Pac-Man

Amr Eid - 900222213
Ismail ElDahshan - 900221719
Kareem ElNaghy - 900223587
Mohamed Azouz - 900225230

CSCE2211
Prof. Amr Goneid

December 11, 2023

Abstract

This project presents the implementation of a Pac-Man game using the C++ programming language and the Qt Creator framework. This project aims to create a visually appealing Pac-Man game using the robust features offered by the Qt Creator framework. This project uses the beauty of shortest path-finding algorithms to move game elements on the map. Algorithms like A* algorithm and Dijkstra were used to help show the four ghosts in the game their path to Pac-Man. In this project, the game map is represented as an adjacency matrix graph data structure.

Introduction

Arcade games were, and partially still are, a big part of modern society. Whether you play them to relax or competitively, everyone enjoys them from time to time. One example of these arcade games, one of the most popular of all time, is Pac-man. Pac-man is a 2-Dimensional game which depicts a small, circular, yellow figure - Pac-man - whose goal is to collect all the treats on the map. It is not as easy as it seems, however, since there are multiple ghosts constantly chasing Pac-man around in an attempt to capture him, stripping him of one of his three lives.

This project will use the robust features of Qt and C++ in order to recreate the Pac-man game using a graph and path-finding algorithms to operate. The report will explore various parts of the code such as how the map was loaded and the movement of the different characters.

Problem Definition

The game revolves around the iconic, circular, yellow character Pac-man whose goal is to collect all the treats around the map. The player must guide Pac-man around this 2-Dimensional environment around the obstacles in order to collect them all. Points are awarded for every treat consumed.

The task is not as easy as it seems, however, as Pac-man is faced with a few obstacles. There are four ghosts constantly chasing Pac-man around the map. It is the player's goal to avoid getting cornered and captured by these ghosts or they may fail the game. The player has three lives in order to collect all the treats or they must restart. There are a few power pellets scattered around the map which grant Pac-man the ability to eat the ghosts, causing them to panic and try to flee from his anger. Eating these ghosts causes them to reset and grants the player some bonus points.

