**Beyond the Endless Castle**



Session: 2022 – 2026

**Submitted by:**

M.IJLAL 2022-CS-04

**Supervised by:**

MAIDA SHAHID

Department of Computer Science

**University of Engineering and Technology**

**Lahore Pakistan**

* **Description and Story**

Alphen is a teenager who has reincarnated into a world of blocks and circles. At first he was sad and terrified, for he has lost all his friends and family of old world. But Later he discovers that the people of this world were exceptional. He made friends and family he could never hope to achieve in [previous world. Thus, he decided to leave the motive of going back to his world and decided to stay here for the rest of his life.

Like all good things comes to an end, Alphen’s good luck didn’t last long. Later he discovers that he was not the only one who was incarnated into this world. There we others before him, others who wish to be the only one of their kind and they wish to destroy Alphen’s friend and family of this new world. One of his friend was kidnapped in the bizarre tension between the two parties, and despite being powerless and alone, he must set on a journey to find his friend and bring him home, for he is the one he loves.

As Alphen advances, he had many hurdles, the evil boss has a large organization, having many goons and evil characters. From some intel he found that his friend is captivated in an old large castle, now he must reach to the top of tower.

Alphen eventually succeeds in clearing the castle and defeating all the enemies. He becomes a hero, known for his bravery, determination, and never-say-die spirit. And he spends the rest of his live happy ever after.

* **Game Characters**

**Player**

There is one human player in the Game.

**Alphen:**  
Alphen is the main character in the game and is known for his high stamina. He is weak at the start of game, but as he defeats other enemies and consume their equipment, he becomes strong.

**Enemies**There are 3 types enemies in the game.

**Boss:**  
Boss if the boss of evil organization. He is rich and is resourceful. He is at the top if castle waiting for his opponent Alphen, to settle to score once and for all.

**Enemy1:**  
Boss’s right hand and most trustworthy companion. He will protect his owner’s life with all he has. To reach the boss, Alphen must pass through this fearsome warrior.

**Enemy2:**  
Small enemies of relative weaker IQ. They will shot at Alphen and try to stop his from clearing the floor.

**Bomber:**  
They will throw bombs and Alphen.

* **Game Objects**

**Equipment:**

As Alphen will advance, he will found many lost equipment which he can wear to become stronger.

**Walls:**

Walls are the barriers in the game which the Alphen and the enemies cannot cross.

**Jump Button:**

Alphen can jump on it to jump very high and reach bomber.

* **Rules & Interactions**

Alphen can jump, move and kill enemies and loot their equipment. If an enemy kills him he loses a life. If Alphen wear stronger equipment, his attack power and durability increases, but doing this will also increase the basic IQ of enemies. When Alphen kills an enemy, he scores, but his main objective is still to reach top of castle.

* **Goal of the Game**

The goal of the game is reach the top of tower and defeat the boss.

* **Wireframes**

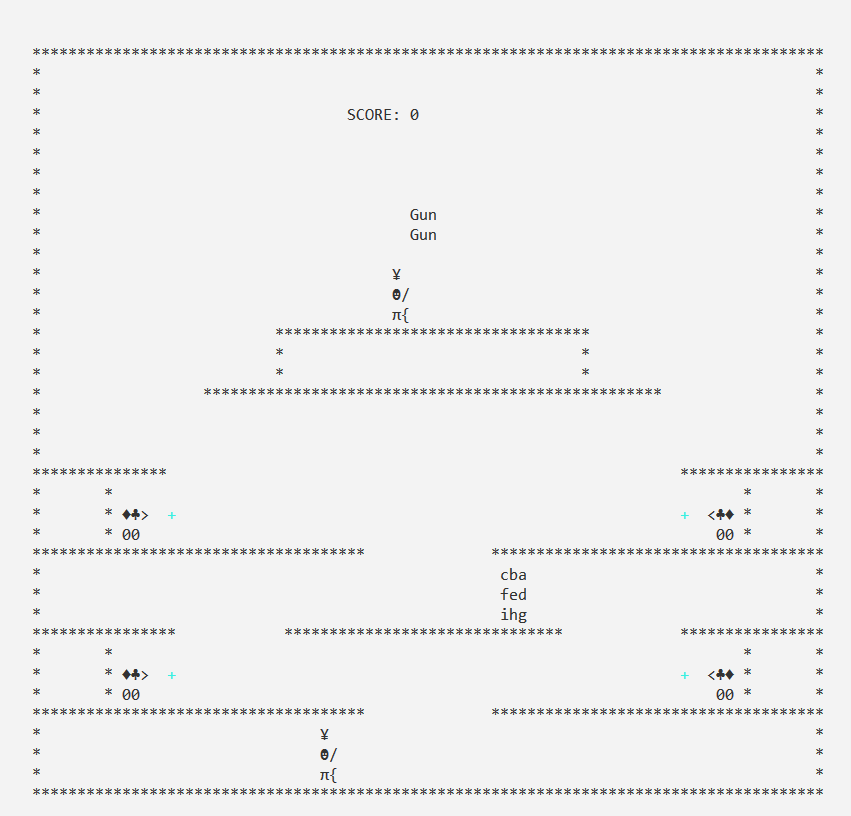
**Fig 1: Startup Menu**

****

**Fig 2: Pause Screen**

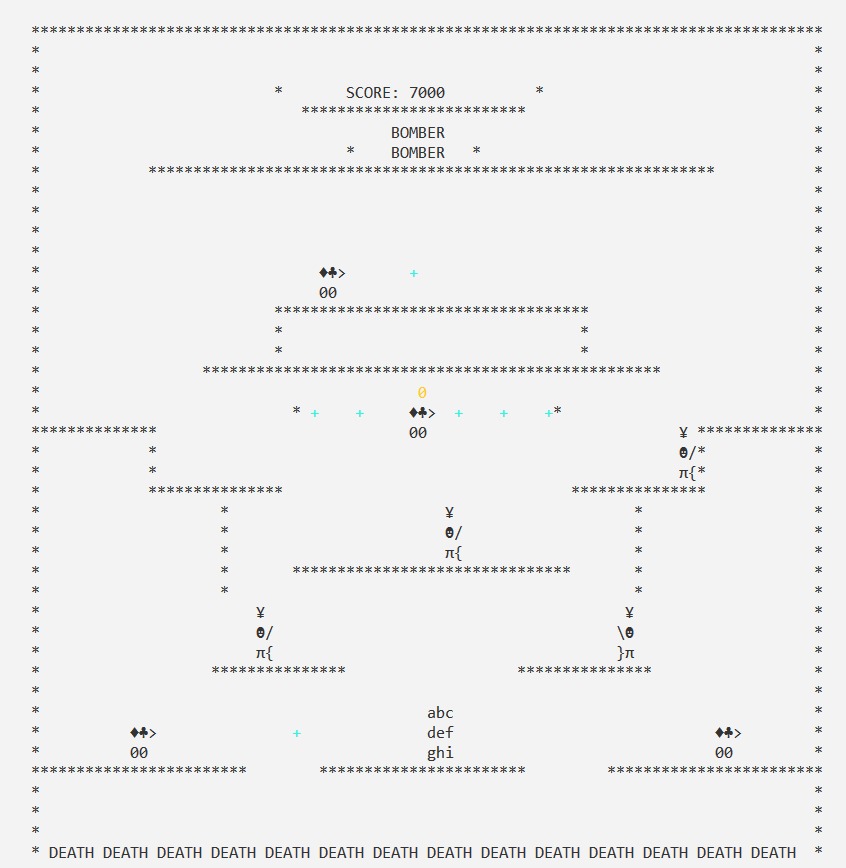
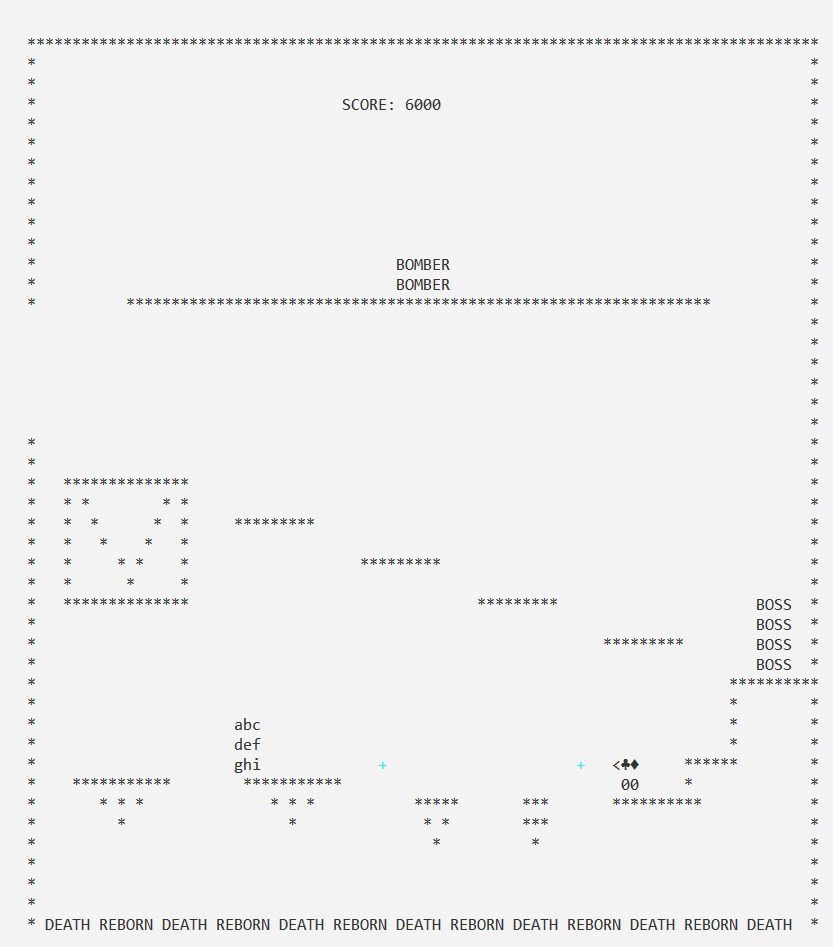
****

**Fig 3: Level 1**

****

**Fig 4: Level 2**

**Fig 5: Level 3**



* **Data Structures (2D Arrays)**

char playerArray[3][3];

char bossArray[4][4] ;

char maze12dArray[38][103];

char maze22dArray[45][103];

char maze32dArray[42][103];

char enemy1Array[2][3];

string enemy2Array[3][2];

