

**OOP LAB PROJECT**

**PAC-MAN**

**Group Members**

**ALI RAZA 23F-0542**

**FAIQ ALI 23F-0592**

**Submitted To:**

**Ms.AREEBA WASEEM**

**Source Code:**

Pacshow cpp

#include <iostream>

#include <cstdlib>

#include <conio.h> // For getch()

#include <SFML/Graphics.hpp>

#include "starter.h"

using namespace std;

using namespace sf;

void setConsoleColour(int colourCode) {

// Placeholder for setting console color. Implement this function for your system.

}

int titleScreen() {

system("cls");

string gameName = "PACMAN C++";

setConsoleColour(7);

setConsoleColour(9);

cout << " ############### \n"

<< " ################## \n"

<< " ####################### \n"

<< " ############## ########### \n"

<< "############################## \n"

<< "####################### \n"

<< "#################### \n"

<< "################ \n"

<< "############# \n"

<< "########### \n"

<< "######### \n"

<< "############ ";

setConsoleColour(14);

cout << gameName << endl;

setConsoleColour(9);

cout << "################## \n"

<< "####################### \n"

<< "########################### \n"

<< "############################## \n"

<< " ####################### \n"

<< " #################### \n"

<< " ################ " << endl;

setConsoleColour(14);

cout << " \_\_\_\_\_ \_\_\_\_\_ \_\_ \_\_ \_ \_ \n"

<< " | \_\_ \\ /\\ / \_\_\_\_| | \\/ | /\\ | \\ | |\n"

<< " | |\_\_) | / \\\ | | | \\ / | / \\\ | \\\| |\n"

<< " | \_\_\_/ / /\\ \\\ | | | |\\/| | / /\\ \\\ | |\n"

<< " | | / \_\_\_\_ \\\ | |\_\_\_ | | | | / \_\_\_\_ \\\ | |\\ | \n"

<< " |\_| /\_/ \\\\_\_\\ \\\\_\_\_\_| |\_| |\_| /\_/ \\\\_\\ |\_| \\\\_|" << endl;

cout << "\nPress s to start, q to quit." << endl;

char inp = \_getch();

cout << " ";

switch (inp) {

case 's':

return 1;

case 'q':

exit(0);

default:

return 0;

}

}

int main() {

// Display title screen

int startGame = titleScreen();

if (startGame == 1) {

// Initialize graphical window

int winW = 25 \* 30; // Window width

int winH = 25 \* 30; // Window height

Texture sprTexture;

while (!sprTexture.loadFromFile("texture/PMSprites.png")) {

// Handle texture loading failure if needed

}

RenderWindow window(VideoMode(winW, winH), "Ali Raza & Faiq Game", Style::Default);

window.setActive(false);

Starter\* starter = new Starter(&window, &sprTexture);

// Main graphical window logic

while (window.isOpen()) {

Event event;

while (window.pollEvent(event)) {

if (event.type == Event::Closed) {

window.close();

}

}

window.clear();

// Add rendering and game logic here

window.display();

}

delete starter;

cout << "Closed window." << endl;

system("pause");

}

return 0;

}

Starte.h

#include <iostream>

#include <thread>// standard library

#include<chrono>//for measuring time intervals

#include "pac.h"

#include "ghost.h"

#include "maze.h"

#include "gameText.h"

#include "fruit.h"

using namespace std;

using namespace sf;

class Starter{

public:

Event sfEvt;

Maze maze;//simple class

Pac<Starter>\* pac;

Ghost<Starter> \*Blinky, \*Pinky, \*Inky,\*Clyde;

thread ghStatusThread;

thread loopThread;

int attackInterval=15; // 15 sec

int scaterInterval=7; // 7 sec

int blueInterval = 66;

int delay;

int curTime;

Texture backText;//class of sfml

Texture backFlashText;//class of sfml

Sprite backSpr; //class of sfml

bool intro = false;

bool isCollid = false;

bool lifeWin = false;

bool toNextLevel = false;

RenderWindow\* win;

GameText \*gameText;

Fruit\* fruit;

RenderWindow\* window;

Starter( RenderWindow \*win, Texture\* sprTexture)

{

window = win;

backText.loadFromFile("texture/map2.png");//map of maze is in picture

backFlashText.loadFromFile("texture/map2.png");//this is background just boxes with low intensity

backSpr.setTexture(backText);

maze.initMaze();

pac = new Pac<Starter>(sprTexture,this);//sprite texture and refrence to current object.

Blinky = new Ghost<Starter>(sprTexture, 1,this);//it is 1

Pinky = new Ghost<Starter>(sprTexture, 2,this);//it is 2

Inky = new Ghost<Starter>(sprTexture, 3,this);//it is 3

Clyde = new Ghost<Starter>(sprTexture, 4,this);//it is 4

gameText = new GameText();//goes in the gametext.h file

fruit = new Fruit(sprTexture);

loopThread = thread ( &Starter::loop,this,win );

while (win->isOpen())

{

while (win->pollEvent(sfEvt))

{

if (sfEvt.type == Event::Closed||sf::Keyboard::isKeyPressed(sf::Keyboard::E)) //yaha per ma na close ka liya E ka button add kia ha khud sa

{

win->close();

}

else if (sfEvt.type == Event::KeyPressed)

{

if (sfEvt.key.code == Keyboard::Enter)

{

startGame();

}

pac->rotation(sfEvt.key.code);

}

}

}

};

~Starter()

{

if (loopThread.joinable()) { loopThread.detach(); }//we detach thread not delete as it we alos made .

if (ghStatusThread.joinable()) { ghStatusThread.detach(); }//same as above line

delete Blinky; //clean up

delete Pinky;

delete Inky;

delete Clyde;

