Zombie game tasks

group programing tasks

LIAM HILL, OLIVER PARKER, ALEX OGDEN

2015

Table of Contents

[Basic code 3](#_Toc417976527)

[PDL diagram 33](#_Toc417976528)

[Extended version 34](#_Toc417976529)

[GROUP CONTRIBUTION 68](#_Toc417976530)

# Basic Code Listing

#include <iostream>

#include <iomanip>

#include <conio.h>

#include <string>

#include <vector>

#include <fstream>

#include <Windows.h>

#include "RandomUtils.h"

#include "ConsoleUtils.h"

#include "TimeUtils.h"

using namespace std;

const int SIZEY(12); //vertical dimension

const int SIZEX(20); //horizontal dimension

const char SPOT('@'); //spot

const char TUNNEL(' '); //open space

const char WALL('#'); //border

const char HOLE('O'); //hole

const char ZOMBIE('Z'); //zombie

const char PILL('.'); //pill (used in basic version insted of structure)

const int UP(72); //up arrow

const int DOWN(80); //down arrow

const int RIGHT(77); //right arrow

const int LEFT(75); //left arrow

const char FREEZ('F'); //stop the zombies moving

const char EXTERMINATE('X'); //remove all zombies

const char EAT('E'); //remove all pills

const char PLAY('P'); //play buttion

const char INFO('I'); //info button

const char QUIT('Q'); //end the game

struct Item {

const char symbol; //symbol on grid

int x, y; //coordinates

};

struct player {

Item baseobject; // the base class of all objects on the map

const string name; // the name of the player

int lives; // the number of lives the player has

int score; // the score the player has acheaved

bool hascheated; // set true if the user has cheated

};

struct zombie {

Item baseobject; // the base class of all objects on the map

int startx; // the start location of the zombie

int starty;

bool imobalized; // set true if the zombie cant move

bool alive; // variable to check if zombie has been killed or not

bool hidden; // variable to check whether a zombie has been hidden by the X cheat

zombie operator= (const zombie& it)

{

zombie a = it;

return a;

}

};

struct pill {

Item baseobject; // the base class of all objects on the map

bool eaten; // a variable to check whether spot has eaten a specific pill

pill operator= (const pill& it)

{

pill a = it;

return a;

}

};

int main()

{

//function declarations (prototypes)

void initialiseGame(char grid[][SIZEX], player& spot, vector<zombie>& zombies, vector<Item>& holes, vector<pill>& pills);

bool isArrowKey(const int k);

bool isCheatKey(const int k);

int getsize(const vector<pill>& pills);

int getKeyPress();

bool endconditions(vector<zombie>& zombies, const int pills, const player &spot, const int key, string& message);

void ApplyCheat(const int key, player& spot, vector<zombie>& zombies, vector<pill>& pills);

void updateGame(char grid[][SIZEX], player& spot, const int key, string& message, vector<zombie>& zombies, vector<pill>& pills, const vector<Item>& holes);

void renderGame(const char g[][SIZEX], const string &mess, const player &spot, const int zomlives, const int remaingpills);

void endProgram(const string &message);

string mainloop();

void savescore(const string &name, const int score);

bool readsavedcore(const string &name, const int score);

// declaration of all functions used in this main method, all pass different paramters and are all called below

char grid[SIZEY][SIZEX]; // grid for display

vector<zombie> zombies; // initalize the 4 zombies

vector<pill> pills; // initalize avalible pills to 8

vector<Item> holes; // sets up the 12 holes to places on the map

string message("LET'S START... "); // creates the message that is going to be sent to the player

int key(' '); // sets up the integer for key that will be used to determine the players input

player spot = { SPOT, 0, 0, mainloop(), 5 }; // sets up the item of the player with the position and name using

// the mainloop function (below)

Clrscr(); // clears the screen of all the introduction menu ('press p to play' etc)

initialiseGame(grid, spot, zombies, holes, pills); //initialise grid and places all the items onto the grid eg spot and holes

renderGame(grid, message, spot, zombies.size(), pills.size()); // this sets up all the on screen information

do {

message = " "; //reset message

key = getKeyPress(); //read in next keyboard event

if (isArrowKey(key)) //does an if statement if the player enters an arrow key

updateGame(grid, spot, key, message, zombies, pills, holes); //calls a function that is the main body of the game