* **Function Prototypes**

                            // Boss Moments

void lev1BossMove();

void lev2BossMove();

char getCharAtxy(short int x, short int y);     // gives character at a given x y

void gotoxy(int x, int y);                      // moves cursor at a given x y

void ShowConsoleCursor(bool showFlag);          // hides cursor, make game beautiful

void printMaze2();                               // print level 2 of maze

void printMaze3();                               // print level 3 of maze

void printPlayer();                             // print player at playerX and player y

void erasePlayer();                             // reomves player from playerX and playerY

void printBoss();

void eraseBoss();

void playerDeath(int, int);

int movePlayer(string);                         // moves left, right jump

void Playerjump();

void playerGravity();

bool detectPlayerCollision(string);             /// 1 means collision upahead

bool isNextCharWall(int, int);                  /// 1 means there is a wall upnext

int collisionsEnemy1(int, string);

int collisionsEnemy2(int, string);

bool enemyCollisionPlayer();

bool enemyBulletCollisionPlayer(int, int);

bool playerBulletCollisionEnemy(int, int);

void printEnemy1();

void eraseEnemy1();

void moveEnemy1();

int isOnGun();

void printGun();

void eraseGun();

bool isPlayerEligibleForFire(string);                // check validations before firing

void movePlayerBullet();                             // move bullet, no collisions yet

void printPlayerBullet();                            // self explanatory

void erasePlayerBullet();                            // self explanatory

void removePlayerBullet(int);                        // unregisters from array

void enemy1Fire(int);

void moveEnemyBullet();

void printEnemyBullet();

void eraseEnemyBullet();

void removeEnemyBullet(int);

void moveEnemy2();

void printEnemy3();

void eraseEnemy3();

void enemy3Fire(string);

void moveEnemy3Bullet();

bool bullet3PlayerCollision(int, int);

bool isEnemy3Alive = true;

bool isEnemy3Fired = false;

void removeEnemy3Bullet();

void printScore();

void increaseScore();

void decreaseScore();

void startLevel1();

void startLevel3();

void level3Prep();

void startLevel2();

void level2Prep();

void level1Intro();

void level2Intro();

void level3Intro();

void printJumpButton();

void jumpButtonJump();

void storePlayerAndBoss();

void loadPlayerAndBoss(string option);

void storeEnemy1();

void loadEnemy1(string option);

void storeEnemy2And3();

void loadEnemy2And3(string option);

void loadAllData(string option);

void storeAllData();      // newGame    load

string getFieldFromRecord(string record, int option);

void StartMenuWithOptions();

void printStartMenuWithOptions();

void printStartUpMenuArrow(int option);

void PauseMenuWithOptions();

void printPauseMenuWithOptions();

void printPauseMenuArrow(int option);

void StartNewGame();

void LoadGame();

void printGameEndScreen();

* **Complete Code**

#include<iostream>

#include<conio.h>

#include<windows.h>

#include<fstream>

using namespace std;

// Player Attributes

int respawnX = 25; // player will respawn at this position

int respawnY = 25; // player will respawn at this position

char wall = '\*'; // what wall will liik like

char playerArray[3][3] = {{'a', 'b', 'c'},{ 'd', 'e', 'f'},{ 'g', 'h', 'i'}}; // totally customizeableplayer array

int playerX = 57; // this will have player corrent position

int playerY = 48; // this will have player corrent position

int playerXsize = 3; // used for arrays, changeable

int playerYsize = 3; // used for arrays, changeable

string playerFace = "right"; // stores what player is facing direction, constantly chaing as game is played

int playerSpeed = 3; // how fast player will move, changes as level changes

// Boss Attributes

char bossArray[4][4] = {{'b', 'b', 'b', 'b'}, {'b', 'b', 'b', 'b'}, {'b', 'b', 'b', 'b'}, {'b', 'b', 'b', 'b'}}; // totally customizeable player array

int bossXSize = 4; // used for arrays, changeable

int bossYSize = 4; // used for arrays, changeable

int bossX = 50; // this will have boss corrent position

int bossY = 50; // this will have boss corrent position

int lvl2BossCondition = 0; // this is used to keep track of how much lvl 2 oss animation has ben done

int lvl2BossIteration = 0; // this is used to keep track of how much lvl 2 oss animation has ben done

bool showlev2Boss = true; // this is used to keep track of how much lvl 2 oss animation has ben done

bool showlev1Boss = false; // this is used to keep track of how much lvl 2 oss animation has ben done

bool islvl1BossShown = false; // this is used to keep track of how much lvl 2 oss animation has ben done

// Boss Moments

void lev1BossMove(); // this holds functionality of how and when boss anmation shoud play

void lev2BossMove(); // this holds functionality of how and when boss anmation shoud play

char getCharAtxy(short int x, short int y); // gives character at a given x y

void gotoxy(int x, int y); // moves cursor at a given x y

void ShowConsoleCursor(bool showFlag); // hides cursor, make game beautiful

void printMaze1(); // print level 1 of maze

void printMaze2(); // print level 2 of maze

void printMaze3(); // print level 3 of maze

// Print Erase Characters

void printPlayer(); // print player at playerX and player y

void erasePlayer(); // reomves player from playerX and playerY

void printBoss(); // print boss

void eraseBoss(); // plsaces empty spaces i.e erase boss

bool donotPrintBoss = false; // to hammer bug, when sometimes we pause and resume and boss still printed

// Player Functions

void playerDeath(int, int); // respawn player and change score

int movePlayer(string); // moves left, right jump

// Jumping and gravity

// jump attributes

int jumpsize = 5; // player will jump n - 1 bloacks

int tempJump = 0; // just a temp

bool currentJump = false;

bool jumpThisIteration = false;

// jump functions

void Playerjump(); // if player is not jumping and up is presed, it will jump

// Gravity Player

void playerGravity(); // always decreasing 1 from playerY unluess a wall beneath

// COLLISIONS

// player collisions with walls

bool detectPlayerCollision(string); /// 1 means collision upahead

bool isNextCharWall(int, int); /// 1 means there is a wall upnext

// Enemy collisions with walls and ground

int collisionsEnemy1(int, string); // returns 1 if path to front if enemy is blocked, i.e wall in front, or no ground above

int collisionsEnemy2(int, string);

// Enemy collisions with player

bool enemyCollisionPlayer(); // returns 1 if any emeny collides with player

// Bullet collisions with player and enemy

bool enemyBulletCollisionPlayer(int, int); // returns 1 if any emeny bullet collides with player

bool playerBulletCollisionEnemy(int, int); // returns 1 if any player bullet collides with enemy, constantly runnimng

// enemy 1 facuilty //

void printEnemy1();

void eraseEnemy1();

// char enemy1Array[6] = {'E', 'E', 'E', 'E', 'E', 'E'};

// string enemy1Array[2][3] = {{ "4", "5", ">"},{ "0", "0", " "}};

char enemy1Array[2][3] = {{ 4, 5, '>'},{ '0', '0', ' '}};

int nrEmeny1 = 4; // total nr of enemies

bool isEnemy1alive[4] = {1,1,1,1}; // self explanatory

int enemy1Xsize = 3;

int enemy1Ysize = 2;

int enemy1X[4] = {49, 66, 49, 66}; // x axis of different enemies

int enemy1Y[4] = {38, 38, 46, 46}; // y axis of different emeies

string enemy1Face[4] = {"left", "right", "left", "right"}; // in which direction they are facing right now

void moveEnemy1(); // behaviour level --> simple, just movve left and right (fire functionality to be added)

// ==================================== //

// Player bullet section

// player firing ability //

// player Shooting

bool isGun = false; // so that player cannot shoot if he has not gun yet

int isOnGun(); // constantly checking if player has colliedde with gun

void printGun(); // prints gun

void eraseGun(); // print spaces where gun was previously

string bulletShape = "#"; // shape of bullet which will be launched

void playerFire(); // just registr shot place and face, donot move

bool isPlayerEligibleForFire(string); // check validations before firing

void movePlayerBullet(); // move bullet, no collisions yet

void printPlayerBullet(); // self explanatory

void erasePlayerBullet(); // self explanatory

void removePlayerBullet(int); // unregisters from array

int playerMaxBulletAtATime = 2; // how many bullets can exist simultaneously

int playerBulletLength = 15; // how far bullet should go, unless stopped

int lengthAfterPlayer2ndBulletAllow = playerBulletLength / playerMaxBulletAtATime; // just some calaulations

bool isPlayerBulletPresent[5] = {0,0}; // to reuse same array

string playerBulletDirection[5] = {"left, left"}; // directions based on face

int playerBulletX[5] = {0,0};

int playerBulletY[5] = {0,0};

int playerBulletDistanceCovered[5] = {0,0};

// note, bullet will be fired from 1 below top of player body // bcz i set it to +1, see playerfire for more//

// ==================================== //

// enemy1 firing ability //

string enemyBulletShape = "+"; // shape of bullet which will be launched

void enemy1Fire(int); // this wil control how enemy1 will fire

void moveEnemyBullet(); // this wil control how enemy1 bullet will move

void printEnemyBullet(); // print bullet of enemy1

void eraseEnemyBullet(); // erase bullet of enemy1

void removeEnemyBullet(int); // remove bullet data from array , beautiful memory optomization, that is

int enemy1BulletDistance = 13; // how far enemy bullet will go

int isEnemyBulletPresent[20]; // keeps track of live and dead bullets, 1 for alive

int emenyBulletX[20]; // keeps bullet information

int emenyBulletY[20]; // keeps bullet information

int enemyBulletXMovement[20]; // keeps bullet information

int enemyBulletYMovement[20]; // keeps bullet information

int enemyBulletLength[20]; // keeps bullet information

// enemy 2 facuilty //

void printEnemy2();

void eraseEnemy2();

string enemy2Array[3][2] = {{"157", "255"},{ "2", "92"},{ "227", "125"}}; // enemy2 shape, completely vchangeable by 1 click

int nrEmeny2 = 4; // total nr of enemies

bool isEnemy2alive[4] = {1,1,0,0}; // self explanatory

int enemy2Xsize = 2;

int enemy2Ysize = 3;

int enemy2X[4] = {50, 57, 0, 0}; // x axis of different enemies

int enemy2Y[4] = {49, 26, 0, 0}; // y axis of different emeies

string enemy2Face[4] = {"left", "right", "left", "right"}; // in which direction they are facing right now

void moveEnemy2(); // behaviour level --> simple, just movve left and right

//===============================//

// enemy 3 facuilty //

void printEnemy3();

void eraseEnemy3();

char enemy3Array[4] = {'E', 'E', 'E', 'E'};

int nrEmeny3 = 1; // total nr of enemies

bool isEnemy3alive = true; // self explanatory

int enemy3Xsize = 2;

int enemy3Ysize = 2;

int enemy3X = 50 ; // x axis of different enemies

int enemy3Y = 25 ; // y axis of different emeies

int enemy3BulletLength = 50;

