Zombie game tasks

group programing tasks

LIAM HILL, OLIVER PARKER, ALEX OGDEN

2015

# Basic code

#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');

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

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;

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(const int zombies, const int pills, const player &spot, const int key, string& message);

void ApplyCheat(const int key, 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);

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

//local variable declarations

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; // 12 holes

string message("LET'S START... "); //current message to player

int key(' ');

player spot = { SPOT, 0, 0, mainloop(), 5 }; //create key to store keyboard events

Clrscr();

initialiseGame(grid, spot, zombies, holes, pills); //initialise grid (incl. walls and spot)

renderGame(grid, message, spot, zombies.size(), pills.size());

do {

message = " "; //reset message

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

if (isArrowKey(key))

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

else if (isCheatKey(key))

{

spot.hascheated = true;

ApplyCheat(key, zombies, pills);

}

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

} while (endconditions(zombies.size(), getsize(pills), spot, key, message)); //while user does not want to quit

if (!spot.hascheated)

{

if (!readsavedcore(spot.name, spot.lives))

savescore(spot.name, spot.lives);

updatescore(spot.name, spot.lives);

}

endProgram(message); //display final message

}

int getsize(const vector<pill> pills)

{

int pils = 0;

for (const pill& item : pills)

if (!item.eaten)

++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 displayhighscores();

void showDescription();

string name = "";

char key = ' ';

while (toupper(key) != PLAY)

{

showTitle();

showgametitle();

showOptions();

showtime();

displayhighscores();

showmenu();

key = getKeyPress();

if (toupper(key) == INFO)

showDescription();

else if (toupper(key) == QUIT)

return 0;

else if (toupper(key) != PLAY)

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 13);

cout << "INVALID KEY! ";

}

}

requestname();

cin >> name;

clearMessage();

int previousscore = getscore(name);

showscore(previousscore);

return name;

}

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

{

ofstream out(name + ".scr");

if (!out.fail())

out << score;

out.close();

}

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

{

ifstream in(name + ".scr");

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

{

int storedscore;

in >> storedscore;

if (storedscore > score)

return true;

else

return false;

}

in.close();

return false;

}

int getscore(const string &name)

{

ifstream in(name + ".scr");

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

{

int storedscore;

in >> storedscore;

return storedscore;

}

in.close();

return -1;

}

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); // zombies move

// this can be just passed a vector<item> made from the .baseobject of all objects needing to be renderd

updateGrid(grid, spot.baseobject, zombies, pills, holes); //update grid information

}

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

{

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

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

{

if (zombies[i].imobalized == false)

{

//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); // if we pass the grid to this we can check to make it rare that the rombie falls down a hole

//check new target position in grid

//and update spot coordinates if move is possible

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

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

switch (g[targetY][targetX])

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

case PILL:

case TUNNEL: //can move

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

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

break;

case SPOT:// dont know if neede

spot.lives--;

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

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

break;

case ZOMBIE:

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;

}

}

break;

case HOLE://remove the zombie from map

zombies.erase(zombies.begin() + i);

}

}

}

}

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

{

if (toupper(key) == EAT)//remove all pils from the grid

pills.clear();

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

zombies.clear();

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;

}

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;

}

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

//----- 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(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);

Seed(); //seed reandom number generator

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

setGrid(grid); //reset empty grid

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

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

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

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

}

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

{

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

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

}

}

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

{

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

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);

}

}

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

{

zombie zom1 = { ZOMBIE, 1, 1, 1, 1 };

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

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

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

zombies.push\_back(zom1);

zombies.push\_back(zom2);

zombies.push\_back(zom3);

zombies.push\_back(zom4);

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;

}

void setSpotInitialCoordinates(Item& spot)

{

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

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

if (!item.eaten)

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;

}

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

{

for (const zombie& item : zombies)

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;

}

}

break;

case HOLE:

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

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

sp.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)