// this takes the inputted key and determines how to move spot and the zombies

else if (isCheatKey(key)) //if the player enters a cheat key and not an arrow key this is called

{

spot.hascheated = true; //this variable changes, as score is not saved if the player has cheated

ApplyCheat(key, spot, zombies, pills); //this calls the function to apply whichever cheat button was pressed

updateGame(grid, spot, key, message, zombies, pills, holes);

}

renderGame(grid, message, spot, zombies.size(), getsize(pills)); //render game state on screen

} while (endconditions(zombies, getsize(pills), spot, key, message));

// this whole loops runs if the player hasn't died, won or quit, if anyone of these conditions change the game ends

if (!spot.hascheated) // this happens if the user hasn't cheated by the time the game ends

{

if (!readsavedcore(spot.name, spot.lives)) // this gets the previous score if there is one

savescore(spot.name, spot.lives); // this saves the new score if it is the users first time

// or it saves over the current score if it is higher

}

endProgram(message); //display final message

}

int getsize(const vector<pill>& pills)

{

int pils = 0;

for (const pill& item : pills) // this loop runs through every item in the pills list

if (!item.eaten) // if the pill is still on the map the pill integer gets increased, this is the size of the pills

// or the pills left one the map

++pils;

return pils;

}

string mainloop()

{

void requestname();

void showTitle();

void showOptions();

void showmenu();

void showtime();

void showgametitle();

int getscore(const string&);

int getKeyPress();

void clearMessage();

void showscore(const int score);

void showDescription();

// these are all the functions used in this part of the program, these are mainly display functions

// that display parts of the menu when the game starts

string name = "";

char key = ' ';

while (toupper(key) != PLAY)

{

// runs whilst the user hasn't clicked to play the game yet

showTitle();

showgametitle();

showOptions();

showtime();

showmenu();

//calls different display functions

key = getKeyPress(); //gets the key press from the user

if (toupper(key) == INFO) // if the key is the i key it displays the game info

showDescription();

else if (toupper(key) == QUIT) // if the key is the q key it quits

return 0;

else if (toupper(key) != PLAY) // if the key is anythoing else it displays an error message

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 13);

cout << "INVALID KEY! ";

}

}

requestname(); // once the user enters their name it

cin >> name;

clearMessage(); // clears the message

int previousscore = getscore(name); // gets the previous score from the name, -1 if first time playing

showscore(previousscore); // displays that score

return name; // returns the value of name for the player

}

void savescore(const string &name, const int score)

{

ofstream out(name + ".scr"); // creates a file stream names out

if (!out.fail()) // if it is available

out << score; // saves the score into this document

out.close(); //closes the stream

}

bool readsavedcore(const string &name, const int score)

{

ifstream in(name + ".scr");

if (!in.fail()) // the file may not be found

{

int storedscore; //sets up where to store it

in >> storedscore; // takes in the stored value

if (storedscore > score) // if the score is greater than original return true

return true;

else

return false;

}

in.close(); //closes the stream

return false;

}

int getscore(const string &name)

{

ifstream in(name + ".scr"); //Create input file stream with player name and extension

if (!in.fail()) //The file may not be found

{

int storedscore; //Gets the score

in >> storedscore;

return storedscore; //Returns it from the stream

}

in.close(); //Close the stream

return -1; //returns -1 if it is the players first game

}

void updateGame(char grid[][SIZEX], player& spot, const int key, string& message, vector<zombie>& zombies, vector<pill>& pills, const vector<Item>& holes)

{

void updateSpotCoordinates(const char g[][SIZEX], player& spot, const int key, string& mess, vector<zombie>& zombies, vector<pill>& pills); // player move

void updatezombieCoordinates(const char g[][SIZEX], player& spot, vector<zombie>& zombies); // zombies move

void updateGrid(char grid[][SIZEX], const Item &spot, const vector<zombie> &zombies, const vector<pill> &pills, const vector<Item> &holes);

updateSpotCoordinates(grid, spot, key, message, zombies, pills); //update spot coordinates