}

void drawLife( RenderWindow \*win)

{

if (pac->pacLife < 1) { return; }

Sprite spr;

for (int i = 0; i < pac->pacLife; i++)

{

spr = pac->getLifeSpr();

spr.setPosition( Vector2f(30\*i, 24\*30) );

(\*win).draw( spr );

}

}

void setBlueGhost()

{

if (CntrGame::pacIsDead) { return; }

if (ghStatusThread.joinable()) { ghStatusThread.detach();

}

sleep(milliseconds(20));

ghostStatus = Blue;

creatGhostThr();//starts new thread to keep status blue its def is next

}

void collidToPac()

{

CntrGame::pacIsDead = true;

pac->pacLife--;

isCollid = true;

stopAll();

if (pac->pacLife < 0)

{

gameOver();

}

}

void gameOver()

{

gameStatus = Demo;

CntrGame::score = 0;

CntrGame::level = 0;

lifeWin = false;

gameText->scoreTxt.setString("SCORE: 0");

pac->stop();

Blinky->stop();

Pinky->stop();

Inky->stop();

Clyde->stop();

resetPacGhost();

resetLevel();

}

private:

//---

void loop( RenderWindow\* win)

{

win->setActive(true);

while (win->isOpen())

{

if (CntrGame::score >= 10000 && !lifeWin)

{

lifeWin = true;

pac->pacLife++;

}

win->clear();

if (gameStatus == Play)

{

drawLife(win);

maze.drawWall(win);//draws maze wall

win->draw(backSpr);//background sprite

win->draw(gameText->gameOverTxt);//this print game over

win->draw(gameText->scoreTxt);//this score

}

win->draw(pac->getSprite());

win->draw(Blinky->getSprite());

win->draw(Pinky->getSprite());

win->draw(Inky->getSprite());

win->draw(Clyde->getSprite());

win->display();

}

}

//---

void changeGhostState()

{

delay = scaterInterval;

if (ghostStatus == Blue)

{

delay = blueInterval;

CntrGame::isBlueGhost = true;

}

changeStatus();

while ( ghStatusThread.joinable() )

{

curTime = time(0);

curTime += delay;

while (true && ghStatusThread.joinable() )

{

if (curTime <= time(0)) {

break;

}

}// wait for change ghost status

sleep(milliseconds(10));

if (ghostStatus == Blue)

{

CntrGame::isBlueGhost = false;

CntrGame::ghostBonus = 100;

}

if (ghostStatus == Attack)

{

ghostStatus = Scater;

delay = scaterInterval;

}

else

{

ghostStatus = Attack;

delay = attackInterval;

}

changeStatus();

}

}

//---

void changeStatus()

{

Blinky->changeGhostState();

Pinky->changeGhostState();

Inky->changeGhostState();

Clyde->changeGhostState();

}

//---

void stopAll()

{

pac->stop();

Blinky->stop();

Pinky->stop();

Inky->stop();

Clyde->stop();

CntrGame::gameRun = false;

gameText->stopThread();

fruit->stop();

if (ghStatusThread.joinable() ){ ghStatusThread.detach(); }

resetLevel();

wait(2);

if (pac->pacLife >= 0) {

startLevel(); }

}

//---

void startGame()

{

maze.redrawDot();

gameText->gameOverTxt.setString("PAC-MAN");

pac->pacLife = 2;

gameStatus = Play;

resetPacGhost();

intro = true;

CntrGame::gameRun = true;

CntrGame::level=1;

CntrGame::score=0;

CntrGame::dotsEat = 0;

blueInterval = 7;

fruit->setLevel(CntrGame::level);

startLevel();

}

//---

void startLevel()

{

if (intro)

{

intro = false;

}

isCollid = false;

ghostStatus = Scater;

CntrGame::gameRun = true;

resetPacGhost();

pac->run();

creatGhostThr();

fruit->start();

}

//---

void creatGhostThr()

{

while (ghStatusThread.joinable())

{

}

ghStatusThread = thread(&Starter::changeGhostState, this);

}

//---

void resetLevel()

{

CntrGame::ghostBonus = 100;

if(pac->pacLife<0)

{

gameText->gameOverTxt.setString("Your Bad Ali"); //final print

}

}

//---

void resetPacGhost()

{

pac->reset();

Blinky->reset();

Pinky->reset();

Inky->reset();

Clyde->reset();

}

void wait(int delayInt)

{

auto curTime = time(0);

int counter=0;

curTime += delayInt;

while (true)

{

counter++;

if (toNextLevel)

{

if (counter % 30 == 0)

{

backSpr.setTexture(backFlashText);

}

else if (counter % 30 == 15)

{

backSpr.setTexture(backText);

}

}

if (curTime < time(0)) {

break;

}

sleep(milliseconds(10));

}

toNextLevel = false;

backSpr.setTexture(backText);

}

};

Target Cal.h

#pragma once

#include <iostream>

#include <string>

#include <SFML/Graphics.hpp>

using namespace std;

//---

enum GhostDirection

{

UP, RIGHT, DOWN, LEFT, NONE

};

class TargetCalc

{

public:

int targetX, targetY;

TargetCalc()

{

}

~TargetCalc() {}

Vector2f getTargetPoint()

{

return Vector2f(targetX/30, targetY/30);//(likely pixels) to another (grid units)

}//its result print the new coordints i.e location

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

//

// Ghosts target calculation Most important part of the game

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

GhostDirection getToTargetDir(int ghId,sf::Vector2f ghostPos,sf::Vector2f blinkyPos,

sf::Vector2f targetPos, sf::Vector2f edgePos, PacDirection pacDir,const char\* crossCh)