{

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) // fix me removing the wrong pill

pills[i].eaten = true; // again needs to be fixed

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));

}

bool isCheatKey(const int key)

{

return ((toupper(key) == EAT) || (toupper(key) == EXTERMINATE) || (toupper(key) == FREEZ));

}

bool wantToQuit(const int key, string& message)

{

bool exit = (toupper(key) == QUIT);

if (exit)

message = "you have quit";

return exit;

}

bool haswon(const int zombies, const int pills, string& message, const player& spot)

{

if (zombies > 0)

return false;

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 16);

cout << "Congratulations, you win ";

Gotoxy(40, 17);

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

return true;

}

bool endconditions(const int zombies, const int pills, const player &spot, const int key, string& message)

{

bool haswon(const int 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)));

}

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

{

if (spot.lives == 0)

{

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)

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 showzomLives(const int lives);

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);

Gotoxy(0, 0);

//display grid contents

paintGrid(gd);

//display game title

showTitle();

showDescription();

showtime();

showLives(spot);

showname(spot.name);

int previousscore = getscore(spot.name);

showscore(previousscore);

//show number of zombie lives

showzomLives(zombielives);

//show number of remaing pills

showrempill(remainingpill);

//display menu options available

showOptions();

//display message if any

showMessage(mess);

}

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 showzomLives(const int lives)

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 11);

cout << "zombie lives: " << lives;

}

void showrempill(const int pils)

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 10);

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

}

void showDescription()

{

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)

{

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: ";

}

void displayhighscores()

{

ifstream in("best.scr");

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

{

Gotoxy(2, 13);

cout << "name score";

for (int i = 0; i != 3; ++i)

{

int storedscore; // the score

string name; // the name

in >> name;

in >> storedscore;

Gotoxy(2, 14 + i);

cout << name << " " << storedscore;

}

}

else

{

Gotoxy(2, 13);

cout << "bob 3";

Gotoxy(2, 14);

cout << "tom 3";

Gotoxy(2, 15);

cout << "jim 3";

}

in.close();

}

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

{

ifstream in("best.scr");

if (!in.fail())

{

Gotoxy(2, 13);

string name1;

string name2;

string name3;

int score1;

int score2;

int score3;

in >> name1;

in >> score1;

in >> name2;

in >> score2;

in >> name3;

in >> score3;

in.close();

ofstream out("best.scr");

if (score1 < score)

{

score3 = score2;

score2 = score1;

name3 = name2;

name2 = name1;

score1 = score;

name1 = name;

}

else if (score2 < score)

{

score3 = score2;

name3 = name2;

score2 = score;

name2 = name;

}

else if (score3 < score)

{

score3 = score;

name3 = name;

}

out << name1 << endl;

out << score1 << endl;

out << name2 << endl;

out << score2 << endl;

out << name3 << endl;

out << score3 << endl;

out.close();

}

else

{

ofstream out("best.scr");

out << "bob" << endl;

out << 3 << endl;

out << "tom" << endl;

out << 2 << endl;

out << "jim" << endl;