//according to key

updatezombieCoordinates(grid, spot, zombies);

//updates the zombies movement and ddoes their actions

updateGrid(grid, spot.baseobject, zombies, pills, holes); //updates the grid

//ie places zombies if alive and doesn't place pills if eaten

}

void updatezombieCoordinates(const char g[][SIZEX], player& spot, vector<zombie>& zombies) // zombies move

{

void getrandommove(const Item &spot, int& x, int& y); // this gets the co ordinates that the zombie will move

for (int i = 0; i < zombies.size(); i++) // loop runs for every zombie

{

if (zombies[i].imobalized == false && zombies[i].hidden == false && zombies[i].alive == true)

{

//calculate direction of movement required by key - if any

int dx(zombies[i].baseobject.x), dy(zombies[i].baseobject.y);

getrandommove(spot.baseobject, dx, dy);

//gets the co ordinates of the move

const int targetY(zombies[i].baseobject.y + dy);

const int targetX(zombies[i].baseobject.x + dx);

//gets the target co ordinate to move at

switch (g[targetY][targetX])

{ //...depending on what's on the target position in grid...

case PILL:

case TUNNEL: //can move if hits tunnel

zombies[i].baseobject.y += dy; //go in that Y direction

zombies[i].baseobject.x += dx; //go in that X direction

break;

case SPOT: // if it hits spot the players lives decrease and then the zombies respawn

spot.lives--;

zombies[i].baseobject.x = zombies[i].startx;

zombies[i].baseobject.y = zombies[i].starty;

break;

case ZOMBIE: // if it hits another zombie they both go back to their spawn

zombies[i].baseobject.x = zombies[i].startx;

zombies[i].baseobject.y = zombies[i].starty;

for (zombie& item : zombies)

{

if (item.baseobject.x == targetX && item.baseobject.y == targetY)

{

item.baseobject.x = item.startx;

item.baseobject.y = item.starty;

//this loop/if statement determine what zombie was hit

}

}

break;

case HOLE://remove the zombie from map if it hits a hole

zombies[i].alive = false;

}

}

}

}

void ApplyCheat(const int key, player& spot, vector<zombie>& zombies, vector<pill>& pills)

{

if (toupper(key) == EAT) // this eats all the pills

{

int livesGained = 0;

for (int i = 0; i < pills.size(); i++)

{

if (pills[i].eaten == true);

livesGained++; //this gets the amount of pills that will be eaten

}

spot.lives = spot.lives + livesGained; // this increases the amount of lives

pills.clear(); //clears all the pills of the board

}

else if (toupper(key) == EXTERMINATE)//remove all zombies from board

{

for (int i = 0; i != zombies.size(); i++)

{

zombies[i].hidden = !zombies[i].hidden; // turns all zombies that aren't hidden to not hidden and vice versa

zombies[i].baseobject.x = zombies[i].startx; // sets all zombies back to their start locations

zombies[i].baseobject.y = zombies[i].starty;

}

}

else if (toupper(key) == FREEZ)// do nothing when it is the zombies turn to move

for (int i = 0; i != zombies.size(); i++)

zombies[i].imobalized = !zombies[i].imobalized; // freezes all zombies and then when reclicked unfreezes them

}

void getrandommove(const Item &spot, int& x, int& y)

{

if (spot.x > x)

x = 1;

else

x = -1;

if (spot.y > y)

y = 1;

else

y = -1;

//this determines which way to go to get to the player

}

//---------------------------------------------------------------------------

//----- initialise game state

//---------------------------------------------------------------------------

void initialiseGame(char grid[][SIZEX], player& spot, vector<zombie>& zombies, vector<Item>& holes, vector<pill>& pills)

{ //initialise grid and place spot in middle

void setGrid(char[][SIZEX]);

void setSpotInitialCoordinates(char grid[][SIZEX], Item& spot);

void placeSpot(char gr[][SIZEX], const Item &spot);

void placepillonmap(char grid[][SIZEX], vector<pill>& pills);

void placeholeonmap(char grid[][SIZEX], vector<Item>& holes);

void placezombiesonmap(char grid[][SIZEX], vector<zombie>& zombies);