Methodology

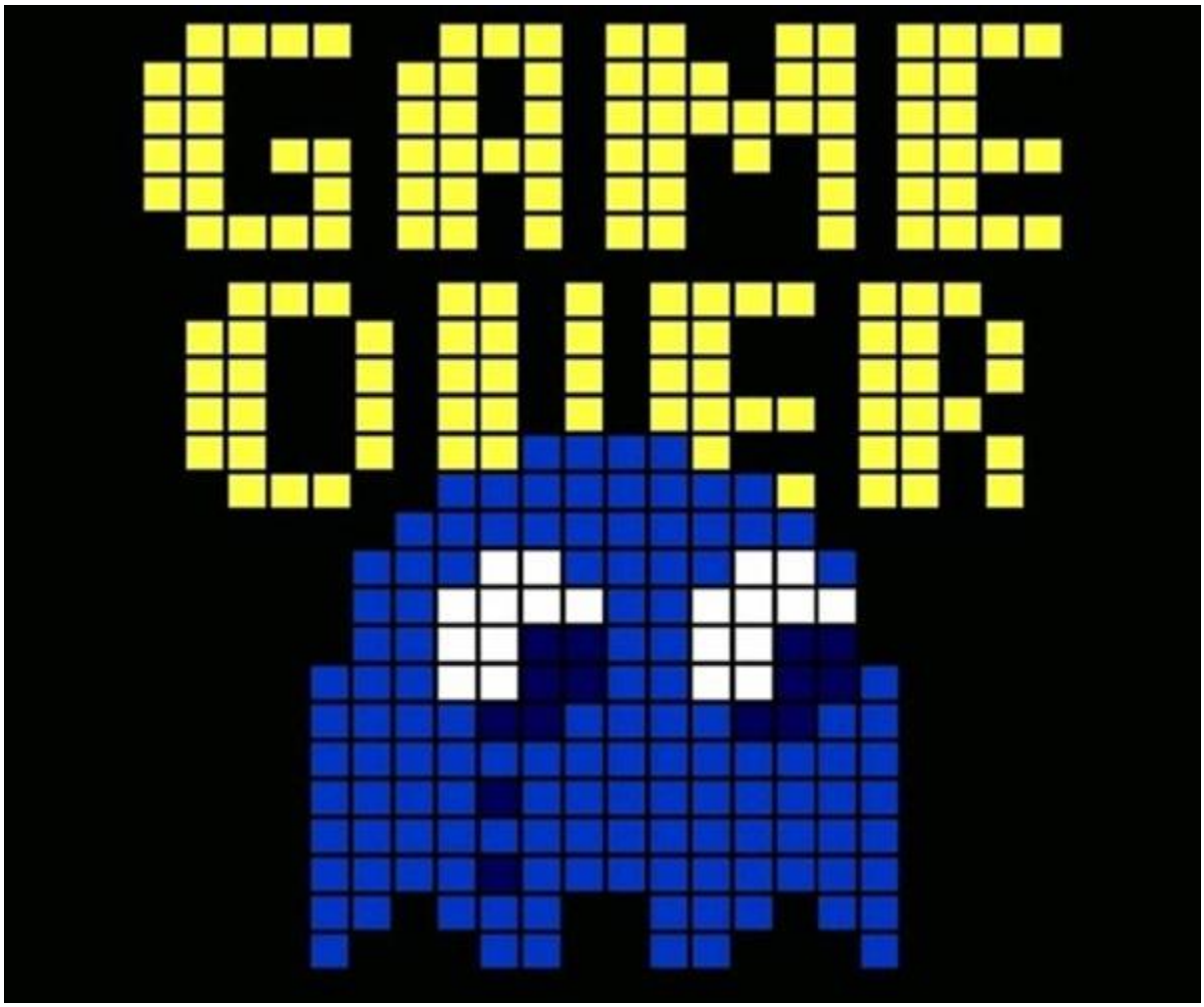
For the methodology part, we chose QT creator framework because it has some robust features and functionalities that would help us develop the game graphics in the way we would like it to be. Starting the game, a start screen is displayed that tells the user some information about the game and how he could play it. Throughout the whole game, we have let QT creator to keep track of the keyboard presses. If the user presses Enter while the start menu is displayed, the game will immediately start. To move Pac-Man, the user can use the arrow keys that are being tracked by QT. On pressing the arrow up key, Pac-Man moves up until he hits a wall or hits a ghost. The same conditions apply for the rest of arrow keys. To pause the game, the user can press the escape button on the keyboard. The user shall reverse this action through pressing the escape button again. On pausing the game, a menu of three

options is displayed. The options are resume, restart, and quit. The first option we have already talked about. For the second and third ones, they are assigned to keyboard buttons such that on pressing ‘R’, the game is restarted, while on pressing ‘Q’ the program exits to the main menu. If the game is over, a game over screen is displayed that asks the user whether he wants to restart the game or exit the main menu. If Pac-Man wins, a winner screen is displayed that also asks the user whether he wants to play again or exit the main menu.

Below is a picture representing the starting screen.

Below is a picture representing the escape menu.

Below is a picture representing the game over screen.



