Maze Escape

The Maze Escape project provides a fun puzzle game that creates unique mazes every time the user plays it. The less steps the player takes to escape the maze, the more points they will receive. Accumulating points will allow the player to purchase upgrades, making maze completion easier.

Features:

//Random Mazes

This feature creates a different maze each time the game is played through procedural generation. It uses characters to function as walls that block the player's movement. The open spaces are accessible to the player to move to. The spawn will be in the center of the maze.

//Player Movement

The character will be indicated visually and will be able to be moved using input from the player. The movement function will not activate if the direction chosen is blocked by a wall.

//Hidden Maze

The player will only be able to view the 8 surrounding tiles of their current position. When they move to a new tile the player’s new position surrounding tiles will then be revealed.

//Move counter

Each time the player makes a move, the counter will increase. This counter will determine the player's score and how many points they receive.

//Star Pickups

Pickups will be randomly generated within the maze. When the player moves to these tiles it will grant them extra points.

//Upgrades

Ghost Dash: Allows the player to walk through a wall

- can’t be used if there is a wall 2 tiles ahead

- can’t be used if it will result in player being outside of the border

Hawk Eye: Allows the user to see the surrounding 24 tiles (2 tiles ahead instead of 1) at the beginning of the game

Temp Sense (Temperature Sense): Gives the player a sense of how close they are to the exit by showing them whether they are “Hot”, “Cold”, etc.

Each upgrade to these abilities gives the player an extra chance to use them

//Save

Automatically records data to a save file every time the player enters the main menu

//Load

Searches for save file when program starts up. Returns the saved upgrades levels and points to the player.

//Menu

Play

Upgrade

Exit

//Exit

Closes the program.

|  |  |
| --- | --- |
| Constant Variables | |
| **Variables** | **Description** |
| \_SIZE = 6 | An int. Determines the size of the maze. |

|  |  |
| --- | --- |
| Functions | |
| **Names** | **Description** |
| main | The main driver for the program. |
| splashScreen | Displays the program's intro. |
| frameMiddle | Declutters the splashScreen by displaying repetitive output. |
| loadData | Reads data from the save.dat file into the ability and points integers. |
| menu | Gives the user the choice to play the game, upgrade abilities, or exit the program. |
| play | Sets up the game and allows the user to play the game. The points variable will be passed by reference and increase based on the outcome of the game. |
| createBlueprint | Creates a TileSet 2D array that will determine how the maze array will be created. |
| setTypes | Sets the values of a TileSet array passed by reference. The array be used to traversed through all possible TileSets. |
| createBpMiddle | Creates the middle areas of a TileSet 2D array that will  determine how the maze array will be created. |
| createMaze | Sets the values of the 2D string array based on the values of the TileSet array. |
| initMaze | Sets the values of the 2D string array to values that  display spaces. |
| displayMaze | Displays the 2D array along with the current ability values which represent how much uses they have left. |
| move | Moves the character across the maze. |
| showMaze | Copies the maze area surrounding the character's location to the hiddenMaze array. |
| upgrades | Displays the upgrade shop and allows the player to level up their abilities with points. |
| saveData | Writes data into the save.dat file from the ability and points integers. |
| exit | Prompts the user if they're sure they want to exit. If yes, an int value is returned that will cause the program to end. If no, the user will return to main menu. |

TileSet Class UML

|  |
| --- |
| **TileSet** |
| - string top  - string right  - string bottom  - string left  - bool midTop  - bool midRight  - bool midBottom  - bool midLeft  - bool doubleMid |
| + TileSet()  + TileSet(string,string,string,string, bool,bool,bool,bool, bool)  + TileSet(TileSet &a)  + string getTop()  + string getRight()  + string getBottom()  + string getLeft()  + bool getMidTop()  + bool getMidRight()  + bool getMidBottom()  + bool getMidLeft()  + bool getDoubleMid()  + void setTop(string)  + void setRight(string)  + void setBottom(string)  + void setLeft(string)  + void setMidTop(bool)  + void setMidRight(bool)  + void setMidBottom(bool)  + void setMidLeft(bool) |

Maze\_driver.cpp

// Name: Hendrick Christian

// COP1334 - Fall 2022 - TR 12:30PM

// Due Date: 12/9/22

// Assignment Description: Maze Project

// Headers //

#include "TileSet.h"

#include <Windows.h>

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

// Global Variables //

const int \_SIZE = 6;

// Prototypes //

void splashScreen(); //intro display

void frameMiddle(); //declutters splashScreen code

int menu();

void play(int, int, int, int&);

void createBlueprint(TileSet mazeBp[][\_SIZE]); //creates Maze Blueprint

void createBpMiddle(TileSet mazeBp[][\_SIZE], int i, int j); //creates the middles of the Maze Blueprint

void createMaze(string maze[][\_SIZE \* 4 + 1], TileSet mazeBp[][\_SIZE]); //creates the maze

void setTypes(TileSet lib[]); //creates every available tile set type and stores in an array

void initMaze(string m[][\_SIZE \* 4 + 1]); //initiallizes empty maze

void displayMaze(string m[][\_SIZE \* 4 + 1], int&, int&, int&); //displays maze

void move(string, int, string m[][\_SIZE \* 4 + 1], string hm[][\_SIZE \* 4 + 1], int&, bool&, int&, int&, int&, int&, int&, int&, bool&);

void showMaze(int, string m[][\_SIZE \* 4 + 1], string hm[][\_SIZE \* 4 + 1], int&, int&); //shows maze when player moves

void upgrades(int& gD, int& hE, int& tS, int& points);

void saveData(int& gD, int& hE, int& tS, int& points);

void loadData(int& gD, int& hE, int& tS, int& points);

int exit();

// Psuedocode //

/\*

//main

Show splash screen

Load data from save file

do

Show menu options. Return user choice.

Call menu options:

1. play

2. upgrades

3. exit

while not exit

Save data

return 0

//play

create maze blueprint

create maze

create hidden maze

show maze corners in hidden maze

spawn character in the middle of maze

show middle of maze in hidden maze

while the game is going give player options:

Movement - W, A, S, D

Ghost Dash - 1

Hawk Eye - 2

Temp Sense - 3

Quit - 4

after game ends, display:

moves made

stars collected

points gained

//upgrade

display:

ability name | ability level | level up cost | ability description

give player options to choose which ability to level up

//exit

make sure player wants to exit

if yes, end program

if no, return to main menu

\*/

// main //

int main()

{

int ghostDash, hawkEye, tempSense, points, menuChoice;

splashScreen();

loadData(ghostDash, hawkEye, tempSense, points);

do

{

menuChoice = menu();

if (menuChoice == 1)

play(ghostDash, hawkEye, tempSense, points);

else if (menuChoice == 2)

upgrades(ghostDash, hawkEye, tempSense, points);

else if (menuChoice == 3)

menuChoice = exit();

saveData(ghostDash, hawkEye, tempSense, points);

} while (menuChoice != 0);

return 0;

}

// Function Definitions //

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* splashScreen \*

\* \*

\* Called by: main \*

\* Displays the program's intro. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void splashScreen()

{

Sleep(1000);

//alternating double lined borders display

for (int i = 1; i <= 24; i++)

{

cout << "\n\n\n\n\n\n\n\n";

//first top line

cout << "\n";

if (i % 2 == 1) //odd

{

for (int j = i; j > 0; j--) //determines how much lines and spaces to display based on i

{

if (j % 2 == 1) //alternates the line display

cout << "----";

else

cout << " ";

}

}

else //even

{ //dispays correctly even when even

for (int j = i; j > 0; j--)

{

if (j % 2 == 0)

cout << "----";

else

cout << " ";

}

}

//second top line (same as the first line)

cout << "\n";

if (i % 2 == 1)

{

for (int j = i; j > 0; j--)

{

if (j % 2 == 1)

cout << "----";

else

cout << " ";

}

}

else

{

for (int j = i; j > 0; j--)

{

if (j % 2 == 0)

cout << "----";

else

cout << " ";

}

}

cout << "\n\n\n\n\n\n\n\n";

//first bottom line (inverse of top)

if (i % 2 == 0) //even

{

for (int j = i; j > 0; j--)

{

if (j % 2 == 1)

cout << "----";

else

cout << " ";

}

}

else //odd

{

for (int j = i; j > 0; j--)

{

if (j % 2 == 0)

cout << "----";

else

cout << " ";

}

}

//second bottom line (same as first line)

cout << "\n";

if (i % 2 == 0)

{

for (int j = i; j > 0; j--)

{

if (j % 2 == 1)

cout << "----";

else

cout << " ";

}

}

else

{

for (int j = i; j > 0; j--)

{

if (j % 2 == 0)

cout << "----";

else

cout << " ";

}

}

cout << "\n"; Sleep(150); //speed

system("CLS");

}

//Title Display

//TD frame 1

cout << "\n\n\n\n\n\n\n\n\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << "\n"; Sleep(600); system("CLS");

//TD frame 2

cout << "\n\n\n\n\n\n\n\n\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n";

cout << "\n";

cout << " Hendrick " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << "\n"; Sleep(600); system("CLS");

//TD frame 3

cout << "\n\n\n\n\n\n\n\n\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n";

cout << "\n";

cout << " Hendrick Christian " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << "\n"; Sleep(600); system("CLS");

//TD frame 4

cout << "\n\n\n\n\n\n\n\n\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n";

cout << "\n";

cout << " Hendrick Christian presents " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << "\n"; Sleep(1500); system("CLS");