{

targetOfsX = 0;

targetOfsY = 0;

targetPos = (Vector2f(targetPos.x - 15, targetPos.y - 15));

if (ghId == 2 || ghId==3) // Pinky Inky

{

switch (pacDir)

{

case Up:

targetOfsY = -4;//for pinky

if (ghId == 3) { targetOfsY = -2; }//for inky

break;

case Right:

targetOfsX = 4;//for pinky

if (ghId == 3) { targetOfsX = 2; }//for inky

break;

case Down:

targetOfsY = 4;

if (ghId == 3) { targetOfsY = 2; }

break;

case Left:

targetOfsX = -4;

if (ghId == 3) { targetOfsX = -2; }//for left subtract

break;

default:

break;

}

}

targetOfsX \*= 30;//(e.g., 30x30 pixels)

targetOfsY \*= 30;//converts the grid units to pixels

if (ghId == 3) // Inky

{

targetOfsX += ( targetPos.x-blinkyPos.x) ;

targetOfsY += ( targetPos.y-blinkyPos.y) ;

}

// targetX = 100 + 60 = 160

// targetY = 150 - 30 = 120

targetX = targetPos.x +targetOfsX;

targetY = targetPos.y +targetOfsY;

if (noCondition(crossCh)) {

return NONE;

}

int dX = abs(ghostPos.x - (targetPos.x+ targetOfsX) );

int dY = abs( ghostPos.y -(targetPos.y+targetOfsY) );

string str = string(1, \*crossCh); //to\_string(\*crossCh);

if (ghId == 4)// Clyde

{

if (ghostStatus == Attack)

{

if (dX/30 < 8 && (dY/30) < 8) {targetPos = edgePos; }

//CntrGame class manages the state

else {targetPos = (\*CntrGame::pacPosition);}

}

}

if (dX > dY)

{

if ( ghostPos.x > targetPos.x)

{//varable of the string

if ( str != "D") {

return LEFT;

}

else if ( ghostPos.y > targetPos.y )

{ return UP ;

}

else { return DOWN; }

}

else

{

if ( str != "F") {

return RIGHT;

}

else if (ghostPos.y > targetPos.y)

{

return UP;

}

else {

return DOWN;

}

}

}

else

{

if ( ghostPos.y > targetPos.y)

{

if ( str != "S" && str != "G") { return UP; }

else if ( ghostPos.x < targetPos.x) { return RIGHT; }

else { return LEFT; }

}

else

{

if ( str != "A" && str != "G") { return DOWN; }

else if (ghostPos.x < targetPos.x) { return RIGHT; }

else { return LEFT; }

}

}

return NONE;

}

private:

int targetOfsX = 0;

int targetOfsY = 0;

//---

bool noCondition(const char\* ch)

{//this indicates no movement of ghost if ture

return \*ch == 'B' || \*ch == 'P' || \*ch == 'I' || \*ch == 'C' ||

\*ch == '0' || \*ch == '=' || \*ch == '1' || \*ch == '@' ||

\*ch == '#' || \*ch == '-' || \*ch == ' ' || \*ch == 'c';

}

};

**Pacshow.h**

#include <thread>

#include "cntrgame.h"

using namespace std;

enum PacDirection

{

Up,Right,Down,Left

} pacDirection,pacPrevDirection;//These are variables of type PacDirection.

template <class C>

class Pac : public CntrGame

{

public:

C\* starter;

int pacLife = 3;

//sf::texture class represents the image loaded from a file

//constructor takes 2 pointer to objects

Pac(sf::Texture\* texture,C \*\_starter)

{

starter = \_starter;

pacSpr.setTexture(\*texture);

init(); //function of pac class to start game thread

};

//getter functions to access the private members of the class

sf::Sprite getSprite()

{

return pacSpr;

};

//---

sf::Sprite getLifeSpr()

{

return lifeSpr;

}

//---

void rotation(sf::Keyboard::Key key)//var representing using enum

{

if (checkKeyPress(key))

{

move(key); //fucntion chek key press adn act accordingly

}

}

~Pac()

{//allow to run independently

if (loopThread->joinable()) { loopThread->detach(); }

}

//---

void reset()

{

CntrGame::pacIsDead = false;

pacAnim = moveX = moveY = 0;//this show animation state or progress of pac

pacRot = 0; //static member floating values

forwX = 1, forwY = 0;

pacDirection = pacPrevDirection = Right;

if (gameStatus == Demo)

{

//calculte the positon in pixels MENUE PAGE

setGridPosition(demoPos[1][1], demoPos[0][1]);

pacDirection = pacPrevDirection =Right;

}

else

{

setGridPosition(startX, startY);

}

pacSpr.setPosition(pacPos);

stop();

}

private:

thread\* loopThread;

float pacAnim = 0;

float pacRot = 0;

//members variables of pac class of sprite type which pick the picture part

sf::Sprite pacSpr;

sf::Sprite lifeSpr;

int forwX = 1,forwY=0;//coordintates fo rpicures

int startX = 11;

int startY = 18;

//---

void init()

{ //width and hieght of the sprite

sprW = 30; sprH = 30; sprOfsX = 0; sprOfsY = 0;

//Speed of pacman you can easilty change from here

speed = 3;

pacSpr.setOrigin(sprW / 2, sprH / 2);//set the origin of the pacman sprite to its centre

pacSpr.setTextureRect(sf::IntRect(sprOfsX, sprOfsY, sprW, sprH));

// lifeSpr.setTextureRect(sf::IntRect(30, 0, sprW, sprH));

//lifeSpr.scale(.8,.8);

reset();

loopThread = new thread([this]() {loop(); });