Algorithms Used

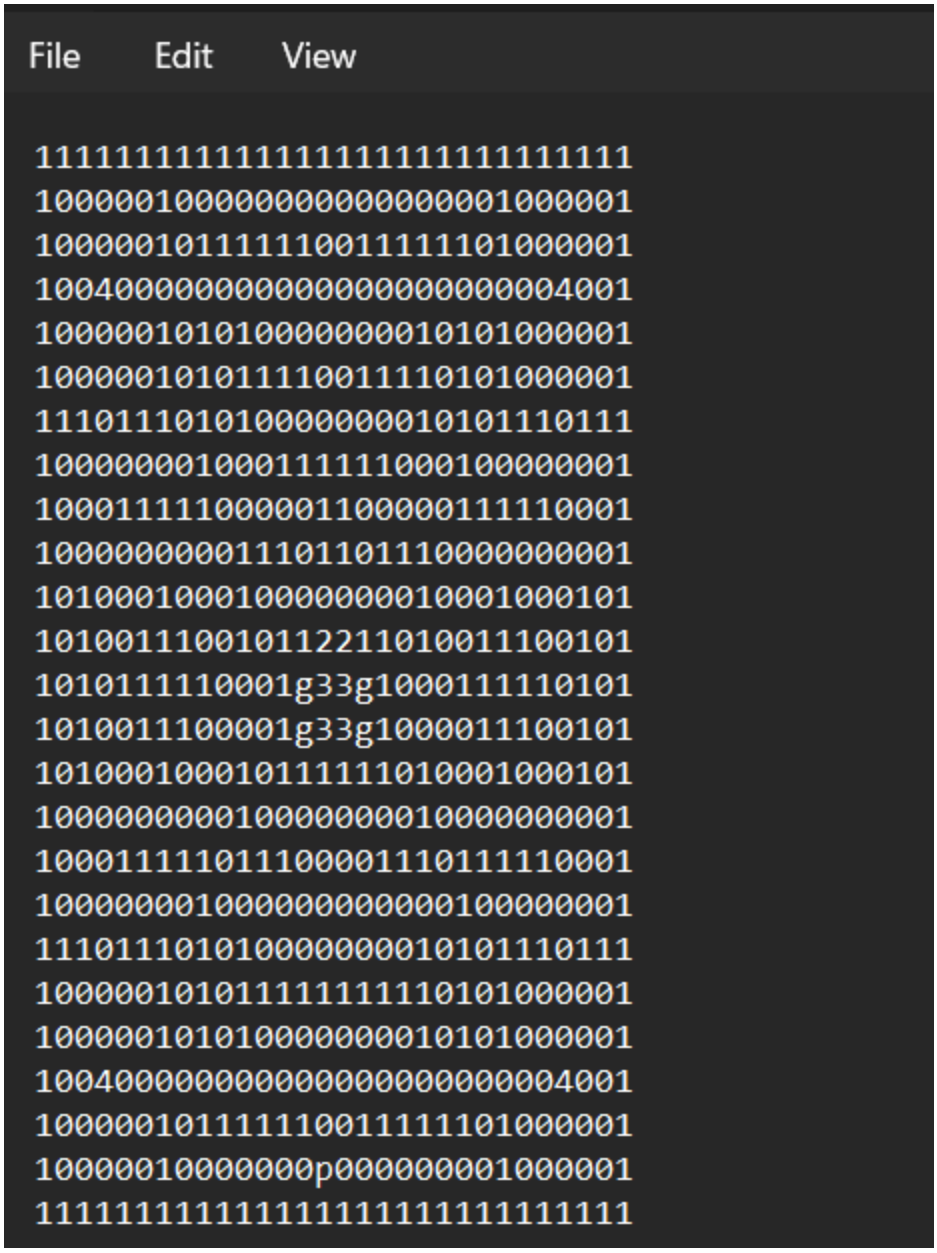
Throughout the project, I have used some famous algorithms. Initially we used A* star algorithm and Dijkstra at the same time for chasing pacman and escaping away from it. However, it was clear to us that they perform very similarly to each other, so we decided to only use

Dijkstra for the movement of ghosts in general. Dijkstra's algorithm, a fundamental graph search method, finds the shortest path between two nodes in a weighted graph by iteratively selecting the vertex with the smallest tentative distance and updating the distances to its neighbors until the destination is reached. It employs a priority queue to efficiently explore and prioritize nodes based on their current tentative distances. We have also used an algorithm to find the furthest location from Pac-Man to help ghosts escape away from it.

Data Specification

For the data specification part, we have used a text file to generate and represent the game map. The text file named “map.txt” contained gameWidth*gameHeight characters, where each character represents an item on the map. The game map text file contained characters like ‘1’, ‘0’, ‘3’, ‘4’, ‘g’, and ‘p’. ‘1’ represents a wall on the game map, ‘0’ represents a ball on the game map, ‘3’ represents an empty space on the game map, ‘4’ represents a power ball, ‘g’ represents a ghost, and finally, ‘p’ represents the Pac-Man. We have also used a text file named “highScore.txt” to save the high score of Pac-Man. This text file is only updated if the Pac-Man was able to reach a score that is higher than the current score high score.