//TD frame 5

cout << "\n\n\n\n\n\n\n\n\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n";

cout << "\n";

cout << " Hendrick Christian presents " << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << "\n"; Sleep(1200); system("CLS");

//Moving Borders

for (int i = 1; i <= 3; i ++)

{

//MB frame 1

cout << "\n\n\n\n\n\n\n\n\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

frameMiddle();

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << "\n"; Sleep(300); system("CLS");

//MB frame 2

cout << "\n\n\n\n\n\n\n\n\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

frameMiddle();

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n"; Sleep(300); system("CLS");

//MB frame 3

cout << "\n\n\n\n\n\n\n\n\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

frameMiddle();

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n"; Sleep(300); system("CLS");

//MB frame 4

cout << "\n\n\n\n\n\n\n\n\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

frameMiddle();

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n"; Sleep(300); system("CLS");

//MB frame 5

cout << "\n\n\n\n\n\n\n\n\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

frameMiddle();

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "\n"; Sleep(300); system("CLS");

//MB frame 6

cout << "\n\n\n\n\n\n\n\n\n";

cout << "- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---" << "\n";

cout << "- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---" << "\n";

frameMiddle();

cout << "--- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- -" << "\n";

cout << "--- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- -" << "\n";

cout << "\n"; Sleep(300); system("CLS");

//MB frame 7

cout << "\n\n\n\n\n\n\n\n\n";

cout << "-- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- --" << "\n";

cout << "-- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- --" << "\n";

frameMiddle();

cout << "-- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- --" << "\n";

cout << "-- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- --" << "\n";

cout << "\n"; Sleep(300); system("CLS");

//MB frame 8

cout << "\n\n\n\n\n\n\n\n\n";

cout << "--- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- -" << "\n";

cout << "--- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- -" << "\n";

frameMiddle();

cout << "- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---" << "\n";

cout << "- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---" << "\n";

cout << "\n"; Sleep(300); system("CLS");

}

//Menu Transition

//MT frame 1

cout << "\n\n\n\n\n\n\n\n\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

frameMiddle();

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << "\n"; Sleep(800); system("CLS");

//MT frame 2

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- " << "\n";

frameMiddle();

cout << " ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----" << "\n";

cout << "\n";

cout << "\n"; Sleep(500); system("CLS");

//MT frame 3

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

frameMiddle();

cout << "\n";

cout << "\n";

cout << "\n"; Sleep(500); system("CLS");

//MT frame 4

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n"; Sleep(500); system("CLS");

//MT frame 5

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n"; Sleep(500); system("CLS");

//MT frame 6

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n"; Sleep(500); system("CLS");

//MT frame 7

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n"; Sleep(500); system("CLS");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* frameMiddle \*

\* \*

\* Called by: splashScreen \*

\* Declutters the splashScreen by displaying repetitive output. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void frameMiddle()

{

cout << "\n\n";

cout << " Hendrick Christian presents ";

cout << "\n\n";

cout << " MAZE ESCAPE ";

cout << "\n\n\n";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* loadData \*

\* \*

\* Called by: main \*

\* Reads data from the save.dat file into the ability and points\*

\* integers. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void loadData(int& gD, int& hE, int& tS, int& points)

{

ifstream inF;

inF.open("save.dat");

if (!inF)

{

gD = 0;

hE = 0;

tS = 0;

points = 0;

}

else

{

inF >> gD >> hE >> tS >> points;

inF.close();

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* menu \*

\* \*

\* Called by: main \*

\* Displays the main menu. Returns an integer value that will \*

\* be used to call functions based on the user's choice. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int menu()

{

int x = -1;

system("CLS");

//Intro

//I frame 1

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n"; Sleep(700); system("CLS");

//I frame 2

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << " |1| Play " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n"; Sleep(250); system("CLS");

//I frame 3

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << " |1| Play " << "\n";

cout << " |2| Upgrade " << "\n";

cout << "\n";

cout << "\n";

cout << "\n";

cout << "\n"; Sleep(250); system("CLS");

do {

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << " |1| Play " << "\n";

cout << " |2| Upgrade " << "\n";

cout << " |3| Exit " << "\n";

cout << "\n";

cout << "\n"; Sleep(250);

cout << "\n\nChoice: ";

cin >> x;

system("CLS");

switch (x) {

case 1:

//case 1

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << " > |1| Play " << "\n";

cout << " |2| Upgrade " << "\n";

cout << " |3| Exit " << "\n";

cout << "\n";

cout << "\n"; Sleep(600); system("CLS"); return x;

break;

case 2:

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << " |1| Play " << "\n";

cout << " > |2| Upgrade " << "\n";

cout << " |3| Exit " << "\n";

cout << "\n";

cout << "\n"; Sleep(600); system("CLS"); return x;

break;

case 3:

cout << "\n\n\n\n\n\n\n\n\n";

cout << "\n";

cout << "\n";

cout << " MAZE ESCAPE " << "\n";

cout << "\n";

cout << " |1| Play " << "\n";

cout << " |2| Upgrade " << "\n";

cout << " > |3| Exit " << "\n";

cout << "\n";

cout << "\n"; Sleep(600); system("CLS"); return x;

break;

default:

return 4;

break;

}

} while (x != 0);

return -1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* play \*

\* \*

\* Called by: main \*

\* Sets up the game and allows the user to play the game. The \*

\* points variable will be passed by reference and increase \*

\* based on the outcome of the game. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void play(int gD, int hE, int tS, int& points)

{

int moveNum = 0, stars = 0, addedPoints;

bool win = false, quit = false,

dash = false, eyes = false, sense = false; //ability on or off

string choice;

int i\_cLoc = (\_SIZE \* 4) / 2, j\_cLoc = (\_SIZE \* 4) / 2, //character location

exitLoc; //exit location (i + j added)

srand(time(0));

//mazeBlueprint determines how the maze array will be created

TileSet mazeBlueprint[\_SIZE][\_SIZE];

createBlueprint(mazeBlueprint);

//create maze from blueprint

string maze[\_SIZE \* 4 + 1][\_SIZE \* 4 + 1];

createMaze(maze, mazeBlueprint);

//what the player will be shown

string hiddenMaze[\_SIZE \* 4 + 1][\_SIZE \* 4 + 1];

initMaze(hiddenMaze);

//show maze corners

hiddenMaze[0][0] = maze[0][0];

hiddenMaze[\_SIZE \* 4][0] = maze[\_SIZE \* 4][0];

hiddenMaze[0][\_SIZE \* 4] = maze[0][\_SIZE \* 4];

hiddenMaze[\_SIZE \* 4][\_SIZE \* 4] = maze[\_SIZE \* 4][\_SIZE \* 4];

//spawn character

maze[(\_SIZE \* 4) / 2][(\_SIZE \* 4) / 2] = " o ";

showMaze(2, maze, hiddenMaze, i\_cLoc, j\_cLoc);

displayMaze(hiddenMaze, gD, hE, tS);

//game

while (win != true && quit != true)

{

cout << "\nEnter choice: ";

cin >> choice;

//movement

if (choice == "w" || choice == "a" || choice == "s" || choice == "d")

{

if (dash == true)

{

move(choice, 2, maze, hiddenMaze, stars, win, gD, hE, tS, moveNum, i\_cLoc, j\_cLoc, eyes);

dash = false;

}

else

move(choice, 1, maze, hiddenMaze, stars, win, gD, hE, tS, moveNum, i\_cLoc, j\_cLoc, eyes);

}

//ghost dash toggle

if (choice == "1")

{

if (gD != 0)

{

gD--;

dash = true;

cout << "Ghost Dash: ACTIVE\n";

}

}

//hawk eye toggle

if (choice == "2")

{

if (hE != 0)

{

hE--;

eyes = true;

cout << "Hawk Eye: ACTIVE\n";

}

}

//temp sense

if (choice == "3")

{

if (tS != 0)

{

for (int i = 0; i < \_SIZE \* 4 + 1; i++)

{

for (int j = 0; j < \_SIZE \* 4 + 1; j++)

if (maze[i][j] == " " && (i == 0 || i == \_SIZE \* 4))

{

//cout << i << "." << j << endl;

exitLoc = i + j;

}

else if (maze[i][j] == " " && (j == 0 || j == \_SIZE \* 4))

{

//cout << i << "." << j << endl;

exitLoc = i + j;

}

}

tS--;

sense = true;

cout << "Temp Sense: ";

if (exitLoc >= i\_cLoc + j\_cLoc)

{

if (exitLoc - (i\_cLoc + j\_cLoc) < \_SIZE \* 2)

cout << "HOT\n";

else if (exitLoc - (i\_cLoc + j\_cLoc) < \_SIZE \* 4)

cout << "WARM\n";

else if (exitLoc - (i\_cLoc + j\_cLoc) < \_SIZE \* 6)

cout << "COLD\n";

else

cout << "FREEZING\n";

}

else

{

if ((i\_cLoc + j\_cLoc) - exitLoc < 10)

cout << "HOT\n";

else if ((i\_cLoc + j\_cLoc) - exitLoc < 25)

cout << "WARM\n";

else if ((i\_cLoc + j\_cLoc) - exitLoc < 36)

cout << "COLD\n";

else

cout << "FREEZING\n";

}

}

}

//quit game

if (choice == "0")

