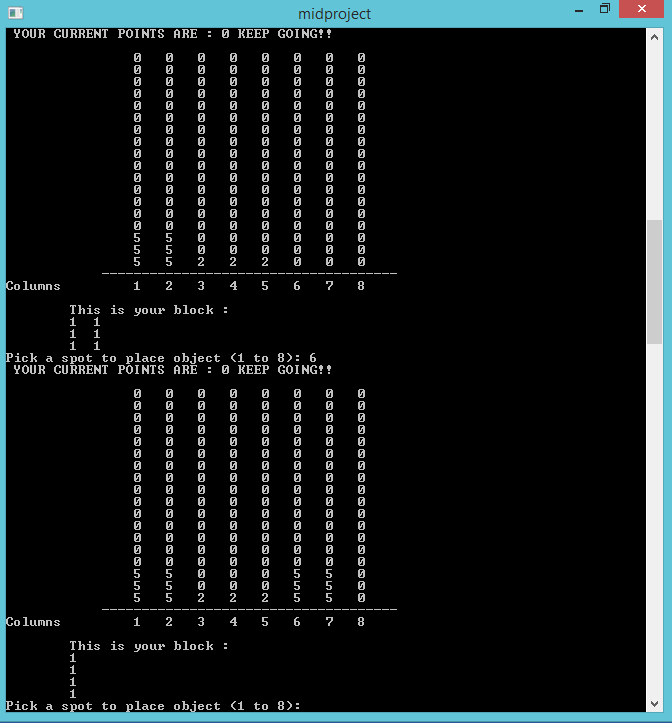
Object is then placed according to column picked by user. When column is picked always placed from left to right starting from column number picked.



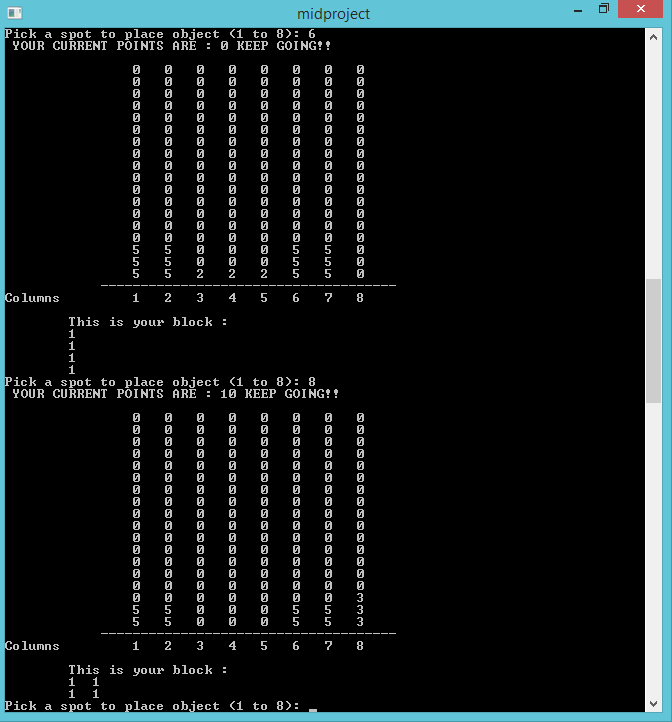
If trying to pick a place where object will go out of the table user is prompted with sentence “Over bounds will occur!” also checks for proper user input notice I typed ‘five’ into the program will only accept numbers 1-8.