Below is a picture representing the game map as a text file.



Experimental Results

Below is a picture that shows the initial phase of the game alongside with the initial positions of each item.

After 5 seconds the red ghost is released as shown below. (One ghosts is released each five seconds.)

When Pac-Man moves in a position where there are balls, the balls disappear and the score increases by 10 for each ball accordingly.

When Pac-Man eats a power ball, the released ghosts turn into panic mode and move away from Pac-Man as shown below.

When Pac-Man meets one of the ghosts, his lives get decremented by one and he returns to his respawn point as shown below.

When Pac-Man meets a ghost while in panic mode, the ghost turns into running mode and gets back to his respawn point as shown below.


pac-man game by the goats
Score: 340
High Score: 4270
Lives:
Press [ESC] to Pause the Game.

When Pac-Man wastes all of his lives, he loses the game as shown below. (The same thing happens for the condition of winning the game)

Analysis and Critique

To begin with, we have the functions included in the gameobject source file which corresponds to the animation of the objects in the game. The move functions for both the PacMan object and ghosts have a time complexity of O(1) since they call on their respective animate function

which changes the frame and calls their respective setpic function which has a complexity of O(1) as well.

Moving onto the other source file, game.cpp, we have the functions that are related to the movements of each object and how the game runs. The functions are all timer based, this includes functions related to the release of the ghosts and how long they chase for between certain intervals, etc. We begin with the constructor that calls the class Game. Basic declarative and initializations, function calls and timer starts will have a complexity of O(1), therefore the overall time complexity in the constructor is O(1). The destructor consists of deleting all the times, deleting the map and deleting the objects. Deleting each timer and object costs O(1) each, however, deleting the map costs O(N*M) where N represents the rows and M represents the columns. The growth of this is equivalent to that of $O(N^2)$. Therefore, the overall complexity of the destructor is $O(N^2)$.

Next we have the generate game function which is called upon in the constructor. This function first initializes the map which will cost O(N*M) where N represents the rows and M represents the columns. However, as we said before this is equivalent to $O(N^2)$. We then read the file to add all the objects to the map which also costs $O(N^2)$. Then there are small processes related to setting certain aspects of the display such as the score, highscore, lives and esc button which will all cost O(1). Therefore the overall time complexity of this function will be $O(N^2)$, since this is the dominant complexity.

We then have the pause and resume game functions which are both similar in overall complexity. The pause function stops the timers which are all constant operations resulting in a time complexity of O(1). The

resume function starts the timers again which is also a constant operation with complexity O(1). The function that proceeds, the restart game function, consists of constant operations such as resetting graphic items and variables and initializing the ghosts and pacman positions. However, when restarting the map, it uses a nested loop resulting in a complexity of $O(N^2)$. Hence the overall complexity of the restart function will be $O(N^2)$.

The release ghosts function consists of a series of conditional statements checking if each ghost released or not. Since this process is independent of the number of ghosts, this results in a complexity of O(1). The games won function that follows has a complexity of $O(N^2)$ since it iterates over the entire matrix to check if PacMan has won by eating all the balls.

The spawn power up function is based on randomizing a location to spawn the power up and so it is difficult to determine the worst case complexity. By considering the average case we could obtain a constant time since it would not take too much time to find a suitable location.

The timer based functions delete timer, start and end panic mode, start and end run mode are all of constant times since they are just starting and stopping timers which are constant time operations. The key press event function handles key presses. The time complexity will also be O(1) since key pressing involves constant time operations. The chasePacma, move pacman and back to cage functions are also constant time operations since they just call upon the move functions and use timer based functions or functions related to setting text objects. The same applies for the move ghost functions. Collision is yet another function which uses previously mentioned functions to determine

whether pac man has collided with a ghost, this results in there being a complexity of O(1).

Now the mapGraph constructor involves reading a file and initializing the graph which will take a time complexity of $O(N^2)$. Furthermore, the boolean functions and getter functions associated with the mapGraph class are all of constant time since they are just returning data, hence an overall complexity of O(1).