{

cout << "\nAre you sure you want to quit (Y or N)? ";

cin >> choice;

if (choice == "yes" || "Yes" || "Y" || "y" || "YES")

quit = true;

}

if (choice != "w" && choice != "a" && choice != "s" && choice != "d" && choice != "1" && choice != "2" && choice != "3")

cout << "Incorrect Choice\n";

}

//end

if (quit != true)

{

system("CLS");

cout << "\nMAZE CLEARED\n";

cout << "\nMoves Made: " << moveNum;

//adds points based on number of moves made

if (moveNum <= 10)

addedPoints = 100;

else if (moveNum <= 20)

addedPoints = 90;

else if (moveNum <= 30)

addedPoints = 70;

else if (moveNum <= 40)

addedPoints = 60;

else if (moveNum <= 50)

addedPoints = 50;

else if (moveNum <= 60)

addedPoints = 40;

else if (moveNum <= 70)

addedPoints = 30;

else if (moveNum <= 80)

addedPoints = 20;

else

addedPoints = 10;

cout << "\nStars Collected: " << stars;

addedPoints = addedPoints + (stars \* 25);

cout << "\nPoints Aquired: " << addedPoints << endl;

points = points + addedPoints;

cout << "\nTOTAL POINTS: " << points << endl;

system("pause");

system("cls");

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* createBlueprint \*

\* \*

\* Called by: play \*

\* Creates a TileSet 2D array that will determine how the maze \*

\* array will be created. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void createBlueprint(TileSet mazeBp[][\_SIZE])

{

TileSet tileSetLib[72]; //Tile Set Library will be used to traverse through tile sets

setTypes(tileSetLib);

TileSet temp;

TileSet array[72];

int count = 0;

int index = 0;

int num = 0;

string tslT; //tileSetLib[k] - Top

string tslL; //tileSetLib[k] - Left

string tslB; //tileSetLib[k] - Bottom

string tslR; //tileSetLib[k] - Right

string tempT; //temp - Top

string tempL; //temp - Left

string mbB; //top right set – Bottom

RETRY:

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

{

//cout << "\nI: " << i << endl;

for (int j = 0; j < \_SIZE; j++) //columns

{

//cout << "\nJ: " << j << endl;

//sets top

if (i == 0)

temp.setTop("00");

else //allows top to connect with the bottom of adjecent tileset

{

if (mazeBp[i - 1][j].getBottom() == "11")

temp.setTop("11");

else if (mazeBp[i - 1][j].getBottom() == "10")

temp.setTop("01");

else if (mazeBp[i - 1][j].getBottom() == "01")

temp.setTop("10");

else if (mazeBp[i - 1][j].getBottom() == "00")

temp.setTop("00");

else

{

cout << "\nERROR: Top\n";

system("pause");

}

}

//sets lefts

if (j == 0)

temp.setLeft("00");

else

{

if (mazeBp[i][j - 1].getRight() == "11")

temp.setLeft("11");

else if (mazeBp[i][j - 1].getRight() == "10")

temp.setLeft("01");

else if (mazeBp[i][j - 1].getRight() == "01")

temp.setLeft("10");

else if (mazeBp[i][j - 1].getRight() == "00")

temp.setLeft("00");

else

{

cout << "\nERROR: Left\n";

system("pause");

}

}

//set right if at border

if (i == \_SIZE - 1)

temp.setRight("00");

//set bottom if at border

if (j == \_SIZE - 1)

temp.setBottom("00");

count = 0;

for (int k = 0; k < 72; k++)

{

//cout << count << endl;

tslT = tileSetLib[k].getTop(); //tileSetLib[k] - Top

tslL = tileSetLib[k].getLeft(); //tileSetLib[k] - Left

tslB = tileSetLib[k].getBottom(); //tileSetLib[k] - Bottom

tslR = tileSetLib[k].getRight(); //tileSetLib[k] - Right

tempT = temp.getTop(); //temp - Top

tempL = temp.getLeft(); //temp - Left

if (j < \_SIZE - 1 && i > 1)

mbB = mazeBp[i - 1][j + 1].getBottom(); //top right set - Bottom

//cout << endl << i << "." << j << "." << k << "." << count << endl;

//cout << "TEMP: " << tempT << tempL << endl;

//cout << "TS: " << tslT << tslL << endl;

if (tslT == tempT && tslL == tempL && i == \_SIZE - 1 && j == \_SIZE - 1) //botttom right conner

{

//cout << "CASE: end corner\n";

if (tslB == "00" && tslR == "00")

{

array[count] = tileSetLib[k];

count++;

}

}

else if (tslT == tempT && tslL == tempL && i == \_SIZE - 1 && tslR != "00") //bottom border

{

//cout << "CASE: bottom border\n";

if (tslB == "00" && j + 1 == \_SIZE - 1 && j < \_SIZE - 1)

{

if (mbB == "00")

{

if (tslR == "11")

{

array[count] = tileSetLib[k];

count++;

}

}

else if (mbB == "10")

{

if (tslR == "01")

{

array[count] = tileSetLib[k];

count++;

}

}

else if (mbB == "01")

{

if (tslR != "00")

{

array[count] = tileSetLib[k];

count++;

}

}

}

else if (tslB == "00" && (mbB == "11" || mbB == "01"))

{

if (tslR == "00" || tslR == "01") //makes sure that future tilesets have possible conditions

{

array[count] = tileSetLib[k];

count++;

}

}

else if (tslB == "00")

{

array[count] = tileSetLib[k];

count++;

}

}

else if (tslT == tempT && tslL == tempL && j == \_SIZE - 1 && tslB != "00") //right border

{

//cout << "CASE: right border\n";

if (tslR == "00")

{

//more frequent double closed gate

/\*

num = 0;

closed = true;

while (closed != false && num < \_SIZE - 1)

{

if (mazeBp[i][num].getBottom() != "00")

closed = false;

num++;

}

if (closed == false)

{

num = \_SIZE - 1;

while (closed != true && num >= 0)

{

if (mazeBp[i][num].getLeft() == "00" && mazeBp[i][num].getBottom() == "00")

closed = true;

num--;

}

}

if (closed == false)

{

if (rand() % 2 == 1 && (mazeBp[i][j - 1].getBottom() == "10" || mazeBp[i][j - 1].getBottom() == "11"))

{

num = rand() % 3;

if (num == 0)

temp.setBottom("00");

else if (num == 1)

temp.setBottom("01");

else

temp.setBottom("10");

if (temp.getBottom() == tslB)

{

array[count] = tileSetLib[k];

count++;

}

}

else if (tslB != "00")

{

array[count] = tileSetLib[k];

count++;

}

}

else if (tslB != "00")

\*/

//{

array[count] = tileSetLib[k];

count++;

}

}

else if (tslT == tempT && tslL == tempL && (tslR != "00" || tslB != "00") && j != \_SIZE - 1 && i != \_SIZE - 1) //other

{

//cout << "CASE: other\n";

if (i > 0 && (mbB == "11" || mbB == "01"))

{

if (tslR == "00" || tslR == "01")

{

array[count] = tileSetLib[k];

count++;

}

}

else if (j == 1 && i == \_SIZE - 2 && mazeBp[\_SIZE - 2][0].getBottom() == "00")

{

if (tslB != "01" || tslB != "11")

{

array[count] = tileSetLib[k];

count++;

}

}

else

{

array[count] = tileSetLib[k];

count++;

}

}

}

if (count == 0)

{

system("CLS");

goto RETRY;

}

index = rand() % count; //gets random position in array

mazeBp[i][j] = array[index];

//

//cout << mazeBp[i][j].getTop() << mazeBp[i][j].getRight() << mazeBp[i][j].getBottom() << mazeBp[i][j].getLeft() << " ";

createBpMiddle(mazeBp, i, j);

}

//cout << endl;

}

system("pause");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* setTypes \*

\* \*

\* Called by: createBlueprint \*

\* Sets the values of a TileSet array passed by reference. The \*

\* array be used to traversed through all possible TileSets. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void setTypes(TileSet lib[])

{

string a = "00", b = "01", c = "10", d = "11";

bool t = true, f = false;

/\*

Tilesets will be represented as a 5 by 5 area in a string array

1 2 3 4 5

1 ~ X ~ X ~

2 X O / O X

3 ~ / ~ / ~

4 X O / O X

5 ~ X ~ X ~

X - Gates that can be open or closed and is represented as 1 or 0 respectively in the TileSet array.

The 1s and 0s come together to form a 2 digit string labeled as a side (top, right, bottom, left).

/ - Middle walls that will be present if the corresponding boolean value is true

O - Tiles that will always be open and traversable to the player

~ - Walls that will be determined based on their surrounding

To connect tilesets, the 1 and 5 lines will overlap with other tilesets that have a left and right or top and bottom that fit together.

Similar to a jigsaw puzzle.

Most tileset are only able to have one middle wall but some can have two. The tileset have the doubleMid boolean set to true.

Non-Explicitly Coded Rules: v(horizontal) (vertical)v

1. open gates cant be next to each other unless its part of the same orientation (top with bottom and left with right)

- Why: Because it affects how the non-gate tiles get decided and how the tilesets will connect.

More than 72 different types of tilesets would need to be considered, making it more difficult to figure out conditions

- example: a tileset with "01" Top cannot have a "10" right

2. Every side that has a double open gate MUST have a corresponding middle wall dividing those gates

- Why: Without it there would be large open areas within the maze

- example: a top with "11" must have midTop = TRUE

3. Every open gate gate MUST be traversable to another open gate

- Why: Without it tilesets could potentially close off paths to the player.