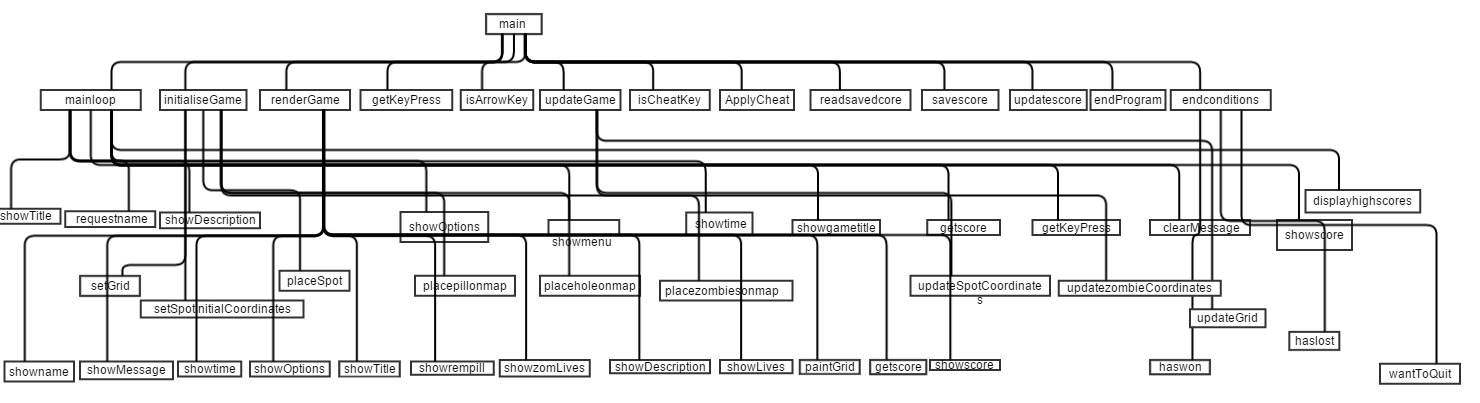
out << 1 << endl;

out.close();

}

}

# PDL diagram



# Extended version

#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 INSIDEWALL('#'); //inside wall

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

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

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

const char MPILL('-'); // magic pills

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');

const char REPLAY('R'); //replay buttion

const char SAVE('s'); // save key

const char LOAD('l'); // load key

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

struct replay

{

char grid[SIZEY][SIZEX]; // store the grid

};

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 levelchoice; // the chosen level

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

bool isProtected; // will be used for magic pill

int protectedcount; // protection only lasts 10 turns

};

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

zombie operator= (const zombie& it)

{

return it;

}

};

struct pill {

Item baseobject; // the base class of all ob jects on the map

bool eaten; // set true if the will not be displayed

pill operator= (const pill& it)

{

return it;

}

};

struct game //will be very helpfull

{

player spot;

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

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

vector<pill> mpills; // initalize magic pills to 8

const vector<Item> holes; // 12 holes

const vector<Item> inwalls; // some walls

game operator= (const game& it)

{

return it;

}

};

int main()

{

//function declarations (prototypes)

game initialiseGame(char grid[][SIZEX]);

bool isArrowKey(const int k);

bool isCheatKey(const int k);

bool issaveKey(const int k);

bool isloadKey(const int k);

bool isreplayKey(const int k);

int getsize(const vector<pill> pills);

int getKeyPress();

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

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

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);

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

void updateGame(char grid[][SIZEX], game& world, const int key, string& message);

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

void endProgram(const string&);

void savegame(const string &name, const game &world);

game loadgame(const string &name);

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

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

void saveboard(vector<replay>& replayer, const char grid[][SIZEX]);

void displayallmoves(const vector<replay> &replayer);

void nextlevel(game& world, char grid[][SIZEX]);

//local variable declarations

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

vector<replay> replayer;

string message("LET'S START... "); //current message to player

int key(' '); //create key to store keyboard events

game world = initialiseGame(grid); //initialise grid (incl. walls and spot)

Clrscr();

do {

nextlevel(world, grid);

int hours, min, seconds; // do we need to reset the timer when we load the next level

GetSystemTime(hours, min, seconds);// populates the varibles

do {

int nhours, nmin, nseconds;

GetSystemTime(nhours, nmin, nseconds);

const int diff = (((nhours - hours) \* 3600) + ((nmin - min) \* 60) + (nseconds - seconds)); // there is a way using delta time but this will work

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

if (\_kbhit() != 0)

{

saveboard(replayer, grid);

message = " "; //reset message

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

if (isArrowKey(key))

updateGame(grid, world, key, message);

else if (isCheatKey(key))

{

world.spot.hascheated = true;

ApplyCheat(key, world.zombies, world.pills);