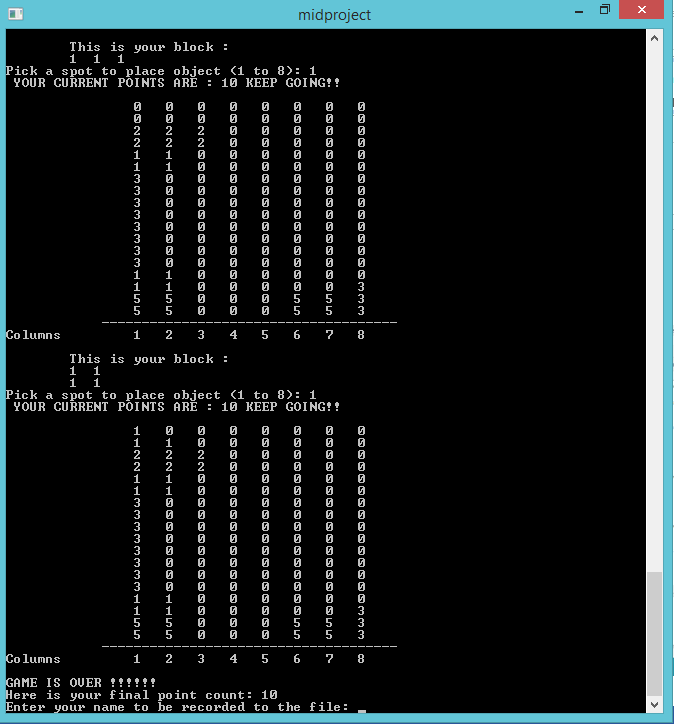
The point is to get a whole row filled with numbers. Different number represent the different blocks that were placed by the user. Once a row is completely full of numbers, row is deleted and user is rewarded with 10 points. Problem I faced was not being able to delete multiple rows at once only does 1 at a time.



PLUS 10 points for completing a row! Everything from top of that row is copied and placed down 1 row.



Game is over once a block reaches the top row. Then user enters their name to be recorded to a file where a structure is used to get names and points of previous games and outputs the high score table to the user at the very end.



Pseudo Code:

*Start program by outputting details of game to user*

*create of 2D array/pointer table and fill it with 0's*

*Start game loop*

*call a random number for number of blocks that can be created this game currently is 1-6*