//sets up all the functions used to place things onto the map

Seed(); // seed reandom number generator

setSpotInitialCoordinates(grid, spot.baseobject);// initialise spot position

setGrid(grid); // setup empty grid

placezombiesonmap(grid, zombies); // place the zombies on the map

placeSpot(grid, spot.baseobject); // set spot in grid

placepillonmap(grid, pills); // place pills on the map

placeholeonmap(grid, holes); // place holes on the map

}

void placepillonmap(char grid[][SIZEX], vector<pill>& pills)

{

bool ocupiedpeace(const char gd[][SIZEX], const int x, const int y); // checks to see if the space is free

for (int i = 0; i != 8; i++) // place 8 pills on the map

{

int x = Random(SIZEX - 2); //

int y = Random(SIZEY - 2); //

while (ocupiedpeace(grid, x, y))

{

Seed();

x = Random(SIZEX - 2); // get new chordinates

y = Random(SIZEY - 2); //

}

pill pilla = { PILL, x, y };

pills.push\_back(pilla);

grid[y][x] = PILL; // place it on the map and adds it to the vector

}

}

void placeholeonmap(char grid[][SIZEX], vector<Item>& holes)

{

bool ocupiedpeace(const char gd[][SIZEX], const int x, const int y); //checks to see if the space is free

for (int i = 0; i != 12; i++) // place 12 holes on the map

{

int x = Random(SIZEX - 2); //

int y = Random(SIZEY - 2); //

while (ocupiedpeace(grid, x, y))

{

Seed();

x = Random(SIZEX - 2); // get new chordinates

y = Random(SIZEY - 2); //

}

Item hole = { HOLE, x, y };

grid[y][x] = HOLE;

holes.push\_back(hole); // places the hole and adds it to the vector

}

}

void placezombiesonmap(char grid[][SIZEX], vector<zombie>& zombies)

{

const zombie zom1 = { ZOMBIE, 1, 1, 1, 1, false, true }; // {{item}, startx, starty, imobilzed, alive}

const zombie zom2 = { ZOMBIE, SIZEX - 2, 1, SIZEX - 2, 1, false, true };

const zombie zom3 = { ZOMBIE, 1, SIZEY - 2, 1, SIZEY - 2, false, true };

const zombie zom4 = { ZOMBIE, SIZEX - 2, SIZEY - 2, SIZEX - 2, SIZEY - 2, false, true };

// creates all the zombies

zombies.push\_back(zom1);

zombies.push\_back(zom2);

zombies.push\_back(zom3);

zombies.push\_back(zom4);

//adds them all to the vector

grid[1][1] = ZOMBIE; // place it on the map

grid[SIZEY - 2][1] = ZOMBIE;

grid[1][SIZEX - 2] = ZOMBIE;

grid[SIZEY - 2][SIZEX - 2] = ZOMBIE;

//places them on the grid

}

void setSpotInitialCoordinates(char grid[][SIZEX], Item& spot)

{

bool ocupiedpeace(const char gd[][SIZEX], const int x, const int y); //checks to make sure the space is free

spot.y = Random(SIZEY - 2); //vertical coordinate in range [1..(SIZEY - 2)]

spot.x = Random(SIZEX - 2); //horizontal coordinate in range [1..(SIZEX - 2)]

while (ocupiedpeace(grid, spot.x, spot.y))

{

Seed();

spot.x = Random(SIZEX - 2); // get new chordinates

spot.y = Random(SIZEY - 2); //

}

}

void setGrid(char grid[][SIZEX])

{ //reset the empty grid configuration

for (int row(0); row < SIZEY; ++row) //for each column

{

for (int col(0); col < SIZEX; ++col) //for each col

{

if ((row == 0) || (row == SIZEY - 1)) //top and bottom walls

grid[row][col] = WALL; //draw a wall symbol

else

if ((col == 0) || (col == SIZEX - 1)) //left and right walls

grid[row][col] = WALL; //draw a wall symbol

else

grid[row][col] = TUNNEL; //draw a space

} //end of row-loop

} //end of col-loop

}

void placeSpot(char gr[][SIZEX], const Item &spot)