}

else if (issaveKey(key))

savegame(world.spot.name, world);

else if (isloadKey(key))

world = loadgame(world.spot.name);

else if (isreplayKey(key))

displayallmoves(replayer);

else

message = "INVALID KEY! ";

}

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

} while (endconditions(world.zombies.size(), world.pills.size(), world.spot, key, message)); //while user does not want to quit

key = ' ';

} while (world.spot.lives != 0 && world.spot.levelchoice <= 3 && key != QUIT && !world.spot.hascheated);

if (!world.spot.hascheated)

{

if (!readsavedcore(world.spot.name, world.spot.score) && !world.spot.hascheated)

savescore(world.spot.name, world.spot.score);

updatescore(world.spot.name, world.spot.lives);

}

endProgram(message); //display final message

}

void nextlevel(game& world, char grid[][SIZEX])

{

void setGrid(char[][SIZEX]);

Item setSpotInitialCoordinates(char[][SIZEX]);

vector<pill> placepillonmap(char grid[][SIZEX], const int levelChoice);

vector<pill> placemagicpills(char grid[][SIZEX]);

vector<Item> placeholeonmap(char grid[][SIZEX], const int levelChoice);

vector<Item> placewallsonmap(char grid[][SIZEX]);

vector<zombie> placezombiesonmap(char grid[][SIZEX]);

Seed(); //seed reandom number generator

setGrid(grid);

game a = { { setSpotInitialCoordinates(grid), world.spot.name , world.spot.levelchoice+1 }, placezombiesonmap(grid), placepillonmap(grid, a.spot.levelchoice), placemagicpills(grid), placeholeonmap(grid, a.spot.levelchoice), placewallsonmap(grid) };//initialise spot position

world = a;

}

void saveboard(vector<replay>& replayer, const char grid[][SIZEX])

{

replay newstep;

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

{

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

{

newstep.grid[row][col] = grid[row][col]; // save the board

}

}

replayer.push\_back(newstep);

}

int level()

{

void showMessage(const string&);

int getKeyPress();

void showtime();

bool isvalidlevel(int level);

int level = 0;

Gotoxy(40, 20);

showMessage("Select level");

while (!isvalidlevel(level))

{

if (\_kbhit() != 0)

level = getKeyPress() - '0';

showtime();

}

return level;

}

bool isvalidlevel(const int level)

{

if (level >=1 && level <= 3)

return true;

return false;

}

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 displayname(const string &name);

stringstream name;

void displayhighscores();

void showDescription();

char key = ' ';

while (key != PLAY)//may work

{

displayname(name.str());

showTitle();

showgametitle();

showOptions();

showtime();

displayhighscores();

showmenu();

key = getKeyPress();

if (toupper(key) == INFO)

showDescription();

else if (toupper(key) == QUIT)

return 0;

else if (toupper(key) != PLAY)

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 13);

cout << "INVALID KEY! ";

}

}

while (key != 13)//may work

{

requestname();

if (\_kbhit() != 0)

{

key = getKeyPress();

if (key != 13)

name << key;

}

}

clearMessage();

return name.str();

}

void displayallmoves(const vector<replay> &replayer)

{

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

int getKeyPress();

void showDescription();

void showTitle();

void showOptions();

void showmenu();

void showtime();

void showMessage(const string&);

int index = 0;

char key = ' ';

Clrscr();

while (index != replayer.size())

{

showDescription();

showTitle();

showOptions();

showmenu();

showtime();

stringstream a;

a << "displaying move " << index << " of " << replayer.size();

showMessage(a.str());

paintGrid(replayer[index].grid);

if (\_kbhit() != 0)

{

key = getKeyPress();

if (key == RIGHT)

index++;

else if (key == LEFT)

index--;

}

}

}

void displayname(const string &name)

{

cout << name;

}

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

{

ofstream out(name + ".scr");

if (!out.fail())

out << score;

out.close();

}

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

{

ifstream in(name + ".scr");