// firing ability //

string enemy3BulletShape = "0"; // completely changeable falling bullet from enemy3

void enemy3Fire(string); // stores previous character for bulltet moving

void moveEnemy3Bullet(); // falls enemy3 bombs as like gravity

bool bullet3PlayerCollision(int, int); // checks if bomb fall on player

bool isEnemy3Alive = true; // checks so tell if to shot bulet

bool isEnemy3Fired = false; // donot fire if already fired

void removeEnemy3Bullet(); // erase when bullet fall on ground

int emeny3BulletX; // keeps track of bulllet location

int emeny3BulletY = enemy3Y+2; // keeps track of bulllet location

char enemy3BulletPreviousChar; // keeps track of bulllet location

int waittime = 20; // yhow long to wait before ersseing previous and printing new vbulllet

int tempWaittime = 0; // just to remember what waitttime is

int enemy3Speed = 5; // completely changeable speed by 1 click

// Scoring section //

void printScore(); // print score on top

void increaseScore();

void decreaseScore();

int currentScore = 0;

int scoreDecrement = 800; // on player death, score to be deducted

int scoreIncrement = 1000; // on enemy death, score to be added

// ==================================================== //

int currentLevel = 1; // // keeps track of what level player is on

void startLevel1(); // this will start by first setting all enemies in position and spawning player, and printing ,maze etc

void startLevel3(); // this will start by first setting all enemies in position and spawning player, and printing ,maze etc

void level3Prep(); // this set all enemies in position and spawning player, and printing ,maze etc

void startLevel2();

void level2Prep(); // this set all enemies in position and spawning player, and printing ,maze etc

void level1Intro(); // just printing text before starting level

void level2Intro(); // just printing text before starting level

void level3Intro(); // just printing text before starting level

// ==================================================== //

// creating jump button

bool isJumpButton = 0; // tells if to show jump button or no, 1 for ues, used in lvl 3

int lvl3jumpsize = 11; // changes how up player will jump on lvl 3 button

int templvl3jump = 0; // just a temp to store how far player hasn jump

void printJumpButton();

void jumpButtonJump();

// ========================================================= //

// file Handling

void storePlayerAndBoss(); // stores data in csv

void loadPlayerAndBoss(string option); // load data in csv

void storeEnemy1(); // sotres data in csv

void loadEnemy1(string option); // load data from csv

void storeEnemy2And3(); // sotres data in csv

void loadEnemy2And3(string option); // load data from csv for either a new game or previous game

void loadAllData(string option); // load data from csv

void storeAllData(); // newGame load

string getFieldFromRecord(string record, int option); // simple function used when loading files from csv

// ====================================================//

// StartUp And Pause

void StartMenuWithOptions(); // selecting startup menu options functionaliity

void printStartMenuWithOptions(); // printing it

void printStartUpMenuArrow(int option); // changiung arraows up and down

void PauseMenuWithOptions(); // stores what options are available in pause menu

void printPauseMenuWithOptions();

void printPauseMenuArrow(int option); // print and erase arraow

int startUpArrowOptionSelected = 1; // just temps

int PauseArrowOptionSelected = 1; // just temps

void StartNewGame(); // loadall new data files from csv

void LoadGame(); // load alll old data from csv

string currentGameMode = "newGame"; // can be load or new game

void printGameEndScreen(); // justi printing ytext you have won bla bla bla...

// for colors

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

int main()

{

getch();

ShowConsoleCursor(false); // hiding ccursor

system("pause");

StartMenuWithOptions(); // goto start up screen with options to load or start a new game

}

void LoadGame()

{

loadAllData("load"); // load from csv to arrays

if (currentLevel == 1) // if previously player was on level 1 when dsaving

{

startLevel1();

level2Intro();

Sleep(6000);

startLevel2();

level3Intro();

Sleep(13000);

system("cls");

startLevel3();

}

else if (currentLevel == 2) // if previously player was on level 2 when dsaving

{

startLevel2();

level3Intro();

Sleep(13000);

system("cls");

startLevel3();

}

else if (currentLevel == 3) // if previously player was on level 3 when dsaving

{

startLevel3();

}

}

void StartNewGame()

{

loadAllData("newGame"); // load from csv to arrays

getch();

level1Intro();

startLevel1();

level2Intro();

Sleep(6000);

startLevel2();

level3Intro();

Sleep(13000);

system("cls");

startLevel3();

}

void level1Intro()

{

system("cls");

gotoxy(50,20);

cout << "LEVEL 1 THE TUTORIAL"; // showing text, nothing fancy

Sleep(3000); // wait for 3 sec as getch not works properly with hide cursor

getch();

}

void level2Intro()

{

system("cls");

gotoxy(50,20);

cout << "LEVEL 2 THE BOSS CHASE"; // showing text, nothing fancy

gotoxy(45,22);

cout << "The Boss is Running, Must catch him !!!";// showing text, nothing fancy in mid of scirren

getch();

}

void level3Intro()

{

system("cls");

gotoxy(50,20);

cout << "LEVEL 3 THE RECKONING"; // showing all text and waiting for some time to read

gotoxy(45,22);

cout << "The Boss has led you to a ambush. With nothin";

gotoxy(45,23);

cout << " to lose, our HERO faces the opponent. ";

gotoxy(30,25);

cout << "YOU HAVE FOUND GOOD EQUIPMENT ! YOU HAVE GROWN STRONGER, BUT SO DOES YOUR OPPONENT";

gotoxy(45,27);

cout << " YOU HAVE BEEN WARNED !!! ";

getch();

}

void playerDeath(int x , int y )

{

Sleep(1000);

erasePlayer(); // erase player from death spot, bccz needs to do this, bcz we are chaning coordinates, so it would be improssible to erase after altering them

playerX = x; // changing to default position for respawning

playerY = y;

decreaseScore(); // decrease svcore by increment variable

}

void lev2BossMove() // control interaction of boss with in lvl2

{

eraseBoss();

if (isEnemy1alive[0] == 0 && lvl2BossCondition == 0) // if enemy1 is bdead and 1st animationn is noit already been done then play this

{

lvl2BossCondition = 1;

}

if ( lvl2BossIteration < 5 && isEnemy1alive[0] == 0) // first 3 jumps and vertical moment of lvl 2 boss anim

{

if ( lvl2BossCondition > 0 && lvl2BossCondition < 7) // checks how far animation has reached

{

bossY -= 1;

bossX -= 1;

lvl2BossCondition += 1;

}

else if( lvl2BossCondition >= 7 && lvl2BossCondition < 8)

{

bossX -= 1;

lvl2BossCondition += 1;

}

else if( lvl2BossCondition >= 8 && lvl2BossCondition < 12)

{

bossY += 1;

bossX -= 1;

lvl2BossCondition += 1;

}

else if ( lvl2BossCondition >=11 && lvl2BossCondition <= 15)

{

bossX -= 1;

lvl2BossCondition += 1;

}

else if (lvl2BossIteration < 4)

{

lvl2BossIteration += 1;

lvl2BossCondition = 1;

}

}

if ( isEnemy1alive[0] == 0 && playerX < 50 && lvl2BossIteration == 4 && playerY < 40) // jump from last platform to new level

{

lvl2BossIteration = 6;

lvl2BossCondition = 1;

}

if ( lvl2BossIteration == 6)

{

if ( lvl2BossCondition > 0 && lvl2BossCondition < 7) // checks how far animation has reached

{

bossY -= 1;

bossX -= 1;

lvl2BossCondition += 1;

}

else if( lvl2BossCondition >= 7 && lvl2BossCondition < 8)

{

bossX -= 1;

lvl2BossCondition += 1;

}

else if( lvl2BossCondition >= 8 && lvl2BossCondition < 12) // checks how far animation has reached

{

bossY += 1;

bossX -= 1;

lvl2BossCondition += 1;

}

else

{

lvl2BossIteration += 1; // on every iteration, add in it, so we know what anim to polay next

showlev2Boss = false;

eraseBoss();

bossX = 10; // hammering bug when boss seems to look as fly up

bossY = 10;

}

}

}

void level2Prep()

{

currentLevel = 2;

respawnX = 27;

respawnY = 45;

playerX = 27;

playerY = 45;

playerSpeed = 3;

bossX = 96;

bossY = 43;

jumpsize = 6; // 5 to 8 is bedst for gameplay

nrEmeny1 = 4; // total nr of enemies

isEnemy1alive[0] = 1;

isEnemy1alive[1] = 0; // setting positions for enemy1

isEnemy1alive[2] = 0;

isEnemy1alive[3] = 0;

enemy1X[0] = 80; // setting positions for enemy1

enemy1X[1] = 0;

enemy1X[2] = 0;

enemy1X[3] = 0;

enemy1Y[0] = 51;

enemy1Y[1] = 0; // setting positions for enemy1

enemy1Y[2] = 0;

enemy1Y[3] = 0;

playerMaxBulletAtATime = 3; // how many bullets can exist simultaneously

playerBulletLength = 25; // how far bullet should go, unless stopped

lengthAfterPlayer2ndBulletAllow = playerBulletLength / playerMaxBulletAtATime; // just some calaulations

isPlayerBulletPresent[0] = 0;

isPlayerBulletPresent[1] = 0; // deleting old bullets for enmnemy 1

isPlayerBulletPresent[2] = 0;

playerBulletDistanceCovered[0] = 0;

playerBulletDistanceCovered[1] = 0; // temp, for tracking how far bullet has moved

playerBulletDistanceCovered[2] = 0;

enemy1BulletDistance = 65; // this is max fdistance bullet will move

// enemy 2 facuilty //

// char enemy2Array[6] = {'E', 'E', 'E', 'E', 'E', 'E'};

nrEmeny2 = 4; // total nr of enemies

// isEnemy2alive[4] = {1,1,1,0};

isEnemy2alive[0] = 0;

isEnemy2alive[1] = 0;

isEnemy2alive[2] = 0; // setting positions for enemy2

isEnemy2alive[3] = 0;

enemy2Xsize = 2;

enemy2Ysize = 3;

enemy2X[0] = 0;

enemy2X[1] = 0;

enemy2X[2] = 0; // setting positions for enemy1

enemy2X[3] = 0;

enemy2Y[0] = 0;

enemy2Y[1] = 0; // setting positions for enemy1

enemy2Y[2] = 0;

enemy2Y[3] = 0;