{ //place spot at its new position in grid

gr[spot.y][spot.x] = spot.symbol;

}

void updateGrid(char grid[][SIZEX], const Item &spot, const vector<zombie> &zombies, const vector<pill> &pills, const vector<Item> &holes)

{

void setGrid(char[][SIZEX]);

void placeSpot(char g[][SIZEX], const Item &spot);

void placezombies(char g[][SIZEX], const vector<zombie> &zombies);

void placepill(char g[][SIZEX], const vector<pill> &pills);

void placeitem(char g[][SIZEX], const vector<Item> &holes);

setGrid(grid); //reset empty grid

placezombies(grid, zombies); //set zombies on map

placepill(grid, pills); //set pills on map

placeitem(grid, holes); // set the holes on the grid

placeSpot(grid, spot); //set spot in grid

}

void placepill(char g[][SIZEX], const vector<pill> &pills)

{

for (const pill& item : pills) // goes through all pills

if (!item.eaten) // if the pill hasnt been eaten place it

g[item.baseobject.y][item.baseobject.x] = item.baseobject.symbol;

}

void placeitem(char g[][SIZEX], const vector<Item> &holes)

{

for (const Item& it : holes)

g[it.y][it.x] = it.symbol; // goes through all holes and places them

}

void placezombies(char g[][SIZEX], const vector<zombie> &zombies)

{

for (const zombie& item : zombies)

if (item.alive == true && item.hidden == false) // if the zombie isnt dead or hidden place it

g[item.baseobject.y][item.baseobject.x] = item.baseobject.symbol;

}

void updateSpotCoordinates(const char g[][SIZEX], player& sp, const int key, string& mess, vector<zombie>& zombies, vector<pill>& pills)

{

void setKeyDirection(const int k, int& dx, int& dy);

//calculate direction of movement required by key - if any

int dx(0), dy(0);

setKeyDirection(key, dx, dy); //find direction indicated by key

//check new target position in grid

//and update spot coordinates if move is possible

const int targetY(sp.baseobject.y + dy);

const int targetX(sp.baseobject.x + dx);

switch (g[targetY][targetX])

{ //...depending on what's on the target position in grid...

case TUNNEL: //can move

sp.baseobject.y += dy; //go in that Y direction

sp.baseobject.x += dx; //go in that X direction

break;

case WALL: //hit a wall and stay there

cout << '\a'; //beep the alarm

mess = "CANNOT GO THERE! ";

break;

case ZOMBIE:

sp.baseobject.y += dy; //go in that Y direction

sp.baseobject.x += dx; //go in that X direction

sp.lives--;

for (zombie& it : zombies)

{

if (sp.baseobject.x == it.baseobject.x && sp.baseobject.y == it.baseobject.y)

{

it.baseobject.x = it.startx;

it.baseobject.y = it.starty;

//finds out what zombie is hit and sends it to the start

}

}

break;

case HOLE:

sp.baseobject.y += dy; //go in that Y direction

sp.baseobject.x += dx; //go in that X direction

sp.lives--; //decreases player lives

break;

case PILL:

sp.baseobject.y += dy; //go in that Y direction

sp.baseobject.x += dx; //go in that X direction

sp.lives++;

for (zombie& it : zombies)

{

if (sp.baseobject.x == it.baseobject.x && sp.baseobject.y == it.baseobject.y)

//this happens when a zombie and a player both hit the same pill

{

sp.lives--;

it.baseobject.x = it.startx;

it.baseobject.y = it.starty;

}

}

for (int i = 0; i < pills.size(); i++)

if (pills[i].baseobject.x == sp.baseobject.x && pills[i].baseobject.y == sp.baseobject.y)

pills[i].eaten = true;

// this checks what pill has been hit and then sets it to eaten

break;

}

}

void setKeyDirection(const int key, int& dx, int& dy)

{

switch (key) //...depending on the selected key...