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

{

int storedscore;

in >> storedscore;

if (storedscore > score)

return true;

else

return false;

}

in.close();

return false;

}

int getscore(const string &name)

{

ifstream in(name + ".scr");

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

{

int storedscore;

in >> storedscore;

return storedscore;

}

in.close();

return -1;

}

void updateGame(char grid[][SIZEX], game& spot,const int key, string& message)

{

void updateSpotCoordinates(const char g[][SIZEX], game& world, const int key, string& mess); // player move

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

void updateGrid(char grid[][SIZEX], const game &world);

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

//according to key

updatezombieCoordinates(grid, spot.zombies,spot.spot); // zombies move

// this can be just passed a vector<item> made from the .baseobject of all objects needing to be renderd

updateGrid(grid, spot); //update grid information

}

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

{

void getrandommove(const player&, int& x, int& y);

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

{

if (zombies[i].imobalized == false)

{

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

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

getrandommove(spot, dx, dy); // if we pass the grid to this we can check to make it rare that the rombie falls down a hole

//check new target position in grid

//and update spot coordinates if move is possible

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

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

switch (g[targetY][targetX])

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

case PILL:

case TUNNEL: //can move

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

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

break;

case SPOT:// dont know if neede

if (!spot.isProtected)

spot.lives--;

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

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

break;

case ZOMBIE:

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;

}

}

break;

case HOLE: // may need to store the starting chord of eatch zombie to make it spawn in its corner

zombies.erase(zombies.begin() + i);

}

}

}

}

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

{

if (toupper(key) == EAT)//remove all pils from the grid

pills.clear();

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

zombies.clear();

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;

}

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

{

if (!spot.isProtected)

{

if (spot.baseobject.x > x)

x = 1;

else

x = -1;

if (spot.baseobject.y > y)

y = 1;

else

y = -1;

}

else

{

if (spot.baseobject.x > x)

x = -1;

else

x = 1;

if (spot.baseobject.y > y)

y = -1;

else

y = 1;

}

}

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

//----- initialise game state

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

game initialiseGame(char grid[][SIZEX])

{ //initialise grid and place spot in middle

void setGrid(char[][SIZEX]);

Item setSpotInitialCoordinates(char[][SIZEX]);

vector<pill> placepillonmap(char grid[][SIZEX], const int levelChoice);

vector<pill> placemagicpills(char grid[][SIZEX]);

vector<Item> placeholeonmap(char grid[][SIZEX], const int levelChoice);

vector<Item> placewallsonmap(char grid[][SIZEX]);

vector<zombie> placezombiesonmap(char grid[][SIZEX]);

string mainloop();

int level();

Seed(); //seed reandom number generator

setGrid(grid);

game a = { { setSpotInitialCoordinates(grid), mainloop(), level() }, placezombiesonmap(grid), placepillonmap(grid, a.spot.levelchoice), placemagicpills(grid), placeholeonmap(grid, a.spot.levelchoice), placewallsonmap(grid) };//initialise spot position

switch (a.spot.levelchoice)

{

case 1:

a.spot.lives = 8;

break;

case 2:

a.spot.lives = 5;

break;

case 3:

a.spot.lives = 3;

break;

}

return a;

}

vector<Item> placewallsonmap(char grid[][SIZEX])

{

vector<Item> holes;

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

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 wall = { WALL, x, y };

grid[y][x] = WALL;

holes.push\_back(wall);

}

return holes;

}

vector<pill> placepillonmap(char grid[][SIZEX], const int levelChoice)

{

vector<pill> occupyPills(const int numberOfPills, char grid[][SIZEX]);

switch (levelChoice)

{

case 1:

return occupyPills(8, grid);

case 2:

return occupyPills(5, grid);

case 3:

return occupyPills(2, grid);

}

}

vector<pill> occupyPills(const int numberOfPills, char grid[][SIZEX])

{

vector<pill> pills;

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

for (int i = 0; i != numberOfPills; 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

