**Introduction**

Title: 2048

This is a number puzzle game. It is played on a 4 x 4 grid which begins with two numbers (of value 2 or 4) placed at arbitrary locations on the grid.

The player can select to move up, down, left, or right. Each move shifts all the tiles on the grid in the chosen direction. If the adjacent tiles have the same value, they will combine and the resulting tile will have a value of the sum of two numbers (or double the number). After the move, a tile of value 2 or 4 will be spawned at a random position on the board.

The objective of the game is to combine equally-valued tiles to reach the 2048 tile without getting stuck. If the grid is full and another tile cannot be spawned, then the game is over.

**Summary**

Project size: about 500 lines of code

The number of variables: about 15

Number of methods: 18

This project was quite challenging to recreate. It took approximately 45 hours to complete. This project includes many of the concepts from chapter 9 through 12 in the book and I also used classes instead of structures. The difficult components of this project included using syntax I was unfamiliar with and having to look up on Google for references, algorithm for moving the tiles in the grid, and converting the Javascript code of the original 2048 to C++ because I don’t have any experience with Javascript.

There is much potential for this project that I plan to develop over the next few weeks for the final project.

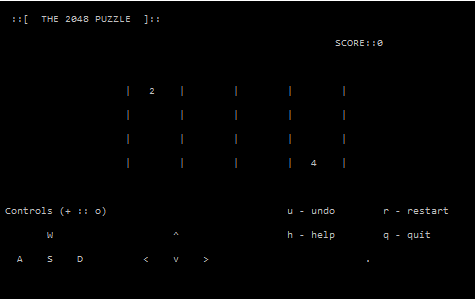
**Description**

Programming the whole game’s algorithm was the most complicated part of my project.

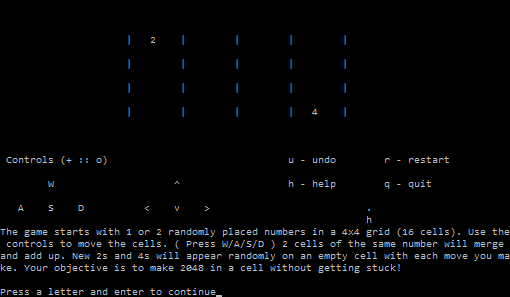
Refer to flowchart.

**Sample input/output** (screenshots)

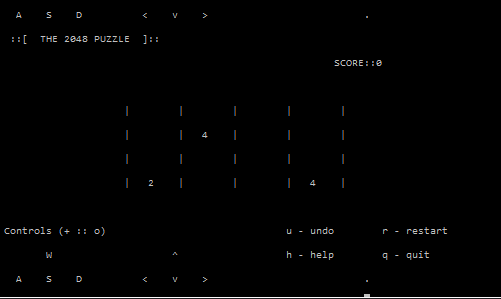
Initial console output:



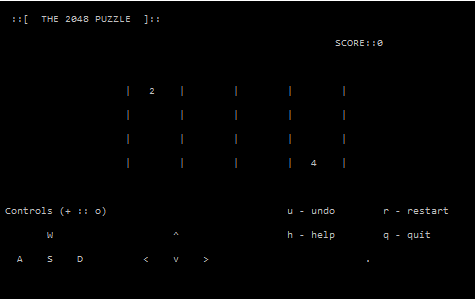
When user enters ‘h’ for help:



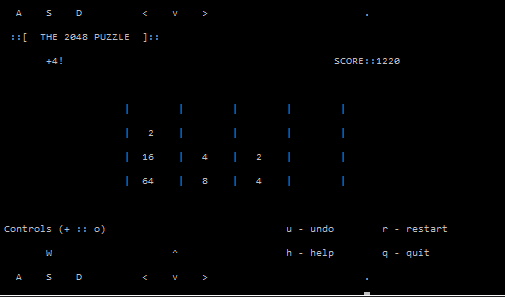
When user enters ‘s’ for down and a new tiles is spawned:



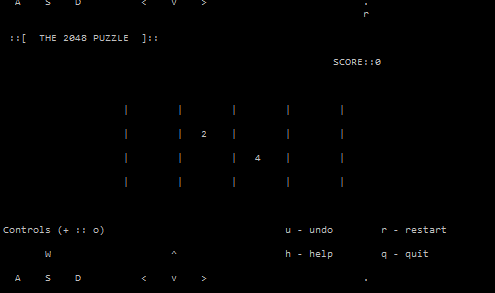
When user enters ‘u’ for undo:



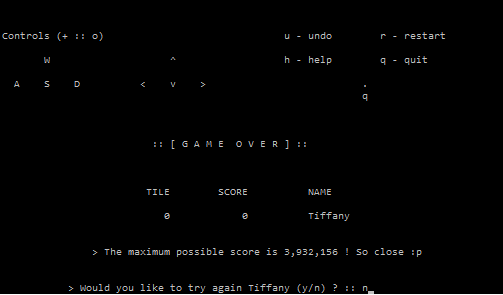
After several moves:



When user enters ‘r’ for restart:



When user enters ‘q’, enters name which is written to binary file, and ‘n’ for try again:



**Pseudocode**

*Initialize grid*

*Display output screen*

*While game has not ended*

*If ‘w’ is pressed*

*Move tiles up*

*Else if ‘a’ is pressed*

*Else if ‘s’ is pressed*

*Else if ‘d’ is pressed*

*Else if ‘q’ is pressed*

*Else if ‘r’ is pressed*

*Else if ‘h’ is pressed*

*Else if ‘u’ is pressed*

*Spawn new tile at random location that is empty*

*Output grid*

*Score of Round = log2(tileValue-1) \* tileValue*

*Update grid*

*Combine equal adjacent tiles*

*Check for game over*

*If grid is full && cannot spawn new tile || maxTile = 2048*

*Input player info*

*exit*

**Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Variable Name | Description | Location |
| int | playerScore | Score of player | Player Class |
| char[] | name[10] | Name of player | Player Class |
| bool | Response | Response to quite by user | Grid Class |
| bool | tileDestroyed | Is tile destroyed | Grid Class |
| char | control | Input by user | Grid Class |
| int | maxTile | Maximum tile | Grid Class |
| int | winValue | Value of tile to win | Grid Class |
| int | scoreRound | Score of the current round | Grid Class |
| int | score | Running total of score | Grid Class |
| int [][] | tiles[4][4] | Grid that is 4x4 matrix | Grid Class |
| int [][] | btiles[4][4] | Backup grid | Grid Class |
| char | option | Option input by user | Game Class |
| Player | player | Player object | Game Class |
| Game | Exec | Game object | int main() |

**Concepts**

|  |  |
| --- | --- |
| Chapter | Location at lines: |
| 9: pointers | 61, 104, 105, 116, 182 |
| 10: Characters and Strings | 24, 102, 169, 551 |
| 12: File operations | 494, 588 |
| 13: classes | 21, 29, 56 |
| Array | 40,41, 295 |