}

//---

void loop()

{

while (true)

{

sleep( milliseconds (delayLoop ) );

pacAnim+=.4;

if (pacAnim > 3) { pacAnim = 0; }

animation();

}

}

//---

//void move(sf::Keyboard::Key key)

//{

// if (gameStatus == Demo) { return; }

// //forwX = forwY = 0;

// switch (key)

// {

// case sf::Keyboard::Up :

// pacDirection = Up;

// break;

// case sf::Keyboard::Right:

// pacDirection = Right;

// break;

// case sf::Keyboard::Down:

// pacDirection = Down;

// break;

// case sf::Keyboard::Left:

// if (pacRot==std::abs(180) ) { return; }

// pacDirection = Left;

// break;

// }

//}

//

////---

//void direction(PacDirection direct)

//{

// forwX = forwY = 0;

// switch (direct)

// {

// case Up:

// moveY = -speed; moveX = 0;

// forwY = -1;

// pacRot = -90;

// break;

// case Right:

// moveY = 0; moveX = speed;

// forwX = 1;

// pacRot = 0;

// break;

// case Down:

// moveY = speed; moveX = 0;

// forwY = 1;

// pacRot = 90;

// break;

// case Left:

// moveY = 0; moveX = -speed;

// forwX = -1;

// pacRot = 180;

// break;

// }

// if (pacRot != pacSpr.getRotation())

// {

// pacRot -= pacSpr.getRotation();

// setRotation(pacRot);

// }

//}

////---

//bool checkKeyPress(sf::Keyboard::Key key)

//{

// return key == sf::Keyboard::Up ||

// key == sf::Keyboard::Right ||

// key == sf::Keyboard::Down ||

// key == sf::Keyboard::Left;

//}

void move(sf::Keyboard::Key key)

{

switch (key)

{

case sf::Keyboard::W: // W key for moving up

pacDirection = Up;

break;

case sf::Keyboard::D: // D key for moving right

pacDirection = Right;

break;

case sf::Keyboard::S: // S key for moving down

pacDirection = Down;

break;

case sf::Keyboard::A: // A key for moving left

if (pacRot == std::abs(180)) { return; }

pacDirection = Left;

break;

}

}

void direction(PacDirection direct)

{

forwX = forwY = 0;

switch (direct)

{

case Up: // Movement upward (W)

moveY = -speed; moveX = 0;

forwY = -1;

pacRot = -90;

break;

case Right: // Movement to the right (D)

moveY = 0; moveX = speed;

forwX = 1;

pacRot = 0;

break;

case Down: // Movement downward (S)

moveY = speed; moveX = 0;

forwY = 1;

pacRot = 90;

break;

case Left: // Movement to the left (A)

moveY = 0; moveX = -speed;

forwX = -1;

pacRot = 180;

break;

}

if (pacRot != pacSpr.getRotation())

{

pacRot -= pacSpr.getRotation();

setRotation(pacRot);

}

}

bool checkKeyPress(sf::Keyboard::Key key)

{

// Update to check for W, A, S, D keys

return key == sf::Keyboard::W ||

key == sf::Keyboard::D ||

key == sf::Keyboard::S ||

key == sf::Keyboard::A;

}

//---

void setRotation(float rot)

{

pacSpr.rotate(rot);//use to represent rotation value

}

//---

int getPosGridX()

{//current grid cell of the pacman in x direction

return (pacSpr.getPosition().x ) / sprW;//width

}

//---

int getPosGridY()

{//current grid cell of the pacman in y direction

return (pacSpr.getPosition().y ) / sprH;

}

//---

void animation()

{

if (getPosGridX() > 21)

{

setGridPosition(0, 11);

}

else if (getPosGridX() < 0)

{

moveX = 0;

setGridPosition(21, 11);

}

if (xyModul())

{

direction(pacDirection);

if (collidWall())

{

if (pacDirection == pacPrevDirection)

{

moveX = moveY = 0;

pacPrevDirection = pacDirection;

}

else

{

direction(pacPrevDirection);

}

}

pacPrevDirection = pacDirection;

}

//Movement and collision detection

if ( xyModul() && collidWall() )

{

moveX = moveY = 0;//stop the movement of pacman

}

else if (!stopMove)//if the movement is not stopped

{

pacPos.x += moveX;

pacPos.y += moveY;

pacSpr.setPosition(pacPos);

if (xyModul() )

{

eatDot(getPosGridX(),getPosGridY());

}

}

// fruit checking of powerpallet and bonnus

if (getMazeStr(getPosGridX(), getPosGridY() )=="1" && starter->fruit->getVisible() )

{

int bonus = 100 ;

setScore(bonus);

starter->fruit->setVisible(false);

}

//Updating animation state of pacman

sprOfsX = (sprW \* (int)pacAnim);//horizantal offset

pacSpr.setTextureRect(sf::IntRect(sprOfsX, sprOfsY, sprW, sprH));

CntrGame::pacPosition=&pacPos;

}

//---updates score and turning ghosts blue)

void eatDot(int pacX, int pacY)

{

for (int i = 0; i < maze->dotsCount; i++)

{

sf::CircleShape dot = maze->dotsArr[i];

sf::Vector2f pos = dot.getPosition();

int dotX = (int)(pos.x / sprW);//grid coordinates

int dotY = (int)(pos.y / sprH);

if (dotX == pacX && dotY == pacY && !starter->isCollid)

{

if (dot.getFillColor() != sf::Color::Black)

{//if it is not eaten

if (dot.getRadius() == 10)// big dot

{

CntrGame::score += 20;

if (starter->blueInterval > 0)

{

CntrGame::isBlueGhost = true;

starter->setBlueGhost();