Note: when I say "exits" I'm referring to the tileset open gates

\*/

//CLASSIFICATION: two - num of exits, A - type of set within "two", 1 - rotation of set

//

// Two Exit tile sets /CONCEPT/

//twoA // │ │ │

TileSet twoA1(d,a,a,a, t,f,f,f, f); lib[0] = twoA1; // │ │ │

TileSet twoA2(a,d,a,a, f,t,f,f, f); lib[1] = twoA2; // │ │ │

TileSet twoA3(a,a,d,a, f,f,t,f, f); lib[2] = twoA3; // │ │

TileSet twoA4(a,a,a,d, f,f,f,t, f); lib[3] = twoA4; // └───────┘

//twoB // │ ┌────

TileSet twoB1(c,c,a,a, t,t,t,t, f); lib[4] = twoB1; // │ │

TileSet twoB2(a,c,c,a, t,t,t,t, f); lib[5] = twoB2; // │ │ │

TileSet twoB3(a,a,c,c, t,t,t,t, f); lib[6] = twoB3; // │ │

TileSet twoB4(c,a,a,c, t,t,t,t, f); lib[7] = twoB4; // └───────┘

//twoC // │ ┌───┐

TileSet twoC1(c,b,a,a, t,t,t,t, f); lib[8] = twoC1; // │ │ │

TileSet twoC2(a,c,b,a, t,t,t,t, f); lib[9] = twoC2; // │ │ │

TileSet twoC3(a,a,c,b, t,t,t,t, f); lib[10] = twoC3; // │

TileSet twoC4(b,a,a,c, t,t,t,t, f); lib[11] = twoC4; // └────────

//twoD // │ ┌───┐

TileSet twoD1(c,a,c,a, t,t,t,t, f); lib[12] = twoD1; // │ │ │

TileSet twoD2(a,c,a,c, t,t,t,t, f); lib[13] = twoD2; // │ │ │

// │ │

// └──── ┘

//twoE // │ ┌───┐

TileSet twoE1(c,a,b,a, t,t,t,t, f); lib[14] = twoE1; // │ │ │

TileSet twoE2(a,c,a,b, t,t,t,t, f); lib[15] = twoE2; // │ │ │

TileSet twoE3(b,a,c,a, t,t,t,t, f); lib[16] = twoE3; // │ │

TileSet twoE4(a,b,a,c, t,t,t,t, f); lib[17] = twoE4; // └ ────┘

//twoF // ┌───┐ │

TileSet twoF1(b,b,a,a, t,t,t,t, f); lib[18] = twoF1; // │ │ │

TileSet twoF2(a,b,b,a, t,t,t,t, f); lib[19] = twoF2; // │ │ │

TileSet twoF3(a,a,b,b, t,t,t,t, f); lib[20] = twoF3; // │

TileSet twoF4(b,a,a,b, t,t,t,t, f); lib[21] = twoF4; // └───────┘

//twoG // ┌───┐ │

TileSet twoG1(b,a,b,a, t,t,t,t, f); lib[22] = twoG1; // │ │ │

TileSet twoG2(a,b,a,b, t,t,t,t, f); lib[23] = twoG2; // │ │ │

// │ │

// └ ────┘

// Three Exit tile sets

//threeA // │ │ │

TileSet threeA1(d,b,a,a, t,f,f,f, f); lib[24] = threeA1; // │ │ │

TileSet threeA2(a,d,b,a, f,t,f,f, f); lib[25] = threeA2; // │ │ │

TileSet threeA3(a,a,d,b, f,f,t,f, f); lib[26] = threeA3; // │

TileSet threeA4(b,a,a,d, f,f,f,t, f); lib[27] = threeA4; // └───────┘

//threeB // │ │ │

TileSet threeB1(d,a,c,a, t,f,f,f, f); lib[28] = threeB1; // │ │ │

TileSet threeB2(a,d,a,c, f,t,f,f, f); lib[29] = threeB2; // │ │ │

TileSet threeB3(c,a,d,a, f,f,t,f, f); lib[30] = threeB3; // │ │

TileSet threeB4(a,c,a,d, f,f,f,t, f); lib[31] = threeB4; // └──── ┘

//threeC // │ │ │

TileSet threeC1(d,a,b,a, t,f,f,f, f); lib[32] = threeC1; // │ │ │

TileSet threeC2(a,d,a,b, f,t,f,f, f); lib[33] = threeC2; // │ │ │

TileSet threeC3(b,a,d,a, f,f,t,f, f); lib[34] = threeC3; // │ │

TileSet threeC4(a,b,a,d, f,f,f,t, f); lib[35] = threeC4; // └ ────┘

//threeD // │ │ │

TileSet threeD1(d,a,a,c, t,f,f,f, f); lib[36] = threeD1; // │ │ │

TileSet threeD2(c,d,a,a, f,t,f,f, f); lib[37] = threeD2; // │ │ │

TileSet threeD3(a,c,d,a, f,f,t,f, f); lib[38] = threeD3; // │

TileSet threeD4(a,a,c,d, f,f,f,t, f); lib[39] = threeD4; // └───────┘

//threeE // │ ┌───┐

TileSet threeE1(c,c,c,a, t,t,t,t, f); lib[40] = threeE1; // │ │

TileSet threeE2(a,c,c,c, t,t,t,t, f); lib[41] = threeE2; // │ │ │

TileSet threeE3(c,a,c,c, t,t,t,t, f); lib[42] = threeE3; // │ │

TileSet threeE4(c,c,a,c, t,t,t,t, f); lib[43] = threeE4; // └──── ┘

//threeF // │ ┌───┐

TileSet threeF1(c,c,b,a, t,t,t,t, f); lib[44] = threeF1; // │ │

TileSet threeF2(a,c,c,b, t,t,t,t, f); lib[45] = threeF2; // │ │ |

TileSet threeF3(b,a,c,c, t,t,t,t, f); lib[46] = threeF3; // │ |

TileSet threeF4(c,b,a,c, t,t,t,t, f); lib[47] = threeF4; // └ ────┘

//threeG // │ ┌───┐

TileSet threeG1(c,b,b,a, t,t,t,t, f); lib[48] = threeG1; // │ │ |

TileSet threeG2(a,c,b,b, t,t,t,t, f); lib[49] = threeG2; // │ │ |

TileSet threeG3(b,a,c,b, t,t,t,t, f); lib[50] = threeG3; // |

TileSet threeG4(b,b,a,c, t,t,t,t, f); lib[51] = threeG4; // └ ────┘

//threeH // ┌───┐ │

TileSet threeH1(b,b,b,a, t,t,t,t, f); lib[52] = threeH1; // │ │ |

TileSet threeH2(a,b,b,b, t,t,t,t, f); lib[53] = threeH2; // │ │ |

TileSet threeH3(b,a,b,b, t,t,t,t, f); lib[54] = threeH3; // │

TileSet threeH4(b,b,a,b, t,t,t,t, f); lib[55] = threeH4; // └ ────┘

// Four Exit tile sets

//fourA // │ │ │

TileSet fourA1(d,a,d,a, t,f,t,f, t); lib[56] = fourA1; // │ │ |

TileSet fourA2(a,d,a,d, f,t,f,t, t); lib[57] = fourA2; // │ │ |

// │ │ |

// │ │ |

//fourB // │ │ │

TileSet fourB1(d,b,b,a, t,f,t,f, t); lib[58] = fourB1; // │ │ |

TileSet fourB2(a,d,b,b, f,t,f,t, t); lib[59] = fourB2; // │ │ |

TileSet fourB3(b,a,d,b, t,f,t,f, t); lib[60] = fourB3; // │ │

TileSet fourB4(b,b,a,d, f,t,f,t, t); lib[61] = fourB4; // └ ────┘

//fourC // │ │ │

TileSet fourC1(d,b,a,c, t,f,t,f, t); lib[62] = fourC1; // │ │ |

TileSet fourC2(c,d,b,a, f,t,f,t, t); lib[63] = fourC2; // │ │ |

TileSet fourC3(a,c,d,b, t,f,t,f, t); lib[64] = fourC3; // │

TileSet fourC4(b,a,c,d, f,t,f,t, t); lib[65] = fourC4; // └───────┘

//fourD // │ │ │

TileSet fourD1(d,a,c,c, t,f,t,f, t); lib[66] = fourD1; // │ │ |

TileSet fourD2(c,d,a,c, f,t,f,t, t); lib[67] = fourD2; // │ │ |

TileSet fourD3(c,c,d,a, t,f,t,f, t); lib[68] = fourD3; // │ |

TileSet fourD4(a,c,c,d, f,t,f,t, t); lib[69] = fourD4; // └──── ┘

//fourE // │ ┌───┐

TileSet fourE1(c,c,c,c, t,t,t,t, t); lib[70] = fourE1; // │ │

// │ │ |

// │ |

// └──── ┘

//fourF // ┌───┐ │

TileSet fourF1(b,b,b,b, t,t,t,t, t); lib[71] = fourF1; // │ |

// │ │ |

// │ │

// └ ────┘

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* createBpMiddle \*

\* \*

\* Called by: createBlueprint \*

\* Creates the middle areas of a TileSet 2D array that will \*

\* determine how the maze array will be created. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void createBpMiddle(TileSet mazeBp[][\_SIZE], int i, int j)