**References**

1. The original 2048 (javascript): <https://github.com/gabrielecirulli/2048>
2. Starting out with C++ by Gaddis

**Program**

|  |  |
| --- | --- |
|  | /\* |
|  | \* File: main.cpp |
|  | \* Author: Tiffany Pan |
|  | \* Date: October 15, 2015 |
|  | \* Course: CSC 17A |
|  | \* Midterm Project: 2048 |
|  | \*/ |
|  |  |
|  | //System Libraries |
|  | #include <iostream> |
|  | #include <ctime> |
|  | #include <string> |
|  | #include <cstdlib> |
|  | #include <cmath> |
|  | #include <fstream> |
|  | #include <cctype> |
|  |  |
|  | using namespace std; |
|  |  |
|  | /\* Definition of Player class \*/ |
|  | class Player |
|  | { |
|  | public: |
|  | char name[10]; //name of player |
|  | int playerScore; //score of player |
|  | }; |
|  |  |
|  | /\* Definition of Grid class \*/ |
|  | class Grid |
|  | { |
|  | public: |
|  | bool response; //response to quit by user |
|  | bool tileDestroyed; //is tile destroyed |
|  | char control; //input by user |
|  |  |
|  | int maxTile; //maximum tile |
|  | int winValue; //value of tile to win |
|  | int scoreRound; //score of the current round |
|  | int score; //running total of score |
|  | int tiles[4][4]; //grid that is a 4x4 matrix |
|  | int btiles[4][4]; //backup grid |
|  |  |
|  | void initializeGrid(); |
|  | void updateGrid(); |
|  | void fillSpace(); |
|  | void spawn(); |
|  | void findGreatestTile(); |
|  | void backupGrid(); |
|  | void undo(); |
|  | bool full(); |
|  | bool blockMoved(); |
|  | }; |
|  |  |
|  | /\* Definition of Game class |
|  | \* inherits Grid class \*/ |
|  | class Game: public Grid |
|  | { |
|  | private: |
|  | char option; //option input by user |
|  | Player player; //player object |
|  | Player\* list; //list of players |
|  |  |
|  | public: |
|  | void displayGrid(); |
|  | void displayHelpScreen(); |
|  | void displayWinScreen(); |
|  | void displayLoserScreen(); |
|  | char displayTryAgainScreen(int); |
|  | void savePlayerInfo(); |
|  | }; |
|  |  |
|  | //function prototypes |
|  | void logic(Game\*); |
|  | void checkGameOver(Game\*); |
|  | int randomNum(int); |
|  |  |
|  | //execution begins here |
|  | int main() |
|  | { |
|  | Game exec; //a Game object |
|  |  |
|  | //set up variables |
|  | exec.score = 0; |
|  | exec.maxTile = 0; |
|  | exec.scoreRound = 0; |
|  | exec.winValue = 2048; |
|  | exec.response = true; |
|  | exec.tileDestroyed = false; |
|  | exec.initializeGrid(); |
|  | char choice; |
|  |  |
|  | //loop for game |
|  | while (1) |
|  | { |
|  | exec.displayGrid(); |
|  | //test for computer to play the game itself |
|  | //char keys[] = "wasd"; |
|  | //exec.control = keys[rand() % 4 + 0]; |
|  |  |
|  | //get input |
|  | cin >> choice; |
|  | exec.control = tolower(choice); |
|  |  |
|  | logic(&exec); |
|  | checkGameOver(&exec); |
|  | }; |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  | /\* Definition for function logic |
|  | \* @param pointer to Game |
|  | \* goes through one loop of the game |
|  | \* depending on user input |
|  | \*/ |
|  | void logic(Game \*execute) |
|  | { |
|  | switch (execute->control) |
|  | { |
|  | case 'w': |
|  | case 'a': |
|  | case 's': |
|  | case 'd': |
|  | //if user inputs any of the four directions (w,a,s,d) |
|  | //make a backupgrid, update grid, check for maxTile, display grid |
|  | execute->backupGrid(); |
|  | execute->fillSpace(); |
|  | execute->updateGrid(); |
|  | execute->fillSpace(); |
|  | execute->findGreatestTile(); |
|  | execute->displayGrid(); |
|  |  |
|  | //check if full |
|  | if (execute->full() && execute->tileDestroyed) |
|  | { |
|  | execute->response = false; |
|  | break; |
|  | } |
|  | else if (execute->blockMoved()) |
|  | { |
|  | execute->spawn(); |
|  | break; |
|  | } |
|  | else |