{

case UP: //when UP arrow pressed...

dx = 0;

dy = -1; //decrease the Y coordinate

break;

case DOWN: //when DOWN arrow pressed...

dx = 0;

dy = 1; //increase the Y coordinate

break;

case LEFT: //when LEFT arrow pressed...

dx = -1; //decrease the X coordinate

dy = 0;

break;

case RIGHT: //when RIGHT arrow pressed...

dx = +1; //increase the X coordinate

dy = 0;

}

}

int getKeyPress()

{

int keyPressed;

keyPressed = getch(); //read in the selected arrow key or command letter

while (keyPressed == 224) //ignore symbol following cursor key

keyPressed = getch();

return(keyPressed);

}

bool isArrowKey(const int key)

{

return ((key == LEFT) || (key == RIGHT) || (key == UP) || (key == DOWN)); // checks if its an arrow key

}

bool isCheatKey(const int key)

{

return ((toupper(key) == EAT) || (toupper(key) == EXTERMINATE) || (toupper(key) == FREEZ)); // checks if cheat key

}

bool wantToQuit(const int key, string& message)

{

bool exit = (toupper(key) == QUIT); // finds out if the key is the QUIT key

if (exit)

message = "you have quit"; // if true display a message

return exit;

}

bool haswon(vector<zombie>& zombies, const int pills, string& message, const player& spot)

{

if (zombies[0].alive == true || zombies[1].alive == true || zombies[2].alive == true || zombies[3].alive == true)

// the the zombies are still alive return false

{

return false;

}

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 16);

cout << "Congratulations, you win ";

Gotoxy(40, 17);

cout << "Your score is: " << spot.lives;

return true;

//if all zombies are dead, display message and return true

}

bool endconditions(vector<zombie>& zombies, const int pills, const player &spot, const int key, string& message)

{

bool haswon(vector<zombie>& zombies, const int pills, string& message, const player &spot);

bool haslost(const player &spot, string& message);

bool wantToQuit(const int k, string& message);

return (!wantToQuit(key, message) && (!haswon(zombies, pills, message, spot) && !haslost(spot, message)));

// this groups all the conditions together

}

bool haslost(const player &spot, string& message)

{

if (spot.lives == 0)

// if the player has ran out of lives the message is displayed

{

message = "you have no lives";

return true;

}

else

return false;

}

bool ocupiedpeace(const char gd[][SIZEX], const int x, const int y)

{

if (gd[y][x] == PILL || gd[y][x] == HOLE || gd[y][x] == ZOMBIE || gd[y][x] == SPOT || gd[y][x] == WALL)

//returns true if the space is aleady taken

return true;

else

return false;

}

void clearMessage()

{

SelectBackColour(clBlack);

SelectTextColour(clWhite);

Gotoxy(40, 8);

string str(20, ' ');

cout << str; //display blank message

}

void renderGame(const char gd[][SIZEX], const string &mess, const player &spot, const int zombielives, const int remainingpill)

{ //display game title, messages, maze, spot and apples on screen

void paintGrid(const char g[][SIZEX]);

void showLives(const player &spot);

void showDescription();

void showrempill(const int pils);

void showTitle();

void showOptions();

void showtime();

void showMessage(const string&);

void showname(const string &name);

void showscore(const int score);

//all the functions used in render game to disply all on the on screen information

Gotoxy(0, 0);

paintGrid(gd);

// pants the grid

showTitle();

// displaays the title

showDescription();

// displays the descriptions

showtime();

// displays the time

showLives(spot);

// displays the number of lives spot has

showname(spot.name);

// displays the players name

int previousscore = getscore(spot.name);

showscore(previousscore);

// displays the previous score

showrempill(remainingpill);

//show number of remaing pills

showOptions();

//display menu options available

showMessage(mess);

//display message if any

}

void paintGrid(const char g[][SIZEX])

{

SelectBackColour(clBlack);

Gotoxy(0, 2);

for (int row(0); row < SIZEY; ++row) //for each row (vertically)

{

for (int col(0); col < SIZEX; ++col) //for each column (horizontally)

{

switch (g[row][col])

{

case SPOT:

case WALL:

SelectTextColour(clWhite);

break;

case ZOMBIE:

SelectTextColour(clGreen);

break;

case HOLE:

SelectTextColour(clRed);

break;

case PILL:

SelectTextColour(clYellow);

}

cout << g[row][col]; //output cell content

} //end of col-loop

cout << endl;