*if 1-6*

*create and fill a 2D block dynamically to be played in table*

*user is promtped with choice of where to place object*

*user validation input is checked by entering acceptable number for placement*

*also checks for multiple columns objects to not cause bounds issues*

*once user choice has been picked game looks for a non 0 number to place block*

*on top of any numbers that could be blocking it from falling all the way down.*

*After that it will place block vertically where it should go however*

*if block reaches top row game is ended*

*else block is set on table accordingly*

*destroy dynamically created block after it has been placed*

*Check for all #'s in a row*

*if all of row == # other than 0 |or| all of row != 0*

*create and copy table dynamically with 2D array/pointer*

*find row that contains full row of #'s then*

*proceed to copy top of that row down exactly one row to replace*

*all the numbers that completed a full row of #'s*

*user is awarded 10 points*

*game ends when a block reaches top row of table*

*use a structure to save names and points parallel*

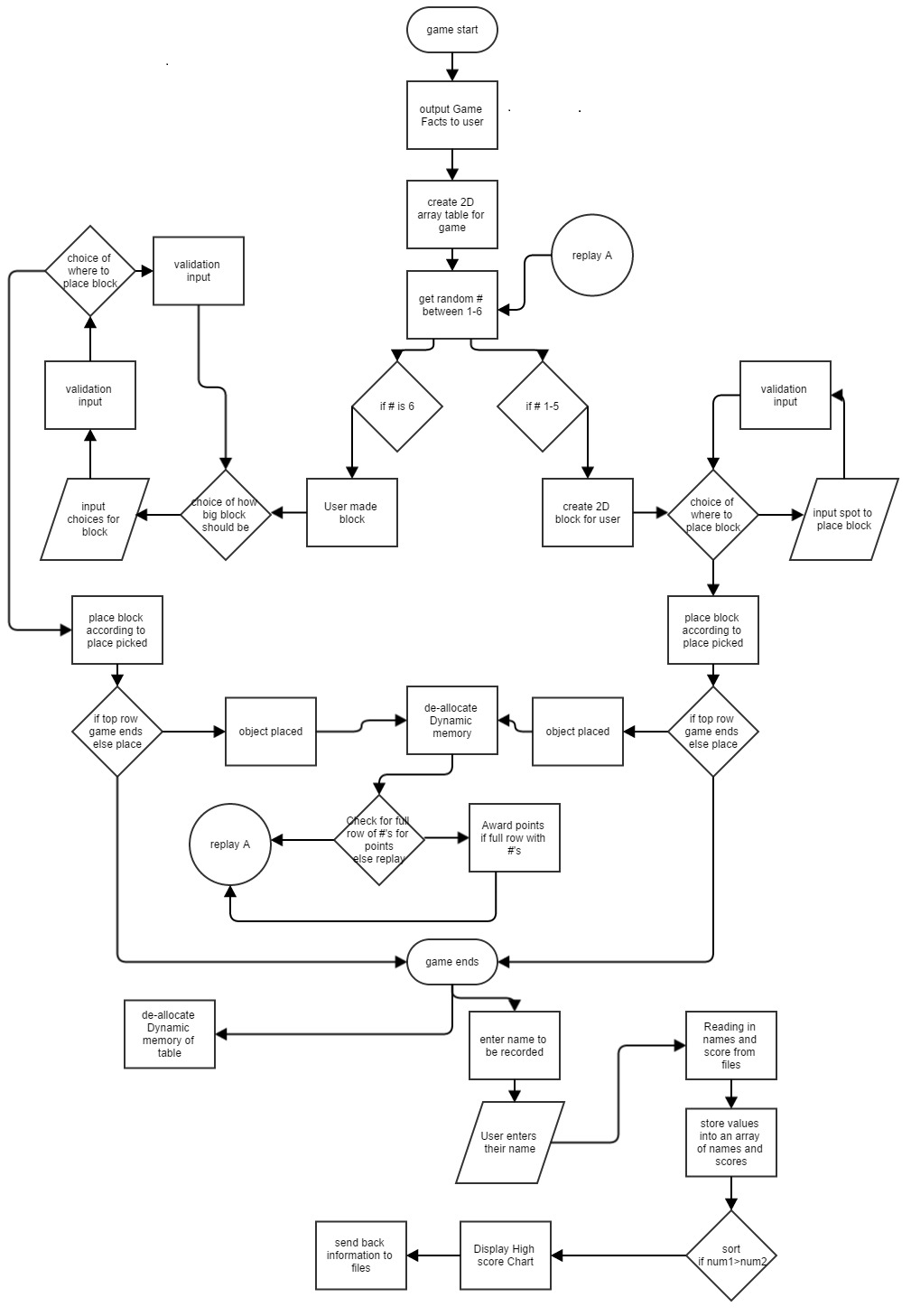
*Read in names from a file save them*

*Read in points from a file save them*

*if one element > second element*

*Sort*

*Output to screen the highscores with names*



Major Variables in the program:

|  |  |  |  |
| --- | --- | --- | --- |
| **TYPE** | **NAME** | **DESCRIPTION** | **LOCATION** |
| Structure | UserObj | User made block | int \*\*objectNum1(int,int,int) |
|  | Filetrack | Point system | void fileScores(int) |
| 2D Array (dynamic) | table | Game table | int main() |
| (dynamic) | object | Created block | int \*\*objectNum(int) |
| (dynamic) | newTble | Copied table | int \*\*newTable(int\*\*,int) |
| Integers | points | Points for game | int \*\*newTable(int\*\*,int) |
|  | spot | Block placement | void spotChoice (int,int) |
| String | name | User name | void fileScores(int) |
|  | valid/choice | User validation | void spotChoice (int,int) |
| Fstream | file | Read and Write to file | void fileScores(int) |
|  |  |  |  |

References:

Gaddis, Tony. Starting Out With C++: From Control Structures Through Objects 8th ed.

Pearson Education Inc. 2015. Text.

Joel. Gave some suggestions and helped with array table shifting.

In class examples.

/\*\*

\* Jonathan Holt

\* C++ objects

\* project

\* i certify this is my work

\*/

#include <cstdlib>

#include <iostream>

#include <iomanip>

#include <cctype>

#include <cstring>

#include <ctime>

#include <fstream>

using namespace std;

/\*\*

\* Constant variables to build the table in this program which is used as

\* 2D array/pointer table.

\*/

const int ROWS = 18;

const int COLS = 8;

/\*\*

\* Enumerator data.

\* Used to create the sizes of objects throughout the program.