// enemy2Face[4] = {"left", "right", "left", "right"}; // in which direction they are facing right now

// char enemy3Array[4] = {'E', 'E', 'E', 'E'};

nrEmeny3 = 1; // total nr of enemies

isEnemy3alive = false; // self explanatory

enemy3Xsize = 2;

enemy3Ysize = 2;

enemy3X = 55 ; // x axis of different enemies

enemy3Y = 26 ; // y axis of different emeies

// firing ability //

enemy3BulletShape = "0";

isEnemy3Alive = false; // fire only if alive

isEnemy3Fired = false; // to stop from contantly firing

// emeny3BulletX;

emeny3BulletY = enemy3Y+2; // spawn bullet from 2 below it

waittime = 10;

tempWaittime = 0; // bllet 3 properties explained above

enemy3Speed = 3;

}

void startLevel2()

{

level2Prep();

if (currentGameMode == "load")

{

loadAllData("load");

currentGameMode = "newGame";

}

printMaze2();

printPlayer();

int i = 0;

while(true) // game starts here

{

printScore();

if (i%3 == 0)

{

if (GetAsyncKeyState(VK\_ESCAPE)) // goto pause menu

{

PauseMenuWithOptions();

}

if (GetAsyncKeyState(VK\_LEFT)) // move player left, sand etc

{

movePlayer("left");

}

if (GetAsyncKeyState(VK\_RIGHT))

{

movePlayer("right");

}

if (GetAsyncKeyState(VK\_UP))

{

movePlayer("up");

}

if (GetAsyncKeyState(VK\_SPACE) && isGun == true) // fire by pessing space only if gun is in hand

{

if (getCharAtxy(playerX - 1, playerY + 1) != '\*' && getCharAtxy(playerX + playerXsize, playerY + 1) != '\*') // helps avaoiding player firing bullet multiuples

playerFire();

}

erasePlayer();

Playerjump();

playerGravity(); // constantly applied, beautifully written funvtipon, do as it says,

printPlayer();

if (showlev2Boss == true)

{

lev2BossMove();

if (bossX != 10 && bossY != 10)

printBoss();

}

else

{

eraseBoss();

}

}

if (i%2 == 0) // player bulltet moves at this speed of 2

{

erasePlayerBullet();

movePlayerBullet();

printPlayerBullet();

}

if (i%4 == 0) // enemy bulltet moves at this speed of 2

{

eraseEnemyBullet();

moveEnemyBullet();

printEnemyBullet();

}

if (i%100 == 0 && isEnemy1alive[0] == 1) // just some functionality so enemy always fires on left

{

enemy1Face[0] = "left";

eraseEnemy1();

enemy1Fire(0);

// moveEnemy1();

// enemy1X[0] += 1;

printEnemy1();

}

if (isEnemy1alive[0] == 0) // when enemy dies, then bring bomber sand starrt fdroping bombs

isEnemy3Alive = true;

if (i% enemy3Speed == 0)

{

if (isEnemy3Alive == true) //fire only if alive, ofvcourse

{

if (isEnemy3Fired == false)

{

enemy3Fire("single");

}

else if (tempWaittime == waittime - 2)

{

if (playerX < 30)

{

emeny3BulletX = 31;

}

else if (playerX > 91)

{

emeny3BulletX = 90;

}

else

emeny3BulletX = playerX;

tempWaittime += 1;

}

else if (tempWaittime == waittime)

{

moveEnemy3Bullet();

}

else{

tempWaittime += 1;

}

}

}

if (playerX < 20 && playerY > 34) // stop level if player goes to new lvl

{

break;

}

if ( enemyCollisionPlayer() == 1) // checks for basic collisions

{

playerDeath(respawnX, respawnY);

}

if ( playerY > 55)

{

playerDeath(respawnX, respawnY);

}

Sleep(20);

i+=1;

if (i == 100001)

{

i = 1;

}

}

getch();

system("cls");

}

void level3Prep() // explained in lvl 2 and in above main when declaring

{

currentLevel = 3;

int respawnX = 25;

int respawnY = 25;

playerX = 57;

playerY = 48;

playerSpeed = 3;

jumpsize = 6; // 5 to 8

// enemy1Array[6] = {'E', 'E', 'E', 'E', 'E', 'E'};

nrEmeny1 = 4; // total nr of enemies

isEnemy1alive[0] = 1;

isEnemy1alive[1] = 1;

isEnemy1alive[2] = 1;

isEnemy1alive[3] = 1;

enemy1X[0] = 19;

enemy1X[1] = 80;

enemy1X[2] = 50;

enemy1X[3] = 57;

enemy1Y[0] = 50;

enemy1Y[1] = 50;

enemy1Y[2] = 27;

enemy1Y[3] = 34;

// enemy1Face[4] = {"left", "right", "left", "right"}; // in which direction they are facing right now

playerMaxBulletAtATime = 3; // how many bullets can exist simultaneously

playerBulletLength = 25; // how far bullet should go, unless stopped

lengthAfterPlayer2ndBulletAllow = playerBulletLength / playerMaxBulletAtATime; // just some calaulations

isPlayerBulletPresent[0] = 0;

isPlayerBulletPresent[1] = 0;

isPlayerBulletPresent[2] = 0;

// playerBulletDirection[5] = {"left, left"};

// playerBulletX[5] = {0,0};

// playerBulletY[5] = {0,0};

playerBulletDistanceCovered[0] = 0;

playerBulletDistanceCovered[1] = 0;

playerBulletDistanceCovered[2] = 0;

enemy1BulletDistance = 35;

// enemy 2 facuilty //

// char enemy2Array[6] = {'E', 'E', 'E', 'E', 'E', 'E'};

nrEmeny2 = 4; // total nr of enemies

// isEnemy2alive[4] = {1,1,1,0}; VK\_ESCAPE

isEnemy2alive[0] = 1;

isEnemy2alive[1] = 1;

isEnemy2alive[2] = 1;

isEnemy2alive[3] = 1;

enemy2Xsize = 2;

enemy2Ysize = 3;

enemy2X[0] = 35;

enemy2X[1] = 80;

enemy2X[2] = 59;

enemy2X[3] = 80;

enemy2Y[0] = 44;

enemy2Y[1] = 44;

enemy2Y[2] = 39;

enemy2Y[3] = 35;

// enemy2Face[4] = {"left", "right", "left", "right"}; // in which direction they are facing right now

// char enemy3Array[4] = {'E', 'E', 'E', 'E'};

nrEmeny3 = 1; // total nr of enemies

isEnemy3alive = true; // self explanatory

enemy3Xsize = 2;

enemy3Ysize = 2;

enemy3X = 55 ; // x axis of different enemies

enemy3Y = 20 ; // y axis of different emeies

enemy3BulletLength = 50;

// firing ability //

enemy3BulletShape = "0";

isEnemy3Alive = true;

isEnemy3Fired = false;

// emeny3BulletX;

emeny3BulletY = enemy3Y+2;

waittime = 10;

tempWaittime = 0;

enemy3Speed = 3;

}

void startLevel3() // explained in lvl 2 and in above main when declaring

{

level3Prep();

if (currentGameMode == "load")

{

loadAllData("load");

currentGameMode = "newGame";

}

printMaze3();

printPlayer();

printEnemy3();

int i = 0;

while(true)

{

if (i%3 == 0)

{

if (GetAsyncKeyState(VK\_ESCAPE))

{

PauseMenuWithOptions();

}

if (GetAsyncKeyState(VK\_LEFT))

{

movePlayer("left");

}

if (GetAsyncKeyState(VK\_RIGHT))

{

movePlayer("right");

}

if (GetAsyncKeyState(VK\_UP))

{

movePlayer("up");

}

if (GetAsyncKeyState(VK\_SPACE) && isGun == true)

{

if (getCharAtxy(playerX - 1, playerY + 1) != '\*' && getCharAtxy(playerX + playerXsize, playerY + 1) != '\*')

playerFire();

}

erasePlayer();

Playerjump();

playerGravity();

printPlayer();

if (playerY > 52)

{

playerDeath(respawnX, respawnY);

}

}

if (i%2 == 0)

{

erasePlayerBullet();

movePlayerBullet();

printPlayerBullet();

eraseEnemyBullet();

moveEnemyBullet();

printEnemyBullet();

eraseEnemy2();

moveEnemy2();

printEnemy2();

}

if (i%5 == 0)

{

eraseEnemy1();

moveEnemy1();

printEnemy1();

if (isJumpButton == 0 && isEnemy1alive[0] == 0 && isEnemy1alive[1] == 0 && isEnemy1alive[2] == 0 && isEnemy1alive[3] == 0 && isEnemy2alive[0] == 0 && isEnemy2alive[1] == 0 && isEnemy2alive[2] == 0 && isEnemy2alive[3] == 0)

{

isJumpButton = 1;

printJumpButton();

}

}

if (i% enemy3Speed == 0)

{

if (isEnemy3Alive == true)

{

if (isEnemy3Fired == false)

{

enemy3Fire("single");

}

else if (tempWaittime == waittime - 2)

{

if (playerX < 27)

{

emeny3BulletX = 28;

}

else if (playerX > 91)

{

emeny3BulletX = 90;

}

else

emeny3BulletX = playerX;

tempWaittime += 1;

}

else if (tempWaittime == waittime)

{

moveEnemy3Bullet();

}

else{

tempWaittime += 1;

}

}

}

if ( enemyCollisionPlayer() == 1)

{

playerDeath(respawnX,respawnY);

}

if (isEnemy3Alive == 0 && isEnemy1alive[0] == 0 && isEnemy1alive[1] == 0 && isEnemy1alive[2] == 0 && isEnemy1alive[3] == 0 && isEnemy2alive[0] == 0 && isEnemy2alive[1] == 0 && isEnemy2alive[2] == 0 && isEnemy2alive[3] == 0)

{

// system("cls");

// Sleep(3000);

printGameEndScreen();

StartMenuWithOptions();

}

printScore();

Sleep(20);

i+=1;

if (i == 100001)

{

i = 1;

}

}

getch();

}

void lev1BossMove() // explained in lvl 2 and in above main when declaring

{

bossX = 105;

bossY = 32;

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

{

bossX -= 2;

printBoss();

Sleep(500);

eraseBoss();

}

printBoss();

Sleep(1000);

eraseBoss();