|  | { |
|  | execute->response = true; |
|  | break; |
|  | } |
|  | //undo |
|  | case 'u': |
|  | if (execute->blockMoved()) |
|  | execute->score -= execute->scoreRound; |
|  | execute->undo(); |
|  | break; |
|  | //restart |
|  | case 'r': |
|  | execute->initializeGrid(); |
|  | execute->score = 0; |
|  | execute->scoreRound = 0; |
|  | break; |
|  | //quit |
|  | case 'q': |
|  | execute->response = false; |
|  | break; |
|  | //help |
|  | case 'h': |
|  | execute->displayHelpScreen(); |
|  | cout << "Press a letter and enter to continue"; |
|  | char cont; |
|  | cin >> cont; |
|  | execute->displayGrid(); |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | /\* definition of function checkGameOver |
|  | \* checks if maxTile has reached a multiple of 2048 |
|  | \* if the user quit |
|  | \* or if the grid is full |
|  | \* @param Game\* |
|  | \*/ |
|  | void checkGameOver(Game \*game) |
|  | { |
|  | if (game->maxTile == game->winValue) |
|  | { |
|  | game->displayWinScreen(); |
|  |  |
|  | if (game->displayTryAgainScreen(0) == 'n') |
|  | game->response = false; |
|  | else |
|  | game->winValue \*= 2; |
|  | } |
|  |  |
|  | else if (game->response == false) |
|  | { |
|  | game->displayLoserScreen(); |
|  |  |
|  | if (game->displayTryAgainScreen(1) == 'y') |
|  | { |
|  | game->initializeGrid(); |
|  | game->response = true; |
|  | } |
|  | } |
|  |  |
|  | if (game->response == false) |
|  | { |
|  | cout << "\n\n\t\t > Thank you for playing. "; |
|  | exit(0); |
|  | } |
|  | } |
|  |  |
|  | /\*definition of Grid class function initializeGrid |
|  | \* resets score and maxTile and each element of grid to 0 |
|  | \* spawns 2 more tiles |
|  | \*/ |
|  | void Grid::initializeGrid() |
|  | { |
|  | int i, j; |
|  |  |
|  | scoreRound = 0; |
|  | score = 0; |
|  | maxTile = 0; |
|  |  |
|  | for (i = 0; i<4; i++) |
|  | for (j = 0; j<4; j++) |
|  | tiles[i][j] = 0; |
|  |  |
|  | spawn(); |
|  | spawn(); |
|  | } |
|  |  |
|  |  |
|  | /\* definition of class Game function displayGrid |
|  | \* displays the formatted output screen of grid |
|  | \*/ |
|  | void Game::displayGrid() |
|  | { |
|  | cout << "\n ::[ THE 2048 PUZZLE ]::\n\n\t"; |
|  |  |
|  | if (scoreRound) |
|  | cout << "+" << scoreRound << "!"; |
|  | else |
|  | cout << " "; |
|  |  |
|  | cout << "\t\t\t\t\t\tSCORE::" << score << " \n\n\n\n"; |
|  |  |
|  | for (int i = 0; i<4; i++) |
|  | { |
|  | cout << "\t\t |"; |
|  |  |
|  | for (int j = 0; j<4; j++) |
|  | { |
|  | if (tiles[i][j]) |
|  | printf("%4d |", tiles[i][j]); |
|  | else |
|  | printf("%4c |", ' '); |
|  | } |
|  |  |
|  | cout << endl << endl; |
|  | } |
|  |  |
|  | cout << "\n\n Controls (+ :: o)\t\t\t\tu - undo\tr - restart\n\n\tW\t\t ^" |
|  | << "\t\t\th - help\tq - quit\n\t\t\t\t\t\t\t\t" |
|  | << " \n A S D\t\t< v >\t\t\t ." |
|  | << " \n\t\t\t\t "; |
|  |  |
|  | } |
|  |  |
|  | /\* Definition of function randomNum |
|  | \* generates random number between 0 and the number passed through param |
|  | \* @param: int |
|  | \* @return: int |
|  | \*/ |
|  | int randomNum(int x) |
|  | { |
|  | srand(time(NULL)); |
|  | int index; |
|  | index = rand() % x + 0; |
|  | return index; |
|  | } |
|  |  |
|  | /\* Definition of Grid class function backupGrid |
|  | \* creates a backup grid based on current grid |
|  | \*/ |
|  | void Grid::backupGrid() |
|  | { |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<4; j++) |
|  | btiles[i][j] = tiles[i][j]; |
|  | } |
|  |  |
|  | /\* Definition of Grid class function fillSpace |
|  | \* depending on direction grid tiles move, 0 will fill spaces |
|  | \*/ |
|  | void Grid::fillSpace() |
|  | { |
|  | switch (control){ |
|  | case 'w': |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<4; j++) |