\*/

enum ObSpots {ONE = 1 , TWO, THREE, FOUR, FIVE};/\*\*< Enum value starting at 1.\*/

struct UserObj /\*\*< Structure used to make object made by user.\*/

{

int \*\*ptr;

int rows;

int cols;

};

struct Filetrack{/\*\*< Structure that takes in names and scores from files.\*/

string name;

int score;

};

void outputBegin ();

int randObject ();

int \*\*objectNum (int num);

int \*\*objectNum1 (int num, int&, int&);

void outputTbl(int \*\*, int, int);

int \*\*fillGrid (int , int);

bool isOver (int \*\*tbl);

void destroy (int\*\*, int);

//void objtPlcmnt(int \*\*tble, int spot);

void objtPlcmnt(int \*\*tble, int spot, int rowOb, int colOb, int num);

int \*\*newTable (int \*\*tble, int &pts);

void spotChoice (int &spot, int col);

int realNum (int n);

void fileScores (int);

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

//making table

int \*\*table = fillGrid (ROWS, COLS); /\*\*< Table to be used for game. \*/

srand (time(NULL));

int \*\*object; /\*\*< 2D ptr that user will be prompted with.\*/

bool game; /\*\*< Bool check to end the game officially. \*/

int rowOb =0; /\*\*< Row size of object.\*/

int colOb =0;/\*\*< Column size of object.\*/

int spot; /\*\*< User choice of placement of objects.\*/

int points = 0;/\*\*< Points to be tracked while playing. \*/

//begin function

outputBegin ();

outputTbl(table, ROWS, COLS);

do

{

int num = randObject ();

switch (num)

{

case 1:

object = objectNum (num);

spotChoice (spot, TWO);

objtPlcmnt(table, spot, TWO, TWO, 1);

destroy (object, TWO);

break;

case 2:

object = objectNum (num);

spotChoice (spot, THREE);

objtPlcmnt(table, spot, ONE, THREE, 2);

destroy (object, ONE);

break;

case 3:

object = objectNum (num);

spotChoice (spot, ONE);

objtPlcmnt(table, spot, FOUR, ONE, 3);

destroy (object, FOUR);

break;

case 4:

object = objectNum (num);

spotChoice (spot, THREE);

objtPlcmnt(table, spot, TWO, THREE, 4);

destroy (object, TWO);

break;

case 5:

object = objectNum (num);

spotChoice (spot, TWO);

objtPlcmnt(table, spot, THREE, TWO, 5);

destroy (object, THREE);

break;

case 6:

object = objectNum1 (num, rowOb, colOb);

spotChoice (spot, colOb);

objtPlcmnt(table, spot, rowOb, colOb, 6);

destroy (object, rowOb);

}

table = newTable (table, points);

cout << " YOUR CURRENT POINTS ARE : " << points << " KEEP GOING!!\n\n";

//int \*\*object = objectNum (num);

//spotChoice (spot, TWO);

outputTbl(table, ROWS, COLS);

//checking if lost

game = isOver(table);

}while(game != false);

destroy(table, ROWS);

cout << "GAME IS OVER !!!!!!" << endl;

cout << "Here is your final point count: " << points << endl;

fileScores (points);

return 0;

}

/\*\*

\* This function is used to output the 2D table to the console that user will

\* be interacting with by placing objects in it. Outputs this table constantly

\* so user can always see it.

\* @param ptr 2D pointer.

\* @param rows size for rows.

\* @param cols size for columns.

\*/

void outputTbl(int \*\*ptr, int rows, int cols)

{

int count = 0;

for (int i = 0; i < ROWS; i++)

{

cout << "\t\t";

for(int j = 0; j < COLS; j++)

{

cout << ptr[i][j] << " ";

count++;

if (count == COLS)

{

cout << endl;

count = 0;

}

}

}

cout << "\t -------------------------------------" << endl;

cout << "Columns \t1 2 3 4 5 6 7 8 "<< endl<<endl;

}

/\*\*

\* Function creates a dynamic two dimensional table which is created by the

\* size of ROWS and COLS and sets all elements to 0. After that it then returns

\* the 2D pointer.

\* @param ROWS size for rows.

\* @param COLS size for columns.

\* @return 2D dynamic pointer.