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

{

bossX += 2;

printBoss();

gotoxy(bossX, bossY+4);

cout << "\*\*\*\*\*";

Sleep(500);

eraseBoss();

}

// gotoxy(103, 36);

// cout << "\*\*\*\*";

showlev1Boss = false;

islvl1BossShown = true;

}

void startLevel1() // explained in lvl 2 and in above main when declaring

{

printMaze1();

printPlayer();

if (isGun == false)

{

printGun();

}

if (currentGameMode == "load")

{

loadAllData("load");

currentGameMode = "newGame";

}

else

{

loadAllData("newGame");

}

int i = 0;

while(true)

{

printScore();

if (isGun == false)

{

isOnGun();

}

if (i%playerSpeed == 0)

{

if (GetAsyncKeyState(VK\_ESCAPE))

{

PauseMenuWithOptions();

}

if (GetAsyncKeyState(VK\_LEFT))

{

movePlayer("left");

}

if (GetAsyncKeyState(VK\_RIGHT))

{

movePlayer("right");

}

if (GetAsyncKeyState(VK\_UP))

{

movePlayer("up");

}

if (GetAsyncKeyState(VK\_SPACE) && isGun == true)

{

if (getCharAtxy(playerX - 1, playerY + 1) != '\*' && getCharAtxy(playerX + playerXsize, playerY + 1) != '\*')

playerFire();

}

erasePlayer();

Playerjump();

playerGravity();

printPlayer();

// lev1BossMove(); // just for testing

if (showlev1Boss == true)

{

lev1BossMove();

}

else if (isEnemy1alive[0] == 0 && isEnemy1alive[1] == 0 && isEnemy1alive[2] == 0 && isEnemy1alive[3] == 0 && isEnemy2alive[0] == 0 && isEnemy2alive[1] == 0 && isEnemy2alive[2] == 0 && isEnemy2alive[3] == 0)

{

if (islvl1BossShown == false)

showlev1Boss = true;

}

}

if (i%2 == 0)

{

erasePlayerBullet();

movePlayerBullet();

printPlayerBullet();

eraseEnemyBullet();

moveEnemyBullet();

printEnemyBullet();

eraseEnemy2();

moveEnemy2();

printEnemy2();

}

if (i%5 == 0)

{

eraseEnemy1();

moveEnemy1();

printEnemy1();

}

if ( enemyCollisionPlayer() == 1)

{

playerDeath(respawnX, respawnY);

}

if (playerY > 32 && playerX > 103)

{

break;

}

Sleep(20);

i+=1;

if (i == 100001)

{

i = 1;

}

}

getch();

system("cls");

}

// ========================= //

// enemy 3 all functionality //

void enemy3Fire(string method = "single") // applied when firing a new bullet

{

enemy3BulletPreviousChar = '\*'; // for a new bullet, previous always \* bcz it comes from a wall whaich is \*

isEnemy3Fired = true;

}

void moveEnemy3Bullet()

{

gotoxy(emeny3BulletX, emeny3BulletY);

cout << enemy3BulletPreviousChar; // printing what was previously here

emeny3BulletY += 1; // moving 1 point downlword

if (emeny3BulletY > enemy3BulletLength) // if reached ground, then destroyyyyyyy

{

removeEnemy3Bullet();

}

else if ( bullet3PlayerCollision(emeny3BulletX, emeny3BulletY) == 1) // if collied with player

{

removeEnemy3Bullet();

playerDeath(respawnX,respawnY);

}

else // prints bulet by chanigng color

{

enemy3BulletPreviousChar = getCharAtxy(emeny3BulletX, emeny3BulletY);

gotoxy(emeny3BulletX, emeny3BulletY);

SetConsoleTextAttribute(hConsole, 1);

cout << enemy3BulletShape;

SetConsoleTextAttribute(hConsole, 7);

}

}

bool bullet3PlayerCollision(int x, int y) // check if player axis matches with enemy, means player dead

{

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

{

if (playerY + j == y)

{

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

{

if (playerX + i == x)

return 1;

}

}

}

return 0;

}

void removeEnemy3Bullet()

{

isEnemy3Fired = false;

tempWaittime = 0;

emeny3BulletY = enemy3Y+2;

}

void printEnemy3() // imp, to be printed just once, as it would not get destroyed fast and will stay in 1 position

{

gotoxy(enemy3X, enemy3Y);

cout << "BOMBER";

gotoxy(enemy3X, enemy3Y+1);

cout << "BOMBER";

}

void eraseEnemy3()

{

gotoxy(enemy3X, enemy3Y);

cout << " ";

gotoxy(enemy3X, enemy3Y+1);

cout << " ";

isEnemy3Alive = false;

removeEnemy3Bullet();

}

// ==============================//

void moveEnemy2()

{

for(int i = 0; i < nrEmeny2; i++)

{

if( isEnemy2alive[i] == 1) // checking if enemy alive, then move

{

int enemyIndex = i;

if ( collisionsEnemy2(enemyIndex, enemy2Face[enemyIndex]) == 1) // if wall in front, vchange dircxtion of moviung

{

if (enemy2Face[enemyIndex] == "left")

{

enemy2Face[enemyIndex] = "right";

// enemy2Fire(enemyIndex);

}

else

{

enemy2Face[enemyIndex] = "left";

// enemy1Fire(enemyIndex);

}

}

else // if clear in fromnt, move

{

if (enemy2Face[enemyIndex] == "right")

{

enemy2X[enemyIndex] += 1;

}

else if (enemy2Face[enemyIndex] == "left")

{

enemy2X[enemyIndex] -= 1;

}

}

}

}

}

int collisionsEnemy2(int enemyNumber, string direction)

{

if (direction == "left") // if enemy direction is left, then checks on left

{

for(int i = 0; i< enemy2Ysize + 1; i++) // if in front of enemy is wall, return 1

{

if ( i == enemy2Ysize)

{

if (getCharAtxy(enemy2X[enemyNumber] - 1, enemy2Y[enemyNumber] + i) != '\*')

{

return 1;

}

}

else if (getCharAtxy(enemy2X[enemyNumber] - 1, enemy2Y[enemyNumber] + i) == '\*')

{

return 1;

}

}

return 0;

}

else if (direction == "right") // if enemy direction is on right, check on right then, not on nleft, it will cause a bug otherwise

{

for(int i = 0; i< enemy2Ysize + 1; i++)

{

if ( i == enemy2Ysize)

{

if (getCharAtxy(enemy2X[enemyNumber] + enemy2Xsize, enemy2Y[enemyNumber] + i) != '\*')

{

return 1;

}

}

else if (getCharAtxy(enemy2X[enemyNumber] + enemy2Xsize, enemy2Y[enemyNumber] + i) == '\*')

{

return 1;

}

}

return 0;

}

}

void printEnemy2()

{

for(int k = 0; k < nrEmeny2; k++)

{

if (isEnemy2alive[k] == 1) // pring only if alive

{

if (enemy2Face[k] == "right") // print according to face direwction

{

for (int i = 0; i < enemy2Ysize; i++)

{

gotoxy(enemy2X[k], enemy2Y[k] + i);

for (int j = 0 ; j < enemy2Xsize; j++)

{

if (i == 1 && j == 1)

{

cout << "/";

}

else if (i == 2 && j == 1)

{

cout << "{";

}

else

{

char tempCharacter = stoi(enemy2Array[i][j]);

cout << tempCharacter;

}

}

}

}

else if (enemy2Face[k] == "left") // print according to face direwction

{

for (int i = 0; i < enemy2Ysize; i++)

{

gotoxy(enemy2X[k], enemy2Y[k] + i);

for (int j = enemy2Xsize - 1 ; j >=0 ; j--)

{

char tempCharacter = stoi(enemy2Array[i][j]);

cout << tempCharacter;

}

}

}

}

}

}

void eraseEnemy2()

{

for(int i = 0; i < nrEmeny2; i++)

{

if (isEnemy2alive[i] == 1)

{

gotoxy(enemy2X[i], enemy2Y[i]);

cout << " ";

gotoxy(enemy2X[i], enemy2Y[i]+1);

cout << " ";

gotoxy(enemy2X[i], enemy2Y[i]+2);

cout << " ";

}

}

}

//===============================//

// enemy 1 fire

void enemy1Fire(int enemyIndex)

{

for(int i = 0; i < 21; i++) // 20 bcz we have space for 20 arrays, ehich is already enough

{

if (emenyBulletX[i] == 0 && emenyBulletY[i] == 0)

{

isEnemyBulletPresent[i] = 1;

if (enemy1Face[enemyIndex] == "right")

{

emenyBulletX[i] = enemy1X[enemyIndex] + enemy1Xsize;

emenyBulletY[i] = enemy1Y[enemyIndex];

enemyBulletXMovement[i] = 1;

// enemy1X[enemyIndex] -=1;

}

else

{

emenyBulletX[i] = enemy1X[enemyIndex] - 1;

emenyBulletY[i] = enemy1Y[enemyIndex];

enemyBulletXMovement[i] = -1;

}

enemyBulletYMovement[i] = 0;

enemyBulletLength[i] = enemy1BulletDistance;

break;

}

}

}

void moveEnemyBullet()

{

for(int i = 0; i < 21; i ++)

{

if (isEnemyBulletPresent[i] == 1)

{

emenyBulletX[i] += enemyBulletXMovement[i]; // moves bullet in a direction , specified when shoot, bcz obviously it wont go in snake, its straight line

emenyBulletX[i] += enemyBulletYMovement[i]; // no y used in this version., intuition was that so we can shot diagonal

if (enemyBulletCollisionPlayer(emenyBulletX[i], emenyBulletY[i]) == 1)

{

removeEnemyBullet(i);

playerDeath(respawnX, respawnY);

}

else if (getCharAtxy(emenyBulletX[i], emenyBulletY[i]) == wall ) // if collide wall, delete

{

removeEnemyBullet(i);

}

else

{

enemyBulletLength[i] -=1;

if (enemyBulletLength[i] == 0)

{

removeEnemyBullet(i);

}

}

}

}

}

void removeEnemyBullet(int i)

{

isEnemyBulletPresent[i] = 0;

emenyBulletX[i] = 0 ;

emenyBulletY[i] = 0 ;

enemyBulletXMovement[i] = 0;

enemyBulletYMovement[i] = 0;

}

void printEnemyBullet()

{

SetConsoleTextAttribute(hConsole, 4);

for(int i = 0; i < 21; i ++)

{

if (isEnemyBulletPresent[i] == 1)