When analyzing the pathfinding functions used in this program, we implemented Dijksatra's algorithm which will have an overall complexity of O(V + ElogV) where E is the number of edges and V is the number of vertices in the graph. This is quite similar to A* algorithm which is considered a more efficient version of Dijkstras.

Overall, throughout the entire program, the overall complexity after considering the complexities of each of the functions will lead to an overall complexity $O(N^2)$ since it is the most dominant.

Possible critiques that we have are not related to the algorithm used since Dijkastra's pathfinding algorithm performs efficiently, but rather the issue lies with the movement of the ghosts where the longer the program runs for, the more likely ghosts are to lag and show significant delays. This may be a result of our timer based functions.

Conclusion

The recreation of the Pac-man game using Qt and C++ has been an interesting experience seeing how pathfinding algorithms behave between moving nodes. It also provided an insight into the use of graphs and their construction for some user-friendly applications. The project provided us with foundational knowledge which will be of great assistance in future projects with teamwork and applications of complex data structures.

The process wasn't as smooth as we had hoped, however, as we faced some difficulties with the project. For example, there were some issues with the movement of the ghosts and the smooth running of the program. However, after some long hours of research and trial and error, it was mostly improved. The game could have been improved much further if we chose to create it with threads for a smoother game which would be our first choice creating a similar program in the future.

Acknowledgements

We would like to express our heartfelt gratitude to Professor Amr Goneid, whose unwavering guidance and expertise have been instrumental in shaping this project. His profound knowledge of the subject matter, coupled with his dedication to fostering a conducive learning environment, has significantly enriched our understanding and contributed to the success of this endeavor. We would also like to extend

our sincere appreciation to Engineer Mohamed Hany, whose support and valuable insights have played a crucial role in navigating the complexities of the course. Their collective efforts have not only enhanced the quality of our work but have also left an indelible mark on our academic journey. We are truly fortunate to have had the privilege of learning under their mentorship.

Appendix

QT Project code

```
QT    += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++17

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000
# disables all the APIs deprecated before Qt 6.0.0

SOURCES += \
        game.cpp \
        gameobject.cpp \
        main.cpp \
        mainwindow.cpp

HEADERS += \
```

```
    game.h \
    gameobject.h \
    mainwindow.h

FORMS += \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

gameobject Header File

```
#include "qtimer.h"
#include<QGraphicsItem>
#ifndef GAMEOBJECT_H
#define GAMEOBJECT_H
#define pacLives 3;
class Game;

enum CellType {open = 0,closed = 1};
enum type{Ball, Wall, Gate, Pacman, Ghost, Blank, PowerBall};
enum direction {Up = 0, Down = 1, Left = 2, Right = 3};
enum ghostColor {Red = 0, Orange = 1, Pink = 2, Green = 3};
enum ghostCondition {Normal, Panic, Running};

struct Cell {
    int x, y;
    CellType type;
```

```
        bool operator<(const Cell& other) const {
        if (x != other.x) {
        return x < other.x;
        }
        return y < other.y;
        }
};


class GameObject: public QGraphicsPixmapItem
{
public:
        static const int Width = 20;
        GameObject(enum type,QPixmap);
        friend class Game;
protected:
        int xcor,ycor;
        enum direction dir;
        enum type objectType;
        virtual void moveUp();
        virtual void moveDown();
        virtual void moveLeft();
        virtual void moveRight();
};


class PacMan : public GameObject{
public:
        PacMan(QPixmap pacpix);
        friend class Game;
```

```
private:
        bool pacmoving,dead,won;
        int initx,inity,lives,currentframe,score,highScore;
        QTimer *pacTimer;
        QPixmap right[6],left[6],up[6],down[6];
        void loadGraphics();
        void moveUp()override;
        void moveDown()override;
        void moveLeft()override;
        void moveRight()override;
        void loadHighScore();
        void saveHighScore();
        void setpic();
        void animate();
};


class Gh0st: public GameObject
{
public:
        const static int GhostNum = 4;
        Gh0st(ghostColor col,QPixmap ghoPix);
        friend class Game;
private:
        bool released;
        ghostColor color;
        ghostCondition condition;
        bool OutofCage,escapepath;
        int initx,inity;
```