\*/

int \*\*fillGrid (int ROWS, int COLS)

{

int \*\*array=new int\*[ROWS];

//creating 2D array

for(int i=0;i<ROWS;i++)

{

array[i]=new int[COLS];

}

//setting to 0

for (int row = 0; row < ROWS; row++)

{

for (int col = 0; col < COLS; col++)

{

array[row][col] = 0;

}

}

return array;

}

/\*\*

\* Information of beginning of game to give a brief description for how the

\* game is played.

\*/

void outputBegin ()

{

cout << " /---------- /|" << endl;

cout << " /\_\_\_\_\_\_\_\_\_\_\_ / | " << endl;

cout << " | | | " << endl;

cout << " | | / \_\_ " << endl;

cout << " |---- ----| / \_\_\_| |\_\_\_\_ ( )" << endl;

cout << " | | | /------\\ | | \_\_\_\_\_\_ \_\_ " << endl;

cout << " | | | | \_\_\_ | | | | \_\_\_| | | /------|\n";

cout << " | | | | |\_\_\_| ---| |--- | / | | / /\_--|\n";

cout << " | | | \\ \\\_\_\_\_\_\_ | | | | | | |\_\_\_\_\_ \\\n";

cout << " | | | \\\_\_\_\_\_\_\_\_/ | | | | | | \_\_\_\_\_/ |\n";

cout << " |\_\_\_\_|/ \_\_ |\_\_| |\_\_| |\_\_\_\_\_\_/\n";

cout << endl << endl;

cout << "This is a console based tetris game." << endl << "The console is"

" going to constantly output a table at the user" << endl << "and"

" the user will be prompted with an object/tetris cube." << endl <<

"User then must pick a column in which to place that object."<< endl

<<"Starting from left to right object will be placed." <<endl

<< "Filling a whole row with any #'s will cause that row to be"

" deleted and"<<endl << "points will be awarded. Continue playing"

" till a number touches the" << endl <<"top row then game is over"

" and points will be saved." <<endl;

}

/\*\*

\* destroy function is used to delete all of the dynamically created objects

\* in the program which most are 2D so need to delete by rows first then delete

\* the entire thing.

\* @param array the dynamic 2D pointer/array.

\* @param rows rows to be deleted first.

\*/

void destroy(int \*\*array,int rows)

{

//Destroy in reverse order of creation

for(int i=0;i<rows;i++)

{

delete []array[i];

}

delete []array;

}

/\*\*

\* Rand function is user to get a number between 1 and 6. It is for the

\* different objects to be randomized by particularly 6 of them.

\* @return an integer value to be used to choose a random object.

\*/

int randObject ()

{

int num;

num = rand()%6+1;

return num;

}

/\*\*

\* This function has multiple objects from which takes a number 1 to 5 and

\* dynamically creates the object. Then outputs that object to the screen

\* for user to see what that dynamic 2D pointer looks like and then

\* to be placed on table.

\* @param num a random integer value used to pick object.

\* @return the object chosen by integer num returns a 2D dynamic array.

\*/

int \*\*objectNum (int num)

{

int count =0;

int \*\*object;

if (num == 1)

{

object=new int\*[TWO];//rows

//creating 2D array

for (int i=0; i < TWO;i++)//rows

{

object[i]=new int[TWO];//cols

}

//once created filling and outputting

cout << "\tThis is your block : " << endl;

for (int i=0; i <TWO; i++){//rows

cout << "\t";

for(int j=0; j <TWO; j++){//cols

object[i][j]= 1;

// TRY AND FINE OUT THIS INDEX !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

//cout << \*\*(object+(i \* COLS)+j) << " ";

cout << object[i][j] << " ";

count++;

if (count == TWO)

{

cout << endl;

count = 0;

}

}

}

}

if (num == 2)

{

object=new int\*[ONE];//rows

//creating 2D array

for (int i=0; i < THREE;i++)//rows

{

object[i]=new int[THREE];//cols

}

//once created filling and outputting

cout << "\tThis is your block : " << endl ;

for (int i=0; i <ONE; i++){//rows

cout << "\t";

for(int j=0; j <THREE; j++){//cols

object[i][j]= 1;

cout << object[i][j] << " ";

count++;

if (count == THREE)

{

cout << endl;

count = 0;

}

}

}

}

if (num == 3)