|  | { |
|  | if (!tiles[j][i]) |
|  | { |
|  | for (int k = j + 1; k<4; k++) |
|  | if (tiles[k][i]) |
|  | { |
|  | tiles[j][i] = tiles[k][i]; |
|  | tiles[k][i] = 0; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | break; |
|  |  |
|  | case 's': |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 3; j >= 0; j--) |
|  | { |
|  | if (!tiles[j][i]) |
|  | { |
|  | for (int k = j - 1; k >= 0; k--) |
|  | if (tiles[k][i]) |
|  | { |
|  | tiles[j][i] = tiles[k][i]; |
|  | tiles[k][i] = 0; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | break; |
|  | case 'a': |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<4; j++) |
|  | { |
|  | if (!tiles[i][j]) |
|  | { |
|  | for (int k = j + 1; k<4; k++) |
|  | if (tiles[i][k]) |
|  | { |
|  | tiles[i][j] = tiles[i][k]; |
|  | tiles[i][k] = 0; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | break; |
|  | case 'd': |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 3; j >= 0; j--) |
|  | { |
|  | if (!tiles[i][j]) |
|  | { |
|  | for (int k = j - 1; k >= 0; k--) |
|  | if (tiles[i][k]) |
|  | { |
|  | tiles[i][j] = tiles[i][k]; |
|  | tiles[i][k] = 0; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | /\* Definition of Grid class function updateGrid |
|  | \* resets the current score of the round to 0 |
|  | \* for each move, grid will combine with adjacent equal tile |
|  | \* and sets tileDestroyed to true if tile was combined |
|  | \* score is recorded |
|  | \*/ |
|  | void Grid::updateGrid() |
|  | { |
|  | scoreRound = 0; |
|  | tileDestroyed = false; |
|  |  |
|  | switch (control) |
|  | { |
|  | case 'w': |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<3; j++) |
|  | { |
|  | if (tiles[j][i] && tiles[j][i] == tiles[j + 1][i]) |
|  | { |
|  | tileDestroyed = true; |
|  | tiles[j][i] += tiles[j + 1][i]; |
|  | tiles[j + 1][i] = 0; |
|  | scoreRound += (((log2(tiles[j][i])) - 1)\*tiles[j][i]); |
|  | score += scoreRound; |
|  | } |
|  | } |
|  | break; |
|  |  |
|  | case 's': |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 3; j>0; j--) |
|  | { |
|  | if (tiles[j][i] && tiles[j][i] == tiles[j - 1][i]) |
|  | { |
|  | tileDestroyed = true; |
|  | tiles[j][i] += tiles[j - 1][i]; |
|  | tiles[j - 1][i] = 0; |
|  | scoreRound += (((log2(tiles[j][i])) - 1)\*tiles[j][i]); |
|  | score += scoreRound; |
|  | } |
|  | } |
|  | break; |
|  |  |
|  | case 'a': |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<3; j++) |
|  | { |
|  | if (tiles[i][j] && tiles[i][j] == tiles[i][j + 1]) |
|  | { |
|  | tileDestroyed = true; |
|  | tiles[i][j] += tiles[i][j + 1]; |
|  | tiles[i][j + 1] = 0; |
|  | scoreRound += ((log2(tiles[i][j])) - 1)\*tiles[i][j]; |
|  | score += scoreRound; |
|  | } |
|  | } |
|  | break; |
|  |  |
|  | case 'd': |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 3; j>0; j--) |
|  | { |
|  | if (tiles[i][j] && tiles[i][j] == tiles[i][j - 1]) |
|  | { |
|  | tileDestroyed = true; |
|  | tiles[i][j] += tiles[i][j - 1]; |
|  | tiles[i][j - 1] = 0; |
|  | scoreRound += ((log2(tiles[i][j])) - 1)\*tiles[i][j]; |
|  | score += scoreRound; |
|  | } |
|  | } |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | /\* Definition of Grid class function spawn |
|  | \* spawns a new tile of value 2 or 4 at an arbitrary location on the grid |
|  | \* that is not currently occupied by a number that is nonzero |
|  | \*/ |
|  | void Grid::spawn() |
|  | { |
|  | int i, j, k; |
|  |  |
|  | do { |
|  | i = randomNum(4); |
|  | j = randomNum(4); |
|  | k = randomNum(10); |
|  | } while (tiles[i][j]); |
|  |  |
|  | if (k<2) |
|  | tiles[i][j] = 4; |
|  |  |
|  | else |
|  | tiles[i][j] = 2; |
|  | } |
|  |  |
|  | /\* Definition of Grid function findGreatestTile |
|  | \* loops through grid to find maxTile |