{

int count = 0;

if (mazeBp[i][j].getMidTop() == true)

count++;

if (mazeBp[i][j].getMidRight() == true)

count++;

if (mazeBp[i][j].getMidBottom() == true)

count++;

if (mazeBp[i][j].getMidLeft() == true)

count++;

if (mazeBp[i][j].getDoubleMid() == false && count > 1) //if count == 1 then values are already correct

{

count = rand() % count + 1;

switch (count) {

case 1:

mazeBp[i][j].setMidRight(false);

mazeBp[i][j].setMidBottom(false);

mazeBp[i][j].setMidLeft(false);

break;

case 2:

mazeBp[i][j].setMidTop(false);

mazeBp[i][j].setMidBottom(false);

mazeBp[i][j].setMidLeft(false);

break;

case 3:

mazeBp[i][j].setMidTop(false);

mazeBp[i][j].setMidRight(false);

mazeBp[i][j].setMidLeft(false);

break;

case 4:

mazeBp[i][j].setMidTop(false);

mazeBp[i][j].setMidRight(false);

mazeBp[i][j].setMidBottom(false);

break;

default:

cout << "\nERROR: invalid count for double mid = " << count << endl;

system("pause");

}

}

else if (mazeBp[i][j].getDoubleMid() == true)

{

count = rand() % 2;

if ((mazeBp[i][j].getTop() == "11" && mazeBp[i][j].getBottom() == "11") || (mazeBp[i][j].getRight() == "11" && mazeBp[i][j].getLeft() == "11"))

;//already has correct values

else if (mazeBp[i][j].getTop() == "11") //50% chance that mid is double or single

{

if (count == 1)

mazeBp[i][j].setMidBottom(false);

//else if count == 0 , do nothing

}

else if (mazeBp[i][j].getRight() == "11")

{

if (count == 1)

mazeBp[i][j].setMidLeft(false);

}

else if (mazeBp[i][j].getBottom() == "11")

{

if (count == 1)

mazeBp[i][j].setMidTop(false);

}

else if (mazeBp[i][j].getLeft() == "11")

{

if (count == 1)

mazeBp[i][j].setMidRight(false);

}

else

{

count = rand() % 4 + 1;

switch (count) {

case 1:

mazeBp[i][j].setMidRight(false);

mazeBp[i][j].setMidBottom(false);

mazeBp[i][j].setMidLeft(false);

count = rand() % 2;

if (count == 1)

mazeBp[i][j].setMidBottom(true);

break;

case 2:

mazeBp[i][j].setMidTop(false);

mazeBp[i][j].setMidBottom(false);

mazeBp[i][j].setMidLeft(false);

count = rand() % 2;

if (count == 1)

mazeBp[i][j].setMidLeft(true);

break;

case 3:

mazeBp[i][j].setMidTop(false);

mazeBp[i][j].setMidRight(false);

mazeBp[i][j].setMidLeft(false);

count = rand() % 2;

if (count == 1)

mazeBp[i][j].setMidTop(true);

break;

case 4:

mazeBp[i][j].setMidTop(false);

mazeBp[i][j].setMidRight(false);

mazeBp[i][j].setMidBottom(false);

count = rand() % 2;

if (count == 1)

mazeBp[i][j].setMidRight(false);

break;

}

}

}

else if (count == 0)

{

cout << "\n\tERROR: Middle - count = 0\n";

system("pause");

}

if (mazeBp[i][j].getMidTop() == false && mazeBp[i][j].getMidRight() == false && mazeBp[i][j].getMidBottom() == false && mazeBp[i][j].getMidLeft() == false)

{

cout << "\n\tERROR: Middle - All False\n";

system("pause");

}

//

//cout << "M" << mazeBp[i][j].getMidTop() << mazeBp[i][j].getMidRight() << mazeBp[i][j].getMidBottom() << mazeBp[i][j].getMidLeft() << "| ";

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* createMaze \*

\* \*

\* Called by: play \*

\* Sets the values of the 2D string array based on the values \*

\* of the TileSet array. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void createMaze(string maze[][\_SIZE \* 4 + 1], TileSet mazeBp[][\_SIZE])

{

initMaze(maze);

int bp\_i = 0;

int bp\_j = 0;

//Horizontal Gates

for (int i = 0; i < \_SIZE \* 4 + 1; i = i + 4)

{

for (int j = 1; j < \_SIZE \* 4 + 1; j = j + 4)

{

//Tops

if (j < \_SIZE \* 4 - 1 && i != \_SIZE \* 4)

{

bp\_i = i / 4;

bp\_j = j / 4;

if (mazeBp[bp\_i][bp\_j].getTop() == "00")

{

maze[i][j] = "─";

maze[i][j + 2] = "─";

}

else if (mazeBp[bp\_i][bp\_j].getTop() == "01")

{

maze[i][j] = "─";

maze[i][j + 2] = " ";

}

else if (mazeBp[bp\_i][bp\_j].getTop() == "10")

{

maze[i][j] = " ";

maze[i][j + 2] = "─";

}

else if (mazeBp[bp\_i][bp\_j].getTop() == "11")

{

maze[i][j] = " ";

maze[i][j + 2] = " ";

}

}

//Bottom

else if (j < \_SIZE \* 4 - 1 && i == \_SIZE \* 4)

{

bp\_j = j / 4;

if (mazeBp[bp\_i][bp\_j].getBottom() == "00")

{

maze[i][j] = "─";

maze[i][j + 2] = "─";

}

else if (mazeBp[bp\_i][bp\_j].getBottom() == "01")

{

maze[i][j] = " ";

maze[i][j + 2] = "─";

}

else if (mazeBp[bp\_i][bp\_j].getBottom() == "10")

{

maze[i][j] = "─";

maze[i][j + 2] = " ";

}

else if (mazeBp[bp\_i][bp\_j].getBottom() == "11")

{

maze[i][j] = " ";

maze[i][j + 2] = " ";

}

}

else

cout << "\nERROR: Horizontal Gate\n";

}

}

//Vertical Gates

for (int i = 1; i < \_SIZE \* 4 + 1; i = i + 4)

{

for (int j = 0; j < \_SIZE \* 4 + 1; j = j + 4)

{

//Lefts

if (i < \_SIZE \* 4 - 1 && j != \_SIZE \* 4)

{

bp\_i = i / 4;

bp\_j = j / 4;

if (j == 0)

{

if (mazeBp[bp\_i][bp\_j].getLeft() == "00")

{

maze[i][j] = "│ ";

maze[i + 2][j] = "│ ";

}

else if (mazeBp[bp\_i][bp\_j].getLeft() == "01")

{

maze[i][j] = " ";

maze[i + 2][j] = "│ ";

}

else if (mazeBp[bp\_i][bp\_j].getLeft() == "10")

{

maze[i][j] = "│ ";

maze[i + 2][j] = " ";

}

else if (mazeBp[bp\_i][bp\_j].getLeft() == "11")

{

maze[i][j] = " ";

maze[i + 2][j] = " ";

}

}

else

{

if (mazeBp[bp\_i][bp\_j].getLeft() == "00")

{

maze[i][j] = " │ ";

maze[i + 2][j] = " │ ";

}

else if (mazeBp[bp\_i][bp\_j].getLeft() == "01")

{

maze[i][j] = " ";

maze[i + 2][j] = " │ ";

}

else if (mazeBp[bp\_i][bp\_j].getLeft() == "10")

{

maze[i][j] = " │ ";

maze[i + 2][j] = " ";

}

else if (mazeBp[bp\_i][bp\_j].getLeft() == "11")

{

maze[i][j] = " ";

maze[i + 2][j] = " ";

}

}

}

//Right

else if (i < \_SIZE \* 4 - 1 && j == \_SIZE \* 4)

{

bp\_j = (\_SIZE \* 4 - 1) / 4;

if (mazeBp[bp\_i][bp\_j].getRight() == "00")

{

maze[i][j] = " │";

maze[i + 2][j] = " │";

}

else if (mazeBp[bp\_i][bp\_j].getRight() == "01")

{

maze[i][j] = " ";

maze[i + 2][j] = " │";

}

else if (mazeBp[bp\_i][bp\_j].getRight() == "10")

{

maze[i][j] = " │";

maze[i + 2][j] = " ";

}

else if (mazeBp[bp\_i][bp\_j].getRight() == "11")

{

maze[i][j] = " ";

maze[i + 2][j] = " ";

}

}

else

cout << "\nERROR: Vertical Gate\n";

}

}

//MIDDLE BARRIERS

for (int i = 0; i < \_SIZE \* 4 - 1; i = i + 4)

{

for (int j = 0; j < \_SIZE \* 4 - 1; j = j + 4)

{

bp\_i = i / 4;

bp\_j = j / 4;

if (mazeBp[bp\_i][bp\_j].getMidTop() == true)

maze[i + 1][j + 2] = " │ ";

if(mazeBp[bp\_i][bp\_j].getMidRight() == true)

maze[i + 2][j + 3] = "─";

if (mazeBp[bp\_i][bp\_j].getMidBottom() == true)

maze[i + 3][j + 2] = " │ ";

if (mazeBp[bp\_i][bp\_j].getMidLeft() == true)

maze[i + 2][j + 1] = "─";

}

}

//Set Spawn

for (int i = (\_SIZE \* 4 + 1) / 2 - 1; i <= (\_SIZE \* 4 + 1) / 2 + 1; i++)