}

}

dot.setFillColor(sf::Color::Black);

maze->dotsArr[i] = dot;

CntrGame::dotsEat++;

setScore(dotBonus);

break;

}

}

}

}

//---//display score on the screen

void setScore(int score)

{

CntrGame::score += score;

starter->gameText->scoreTxt.setString("SCORE: " + to\_string(CntrGame::score));

}

//---

bool collidWall()

{

return checkMazeGrid(getPosGridX() + forwX, getPosGridY() + forwY);

}

//---

bool xyModul()//allignment in grid check

{

modulX = (int)pacSpr.getPosition().x % sprW;

modulY = (int)pacSpr.getPosition().y % sprH;

return modulX == 15 && modulY == 15;

}

//---

void stopLoop()

{

loopThread->detach();

}

//---

void setGridPosition(int x, int y)

{

pacPos = sf::Vector2f((x \* 30) + (sprW / 2), (y \* 30) + (sprH / 2));

}

};

Maze.h

#pragma once

#include <iostream>

#include <string>

#include <SFML/Graphics.hpp>

using namespace std;

using namespace sf;

class Maze

{

public:

static Maze\* instance;

string mazePattern[25] =

{ "#######################" , // 0 // #= Bricks

"#Q S W#Q S W#" , // 1 // X= Ghosts direction condition

"#@### #### # #### ###@#" , // 2 // c= Pacman char

"# ### #### # #### ### #" , // 3 // Q=RD

"#D X S A A S X F#" , // 4 // W=LD

"# ### # ####### # ### #" , // 5 // E=RU

"#E F# ####### #D R#" , // 6 // R=LU

"##### #E W#Q R# #####" , // 7 //-------

"##### #### # #### #####" , // 8 // A=LRU

"##### #Q AGA W# #####" , // 9 // S=LRD

"##### # ###-### # #####" , // 12 // D=UDR

"0====X F#BP\*IC#D X====0" , // 11 // F=UDL

"##### # ####### # #####" , // 12 // G=LR

"##### #D0001000F# #####" , // 13 // 1=Fruit location

"##### # ####### # #####" , // 14 // L=Low speed in tunel for ghosts

"##### # ####### # #####" , // 15 // BPIC=Ghosts char

"#Q X A W#Q A X W#" , // 16

"# ### #### # #### ### #" , // 17

"#E W#D S AcA S F#Q R#" , // 18

"### # # ####### # # ###" , // 19

"#Q A R#E W#Q R#E A W#" , // 20

"# ######## # ######## #" , // 24

"#@######## # ########@#" , // 22

"#E A A R#" , // 23

"#######################" };// 24

//// 01234567890123456789012

//

//# = Bricks

//@ = Big dot

//0 = empty space or path

//= = tunnel or short path

//B = Blinky

//P = Pinky

//I = Inky

//C = Clyde

//\* = Big power pallet

//1 = specific location of the Powerpallet location

//• Q: Right-Down (RD)

/\*• W : (LD)

E : (RU)

R : (LU)

D : (UDR)

F : (UDL)

A : (LRU)

S : (LRD)

G : (LR)

\*/

string mazeGrid[23][25];

CircleShape dotsArr[222]; // 222 dots

int dotsCount = 0;

Maze()

{

instance = this;

}

~Maze()

{

}

//----

void initMaze()

{

int colum = 0; int row = 0;

for (int i = 0; i < (Maze::mazeW\*Maze::mazeH) ; i++)

{ //total positions

if (colum >= Maze::mazeW)

{

colum =0;

row++;

}//extract single character from maze

string str = mazePattern[row].substr(colum, 1);

mazeGrid[colum][row] = str;

if (str == "#")

{

// wallShapeArr[wallShapeCount] = getWallShape(colum,row);

wallShapeCount++;

}

else

{// check if dot is this

if (isDot(str) )

{

dotsArr[dotsCount] = getDots(colum, row,str);

dotsCount++;

}

}

colum++;

}

}

//---

void drawWall( RenderWindow \*window )

{

int i = 0;

for ( i = 0; i < wallShapeCount; i++)

{

// window->draw (wallShapeArr[i]);

if (i < dotsCount)// draw dots with this condition

{

window->draw(dotsArr[i]);

}

}

}

//

void redrawDot()

{

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

{ //Dots colour is controlled here

CircleShape dot = dotsArr[i];

dot.setFillColor(Color::Green);

dotsArr[i] = dot;

}

}

private:

sf::RectangleShape wallShapeArr[23\*24];

static int mazeW, mazeH;

int wallShapeCount = 0;

//---

bool isDot(string str)//means @

{

return str != "0" && str != "=" &&

str != "B" && str != "P" &&

str != "I" && str != "C" &&

str != "G" && str != "\*" &&

str != "1" && str != "-" && str!="c";

}

//---

RectangleShape getWallShape(int col,int row)

{

RectangleShape rectangle;

//vector2f class is specify width and height

rectangle.setSize(sf::Vector2f(26, 26));// 30 28

rectangle.setFillColor(Color::Green);

rectangle.setOutlineColor(sf::Color::Green);

rectangle.setOutlineThickness(1.);

rectangle.setPosition(col\*30, row\*30);

return rectangle;

}

//--- circle shape of dot.