{

object=new int\*[FOUR];//rows

//creating 2D array

for (int i=0; i < FOUR;i++)//rows

{

object[i]=new int[ONE];//cols

}

//once created filling and outputting

cout << "\tThis is your block : " << endl ;

for (int i=0; i <FOUR; i++){//rows

cout << "\t";

for(int j=0; j <ONE; j++){//cols

object[i][j]= 1;

cout << object[i][j] << " ";

count++;

if (count == ONE)

{

cout << endl;

count = 0;

}

}

}

}

if (num == 4)

{

object=new int\*[TWO];//rows

//creating 2D array

for (int i=0; i < TWO;i++)//rows

{

object[i]=new int[THREE];//cols

}

//once created filling and outputting

cout << "\tThis is your block : " << endl ;

for (int i=0; i < TWO; i++){//rows

cout << "\t";

for(int j=0; j < THREE; j++){//cols

object[i][j]= 1;

cout << object[i][j] << " ";

count++;

if (count == THREE)

{

cout << endl;

count = 0;

}

}

}

}

if (num == 5)

{

object=new int\*[THREE];//rows

//creating 2D array

for (int i=0; i < THREE;i++)//rows

{

object[i]=new int[TWO];//cols

}

//once created filling and outputting

cout << "\tThis is your block : " << endl ;

for (int i=0; i < THREE; i++){//rows

cout << "\t";

for(int j=0; j < TWO; j++){//cols

object[i][j]= 1;

cout << object[i][j] << " ";

count++;

if (count == TWO)

{

cout << endl;

count = 0;

}

}

}

}

return object;

}

/\*\*

\* A function that creates a 2 dimensional pointer through a structure. Allows

\* user to make it and checks for valid user input. User enters a row size

\* then a column size not allowing to be bigger than 5 or less than 1 for both.

\* Keeps track of row and column integers to be used for delete later.

\* @see objectNum().

\* @param num a random integer value used to pick object.

\* @param rowOb an integer to save the row value.

\* @param colOb an integer to save the column value.

\* @return the object chosen by integer number returns a 2D dynamic array.

\*/

int \*\*objectNum1 (int num, int &rowOb, int &colOb)

{

UserObj object ;//= new UserObj;

string valid;

if (num == 6)

{

cout << "This is a BONUS PLAY!" << endl << "User is allowed to enter"

" an object of their own choice." << endl << "The max size"

" you may enter is a 5x5." << endl;

cout << "Enter a row(s) number between 1 n 5" << endl;

//for (int i = 0 ; i < 5; i ++)

int count=0;

cin >> valid;

while (valid != "1" &&valid != "2" &&valid!= "3" &&valid != "4" &&

valid != "5")

{

cout << "Only enter a number 1 thru 5!!!" << endl;

cin >> valid;

//object.rows = realNum (valid[0]);

}

object.rows = realNum (valid[0]);

cout << "Enter a column(s) number between 1 n 5" << endl;

cin >> valid;

while (valid != "1" &&valid != "2" &&valid!= "3" &&valid != "4" &&

valid != "5"){

cout << "Only enter a number 1 thru 5!!!" << endl;

cin >> valid;

}

object.cols = realNum (valid[0]);

//creating user object

object.ptr = new int \*[object.rows];//rows

for (int i = 0; i < object.rows; i++)//rows

{

object.ptr[i] = new int [object.cols];//cols

}

//once created filling and outputting

cout << "\tThis is your block : " << endl;

for (int i=0; i <object.rows; i++){//rows

cout << "\t";

for(int j=0; j <object.cols; j++){//cols

object.ptr[i][j]= 1;

cout << object.ptr[i][j] << " ";

count++;

if (count == object.cols)

{

cout << endl;

count = 0;

}

}

}

rowOb = object.rows;

colOb = object.cols;

/\*for(int i=0;i<object.rows;i++)

{

delete []object.ptr[i];

}

delete []object.ptr;\*/

//}

return object.ptr;

}

}

/\*\*

\* This function spotChoice allows user to input data that will then decide

\* where they wish to place their object. Checks for user validation by taking

\* a string first thing converting it to an integer. The integer spot is

\* referenced to be changed throughout the program. Returns nothing.