{

gotoxy(emenyBulletX[i], emenyBulletY[i]);

cout << enemyBulletShape;

}

}

SetConsoleTextAttribute(hConsole, 7);

}

void eraseEnemyBullet()

{

for(int i = 0; i < 21; i ++)

{

if (isEnemyBulletPresent[i] == 1)

{

gotoxy(emenyBulletX[i], emenyBulletY[i]);

for (int j = 0; j < enemyBulletShape.length() ; j++)

{

cout << " ";

}

}

}

}

// collisions section

bool enemyBulletCollisionPlayer(int x, int y)

{

// char temp = getCharAtxy(x, y);

// for (int i = 0; i < (playerXsize \* playerYsize) ; i++)

// {

// if (playerArray[i] == temp)

// {

// return 1;

// }

// }

if ((x >= playerX && x <= (playerX + playerXsize-1)) && (y >= playerY && y <= (playerY + playerYsize -1)))

{

return 1;

}

return 0;

}

bool playerBulletCollisionEnemy(int x, int y)

{

char temp = getCharAtxy(x, y);

if ((y == enemy3Y || y == enemy3Y+1) && isEnemy3Alive == true)

{

if (x == enemy3X || x == enemy3X + 1)

{

eraseEnemy3();

// removeEnemy3Bullet();

return 1;

}

}

for (int i = 0; i < nrEmeny1 ; i++)

{

if ((x >= enemy1X[i] && x <= (enemy1X[i] + enemy1Xsize -1)) && (y >= enemy1Y[i] && y <= (enemy1Y[i] + enemy1Ysize -1)))

{

eraseEnemy1();

enemy1X[i] = 0;

enemy1Y[i] = 0;

isEnemy1alive[i] = 0;

printEnemy1();

return 1;

}

}

for (int i = 0; i < nrEmeny2 ; i++)

{

if ((x >= enemy2X[i] && x <= (enemy2X[i] + enemy2Xsize -1)) && (y >= enemy2Y[i] && y <= (enemy2Y[i] + enemy2Ysize -1)))

{

eraseEnemy2();

enemy2X[i] = 0;

enemy2Y[i] = 0;

isEnemy2alive[i] = 0;

printEnemy2();

return 1;

}

}

return 0;

}

bool enemyCollisionPlayer()

{

for (int i = 0; i < nrEmeny1; i++)

{

int enemyx = enemy1X[i];

int enemyy = enemy1Y[i];

for (int j = 0; j < enemy1Xsize; j++) // cehcking if axis of enemy 1 matches withp player

{

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

{

if ( (enemyx + j == playerX + k) && (enemyy == playerY+(playerYsize-1)) )

{

return 1;

}

if ( (enemyx + j == playerX + k) && ((enemyy + (enemy1Ysize-1)) == playerY) )

{

return 1;

}

}

}

for (int j = 0; j < enemy1Ysize; j++) // cehcking if axis of enemy 1 matches withp player

{

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

{

if ( (enemyx == playerX + (playerXsize-1)) && (enemyy + j == playerY+k) )

{

return 1;

}

if ( (enemyx + (enemy1Xsize - 1) == playerX) && (enemyy + j == playerY+k) )

{

return 1;

}

}

}

}

for (int i = 0; i < nrEmeny2; i++)// cehcking if axis of enemy 2 matches withp player

{

int enemyx = enemy2X[i];

int enemyy = enemy2Y[i];

for (int j = 0; j < enemy2Xsize; j++)

{

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

{

if ( (enemyx + j == playerX + k) && (enemyy == playerY+(playerYsize-1)) )

{

return 1;

}

if ( (enemyx + j == playerX + k) && ((enemyy + (enemy2Ysize-1)) == playerY) )

{

return 1;

}

}

}

for (int j = 0; j < enemy2Ysize; j++)// cehcking if axis of enemy 1 matches withp player

{

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

{

if ( (enemyx == playerX + (playerXsize-1)) && (enemyy + j == playerY+k) )

{

return 1;

}

if ( (enemyx + (enemy2Xsize - 1) == playerX) && (enemyy + j == playerY+k) )

{

return 1;

}

}

}

}

return 0;

}

int collisionsEnemy1(int enemyNumber, string direction)// cehcking if axis of enemy 1 matches withp player

{

if (direction == "left")

{

for(int i = 0; i< enemy1Ysize + 1; i++)

{

if ( i == enemy1Ysize)

{

if (getCharAtxy(enemy1X[enemyNumber] - 1, enemy1Y[enemyNumber] + i) != '\*')

{

return 1;

}

}

else if (getCharAtxy(enemy1X[enemyNumber] - 1, enemy1Y[enemyNumber] + i) == '\*')

{

return 1;

}

}

return 0;

}

else if (direction == "right")

{

for(int i = 0; i< enemy1Ysize + 1; i++)

{

if ( i == enemy1Ysize)

{

if (getCharAtxy(enemy1X[enemyNumber] + enemy1Xsize, enemy1Y[enemyNumber] + i) != '\*')

{

return 1;

}

}

else if (getCharAtxy(enemy1X[enemyNumber] + enemy1Xsize, enemy1Y[enemyNumber] + i) == '\*')

{

return 1;

}

}

return 0;

}

}

// enemy options //

void moveEnemy1()

{

for(int i = 0; i < nrEmeny1; i++)

{

if( isEnemy1alive[i] == 1) // move only if alive

{

int enemyIndex = i;

if ( collisionsEnemy1(enemyIndex, enemy1Face[enemyIndex]) == 1) // if wall in fornt, then change dirctions

{

if (enemy1Face[enemyIndex] == "left")

{

enemy1Face[enemyIndex] = "right";

enemy1Fire(enemyIndex);

}

else

{

enemy1Face[enemyIndex] = "left"; // if wall in fornt, then change dirctions

enemy1Fire(enemyIndex);

}

}

else // if clear in front, then just omve forward

{

if (enemy1Face[enemyIndex] == "right")

{

enemy1X[enemyIndex] += 1;

}

else if (enemy1Face[enemyIndex] == "left")

{

enemy1X[enemyIndex] -= 1;

}

}

}

}

}

void printEnemy1()

{

for(int k = 0; k < nrEmeny1; k++) // looping all enemies in array

{

if (isEnemy1alive[k] == 1)

{

if (currentLevel == 3 && k == 3) // we have special tratment for 3 in lvl 3, i.e fliying fdouble shooter

{

for (int i = 0; i < enemy1Ysize; i++)

{

gotoxy(enemy1X[k], enemy1Y[k] + i);

for (int j = 0 ; j < enemy1Xsize; j++)

{

cout << enemy1Array[i][j];

}

}

}

else if (enemy1Face[k] == "right") // print according to face firection

{

for (int i = 0; i < enemy1Ysize; i++)

{

gotoxy(enemy1X[k], enemy1Y[k] + i);

for (int j = 0 ; j < enemy1Xsize; j++)

{

cout << enemy1Array[i][j];

}

}

}

else if (enemy1Face[k] == "left")

{

for (int i = 0; i < enemy1Ysize; i++)

{

gotoxy(enemy1X[k], enemy1Y[k] + i);

for (int j = enemy1Xsize -1 ; j >= 0 ; j--)

{

if (i==0 && j == 2)

{

cout << "<";

}

else

{

cout << enemy1Array[i][j];

}

}

}

}

}

}

}

void eraseEnemy1()

{

for(int i = 0; i < nrEmeny1; i++)

{

if (isEnemy1alive[i] == 1)

{

gotoxy(enemy1X[i], enemy1Y[i]);

cout << " ";

gotoxy(enemy1X[i], enemy1Y[i]+1);

cout << " ";

}

}

}

// Player Firing //

void playerFire()

{

if (isPlayerEligibleForFire("first") == 1) // in case no fire has been shot

{

isPlayerBulletPresent[0] = 1; // say yes, there is indeed a fire

playerBulletDirection[0] = playerFace; // set direction to move

if ( playerBulletDirection[0] == "left")

{

playerBulletX[0] = playerX - 1;

}

else

{

playerBulletX[0] = playerX+playerXsize ;

}

playerBulletY[0] = playerY +1 ;

playerBulletDistanceCovered[0] = 0;

}

if (isPlayerEligibleForFire("next") == 1) // in case of a fire has been shot

{

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

{

if (isPlayerBulletPresent[i] == 0) // if bullet present, move according to face at that time

{

isPlayerBulletPresent[i] = 1;

playerBulletDirection[i] = playerFace;

if ( playerBulletDirection[i] == "left")

{

playerBulletX[i] = playerX - 1;

}

else

{

playerBulletX[i] = playerX+playerXsize ;

}

playerBulletY[i] = playerY +1 ;

playerBulletDistanceCovered[i] = 0;

break;

}

}

}

}

bool isPlayerEligibleForFire(string order)

{

if (order == "first") // return 1 if no bullet is fired by player

{

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

{

if (isPlayerBulletPresent[i] != 0)

{

return 0;

}

}

return 1;

}

else if (order == "next") // returns 1 if some time has passed since player has shot last and now he can do a 2nd 🙄

{

int minDistance = playerBulletLength; // setting how far bulet will go

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

{

if (isPlayerBulletPresent[i] == 1)

{

int tempLength = playerBulletDistanceCovered[i];

if (tempLength < minDistance)

{

minDistance = tempLength;

}

}

}

if ( minDistance > (lengthAfterPlayer2ndBulletAllow)) // this checks if there is a certains distance between last fired bullet and this

{

return 1;

}

return 0;

}

}

void printPlayerBullet()

{

SetConsoleTextAttribute(hConsole, 13); // chanigng color

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

{

if (isPlayerBulletPresent[i] == 1)

{

gotoxy(playerBulletX[i], playerBulletY[i]);

cout << bulletShape;

}

}

SetConsoleTextAttribute(hConsole, 7); // reverting back color

}

void movePlayerBullet()

{

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

{

if (isPlayerBulletPresent[i] == 1)

{

if ( playerBulletDirection[i] == "left")

{

// stop bullets and check for collision with enemy here

playerBulletX[i] -= 1;

}

else

{

// stop bullets and check for collision with enemy here

playerBulletX[i] += 1;

}

playerBulletDistanceCovered[i] += 1;

if ( playerBulletCollisionEnemy(playerBulletX[i], playerBulletY[i]) == 1)

{

increaseScore();

removePlayerBullet(i);

}

else if (getCharAtxy(playerBulletX[i], playerBulletY[i]) == wall)

{

removePlayerBullet(i);

}

else if (playerBulletDistanceCovered[i] == playerBulletLength)