CircleShape getDots(int col, int row, string str)

{

float radius = 3; int offset = 12;

Color color = Color::Green;

if (str == "@")

{

radius = 10; offset = 6;

}

CircleShape \_dot;

\_dot.setRadius(radius);

\_dot.setFillColor(color);

\_dot.setPosition(col \* 30 + offset, row \* 30 + offset);

return \_dot;

}

};

Maze \*Maze::instance;

int Maze::mazeW = 23;

int Maze::mazeH = 25;

Ghost.h

#pragma once

#include <SFML/Graphics.hpp>

#include <thread>

#include <iostream>

#include "targetCalc.h"

#include "gameText.h"

using namespace std;

using namespace sf;

template <class C>

class Ghost : public CntrGame

{

public:

C\* starter;

Ghost(Texture\* texture, int id, C\* \_starter)

{

Id = id;

Name = nameArr[id];

starter = \_starter;

ghostSpr.setTexture(\*texture);

cout << "Ghost class OK Name: " << Name << endl;

init();

}

~Ghost()

{

if (loopThread->joinable()) {

loopThread->detach();

}

}

//---

Sprite getSprite()

{

return ghostSpr;

};

//---

Text getNameTxt()

{

return ghostNameTxt;

}

//---

void reset()

{

ghostAnim = moveX = moveY = 0;

speed = 2;

if (gameStatus == Demo)

{

setGridPosition(demoPos[Id][0], demoPos[Id][1]);

ghostNameTxt.setPosition(Vector2f(demoPos[Id][0] \* 40, demoPos[Id][1] \* 30));

direction(GhostDirection::DOWN);

}

else

{

goHome();

}

thisGhostStatus = Scater;

stop();

}

private:

Sprite ghostSpr;

GameText gameText;

Text ghostNameTxt;

TargetCalc targetCalc;

GhostDirection currGhostDir;

thread\* loopThread;

sf::Vector2f ghostPos;

float ghostAnim = 0;

int startX;

int startY;

int sprOfsXX = 0;

int startPosArr[4][2] = { {9,11},{10,11},{12,11},{13,11} };

int edgePosArr[4][2] = { {-2,-5},{22,-2},{3,19},{18,19} };

string nameArr[5] = { "","Blinky","Pinky","Inky","Clyde" };

//---

void init()

{

startX = startPosArr[Id - 1][0];

startY = startPosArr[Id - 1][1];

edgePos = sf::Vector2f(edgePosArr[Id - 1][0] \* 30, edgePosArr[Id - 1][1] \* 30);

targetPos = &edgePos;

sprW = 30; sprH = 30; sprOfsX = 0; sprOfsY = 30 + sprW \* (Id - 1);

ghostNameTxt = gameText.getGhostNameTxt(Name);

reset();

ghostSpr.setTextureRect(sf::IntRect(sprOfsX, sprOfsY, sprW, sprH));

loopThread = new thread([this]() {loop(); });

}

//---

void loop()

{

while (true)

{

sleep(milliseconds(delayLoop));

animation();

if (!stopMove )

{

if (xyModul())

{

checkCross();

}

ghostPos.x += moveX; ghostPos.y += moveY;

}

ghostAnim += .1;

if (ghostAnim > 2) { ghostAnim = 0; }

}

}

//---

void checkCross()

{

if (getPosGridX() > 21)

{

setGridPosition(0, 11);

}

else if (getPosGridX() < 0)

{

setGridPosition(21, 11);

}

string str = getMazeStr(getPosGridX(), getPosGridY());

if (str == "=" || thisGhostStatus == Blue) {

delayLoop = 20;

}

else {

delayLoop = 15;

}

if (str == "-") {

isInHome = false;

}

else if (str == "#")

{

cout <<"Ghost error stoped. Name = " << Name << endl;

stop();

}

const char\* ch = str.c\_str();

GhostDirection dir = getDirection(ch,currGhostDir);

if (dir == NONE)

{

//Blinky position for calculate Inky target

if (Id == 1) {

CntrGame::blinkyPosition = ghostPos;

}

dir = targetCalc.getToTargetDir(Id, ghostSpr.getPosition(),CntrGame::blinkyPosition,

\*targetPos, edgePos, pacDirection, ch );

if (dir == NONE) { dir = currGhostDir; }

}

direction(dir);

}

//---

GhostDirection getDirection(const char \*crossCh,GhostDirection currDir)

{

switch (\*crossCh)

{

case '\*':

return UP;

break;

case 'Q': // RD

if (currDir == LEFT) {

return DOWN;

}

else {

return RIGHT;

}

break;

case 'W': // W=LD

if (currDir == RIGHT) {

return DOWN;

}

else {

return LEFT;

}

break;

case 'E': // E=RU

if (currDir == LEFT) {

return UP;

}

else {

return RIGHT;

}

break;

case 'R': // R=LU

if (currDir == RIGHT) {

return UP;

}

else {

return LEFT;

}

break;

}

return NONE;

}

//---

void direction(GhostDirection direct)

{

switch (direct)

{

case UP:

moveY = -speed; moveX = 0;

break;

case RIGHT:

moveY = 0; moveX = speed;

break;

case DOWN:

moveY = speed; moveX = 0;

break;

case LEFT:

moveY = 0; moveX = -speed;

break;

}

currGhostDir = direct;

}

//---

void animation()

{

if (thisGhostStatus != Blue)

{

switch (currGhostDir)

{

case UP:

sprOfsXX = sprW \* 4;

break;

case RIGHT:

sprOfsXX = 0;

break;

case DOWN:

sprOfsXX = sprW \* 2;

break;

case LEFT:

sprOfsXX = sprW \* 6;

break;

}

}

else

{

sprOfsY = 5 \* sprH;

sprOfsXX = 0;

}

sprOfsX = (sprW \* (int)ghostAnim) + sprOfsXX;

ghostSpr.setTextureRect(sf::IntRect(sprOfsX, sprOfsY, sprW, sprH));

ghostSpr.setPosition(ghostPos);

if (gameStatus == Play && !CntrGame::pacIsDead)