\* @param spot an integer value that represents spot to be placed on table.

\* @param col an integer value needed to check for bounds issues.

\*/

void spotChoice (int &spot, int col){

string choice;

cout << "Pick a spot to place object (1 to 8): " ;

cin >> choice;

while (choice != "1" &&choice != "2" &&choice != "3" &&choice != "4" &&

choice != "5" &&choice != "6" &&choice != "7" &&choice != "8")

{

cout << "Only enter a number 1 thru 8!!!" << endl;

cin >> choice;

}

spot = realNum (choice[0]);

//cout << "SPOT: " << spot << "SPPOT!!"<<endl;

//IMPORTANT BOUNDS CHECKING AND PLACEMENT

//first area checks spot choice compared to column size of object

//rest makes sure number is 1 - 8

while ((spot-1) + col > 8 || spot < 1 || spot > 8){

cout << "Over Bounds will occur!!!!" << endl;

cout << "Cannot place there please pick another:" << endl;

cin >> choice;

while(choice != "1" &&choice != "2" &&choice != "3" &&choice != "4" &&

choice != "5" &&choice != "6" &&choice != "7" &&choice != "8"){

cout << "Only enter a number 1 thru 8!!!" << endl;

cin >> choice;

spot = realNum (choice[0]);

}

}

}

/\*\*

\* The function realNum takes an integer number between 1 and 8 that then

\* references it back to the ascII table to be able to convert it to its

\* actual number.

\* @param n an integer value that inputted by user.

\* @return returns an integer value.

\*/

int realNum (int n){

int realOne;

if (n == 49){

realOne = 1;

}if (n == 50) {

realOne = 2;

}if (n == 51){

realOne = 3;

}if (n == 52){

realOne = 4;

}if (n == 53){

realOne = 5;

}if (n == 54){

realOne = 6;

}if (n == 55){

realOne = 7;

}if (n == 56){

realOne = 8;

}

return realOne;

}

/\*\*

\* The function isOver is checking for a non 0 value on the first row of the

\* table. If it finds this value it instantly breaks from stacking objects on

\* the table that way over bounds issues will not come into play.

\* @param tbl 2D pointer that represents a table of elements.

\* @return condition of true or false.

\*/

bool isOver (int \*\*tbl)

{

bool lose;

//for (int i =0; i < ROWS; i++){ actually dont need to check columns

int i = 0;

for (int col=0; col < COLS;col++){

//[0][j] because this would start from the top left of table

if (tbl[i][col] != 0)

{

lose = false;

break;

}

else

lose = true;

}

return lose;

}

/\*\*

\* The function objtPlcmnt is first going take 2D table then since object is

\* of a certain size looking at size of column spots to check for a #. If there

\* is a # in either spot i set rows to that spot to then place object on top

\* of the numbers in table. Uses a break statement to break from going over

\* bounds when placing an object.

\* @param tble 2D table.

\* @param spot integer value entered by user.

\* @param rowOb integer row size of object.

\* @param colOb integer column size of object.

\* @param num integer value to represent numbers in the object.

\*/

void objtPlcmnt(int \*\*tble, int spot, int rowOb, int colOb, int num)

{

//columns

int col = spot-1;

int row = ROWS;

//starting from bottom left to top

for (int i = ROWS-1; i >= 0; i--){

for (int k = 0; k < colOb; k++)

{

if (tble[i][col+k] != 0 )//||

{

//setting row

row = i;

}

}

}

for (int i=row-1; i >= row-rowOb; i--){

for(int j=0; j < colOb; j++) {

//checking if spot[col] top of table = 1 if so break from placing

//one

if (tble[0][col] != 0){

//breaking from loop cycle

break;

}

else

tble [i][j+spot-1] = num;

}

}

}

/\*\*

\* This function newTable is designed to act as the point system for the game.

\* It creates a new dynamic 2D table and then copies the one currently being

\* use by user. It loops through all the rows in the table by checking for

\* non zero #s and if it finds that row saves that row. After that goes

\* through two loops to copy the row above that row and everything above it to

\* be placed where the row with all #'s were.

\* @param tble 2D pointer of table for game play.

\* @param pts integer value that is used to keep track of user points.