}

return pills;

}

vector<pill> placemagicpills(char grid[][SIZEX])

{

vector<pill> pills;

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

for (int i = 0; i != 4; i++) // place 4 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 = { MPILL, x, y, false };

pills.push\_back(pilla);

grid[y][x] = MPILL; // place it on the map

}

return pills;

}

vector<Item> placeholeonmap(char grid[][SIZEX], const int levelChoice)

{

vector<Item> occupyHoles(char grid[][SIZEX], const int numberOfHoles);

switch (levelChoice)

{

case 1:

return occupyHoles(grid, 12);

case 2:

return occupyHoles(grid, 5);

case 3:

return occupyHoles(grid, 2);

}

}

vector<Item> occupyHoles(char grid[][SIZEX], const int numberOfHoles)

{

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

vector<Item> holes;

for (int i = 0; i != numberOfHoles; 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); //

}

const Item hole = { HOLE, x, y };

grid[y][x] = HOLE;

holes.push\_back(hole);

}

return holes;

}

vector<zombie> placezombiesonmap(char grid[][SIZEX])

{

zombie zom1 = { ZOMBIE, 1, 1, 1, 1 };

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

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

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

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;

return { zom1, zom2, zom3, zom4 };

}

Item setSpotInitialCoordinates(char gr[][SIZEX])

{

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

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

gr[y][x] = SPOT;

return{ SPOT, x, y };

}

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 updateGrid(char grid[][SIZEX],const game& world)

{

void setGrid(char[][SIZEX]);

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

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

setGrid(grid); //reset empty grid todo: we are only intrested in the item structure located in all objects when placing them on the grid

for (const zombie& item : world.zombies)

placeSpot(grid, item.baseobject); //set zombies on map

for (const pill& item : world.pills)

if (!item.eaten)

placeSpot(grid, item.baseobject); //set pills on map

for (const pill& item : world.mpills)

placeSpot(grid, item.baseobject); //set magic pills on map

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

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

placeitem(grid, world.inwalls);

}

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

{ //place spot at its new position in grid

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

}

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

{

for (Item it : holes)

g[it.y][it.x] = it.symbol;

}

void updateSpotCoordinates(const char g[][SIZEX], game& world,const int key, string& mess)

{

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(world.spot.baseobject.y + dy);

const int targetX(world.spot.baseobject.x + dx);

switch (g[targetY][targetX])

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

case TUNNEL: //can move

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

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

world.spot.protectedcount--;

break;

case WALL: //hit a wall and stay there

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

mess = "CANNOT GO THERE! ";

world.spot.protectedcount--;

break;

case ZOMBIE:

if (world.spot.isProtected)

world.spot.protectedcount--;

else

world.spot.lives--;

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

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

for (zombie& it : world.zombies)

{

if (world.spot.baseobject.x == it.baseobject.x && world.spot.baseobject.y == it.baseobject.y)

{

it.baseobject.x = it.startx;

it.baseobject.y = it.starty;

}

}

break;

case HOLE:

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

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

world.spot.lives--;

world.spot.protectedcount--;

break;

case PILL:

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

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

world.spot.lives++;

world.spot.protectedcount--;

for (zombie& it : world.zombies)

{

if (world.spot.baseobject.x == it.baseobject.x && world.spot.baseobject.y == it.baseobject.y)

{

world.spot.lives--;

it.baseobject.x = it.startx;

it.baseobject.y = it.starty;

}

}

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

if (world.pills[i].baseobject.x == world.spot.baseobject.x && world.pills[i].baseobject.y == world.spot.baseobject.y) // fix me removing the wrong pill

world.pills[i].eaten = true; // again needs to be fixed

break;

case MPILL:

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

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

world.spot.isProtected = true; // protect the player

for (zombie& it : world.zombies)

{

if (world.spot.baseobject.x == it.baseobject.x && world.spot.baseobject.y == it.baseobject.y)