{

if (collidToPac())

{

if (CntrGame::isBlueGhost)//thisGhostStatus==Blue)

{

if (CntrGame::ghostBonus > 1600)

{

CntrGame::ghostBonus = 100;

}

CntrGame::ghostBonus \*= 2;

CntrGame::score += CntrGame::ghostBonus;

starter->gameText->showBonus(CntrGame::ghostBonus, ghostSpr.getPosition(),false);

starter->gameText->scoreTxt.setString("SCORE: "+to\_string(CntrGame::score));

goHome();

}

else { starter->collidToPac(); }

cout << "Collided to Pac " << endl;

}

}

if (getPosGridX()>23 || getPosGridX()<-5 ||

getPosGridY()>24 || getPosGridY()<-5

)

{

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\* Ghost out of Maze. Name=" << Name<<endl;

cout << "isInHome=" << isInHome << endl;

cout << "Direction=" << currGhostDir << endl;

cout << "Status=" << thisGhostStatus << endl;

stop();

goHome();

}

}

//---

void goHome()

{

setGridPosition(startX, startY);

isInHome = true;

if (Id < 3) { direction(GhostDirection::RIGHT); }

else

{

{direction(GhostDirection::LEFT); }

}

sprOfsY = 30 + sprW \* (Id - 1);

targetPos = &edgePos;

thisGhostStatus = Scater;

stop();

}

//---

void setGridPosition(int x, int y)

{

ghostPos = sf::Vector2f((x \* sprW) , (y \* sprH) );

}

//---

int getPosGridX()

{

return (ghostSpr.getPosition().x) / sprW;

}

//---

int getPosGridY()

{

return (ghostSpr.getPosition().y) / sprH;

}

//---

bool collidToPac()

{

int dx = abs((ghostSpr.getPosition().x+sprW/2) - CntrGame::pacPosition->x);

int dy = abs((ghostSpr.getPosition().y+sprH/2) - CntrGame::pacPosition->y);

return dx < sprW/2 && dy < sprH/2;

}

//---

bool xyModul()

{

modulX = (int)ghostSpr.getPosition().x % sprW;

modulY = (int)ghostSpr.getPosition().y % sprH;

return modulX == 0 && modulY == 0 ;

}

};

Gametext.cpp

#include "gameText.h"

// Constructor

GameText::GameText() {

fnt.loadFromFile("font/comicbd.ttf");

setText();

}

// Destructor

GameText::~GameText() {

if (bonusThr.joinable()) {

bonusThr.detach();

}

}

// Show bonus text

void GameText::showBonus(int bonus, Vector2f pos, bool fruit) {

if (fruit) {

bonusTxt.setFillColor(Color::Green);

}

else {

bonusTxt.setFillColor(Color::Blue);

}

bonusTxt.setString(to\_string(bonus));

bonusTxt.setPosition(pos);

curTime = time(0);

curTime += delay;

totalTime = &curTime;

sleep(milliseconds(100));

if (!bonusThr.joinable()) {

bonusThr = thread(&GameText::show, this);

}

}

// Stop bonus thread

void GameText::stopThread() {

if (bonusThr.joinable()) {

bonusThr.detach();

}

}

// Get ghost name text

Text GameText::getGhostNameTxt(string name) {

Text\* txt = new Text();

txt->setFont(fnt);

txt->setString(name);

txt->setFillColor(Color::White);

txt->setCharacterSize(25);

return \*txt;

}

// Internal show method

void GameText::show() {

// Placeholder for functionality

}

// Initialize texts

void GameText::setText() {

// Score text

scoreTxt.setFont(fnt);

scoreTxt.setString("SCORE:");

scoreTxt.setFillColor(Color::Green);

scoreTxt.setCharacterSize(27);

scoreTxt.setPosition(getGridPosXY(15, 0));

// Game Over text

gameOverTxt.setFont(fnt);

gameOverTxt.setString("");

gameOverTxt.setCharacterSize(35);

gameOverTxt.setFillColor(Color::Green);

gameOverTxt.setOutlineColor(Color::White);

gameOverTxt.setOutlineThickness(1);

gameOverTxt.setPosition(getGridPosXY(8, 14));

}

// Convert grid to position

Vector2f GameText::getGridPosXY(float x, float y) {

return Vector2f(x \* 30, y \* 30);

}

Gametext.h

#pragma once

#include <iostream>

#include <SFML/Graphics.hpp>

#include <thread>

#include <chrono>

#include <ctime>

using namespace std;

using namespace sf;

class GameText {

public:

Font fnt;

Text enterTxt;

Text gameOverTxt;

Text scoreTxt;

Text bonusTxt;

Text levelTxt;

Text countTxt;

Text targetPointTxt;

RenderWindow\* window;

thread bonusThr;

int\* totalTime;

int curTime;

int delay = 2; // seconds

GameText(); // Constructor

~GameText(); // Destructor

void showBonus(int bonus, Vector2f pos, bool fruit); // Show bonus text

void stopThread(); // Stop bonus thread

Text getGhostNameTxt(string name); // Get ghost name text

private:

void show(); // Internal show method

void setText(); // Initialize texts

Vector2f getGridPosXY(float x, float y); // Convert grid to position

};

Fruitdef.cpp

#include"fruit.h"

#include <iostream>

#include <SFML/Graphics.hpp>

#include <thread>

using namespace std;

using namespace sf;

Fruit::Fruit(Texture\* texture)

{

fruitSpr.setTexture(\*texture);

setGridPos(11, 13);

}

Fruit::~Fruit() {}

//---

Sprite Fruit::getSrpite()

{

return fruitSpr;