} //end of row-loop

}

void showrempill(const int pils)

//shows pills left

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 10);

cout << "pills left: " << pils;

}

void showDescription()

//shows information about game

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 2);

cout << "This is a game where you must escape";

Gotoxy(40, 3);

cout << "the zombies and survive. Pills mean";

Gotoxy(40, 4);

cout << "a life is gained.";

Gotoxy(40, 5);

cout << "Contact with a hole(0) or zombie(Z)";

Gotoxy(40, 6);

cout << "means a life is lost ";

}

void showTitle()

{ //display game title

SelectTextColour(clYellow);

Gotoxy(0, 0);

cout << "\_\_\_ZOMBIES GAME SKELETON\_\_\_\n" << endl;

SelectBackColour(clWhite);

SelectTextColour(clRed);

Gotoxy(40, 0);

cout << "Oliver Parker, Liam Hill, Alex Odgen";

Gotoxy(40, 1);

cout << "1RR - COMPUTER SCIENCE";

}

void showname(const string &name)

//shows players name

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 13);

cout << "your name: " << name;

}

void showOptions()

{ //show game options

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 7);

cout << "TO MOVE USE KEYBOARD ARROWS ";

Gotoxy(40, 8);

cout << "TO QUIT ENTER 'Q' ";

}

void showLives(const player &spot)

{ //show game options

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 9);

cout << spot.lives << " lives left";

}

void showMessage(const string &m)

{ //print auxiliary messages if any

SelectBackColour(clBlack);

SelectTextColour(clWhite);

Gotoxy(40, 8);

cout << m; //display current message

}

void endProgram(const string &message)

{ //end program with appropriate message

SelectBackColour(clBlack);

SelectTextColour(clYellow);

Gotoxy(40, 8);

cout << message;

//hold output screen until a keyboard key is hit

Gotoxy(40, 9);

system("pause");

}

void showmenu()

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 10);

cout << "press p to play";

Gotoxy(40, 11);

cout << "press i to get infomation";

}

void showscore(const int score)

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 15);

cout << "player score: " << score;

}

void showtime()

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 11);

cout << GetDate();

Gotoxy(40, 12);

cout << GetTime();

}

void showgametitle()

{

SelectBackColour(clBlue);

SelectTextColour(clYellow);

Gotoxy(2, 4);

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

Gotoxy(2, 5);

cout << "| SPOT AND ZOMBIE GAME |";

Gotoxy(2, 6);

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

}

void requestname()

{

SelectBackColour(clBlue);

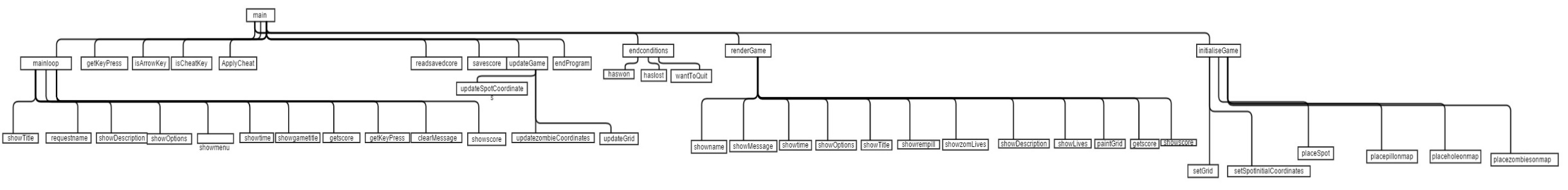
SelectTextColour(clYellow);

Gotoxy(2, 11);

cout << "please enter your name: ";

}

# PDL diagram



# GROUP CONTRIBUTION

|  |  |  |  |
| --- | --- | --- | --- |
| **Group Number:** |  | **Individual Contribution** | |
| **Member name:** | **Brief Description of work done** | **Effort** | **Achievement** |
| 1. **Liam hill** | **Good work** | **100** | **100** |
| 1. **OLIVER PARKER** | **Good work** | **100** | **100** |
| 1. **ALEX OGDEN** | **Good work** | **100** | **100** |
| **Total** | | **100/100** | **100/100** |