|  | \*/ |
|  | void Grid::findGreatestTile() |
|  | { |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<4; j++) |
|  | if (tiles[i][j]>maxTile) |
|  | maxTile = tiles[i][j]; |
|  | } |
|  |  |
|  | /\* Definition of Grid class function full |
|  | \* loops through grid to check if each element is a nonzero |
|  | \* returns true if so, false if not |
|  | \* @return bool |
|  | \*/ |
|  | bool Grid::full() |
|  | { |
|  | bool k = false; |
|  |  |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<4; j++) |
|  | { |
|  | if (!tiles[i][j]) |
|  | k = 0; |
|  | } |
|  | return k; |
|  | } |
|  |  |
|  | /\* Definition of Game class function displayHelpScreen |
|  | \* opens directions.txt and reads the strings from it |
|  | \*/ |
|  | void Game::displayHelpScreen() |
|  | { |
|  | string str; |
|  | ifstream openFile; |
|  | openFile.open("directions.txt"); |
|  |  |
|  | while (openFile >> str) |
|  | { |
|  | cout << str << " "; |
|  | } |
|  |  |
|  | cout << endl << endl; |
|  |  |
|  | openFile.close(); |
|  | } |
|  |  |
|  | /\* definition of Game class function displayWinScreen |
|  | \* displays formatted output to screen |
|  | \*/ |
|  | void Game::displayWinScreen() |
|  | { |
|  | cout << endl << endl; |
|  | cout << "\n\t\t\t :: [ YOU MADE " << winValue << "! ] ::" |
|  | << "\n\n\t\t\t :: [ YOU WON THE GAME ] ::" |
|  | << "\n\n\n\n\t\t\t TILE\t SCORE\t NAME"; |
|  | printf("\n\n\t\t\t %4d\t %6d\t ", maxTile, score); |
|  | cin >> player.name; |
|  | savePlayerInfo(); |
|  |  |
|  | cout << "\n\n\t\t> The maximum possible tile is 65,536 ! So go on :)"; |
|  | } |
|  |  |
|  | /\* definition of Game class function displayLoserScreen |
|  | \* displays formatted output to screen |
|  | \*/ |
|  | void Game::displayLoserScreen() |
|  | { |
|  | cout << "\n\n\n\t\t\t :: [ G A M E O V E R ] ::" |
|  | << "\n\n\n\n\t\t\t TILE\t SCORE\t NAME"; |
|  | printf("\n\n\t\t\t %4d\t %6d\t ", maxTile, score); |
|  | cin >> player.name; |
|  | savePlayerInfo(); |
|  |  |
|  | cout << "\n\n\t\t> The maximum possible score is 3,932,156 ! So close :p"; |
|  | } |
|  |  |
|  | /\* definition of Game class function displayTryAgainScreen |
|  | \* displays formatted output to screen |
|  | \* prompts user if they want to play again |
|  | \*/ |
|  | char Game::displayTryAgainScreen(int lose) |
|  | { |
|  | if (lose) |
|  | cout << "\n\n\n\t > Would you like to try again " << player.name << " (y/n) ? :: "; |
|  | else |
|  | cout << "\n\n\n\t > Would you like to continue playing " << player.name << " (y/n) ? :: "; |
|  |  |
|  | cin >> option; |
|  |  |
|  | return option; |
|  | } |
|  |  |
|  | /\* definition of Grid class function undo |
|  | \* sets current grid to backup grid |
|  | \*/ |
|  | void Grid::undo() |
|  | { |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<4; j++) |
|  | tiles[i][j] = btiles[i][j]; |
|  | } |
|  |  |
|  | /\* definition for Grid class function blockMoved |
|  | \* loops through the grid to check if backup grid is same as current grid |
|  | \* returns true if false true if so |
|  | \*@return bool |
|  | \*/ |
|  | bool Grid::blockMoved() |
|  | { |
|  | bool k = 0; |
|  |  |
|  | for (int i = 0; i<4; i++) |
|  | for (int j = 0; j<4; j++) |
|  | if (btiles[i][j] != tiles[i][j]) |
|  | { |
|  | k = 1; |
|  | break; |
|  | } |
|  | return k; |
|  | } |
|  |  |
|  | /\* definition for Game class function savePlayerInfo |
|  | \* writes to binary file the player's information |
|  | \*/ |
|  | void Game::savePlayerInfo() |
|  | { |
|  | fstream file; |
|  | file.open("players.dat", ios::out | ios::binary | ios::app); |
|  | file.write(player.name, strlen(player.name)); |
|  | file.write((reinterpret\_cast<const char \*>(&score)), sizeof(score)); |
|  | file.close(); |
|  | } |