{

it.baseobject.x = it.startx;

it.baseobject.y = it.starty;

}

}

switch (world.spot.levelchoice)

{

case 1:

world.spot.protectedcount = 10;// set number of levels to protect

break;

case 2:

world.spot.protectedcount = 8;// set number of levels to protect

break;

case 3:

world.spot.protectedcount = 5;// set number of levels to protect

}

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

if (world.mpills[i].baseobject.x == world.spot.baseobject.x && world.mpills[i].baseobject.y == world.spot.baseobject.y) // fix me removing the wrong pill

world.mpills[i].eaten = true; // again needs to be fixed

}

}

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);

}

void savegame(const string &name,const game& world)

{

ofstream writer(name + ".save");

writer << world.spot.baseobject.x << endl;

writer << world.spot.baseobject.y << endl;

writer << world.spot.hascheated << endl;

writer << world.spot.isProtected << endl;

writer << world.spot.lives << endl;

writer << world.spot.protectedcount << endl;

writer << world.spot.score << endl;

writer << world.spot.levelchoice << endl;

writer << world.zombies.size() << endl;

for (const zombie& a : world.zombies)

{

writer << a.baseobject.x << endl;

writer << a.baseobject.y << endl;

writer << a.imobalized << endl;

writer << a.startx << endl;

writer << a.starty << endl;

}

writer << world.pills.size() << endl;

for (const pill& a : world.pills)

{

writer << a.baseobject.x << endl;

writer << a.baseobject.y << endl;

writer << a.eaten << endl;

}

writer << world.mpills.size() << endl;

for (const pill& a : world.mpills)

{

writer << a.baseobject.x << endl;

writer << a.baseobject.y << endl;

writer << a.eaten << endl;

}

writer << world.holes.size() << endl;

for (const Item& a : world.holes)

{

writer << a.x << endl;

writer << a.y << endl;

}

writer << world.inwalls.size() << endl;

for (const Item& a : world.inwalls)

{

writer << a.x << endl;

writer << a.y << endl;

}

}

game loadgame(const string &name)

{

ifstream reader(name + ".save");

player spot = { {SPOT}, name };

reader >> spot.baseobject.x;

reader >> spot.baseobject.y;

reader >> spot.hascheated;

reader >> spot.isProtected;

reader >> spot.lives;

reader >> spot.protectedcount;

reader >> spot.score;

reader >> spot.levelchoice;

int numofzom;

reader >> numofzom;

vector<zombie> zombies;

for (int i = 0; i != numofzom; i++)

{

zombie a = { ZOMBIE };

reader >> a.baseobject.x;

reader >> a.baseobject.y;

reader >> a.imobalized;

reader >> a.startx;

reader >> a.starty;

zombies.push\_back(a);

}

reader >> numofzom;

vector<pill> pills;

for (int i = 0; i != numofzom; i++)

{

pill a = { PILL };

reader >> a.baseobject.x;

reader >> a.baseobject.y;

reader >> a.eaten;

pills.push\_back(a);

}

reader >> numofzom;

vector<pill> mpills;

for (int i = 0; i != numofzom; i++)

{

pill a = { PILL };

reader >> a.baseobject.x;

reader >> a.baseobject.y;

reader >> a.eaten;

mpills.push\_back(a);

}

reader >> numofzom;

vector<Item> holes;

for (int i = 0; i != numofzom; i++)

{

Item a = { HOLE };

reader >> a.x;

reader >> a.y;

holes.push\_back(a);

}

reader >> numofzom;

vector<Item> inwalls;

for (int i = 0; i != numofzom; i++)

{

Item a = { HOLE };

reader >> a.x;

reader >> a.y;

inwalls.push\_back(a);

}

return { spot, zombies, pills, mpills, holes, inwalls };

}

bool isArrowKey(const int key)

{

return ((key == LEFT) || (key == RIGHT) || (key == UP) || (key == DOWN));