{

for (int j = (\_SIZE \* 4 + 1) / 2 - 1; j <= (\_SIZE \* 4 + 1) / 2 + 1; j++)

{

if (j % 2 == 1)

maze[i][j] = " ";

else

maze[i][j] = " ";

}

}

//Set Exit

int exitPos = 1, border = rand() % 4 + 1;

do

exitPos = rand() % (\_SIZE \* 4) + 1;

while (exitPos % 2 == 0);

switch (border) {

case 1: //Top

maze[0][exitPos] = " ";

break;

case 2: //Right

maze[exitPos][\_SIZE \* 4] = " ";

break;

case 3: //Bottom

maze[\_SIZE \* 4][exitPos] = " ";

break;

case 4: //Left

maze[exitPos][0] = " ";

break;

default:

cout << "\n\tERROR: Exit Positon\n";

system("pause");

}

//CONNECT PIECES

string mTop, mRight, mBottom, mLeft;

for (int i = 0; i < \_SIZE \* 4 + 1; i = i + 2)

{

for (int j = 0; j < \_SIZE \* 4 + 1; j = j + 2)

{

//bp\_i = i / 4;

//bp\_j = j / 4;

if (i > 1)

mTop = maze[i - 1][j];

if (j < \_SIZE \* 4)

mRight = maze[i][j + 1];

if (i < \_SIZE \* 4)

mBottom = maze[i + 1][j];

if (j > 1)

mLeft = maze[i][j - 1];

if (i == 0 && j == 0) //top left corner

maze[i][j] = "┌─";

else if (i == 0 && j == \_SIZE \* 4) //top right corner

maze[i][j] = "─┐";

else if (i == \_SIZE \* 4 && j == 0) //bottom left corner

maze[i][j] = "└─";

else if (i == \_SIZE \* 4 && j == \_SIZE \* 4) //bottom right corner

maze[i][j] = "─┘";

else if (j == 0) //Left border

{

if (maze[i][j + 1] != " ")

maze[i][j] = "├─";

else

maze[i][j] = "│ ";

}

else if (j == \_SIZE \* 4) //Right border

{

if (maze[i][j - 1] != " ")

maze[i][j] = "─┤";

else

maze[i][j] = " │";

}

else if (i == 0) //Top border

{

if ( maze[i + 1][j] != " ")

maze[i][j] = "─┬─";

else

maze[i][j] = "───";

}

else if (i == \_SIZE \* 4) //Bottom border

{

if (maze[i - 1][j] != " ")

maze[i][j] = "─┴─";

else

maze[i][j] = "───";

}

else //Middle area

{

if (mTop != " " && mRight != " " && mBottom != " " && mLeft != " ")

maze[i][j] = "─┼─";

else if (mTop == " " && mRight != " " && mBottom != " " && mLeft != " ")

maze[i][j] = "─┬─";

else if (mTop != " " && mRight == " " && mBottom != " " && mLeft != " ")

maze[i][j] = "─┤ ";

else if (mTop != " " && mRight != " " && mBottom == " " && mLeft != " ")

maze[i][j] = "─┴─";

else if (mTop != " " && mRight != " " && mBottom != " " && mLeft == " ")

maze[i][j] = " ├─";

else if (mTop == " " && mRight == " " && mBottom != " " && mLeft != " ")

maze[i][j] = "─┐ ";

else if (mTop == " " && mRight != " " && mBottom == " " && mLeft != " ")

maze[i][j] = "───";

else if (mTop == " " && mRight != " " && mBottom != " " && mLeft == " ")

maze[i][j] = " ┌─";

else if (mTop != " " && mRight == " " && mBottom == " " && mLeft != " ")

maze[i][j] = "─┘ ";

else if (mTop != " " && mRight == " " && mBottom != " " && mLeft == " ")

maze[i][j] = " │ ";

else if (mTop != " " && mRight != " " && mBottom == " " && mLeft == " ")

maze[i][j] = " └─";

else if (mTop == " " && mRight == " " && mBottom == " " && mLeft != " ")

maze[i][j] = "── ";

else if (mTop == " " && mRight == " " && mBottom != " " && mLeft == " ")

maze[i][j] = " │ ";

else if (mTop == " " && mRight != " " && mBottom == " " && mLeft == " ")

maze[i][j] = " ──";

else if (mTop != " " && mRight == " " && mBottom == " " && mLeft == " ")

maze[i][j] = " │ ";

else

maze[i][j] = " ";

}

}

}

//EXP

int rand\_i;

int rand\_j;

for (int k = 0; k < 5; k++)

{

do rand\_i = rand() % (\_SIZE \* 4 + 1);

while (rand\_i % 2 != 1);

do rand\_j = rand() % (\_SIZE \* 4 + 1);

while (rand\_j % 2 != 1);

if (maze[rand\_i][rand\_j] != "\*")

maze[rand\_i][rand\_j] = "\*";

else k--;

}

//└ ┐ ┘ ┌ │ ─ ┬ ┤ ├ ┴ ┼

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* initMaze \*

\* \*

\* Called by: play, createMaze \*

\* Sets the values of the 2D string array to values that \*

\* display spaces. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initMaze(string m[][\_SIZE \* 4 + 1])

{

for (int i = 0; i < \_SIZE \* 4 + 1; i++)

{

for (int j = 0; j < \_SIZE \* 4 + 1; j++)

{

if (j == 0 || j == \_SIZE \* 4)

m[i][j] = " ";

else if (j % 2 == 1)

m[i][j] = " ";

else

m[i][j] = " ";

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* displayMaze \*

\* \*

\* Called by: play, displayMaze \*

\* Displays the 2D array along with the current ability values \*

\* which represent how much uses they have left. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void displayMaze(string m[][\_SIZE \* 4 + 1], int& gD, int& hE, int& tS)

{

cout << endl;

//cout << "\t0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4\n";

for (int i = 0; i < \_SIZE \* 4 + 1; i++) //25 rows

{

cout << "\t";

//cout << i << "\t";

for (int j = 0; j < \_SIZE \* 4 + 1; j++) //25 cols

cout << m[i][j];

if (i == 1)

cout << "\t\tCONTROLS:\n";

else if (i == 3)

cout << "\t\t\tUp - W\n";

else if (i == 4)

cout << "\t\t\tRight - D\n";

else if (i == 5)

cout << "\t\t\tDown - S\n";

else if (i == 6)

cout << "\t\t\tLeft - A\n";

else if (i == 8)

cout << "\t\t\tGhost Dash - 1\n";

else if (i == 9)

cout << "\t\t\tHawk Eye - 2\n";

else if (i == 10)

cout << "\t\t\tTemp Sense - 3\n";

else if (i == 12)

cout << "\t\t\tQuit - 0\n";

else if (i == 19)

cout << "\t\tGHOST DASH: " << gD << endl;

else if (i == 21)

cout << "\t\tHAWK EYE: " << hE << endl;

else if (i == 23)

cout << "\t\tTEMP SENSE: " << tS << endl;

else

cout << endl;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* move \*

\* \*

\* Called by: play \*

\* Moves the character across the maze. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void move(string direction, int distance, string m[][\_SIZE \* 4 + 1], string hm[][\_SIZE \* 4 + 1], int &stars, bool &win, int& gD, int& hE, int& tS, int& moveNum, int& i\_cLoc, int& j\_cLoc, bool& eyes)

{

if (distance == 1) //no dash

{

if (direction == "w") //up

{

if (m[i\_cLoc - 1][j\_cLoc] == " " && i\_cLoc - 1 != 0)

{

m[i\_cLoc - 1][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

i\_cLoc--;

moveNum++;

}

else if (m[i\_cLoc - 1][j\_cLoc] == " ")

{

m[i\_cLoc - 1][j\_cLoc] = " o ";

m[i\_cLoc][j\_cLoc] = " ";

i\_cLoc--;

moveNum++;

}

else if (m[i\_cLoc - 1][j\_cLoc] == "\*")

{

m[i\_cLoc - 1][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

stars++;

i\_cLoc--;

moveNum++;

}

else if (m[i\_cLoc - 1][j\_cLoc] == " ")

{

m[i\_cLoc - 1][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

i\_cLoc--;

moveNum++;

win = true;

}

}

else if (direction == "d") //Right

{

if (m[i\_cLoc][j\_cLoc + 1] == " ")

{

m[i\_cLoc][j\_cLoc + 1] = "o";

m[i\_cLoc][j\_cLoc] = " ";

j\_cLoc++;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc + 1] == " ")

{

m[i\_cLoc][j\_cLoc + 1] = " o ";

m[i\_cLoc][j\_cLoc] = " ";

j\_cLoc++;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc + 1] == "\*")

{

m[i\_cLoc][j\_cLoc + 1] = "o";

m[i\_cLoc][j\_cLoc] = " ";

stars++;

j\_cLoc++;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc + 1] == " ")

{

m[i\_cLoc][j\_cLoc + 1] = " o";

m[i\_cLoc][j\_cLoc] = " ";

j\_cLoc++;

moveNum++;

win = true;

}

}

else if (direction == "s") //bottom

{

if (m[i\_cLoc + 1][j\_cLoc] == " " && i\_cLoc + 1 != \_SIZE \* 4)

{

m[i\_cLoc + 1][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

i\_cLoc++;

moveNum++;

}

else if (m[i\_cLoc + 1][j\_cLoc] == " ")

{

m[i\_cLoc + 1][j\_cLoc] = " o ";

m[i\_cLoc][j\_cLoc] = " ";

i\_cLoc++;

moveNum++;