}

//---

void Fruit::setVisible(bool vis)

{

visible = vis;

}

//---

void Fruit::start()

{

if (!fruitThr.joinable())

{

fruitThr = thread(&Fruit::loop, this);

//Fruit thread start

}

}

//---

void Fruit::stop()

{

if (fruitThr.joinable())

{

fruitThr.detach();

//Fruit thread stop

}

}

//---

void Fruit::setLevel(int lvl)

{

switch (lvl)

{

case 1:

level = 1;

break;

default:

level = lvl;

break;

}

}

//---

bool Fruit::getVisible() { return visible; }

//---

Vector2f Fruit::getPosition() { return fruitSpr.getPosition(); }

//---

void Fruit::loop()

{

curTime = time(0);

curTime += delay;

while (fruitThr.joinable())

{

if (curTime < time(0))

{

setVisible(!visible);

curTime = time(0);

curTime += delay;

}

sleep(milliseconds(10));

}

setVisible(false);

}

//---

void Fruit::setGridPos(int x, int y)

{//ir places fruit on the grid specific part and each is 30\*30 PIXELS

fruitSpr.setPosition(Vector2f(x \* 30, y \* 30));

}

Fruit.h

#include <iostream>

#include <SFML/Graphics.hpp>

#include <thread>

using namespace std;

using namespace sf;

class Fruit

{

public:

Fruit(Texture\* texture);

~Fruit();

//---

Sprite getSrpite();

//---

void setVisible(bool vis);

//---

void start();

//---

void stop();

//---

void setLevel(int lvl);

//---

bool getVisible();//it shows wheter fruit is visible or not.

//---

Vector2f getPosition();

private:

Sprite fruitSpr;

int ofsX=0, ofsY=6\*30;

int level=0;

int delay = 30;// wait 30 sec for visible fruit

bool visible = false;

int curTime;

thread fruitThr;

//---

void loop();

//---

void setGridPos(int x, int y);

};

Game centr.h

#pragma once

#include <iostream>

#include <SFML/Graphics.hpp>

#include <map>

#include "maze.h"

using namespace std;

//......................................................

// The CntrGame class manages the state and behavior of a game,

// including ghost statuses, player positions,

// game dimensions, and interactions with the game maze.

//

// ...................................................

//--

enum GhostStatus

{

Attack,

Scater,

Blue,

Death

}ghostStatus;

//---

enum GameStatus

{

Demo,Play

}gameStatus;

class CntrGame

{

public:

//--

enum GhostStatus

{

Attack,

Scater,

Blue,

Death

}thisGhostStatus;

static int winW,winH;

static sf::Vector2f \*pacPosition;

static sf::Vector2f blinkyPosition;

static bool pacIsDead;

static bool isBlueGhost;

static int ghostBonus;

static int score;

static int level;

static int dotsEat;

static bool gameRun;

sf::Vector2f pacPos;

int sprW = 0, sprH = 0, sprOfsX = 0, sprOfsY = 0;

int demoPos[5][2] = { {6,11}, {6,6},{6,7},{6,8},{6,9} };

int modulX, modulY;

int dotBonus=10;

float moveX = 0, moveY = 0;

float speed=0;

bool stopMove = false;

bool isInHome;

sf::Vector2f \*targetPos;

sf::Vector2f edgePos;

sf::VideoMode videoMode;

//speed of pacman

int delayLoop = 15;

string Name;

int Id;

Maze \*maze;

CntrGame()

{

maze = Maze::instance;

winW = videoMode.getDesktopMode().width;

winH = videoMode.getDesktopMode().height;

}

~CntrGame()

{

}

//---run time polymorphism

bool virtual checkMazeGrid(int x, int y)

{

string str = maze->mazeGrid[x][y];

return str == '#' || str=="-";

}

//---

void changeGhostState()

{

switch (ghostStatus)

{

case Attack:

thisGhostStatus = Attack;

break;

case Scater:

thisGhostStatus = Scater;

break;

case Blue:

thisGhostStatus = Blue;

break;

case Death:

thisGhostStatus = Death;

break;

}

if (Id == 1 && thisGhostStatus == Scater)// for Blinky

{

if ((maze->dotsCount - CntrGame::dotsEat) < 30)

{

thisGhostStatus = Attack;

}

}

if ( thisGhostStatus == Attack)

{

targetPos = CntrGame::pacPosition;

}

else if (thisGhostStatus == Scater || thisGhostStatus==Blue) { targetPos = &edgePos; }

if (thisGhostStatus != Blue)

{

sprOfsY = 30 + sprW \* (Id - 1);

}

if (isInHome && CntrGame::gameRun ) { run(); }

}

//---

string getMazeStr(int x, int y)

{

return maze->mazeGrid[x][y];

}

//---

bool virtual checkWall(sf::Vector2f sprPos)

{

return (sprPos.x) + sprW / 2 >= CntrGame::winW ||

(sprPos.y) + sprH/2 >= CntrGame::winH ||

(sprPos.x) < (sprW/2) ||

(sprPos.y) < (sprH/2);

}

//---

void run()

{

stopMove = false;

}

//---

void stop()

{

moveX = moveY = 0;

stopMove = true;

}

//---

void endGame()

{

}

private:

};

int CntrGame::winW=800;

int CntrGame::winH=800;

int CntrGame::ghostBonus = 100;

int CntrGame::score=0;

int CntrGame::level = 0;

int CntrGame::dotsEat = 0;

bool CntrGame::pacIsDead = false;

bool CntrGame::isBlueGhost = false;

bool CntrGame::gameRun = false;

sf::Vector2f \*CntrGame::pacPosition=0;

sf::Vector2f CntrGame::blinkyPosition;