}

bool isCheatKey(const int key)

{

return ((toupper(key) == EAT) || (toupper(key) == EXTERMINATE) || (toupper(key) == FREEZ));

}

bool isreplayKey(const int key)

{

return (toupper(key) == REPLAY);

}

bool wantToQuit(const int key, string& message)

{

bool exit = (toupper(key) == QUIT);

if (exit)

message = "you have quit";

return exit;

}

bool haswon(const int zombies, const int pills, string& message, const player& spot)

{

if (zombies > 0)

return false;

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 16);

cout << "Congratulations, you win ";

Gotoxy(40, 17);

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

return true;

}

bool endconditions(const int zombies, const int pills, const player &spot, const int key, string& message)

{

bool haswon(const int 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)));

}

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

{

if (spot.lives == 0)

{

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)

return true;

else

return false;

}

bool issaveKey(const int k)

{

if (toupper(k) == SAVE)

return true;

else

return false;

}

bool isloadKey(const int k)

{

if (toupper(k) == LOAD)

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,const int diff)

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

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

void showLives(const player &spot);

void showDescription();

void showzomLives(const int lives);

void showrempill(const int pils);

void showTitle();

void showOptions();

void showtime();

void showMessage(const string&);

void showname(const string &name);

void showdiftime(const int diff);

void showscore(const int score);

Gotoxy(0, 0);

//display grid contents

paintGrid(gd);

//display game title

showTitle();

showDescription();

showtime();

showLives(spot);

showname(spot.name);

int previousscore = getscore(spot.name);

showscore(previousscore);

showdiftime(diff);

//show number of zombie lives

showzomLives(zombielives);

//show number of remaing pills

showrempill(remainingpill);

//display menu options available

showOptions();

//display message if any

showMessage(mess);

}

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);

break;

case MPILL:

SelectTextColour(clGrey);

}

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

} //end of col-loop

cout << endl;

} //end of row-loop

}

void showzomLives(const int lives)

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 11);

cout << "zombie lives: " << lives;

}

void showrempill(const int pils)

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 10);

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

}

void showDescription()

{

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)

{

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: ";

}

void displayhighscores()

{

ifstream in("best.scr");

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

{

Gotoxy(2, 13);

cout << "name score";

for (int i = 0; i != 3; ++i)

{

int storedscore; // the score

string name; // the name

in >> name;

in >> storedscore;

Gotoxy(2, 14 + i);

cout << name << " " << storedscore;

}

}

else

{

Gotoxy(2, 13);

cout << "bob 3";

Gotoxy(2, 14);

cout << "tom 3";

Gotoxy(2, 15);

cout << "jim 3";

}

in.close();

}

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

{

ifstream in("best.scr");

if (!in.fail())

{

Gotoxy(2, 13);

string name1;

string name2;

string name3;

int score1;

int score2;

int score3;

in >> name1;

in >> score1;

in >> name2;

in >> score2;

in >> name3;

in >> score3;

in.close();

ofstream out("best.scr");

if (score1 < score)

{

score3 = score2;

score2 = score1;

name3 = name2;

name2 = name1;

score1 = score;

name1 = name;

}

else if (score2 < score)

{

score3 = score2;

name3 = name2;

score2 = score;

name2 = name;

}

else if (score3 < score)

{

score3 = score;

name3 = name;

}

out << name1 << endl;

out << score1 << endl;

out << name2 << endl;

out << score2 << endl;

out << name3 << endl;

out << score3 << endl;

out.close();

}

else

{

ofstream out("best.scr");

out << "bob" << endl;

out << 3 << endl;

out << "tom" << endl;

out << 2 << endl;

out << "jim" << endl;

out << 1 << endl;

out.close();

}

}

void showdiftime(const int diff)

{

SelectBackColour(clRed);

SelectTextColour(clYellow);

Gotoxy(40, 16);

cout << "time spent in game: " << diff;

}

# 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** |