{

removePlayerBullet(i);

}

}

}

}

void removePlayerBullet(int index)

{

isPlayerBulletPresent[index] = 0;

playerBulletX[index] = 0; // resetting array, i.e telling nre bullet there is space avail

playerBulletY[index] = 0;

playerBulletDistanceCovered[index] = 0;

}

void erasePlayerBullet()

{

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

{

if (isPlayerBulletPresent[i] == 1)

{

gotoxy(playerBulletX[i], playerBulletY[i]);

for(int j = 0; j < bulletShape.length(); j++)

{

cout << " ";

}

}

}

}

// player gravity and player jump

void Playerjump()

{

if (tempJump > 0) // checking if jump already pressed

{

tempJump -=1;

if (tempJump == 0) // enabling gravity again

{

currentJump = false;

}

int currentTemp = detectPlayerCollision("up");

if ( (currentTemp != 1)) // go up only if no wall above

{

playerY -= 1;

}

}

}

void playerGravity()

{

if (detectPlayerCollision("down") != 1 && currentJump == false) // if not already jumping and no wall below, then apply gravity

{

playerY += 1;

// printPlayer();

}

if (isJumpButton == 1)

{

jumpButtonJump();

}

}

// player collisions with walls

bool detectPlayerCollision(string direction)

{

if (direction == "left") // dpends on directions

{

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

{

if ( isNextCharWall(playerX-1, playerY + i) == 1) // if wall in front, cant move farward,change course will ya

{

return 1;

}

}

return 0;

}

else if (direction == "right")

{

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

{

if ( isNextCharWall(playerX + playerXsize, playerY + i) == 1) // if wall in front, cant move farward,change course will ya

{

return 1; // 1 means cannot go forward

}

}

return 0;

}

else if (direction == "up") // checking if we can go up, i.e which jumping

{

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

{

if ( isNextCharWall(playerX + i, playerY -1) == 1)

{

return 1;

}

}

return 0; // remember me ?

}

if (direction == "down") // checking if there is floor belw is

{

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

{

if ( isNextCharWall(playerX + i, playerY + playerYsize) == 1)

{

return 1;

}

}

return 0;

}

}

bool isNextCharWall(int x, int y) // basic simple functionality for if thetr e is a wall in giuven position

{

if (getCharAtxy(x,y) == '\*')

{

return 1;

}

return 0;

}

// player functionality

int movePlayer(string direction)

{

erasePlayer();

if (direction == "left") // move player left if nothing on left

{

if (detectPlayerCollision("left") != 1) // check here if wall in fronyt

{

playerFace = "left";

playerX -=1;

}

}

if (direction == "right")

{

if (detectPlayerCollision("right") != 1 ) // check here if wall in fronyt

{

playerFace = "right";

playerX +=1;

}

}

if (direction == "up")

{

if (detectPlayerCollision("down") == 1 && currentJump == false) // player on ground ( experimental player not already jumping, worked good, keeping it)

{

tempJump = jumpsize;

currentJump = true;

}

}

if (direction == "down") // fasll player if no floor bbelow

{

if (detectPlayerCollision("down") != 1 )

{

playerY +=1;

}

}

}

// printing faculity

void printPlayer()

{

if (playerFace == "right")

{

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

{

gotoxy(playerX, playerY + i);

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

{

cout << playerArray[i][j];

}

}

}

else if (playerFace == "left")

{

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

{

gotoxy(playerX, playerY + i);

for (int j = playerYsize-1 ; j >= 0; j--)

{

cout << playerArray[i][j];

}

}

}

gotoxy(playerX, playerY); // returning to original position, to top left

}

void printMaze1()

{

gotoxy(0, 15);

for (int i = 0; i < 38; i++)

{

for (int j = 0; j < 103; j++)

{

cout << maze12dArray[i][j];

}

cout << endl;

}

}

void printMaze2()

{

gotoxy(0, 15);

for (int i = 0; i < 45; i++)

{

for (int j = 0; j < 103; j++)

{

cout << maze22dArray[i][j];

}

cout << endl;

}

} // base 16, 56

void printMaze3()

{

gotoxy(0, 15);

for (int i = 0; i < 42; i++)

{

for (int j = 0; j < 103; j++)

{

cout << maze32dArray[i][j];

}

cout << endl;

}

printEnemy3();

if (isJumpButton == 1)

{

printJumpButton();

}

}

void printGameEndScreen()

{

system("cls");

Sleep(300);

getch();

gotoxy(50,25);

cout << "CONGRATS ! YOU HAVE CLEARED THE GAME" << endl;

gotoxy(50,26);

cout << " Press Anything To Continue";

Sleep(6000);

// getch();

}

void printBoss()

{

gotoxy(bossX, bossY);

cout << "BOSS";

gotoxy(bossX, bossY+1);

cout << "BOSS";

gotoxy(bossX, bossY+2);

cout << "BOSS";

gotoxy(bossX, bossY+3);

cout << "BOSS";

}

void eraseBoss()

{

gotoxy(bossX, bossY);

cout << " ";

gotoxy(bossX, bossY+1);

cout << " ";

gotoxy(bossX, bossY+2);

cout << " ";

gotoxy(bossX, bossY+3);

cout << " ";

}

void erasePlayer()

{

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

{

gotoxy(playerX, playerY+i);

cout << " ";

}

}

void printGun()

{

gotoxy(57,23);

cout << "Gun";

gotoxy(57,24);

cout << "Gun";

}

void eraseGun()

{

gotoxy(57,23);

cout << " ";

gotoxy(57,24);

cout << " ";

}

int isOnGun()

{

if (playerX == 57 || playerX == 58 || playerX == 59 || playerX + 2 == 57 || playerX+2 == 58 || playerX+2 == 59)

{

if (playerY == 23 || playerY == 22 || playerY == 21)

{

eraseGun();

isGun = true;

return 1;

}

}

return 0;

}

void increaseScore()

{

currentScore += scoreIncrement;

}

void decreaseScore()

{

currentScore -= scoreDecrement;

}

void printScore()

{

gotoxy(50, 18);

cout << " ";

gotoxy(50, 18);

cout << "SCORE: " << currentScore;

}

// =======================================================//

// Jump Button

void printJumpButton()

{

if (!(playerX <= 23 && (playerY >= 30 && playerY < 34)))

{

gotoxy(16,32);

cout << "\*\*\*\*\*\*\*";

gotoxy(16,33);

cout << " JUMP \*";

gotoxy(16,34);

cout << " \*";

}

}

void jumpButtonJump()

{

if ((playerX >= 16 && playerX <=20) && (playerY == 29))

{

templvl3jump = 1;

}

if (templvl3jump != 0)

{

erasePlayer();

playerY -= 2;

printPlayer();

templvl3jump += 1;

if (templvl3jump == lvl3jumpsize)

{

templvl3jump = 0;

}

}

}

// =======================================================//

// file Handling

void storeEnemy2And3()

{

fstream file1;

file1.open("txtFiles/enemy2And3.txt", ios::out);

file1 << nrEmeny2 << endl;

file1 << isEnemy2alive[0] << "," << isEnemy2alive[1] << "," << isEnemy2alive[2] << "," << isEnemy2alive[3] << endl;

file1 << enemy2X[0] << "," << enemy2X[1] << "," << enemy2X[2] << "," << enemy2X[3] << endl;

file1 << enemy2Y[0] << "," << enemy2Y[1] << "," << enemy2Y[2] << "," << enemy2Y[3] << endl;

file1 << enemy2Face[0] << "," << enemy2Face[1] << "," << enemy2Face[2] << "," << enemy2Face[3] << endl;

file1 << isEnemy3alive << "," << enemy3X << "," << enemy3Y << "," << enemy3BulletLength << "," << isEnemy3Alive << "," << isEnemy3Fired << endl;

file1 << emeny3BulletX << "," << emeny3BulletY << "," << enemy3BulletPreviousChar << "," << waittime << "," << tempWaittime << endl;

file1 << enemy3Speed << "," << currentScore << "," << currentLevel << "," << isJumpButton << "," << lvl3jumpsize << "," << templvl3jump << endl;

file1 << isGun << endl;

file1.close();

}

void loadEnemy2And3(string option)

{

string record = "";

fstream file1;

if (option == "newGame")

file1.open("txtFiles - NewGame/enemy2And3.txt", ios::in);

else if (option == "load")

file1.open("txtFiles/enemy2And3.txt", ios::in);

getline(file1, record);

nrEmeny2 = stoi(getFieldFromRecord(record, 0));

getline(file1, record);

for (int i = 0; i < 4; i++)

{

isEnemy2alive[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 4; i++)

{

enemy2X[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 4; i++)

{

enemy2Y[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 4; i++)

{

enemy2Face[i] = (getFieldFromRecord(record, i));

}

getline(file1, record);

isEnemy3alive = stoi(getFieldFromRecord(record, 0));

enemy3X = stoi(getFieldFromRecord(record, 1));

enemy3Y = stoi(getFieldFromRecord(record, 2));

enemy3BulletLength = stoi(getFieldFromRecord(record, 3));

isEnemy3Alive = stoi(getFieldFromRecord(record, 4));

isEnemy3Fired = stoi(getFieldFromRecord(record, 5));

getline(file1, record);

emeny3BulletX = stoi(getFieldFromRecord(record, 0));

emeny3BulletY = stoi(getFieldFromRecord(record, 1));

enemy3BulletPreviousChar = (getFieldFromRecord(record, 2))[0];

waittime = stoi(getFieldFromRecord(record, 3));

tempWaittime = stoi(getFieldFromRecord(record, 4));

getline(file1, record);

enemy3Speed = stoi(getFieldFromRecord(record, 0));

currentScore = stoi(getFieldFromRecord(record, 1));

currentLevel = stoi(getFieldFromRecord(record, 2));

isJumpButton = stoi(getFieldFromRecord(record, 3));

lvl3jumpsize = stoi(getFieldFromRecord(record, 4));

templvl3jump = stoi(getFieldFromRecord(record, 5));

getline(file1, record);

isGun = stoi(getFieldFromRecord(record, 0));

file1.close();

}

void storeEnemy1()

{

fstream file1;

file1.open("txtFiles/enemy1.txt", ios::out);