}

else if (m[i\_cLoc + 1][j\_cLoc] == "\*")

{

m[i\_cLoc + 1][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

stars++;

i\_cLoc++;

moveNum++;

}

else if (m[i\_cLoc + 1][j\_cLoc] == " ")

{

m[i\_cLoc + 1][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

i\_cLoc++;

moveNum++;

win = true;

}

}

else if (direction == "a") //left

{

if (m[i\_cLoc][j\_cLoc - 1] == " ")

{

m[i\_cLoc][j\_cLoc - 1] = "o";

m[i\_cLoc][j\_cLoc] = " ";

j\_cLoc--;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc - 1] == " ")

{

m[i\_cLoc][j\_cLoc - 1] = " o ";

m[i\_cLoc][j\_cLoc] = " ";

j\_cLoc--;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc - 1] == "\*")

{

m[i\_cLoc][j\_cLoc - 1] = "o";

m[i\_cLoc][j\_cLoc] = " ";

stars++;

j\_cLoc--;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc - 1] == " ")

{

m[i\_cLoc][j\_cLoc - 1] = " o";

m[i\_cLoc][j\_cLoc] = " ";

j\_cLoc--;

moveNum++;

win = true;

}

}

//show maze

if (eyes == true)

{

showMaze(2, m, hm, i\_cLoc, j\_cLoc);

eyes = false;

}

else

showMaze(1, m, hm, i\_cLoc, j\_cLoc);

}

else //dash is active

{

if (direction == "w" && i\_cLoc - 2 > -1) //up

{

if (m[i\_cLoc - 2][j\_cLoc] == " " && i\_cLoc - 2 != 0)

{

m[i\_cLoc - 2][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

i\_cLoc = i\_cLoc - 2;

moveNum++;

}

else if (m[i\_cLoc - 2][j\_cLoc] == " ")

{

m[i\_cLoc - 2][j\_cLoc] = " o ";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

i\_cLoc = i\_cLoc - 2;

moveNum++;

}

else if (m[i\_cLoc - 2][j\_cLoc] == "\*")

{

m[i\_cLoc - 2][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

stars++;

i\_cLoc = i\_cLoc - 2;

moveNum++;

}

else if (m[i\_cLoc - 2][j\_cLoc] == " ")

{

m[i\_cLoc - 2][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

i\_cLoc = i\_cLoc - 2;

moveNum++;

win = true;

}

}

else if (direction == "d" && j\_cLoc + 2 < 25) //Right

{

if (m[i\_cLoc][j\_cLoc + 2] == " ")

{

m[i\_cLoc][j\_cLoc + 2] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

j\_cLoc = j\_cLoc + 2;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc + 2] == " ")

{

m[i\_cLoc][j\_cLoc + 2] = " o ";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

j\_cLoc = j\_cLoc + 2;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc + 2] == "\*")

{

m[i\_cLoc][j\_cLoc + 2] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

stars++;

j\_cLoc = j\_cLoc + 2;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc + 2] == " ")

{

m[i\_cLoc][j\_cLoc + 2] = " o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

j\_cLoc = j\_cLoc + 2;

moveNum++;

win = true;

}

}

else if (direction == "s" && i\_cLoc + 2 < 25) //bottom

{

if (m[i\_cLoc + 2][j\_cLoc] == " " && i\_cLoc + 2 != \_SIZE \* 4)

{

m[i\_cLoc + 2][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

i\_cLoc = i\_cLoc + 2;

moveNum++;

}

else if (m[i\_cLoc + 2][j\_cLoc] == " ")

{

m[i\_cLoc + 2][j\_cLoc] = " o ";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

i\_cLoc = i\_cLoc + 2;

moveNum++;

}

else if (m[i\_cLoc + 2][j\_cLoc] == "\*")

{

m[i\_cLoc + 2][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

stars++;

i\_cLoc = i\_cLoc + 2;

moveNum++;

}

else if (m[i\_cLoc + 2][j\_cLoc] == " ")

{

m[i\_cLoc + 2][j\_cLoc] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

i\_cLoc = i\_cLoc + 2;

moveNum++;

win = true;

}

}

else if (direction == "a" && j\_cLoc - 2 > -1) //left

{

if (m[i\_cLoc][j\_cLoc - 2] == " ")

{

m[i\_cLoc][j\_cLoc - 2] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

j\_cLoc = j\_cLoc - 2;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc - 2] == " ")

{

m[i\_cLoc][j\_cLoc - 2] = " o ";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

j\_cLoc = j\_cLoc - 2;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc - 2] == "\*")

{

m[i\_cLoc][j\_cLoc - 2] = "o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

stars++;

j\_cLoc = j\_cLoc - 2;

moveNum++;

}

else if (m[i\_cLoc][j\_cLoc - 2] == " ")

{

m[i\_cLoc][j\_cLoc - 2] = " o";

m[i\_cLoc][j\_cLoc] = " ";

hm[i\_cLoc][j\_cLoc] = " ";

j\_cLoc = j\_cLoc - 2;

moveNum++;

win = true;

}

}

//show maze

if (eyes == true)

{

showMaze(2, m, hm, i\_cLoc, j\_cLoc);

eyes = false;

}

else

showMaze(1, m, hm, i\_cLoc, j\_cLoc);

}

system("CLS");

displayMaze(hm, gD, hE, tS);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* showMaze \*

\* \*

\* Called by: play, move \*

\* Copies the maze area surrounding the character's location to \*

\* the hiddenMaze. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void showMaze(int radius, string m[][\_SIZE \* 4 + 1], string hm[][\_SIZE \* 4 + 1],int& i\_cLoc, int& j\_cLoc )

{

if (radius == 1) //no hawk eye

{

hm[i\_cLoc][j\_cLoc] = m[i\_cLoc][j\_cLoc]; //center

if (i\_cLoc - 1 >= 0 && j\_cLoc - 1 >= 0) //top left

hm[i\_cLoc - 1][j\_cLoc - 1] = m[i\_cLoc - 1][j\_cLoc - 1];

if (i\_cLoc - 1 >= 0) //top

hm[i\_cLoc - 1][j\_cLoc] = m[i\_cLoc - 1][j\_cLoc];

if (i\_cLoc - 1 >= 0 && j\_cLoc + 1 <= \_SIZE \* 4) //top right

hm[i\_cLoc - 1][j\_cLoc + 1] = m[i\_cLoc - 1][j\_cLoc + 1];

if (j\_cLoc + 1 <= \_SIZE \* 4) //right

hm[i\_cLoc][j\_cLoc + 1] = m[i\_cLoc][j\_cLoc + 1];

if (i\_cLoc + 1 <= \_SIZE \* 4 && j\_cLoc + 1 <= \_SIZE \* 4) //bottom right

hm[i\_cLoc + 1][j\_cLoc + 1] = m[i\_cLoc + 1][j\_cLoc + 1];

if (i\_cLoc + 1 <= \_SIZE \* 4) //bottom

hm[i\_cLoc + 1][j\_cLoc] = m[i\_cLoc + 1][j\_cLoc];

if (i\_cLoc + 1 <= \_SIZE \* 4 && j\_cLoc - 1 >= 0) //bottom left

hm[i\_cLoc + 1][j\_cLoc - 1] = m[i\_cLoc + 1][j\_cLoc - 1];

if (j\_cLoc - 1 >= 0) //left

hm[i\_cLoc][j\_cLoc - 1] = m[i\_cLoc][j\_cLoc - 1];

}

else //hawk eye active

{

hm[i\_cLoc][j\_cLoc] = m[i\_cLoc][j\_cLoc];

if (i\_cLoc - 1 >= 0 && j\_cLoc - 1 >= 0)

hm[i\_cLoc - 1][j\_cLoc - 1] = m[i\_cLoc - 1][j\_cLoc - 1];

if (i\_cLoc - 1 >= 0)

hm[i\_cLoc - 1][j\_cLoc] = m[i\_cLoc - 1][j\_cLoc];

if (i\_cLoc - 1 >= 0 && j\_cLoc + 1 <= \_SIZE \* 4)

hm[i\_cLoc - 1][j\_cLoc + 1] = m[i\_cLoc - 1][j\_cLoc + 1];

if (j\_cLoc + 1 <= \_SIZE \* 4)

hm[i\_cLoc][j\_cLoc + 1] = m[i\_cLoc][j\_cLoc + 1];

if (i\_cLoc + 1 <= \_SIZE \* 4 && j\_cLoc + 1 <= \_SIZE \* 4)

hm[i\_cLoc + 1][j\_cLoc + 1] = m[i\_cLoc + 1][j\_cLoc + 1];

if (i\_cLoc + 1 <= \_SIZE \* 4)

hm[i\_cLoc + 1][j\_cLoc] = m[i\_cLoc + 1][j\_cLoc];

if (i\_cLoc + 1 <= \_SIZE \* 4 && j\_cLoc - 1 >= 0)

hm[i\_cLoc + 1][j\_cLoc - 1] = m[i\_cLoc + 1][j\_cLoc - 1];

if (j\_cLoc - 1 >= 0)

hm[i\_cLoc][j\_cLoc - 1] = m[i\_cLoc][j\_cLoc - 1];