\* @return the 2D pointer that represents new table.

\*/

int \*\*newTable (int \*\*tble, int &pts){

int dstryRow =0; /\*\*< Integer value of row where replacing happens.\*/

int \*\*newTble = fillGrid (ROWS, COLS);/\*\*< Creating new table to copy.\*/

int count=0;

for (int i =0; i < ROWS; i++){

for (int j =0; j < COLS; j++){

newTble[i][j] = tble[i][j]; /\*\*< Copying new table.\*/

}

}

for (int i = 0; i < ROWS; i++)

{

if (tble[i][0] != 0 && tble[i][1] != 0 &&tble[i][2] != 0 &&

tble[i][3] != 0 &&tble[i][4] != 0 &&tble[i][5] != 0 &&

tble[i][6] != 0 &&tble[i][7] != 0)

{

count++;

}

}

for (int i = 0; i < ROWS; i++)

{

if (tble[i][0] != 0 && tble[i][1] != 0 &&tble[i][2] != 0 &&

tble[i][3] != 0 &&tble[i][4] != 0 &&tble[i][5] != 0 &&

tble[i][6] != 0 &&tble[i][7] != 0)

{

//count++;

pts += 10;

dstryRow=i;

//for (int i=row-1; i >= row-rowOb; i--)

//for (int i = ROWS-1; i >= 0; i--)

for (int k = 0; k < dstryRow; k++){

for (int j =0; j < COLS ; j++){

tble[k+1][j] = newTble[k][j];

}

}

}

}

destroy (newTble, ROWS);

return tble;

}

/\*\*

\* The function fileScores serves the purpose of a high score tracking system.

\* It first goes through the name file to get a count to find out how big of

\* a dynamic structure needs to be allocated. Then 2D dynamic structure is

\* created to keep track of parallel names and points. First, names are read

\* and saved and then last person played name is saved and written back to the

\* file. After, points are read in and then points of last played game are also

\* recorded and then all sent back to a different file. After that, names and

\* points are then sorted parallel to find highest to lowest then printed to

\* user.

\* @param points integer value of user's points after game is done.

\*/

void fileScores (int points){

fstream file;

string output;

string name;

int scores;

int count=1;

cout << "Enter your name to be recorded to the file: ";

cin.ignore();

getline (cin, name);

file.open("names.txt", ios:: in | ios::out);// ios::app |

if(file){

while (getline(file, output))

{

count++;

}

}

file.close();

//allNames = new string [count];

//allpoints = new int [count];

Filetrack \*stats = new Filetrack [count];

file.open("names.txt", ios:: in );//| ios::ate

int i=0;

if(file){

getline(file, output); //WORKING ON GETLINE

while (file)

{

stats[i].name=output;

i++;

getline(file, output);

}

}

file.close();

//setting name entered to last element

stats[count-1].name=name;

cout<<endl;

file.open("names.txt", ios:: out);

for (int i=0;i<count;i++){

if (i != count-1)

{

file << stats[i].name <<endl;

}

else

file << stats[i].name;

}

file.close();

file.open("scores.dat", ios:: in );//| ios::ate

int i1=0;

if(file){

while (file >> scores)

{

stats[i1].score=scores;

i1++;

}

}

file.close();

//setting points of last person played to last element

stats[count-1].score=points;

file.open("scores.dat", ios:: out);

for (int i=0;i<count;i++){

if (i != count-1)

{

file << stats[i].score <<endl;

}

else

file << stats[i].score;

}

file.close();

int temp = 0;

string temp1="";

for (int i = 0; i < count - 1; i++)

{

for (int j = i + 1; j < count; j++)

{

if (stats[j].score < stats[i].score)

{

temp = stats[j].score;

stats[j].score = stats[i].score;

stats[i].score = temp;

temp1 = stats[j].name;

stats[j].name = stats[i].name;

stats[i].name = temp1;

}

}

}

cout << "\nHIGH SCORES!!!!!!!!!!!!!!!!!!!!" <<endl;

cout << "NAME: SCORES:"<<endl;

for (int i=count-1; i >=0;i--){

cout<<left<<setw(20)<<stats[i].name << stats[i].score <<endl;

}

delete []stats;

}