// Entering Player

file1 << nrEmeny1 << endl;

file1 << isEnemy1alive[0] << "," << isEnemy1alive[1] << "," << isEnemy1alive[2] << "," << isEnemy1alive[3] << endl;

file1 << enemy1X[0] << "," << enemy1X[1] << "," << enemy1X[2] << "," << enemy1X[3] << endl;

file1 << enemy1Y[0] << "," << enemy1Y[1] << "," << enemy1Y[2] << "," << enemy1Y[3] << endl;

file1 << enemy1Face[0] << "," << enemy1Face[1] << "," << enemy1Face[2] << "," << enemy1Face[3] << endl;

file1 << enemy1BulletDistance << endl;

for(int i = 0 ; i < 20; i++)

{

file1 << isEnemyBulletPresent[i] << ",";

}

file1 << endl;

for(int i = 0 ; i < 20; i++)

{

file1 << emenyBulletX[i] << ",";

}

file1 << endl;

for(int i = 0 ; i < 20; i++)

{

file1 << emenyBulletY[i] << ",";

}

file1 << endl;

for(int i = 0 ; i < 20; i++)

{

file1 << enemyBulletXMovement[i] << ",";

}

file1 << endl;

for(int i = 0 ; i < 20; i++)

{

file1 << enemyBulletYMovement[i] << ",";

}

file1 << endl;

for(int i = 0 ; i < 20; i++)

{

file1 << enemyBulletLength[i] << ",";

}

file1 << endl;

file1.close();

}

void loadEnemy1(string option)

{

string record = "";

fstream file1;

if (option == "newGame")

file1.open("txtFiles - NewGame/enemy1.txt", ios::in);

else if (option == "load")

file1.open("txtFiles/enemy1.txt", ios::in);

getline(file1, record);

nrEmeny1 = stoi(getFieldFromRecord(record, 0));

getline(file1, record);

for (int i = 0; i < 4; i++)

{

isEnemy1alive[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 4; i++)

{

enemy1X[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 4; i++)

{

enemy1Y[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 4; i++)

{

enemy1Face[i] = (getFieldFromRecord(record, i));

}

getline(file1, record);

enemy1BulletDistance = stoi(getFieldFromRecord(record, 0));

getline(file1, record);

for (int i = 0; i < 20; i++)

{

isEnemyBulletPresent[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 20; i++)

{

emenyBulletX[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 20; i++)

{

emenyBulletY[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 20; i++)

{

enemyBulletXMovement[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 20; i++)

{

enemyBulletYMovement[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

for (int i = 0; i < 20; i++)

{

enemyBulletLength[i] = stoi(getFieldFromRecord(record, i));

}

file1.close();

}

void storePlayerAndBoss()

{

fstream file1;

file1.open("txtFiles/playerAndBoss.txt", ios::out);

// Entering Player

file1 << respawnX << "," << respawnY << "," << playerX << "," << playerY << "," << playerFace << "," << playerSpeed << endl;

file1 << playerMaxBulletAtATime << "," << playerBulletLength << endl;

file1 << isPlayerBulletPresent[0] << "," << isPlayerBulletPresent[1] << "," << isPlayerBulletPresent[2] << "," << isPlayerBulletPresent[3] << "," << isPlayerBulletPresent[4] << endl;

file1 << playerBulletDirection[0] << "," << playerBulletDirection[1] << "," << playerBulletDirection[2] << "," << playerBulletDirection[3] << "," << playerBulletDirection[4] << endl;

file1 << playerBulletX[0] << "," << playerBulletX[1] << "," << playerBulletX[2] << "," << playerBulletX[3] << "," << playerBulletX[4] << endl;

file1 << playerBulletY[0] << "," << playerBulletY[1] << "," << playerBulletY[2] << "," << playerBulletY[3] << "," << playerBulletY[4] << endl;

file1 << playerBulletDistanceCovered[0] << "," << playerBulletDistanceCovered[1] << "," << playerBulletDistanceCovered[2] << "," << playerBulletDistanceCovered[3] << "," << playerBulletDistanceCovered[4] << endl;

// storing boss

file1 << bossX << "," << bossY << "," << lvl2BossCondition << "," << lvl2BossIteration << "," << showlev2Boss << "," << showlev1Boss << "," << islvl1BossShown << endl;

file1 << jumpsize << "," << tempJump << "," << currentJump << "," << jumpThisIteration << endl;

file1.close();

}

void loadPlayerAndBoss(string option)

{

string record = "";

fstream file1;

if (option == "newGame")

file1.open("txtFiles - NewGame/playerAndBoss.txt", ios::in);

else if (option == "load")

file1.open("txtFiles/playerAndBoss.txt", ios::in);

getline(file1, record);

respawnX = stoi(getFieldFromRecord(record, 0));

respawnY = stoi(getFieldFromRecord(record, 1));

playerX = stoi(getFieldFromRecord(record, 2));

playerY = stoi(getFieldFromRecord(record, 3));

playerFace = (getFieldFromRecord(record, 4));

playerSpeed = stoi(getFieldFromRecord(record, 5));

getline(file1, record);

playerMaxBulletAtATime = stoi(getFieldFromRecord(record, 0));

playerBulletLength = stoi(getFieldFromRecord(record, 1));

getline(file1, record);

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

{

isPlayerBulletPresent[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

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

{

playerBulletDirection[i] = (getFieldFromRecord(record, i));

}

getline(file1, record);

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

{

playerBulletX[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

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

{

playerBulletY[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

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

{

playerBulletDistanceCovered[i] = stoi(getFieldFromRecord(record, i));

}

getline(file1, record);

bossX = stoi(getFieldFromRecord(record, 0));

bossY = stoi(getFieldFromRecord(record, 1));

lvl2BossCondition = stoi(getFieldFromRecord(record, 2));

lvl2BossIteration = stoi(getFieldFromRecord(record, 3));

showlev2Boss = stoi(getFieldFromRecord(record, 4));

showlev1Boss = stoi(getFieldFromRecord(record, 5));

islvl1BossShown = stoi(getFieldFromRecord(record, 6));

file1.close();

}

//=========================================================//

// Startup And Pause

void PauseMenuWithOptions()

{

printPauseMenuWithOptions();

printPauseMenuArrow(1);

PauseArrowOptionSelected = 1;

while(true)

{

if (GetAsyncKeyState(VK\_UP))

{

if (PauseArrowOptionSelected == 2)

{

printPauseMenuArrow(1);

PauseArrowOptionSelected = 1;

}

else if (PauseArrowOptionSelected == 3)

{

printPauseMenuArrow(2);

PauseArrowOptionSelected = 2;

}

Sleep(100);

}

if (GetAsyncKeyState(VK\_DOWN))

{

if (PauseArrowOptionSelected == 1)

{

printPauseMenuArrow(2);

PauseArrowOptionSelected = 2;

}

else if (PauseArrowOptionSelected == 2)

{

printPauseMenuArrow(3);

PauseArrowOptionSelected = 3;

}

Sleep(100);

}

if (GetAsyncKeyState(VK\_NUMPAD0))

{

// load or new game based on startUpArrowOptionSelected

if (PauseArrowOptionSelected == 1)

{

if (currentLevel == 1)

{

printMaze1();

if (isGun == false)

{

printGun();

}

}

else if (currentLevel == 2)

{

printMaze2();

printEnemy2();

if (isEnemy1alive[0] == 1)

{

printEnemy1();

}

}

else if (currentLevel == 3)

{

printMaze3();

printEnemy3();

if (isJumpButton == 1)

printJumpButton();

}

break;

}

else if (PauseArrowOptionSelected == 2)

{

if (currentLevel == 1)

{

printMaze1();

if (isGun == false)

{

printGun();

}

}

else if (currentLevel == 2)

{

printMaze2();

printEnemy2();

printEnemy1();

}

else if (currentLevel == 3)

{

printMaze3();

// printEnemy3();

// if (isJumpButton == 1)

// printJumpButton();

}

storeAllData();

break;

}

else if (PauseArrowOptionSelected == 3)

{

Sleep(300);

getch();

StartMenuWithOptions();

}

}

}

}

void printPauseMenuArrow(int option)

{ // 47, 25

int x = 41, y = 25;

gotoxy(x,y);

cout << " ";

gotoxy(x,y+1);

cout << " ";

gotoxy(x,y+2);

cout << " ";

if (option == 1)

{

gotoxy(x,y);

cout << "---->";

}

else if (option == 2)

{

gotoxy(x,y+1);

cout << "---->";

}

else if (option == 3)

{

gotoxy(x,y+2);

cout << "---->";

}

}

void StartMenuWithOptions()

{

printStartMenuWithOptions();

printStartUpMenuArrow(1);

startUpArrowOptionSelected = 1;

while(true)

{

if (GetAsyncKeyState(VK\_UP))

{

printStartUpMenuArrow(1);

startUpArrowOptionSelected = 1;

}

else if (GetAsyncKeyState(VK\_DOWN))

{

printStartUpMenuArrow(2);

startUpArrowOptionSelected = 2;

}

else if (GetAsyncKeyState(VK\_NUMPAD0))

{

// load or new game based on startUpArrowOptionSelected

if (startUpArrowOptionSelected == 1)

{

currentGameMode = "newGame";

StartNewGame();

}

else if (startUpArrowOptionSelected == 2)

{

currentGameMode = "load";

LoadGame();

}

}

}

}

void printStartUpMenuArrow(int option)

{ // 47, 25

int x = 41, y = 25;

gotoxy(x,y);

cout << " ";

gotoxy(x,y+1);

cout << " ";

if (option == 1)

{

gotoxy(x,y);

cout << "---->";

}

else if (option == 2)

{

gotoxy(x,y+1);

cout << "---->";

}

}

string getFieldFromRecord(string record, int option)

{

int count = 0;

string field = "";

for (int i = 0; i < record.length(); i++)

{

if (record[i] == ',')

{

count ++;

}

else if (count == option)

{

field = field + record[i];

}

}

return field;

}

void storeAllData()

{

storePlayerAndBoss();

storeEnemy1();

storeEnemy2And3();

}

void loadAllData(string option)

{

loadPlayerAndBoss(option);

loadEnemy1(option);

loadEnemy2And3(option);

}

// ======================================================= //

void gotoxy(int x, int y)

{

COORD coordinates;

coordinates.X = x;

coordinates.Y = y;

SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), coordinates);

}

char getCharAtxy(short int x, short int y)

{

CHAR\_INFO ci;

COORD xy = {0,0};

SMALL\_RECT rect = {x,y,x,y};

COORD coordBufSize;

coordBufSize.X = 1;

coordBufSize.Y = 1;

return ReadConsoleOutput(GetStdHandle(STD\_OUTPUT\_HANDLE), &ci, coordBufSize, xy, &rect) ? ci.Char.AsciiChar : ' ';

}