//OUTER

if (i\_cLoc - 2 >= 0 && j\_cLoc - 2 >= 0) //top left

hm[i\_cLoc - 2][j\_cLoc - 2] = m[i\_cLoc - 2][j\_cLoc - 2];

if (i\_cLoc - 2 >= 0 && j\_cLoc - 1 >= 0) //top left mid

hm[i\_cLoc - 2][j\_cLoc - 1] = m[i\_cLoc - 2][j\_cLoc - 1];

if (i\_cLoc - 2 >= 0) //top

hm[i\_cLoc - 2][j\_cLoc] = m[i\_cLoc - 2][j\_cLoc];

if (i\_cLoc - 2 >= 0 && j\_cLoc + 1 <= \_SIZE \* 4) //top right mid

hm[i\_cLoc - 2][j\_cLoc + 1] = m[i\_cLoc - 2][j\_cLoc + 1];

if (i\_cLoc - 2 >= 0 && j\_cLoc + 2 <= \_SIZE \* 4) //top right

hm[i\_cLoc - 2][j\_cLoc + 2] = m[i\_cLoc - 2][j\_cLoc + 2];

if (i\_cLoc - 1 >= 0 && j\_cLoc + 2 <= \_SIZE \* 4) //right top mid

hm[i\_cLoc - 1][j\_cLoc + 2] = m[i\_cLoc - 1][j\_cLoc + 2];

if (j\_cLoc + 2 <= \_SIZE \* 4) //right

hm[i\_cLoc][j\_cLoc + 2] = m[i\_cLoc][j\_cLoc + 2];

if (i\_cLoc + 1 <= \_SIZE \* 4 && j\_cLoc + 2 <= \_SIZE \* 4) //right bottom mid

hm[i\_cLoc + 1][j\_cLoc + 2] = m[i\_cLoc + 1][j\_cLoc + 2];

if (i\_cLoc + 2 <= \_SIZE \* 4 && j\_cLoc + 2 <= \_SIZE \* 4) //bottom right

hm[i\_cLoc + 2][j\_cLoc + 2] = m[i\_cLoc + 2][j\_cLoc + 2];

if (i\_cLoc + 2 <= \_SIZE \* 4 && j\_cLoc + 1 <= \_SIZE \* 4) //bottom right mid

hm[i\_cLoc + 2][j\_cLoc + 1] = m[i\_cLoc + 2][j\_cLoc + 1];

if (i\_cLoc + 2 <= \_SIZE \* 4) //bottom

hm[i\_cLoc + 2][j\_cLoc] = m[i\_cLoc + 2][j\_cLoc];

if (i\_cLoc + 2 <= \_SIZE \* 4 && j\_cLoc - 1 >= 0) //bottom left mid

hm[i\_cLoc + 2][j\_cLoc - 1] = m[i\_cLoc + 2][j\_cLoc - 1];

if (i\_cLoc + 2 <= \_SIZE \* 4 && j\_cLoc - 2 >= 0) //bottom left

hm[i\_cLoc + 2][j\_cLoc - 2] = m[i\_cLoc + 2][j\_cLoc - 2];

if (i\_cLoc + 1 <= \_SIZE \* 4 && j\_cLoc - 2 >= 0) //left bottom mid

hm[i\_cLoc + 1][j\_cLoc - 2] = m[i\_cLoc + 1][j\_cLoc - 2];

if (j\_cLoc - 2 >= 0) //left

hm[i\_cLoc][j\_cLoc - 2] = m[i\_cLoc][j\_cLoc - 2];

if (i\_cLoc - 1 >= 0 && j\_cLoc - 2 >= 0) //left top mid

hm[i\_cLoc - 1][j\_cLoc - 2] = m[i\_cLoc - 1][j\_cLoc - 2];

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* upgrades \*

\* \*

\* Called by: main \*

\* Displays the upgrade shop and allows the player to level up \*

\* their abilities with points. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void upgrades(int& gD, int& hE, int& tS, int& points)

{

int gD\_Cost, hE\_Cost, tS\_Cost, x = 0;

do {

gD\_Cost = (gD + 1) \* 25;

hE\_Cost = (hE + 1) \* 25;

tS\_Cost = (tS + 1) \* 25;

system("cls");

cout << "\nUPGRADE SHOP \t\t\t\t\t\t\t\t\t\t\t\tPoints: " << points << "\n\n\n";

cout << " ABILITY\t | LEVEL | COST\t | DESCRIPTION " << "\n\n";

cout << "1. Ghost Dash\t | " << gD << "\t | " << gD\_Cost << "\t | Dash 2 tiles ahead. Works through walls." << "\n";

cout << "2. Hawk Eye\t | " << hE << "\t | " << hE\_Cost << "\t | Vision increases to 2 tiles ahead." << "\n";

cout << "3. Temp Sense\t | " << tS << "\t | " << tS\_Cost << "\t | Sense the temperature based on how close the exit is." << "\n";

cout << "\n4. Back to menu\n";

cout << "\n\nChoice: ";

cin >> x;

system("CLS");

switch (x) {

case 1:

if (points >= gD\_Cost)

{

points -= gD\_Cost;

gD++;

cout << "\n\n ^^ Ghost Dash: LEVEL UP ^^ \n\n";

}

else

cout << "\n\n Insufficient points \n\n";

system("pause");

break;

case 2:

if (points >= hE\_Cost)

{

points -= hE\_Cost;

hE++;

cout << "\n\n ^^ Hawk Eye: LEVEL UP ^^ \n\n";

}

else

cout << "\n\n Insufficient points \n\n";

system("pause");

break;

case 3:

if (points >= tS\_Cost)

{

points -= tS\_Cost;

tS++;

cout << "\n\n ^^ Temp Sense: LEVEL UP ^^ \n\n";

}

else

cout << "\n\n Insufficient points \n\n";

system("pause");

break;

default:

break;

}

} while (x != 4);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* saveData \*

\* \*

\* Called by: main \*

\* Writes data into the save.dat file from the ability and \*

\* points integers. \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void saveData(int& gD, int& hE, int& tS, int& points)

{

ofstream outF;

outF.open("save.dat");

outF << gD << " " << hE << " " << tS << " " << points;

outF.close();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* exit \*

\* \*

\* Called by: main \*

\* Prompts the user if they're sure they want to exit. If yes, \*

\* an int value is returned that will cause the program to end. \*

\* If no, the user will return to main menu \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int exit()

{

int YorN;

cout << "Are you sure want to exit? Yes or No: \n\n";

cout << "1. Yes\n"

<< "2. No\n";

cout << "\n\nChoice: ";

cin >> YorN;

if (YorN == 1)

return(0);

else

return (-1);

}

TileSet.h

#pragma once

#ifndef TILESET\_H

#define TILESET\_H

#include <string>

using namespace std;

class TileSet

{

private:

string top, right, bottom, left;

bool midTop, midRight, midBottom, midLeft, doubleMid;

public:

TileSet(); //default constructor

TileSet(string,string,string,string, bool,bool,bool,bool, bool); // clockwise order from top to left

TileSet(TileSet &a); // copy constructor

string getTop();

string getRight();

string getBottom();

string getLeft();

bool getMidTop();

bool getMidRight();

bool getMidBottom();

bool getMidLeft();

bool getDoubleMid();

void setTop(string);

void setRight(string);

void setBottom(string);

void setLeft(string);

void setMidTop(bool);

void setMidRight(bool);

void setMidBottom(bool);

void setMidLeft(bool);

};

#endif

TileSet.cpp

// HEADERS //

#include "TileSet.h"

#include <string>

using namespace std;

// CLASS DEFINITIONS //

TileSet::TileSet()

{

top = "XX";

right = "XX";

bottom = "XX";

left = "XX";

midTop = false;

midRight = false;

midBottom = false;

midLeft = false;

doubleMid = false;

}

TileSet::TileSet(string t, string r, string b, string l, bool mt, bool mr, bool mb, bool ml, bool dm)

{

top = t;

right = r;

bottom = b;

left = l;

midTop = mt;

midRight = mr;

midBottom = mb;

midLeft = ml;

doubleMid = dm;

}

TileSet::TileSet(TileSet& a)

{

top = a.getTop();

right = a.getRight();

bottom = a.getBottom();

left = a.getLeft();

midTop = a.getMidTop();

midRight = a.getMidRight();

midBottom = a.getMidBottom();

midLeft = a.getMidLeft();

doubleMid = a.getDoubleMid();

}

string TileSet::getTop()

{

return top;

}

string TileSet::getRight()

{

return right;

}

string TileSet::getBottom()

{

return bottom;

}

string TileSet::getLeft()

{

return left;

}

bool TileSet::getMidTop()

{

return midTop;

}

bool TileSet::getMidRight()

{

return midRight;

}

bool TileSet::getMidBottom()

{

return midBottom;

}

bool TileSet::getMidLeft()

{

return midLeft;

}

bool TileSet::getDoubleMid()

{

return doubleMid;

}

void TileSet::setTop(string s)

{

top = s;

}

void TileSet::setRight(string s)

{

right = s;

}

void TileSet::setBottom(string s)

{

bottom = s;

}

void TileSet::setLeft(string s)

{

left = s;

}

void TileSet::setMidTop(bool b)

{

midTop = b;

}

void TileSet::setMidRight(bool b)

{

midRight = b;

}

void TileSet::setMidBottom(bool b)

{

midBottom = b;

}

void TileSet::setMidLeft(bool b)

{

midLeft = b;

}

d

Program Output

Text

Description automatically generatedText

Description automatically generatedText

Description automatically generatedA picture containing shape

Description automatically generated