```
        QTimer
*chaseTimer,*escapeTimer,*panicModeTimer,*runTimer,*runMode;
        QPixmap
orange[4][2],red[4][2],pink[4][2],green[4][2],peacful[2],eyes;
        int currentframe;
        void loadGraphics();
        void setGhostPic();
        void animateGhost();
        void moveUp()override;
        void moveDown()override;
        void moveLeft()override;
        void moveRight()override;
};


#endif // GAMEOBJECT_H
```

gameobject Implementation File

```
#include "gameobject.h"
#include<QTimer>
#include<QMainWindow>
#include<QFile>

GameObject::GameObject(enum type t, QPixmap pixmap)
        : QGraphicsPixmapItem(pixmap)
{
        objectType = t;
}

void GameObject::moveUp()
```

```
{
	ycor--;
}

void GameObject::moveDown()
{
	ycor++;
}

void GameObject::moveLeft()
{
	xcor--;
}

void GameObject::moveRight()
{
	xcor++;
}




PacMan::PacMan(QPixmap pacpix):GameObject(type::Pacman,pacpix)
{
	pacmoving=dead=won=false;
	lives=pacLives;currentframe=0;score=0;highScore=0;
	loadHighScore();
	loadGraphics();
}
```

```
void PacMan::loadGraphics()
{
	right[0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/common.png");
	right[1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/r2.png");
	right[2].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/r3.png");
	right[3].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/r4.png");
	right[4].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/r5.png");
	right[5].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/r6.png");

	left[0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/common.png");
	left[1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/l2.png");
	left[2].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/l3.png");
	left[3].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/l4.png");
	left[4].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/l5.png");
	left[5].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/l6.png");

	up[0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/common.png");
```

```
    up[1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/u2.png");
    up[2].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/u3.png");
    up[3].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/u4.png");
    up[4].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/u5.png");
    up[5].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/u6.png");

    down[0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/common.png");
    down[1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/d2.png");
    down[2].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/d3.png");
    down[3].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/d4.png");
    down[4].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/d5.png");
    down[5].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/d6.png");
}

void PacMan::loadHighScore()
{
    QFile file("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/PAC/highScore.txt");
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
```

```
        qDebug() << "Failed to open the text file";
        return;
        }
        QTextStream in(&file);
        QString line=in.readLine();
        highScore=line.toInt();
        file.close();
}

void PacMan::saveHighScore()
{
        QFile file("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/PAC/highScore.txt");
        if (file.open(QIODevice::WriteOnly | QIODevice::Text)) {
        QTextStream out(&file);
        out << highScore;
        file.close();
        }
}


void PacMan::moveUp()
{
        ycor--;
        animate();
}
void PacMan::moveDown()
{
        ycor++;
        animate();
```

```
}
void PacMan::moveLeft()
{
	xcor--;
	animate();
}
void PacMan::moveRight()
{
	xcor++;
	animate();
}

void PacMan::setpic()
{
	if (dir == direction::Up) {
	setPixmap(up[currentframe]);
	} else if (dir == direction::Down) {
	setPixmap(down[currentframe]);
	} else if (dir == direction::Left) {
	setPixmap(left[currentframe]);
	} else if (dir == direction::Right) {
	setPixmap(right[currentframe]);
	}
}

void PacMan::animate()
{
	currentframe = (currentframe + 1) % 6;
	setpic();
}
```

```
Gh0st::Gh0st(ghostColor col,QPixmap
ghoPix):GameObject(type::Ghost,ghoPix)
{
	released=false;
	this->color=col;
	escapepath=false;currentframe=0;
	condition=ghostCondition::Normal;
	loadGraphics();
	dir=direction::Up;
	if(col==ghostColor::Green)
	{
	setPixmap(green[0][0]);
	}
	else if(col==ghostColor::Red)
	{
	setPixmap(red[0][0]);
	}
	else if(col==ghostColor::Orange)
	{
	setPixmap(orange[0][0]);
	}
	else if(col==ghostColor::Pink)
	{
	setPixmap(pink[0][0]);
	}
}
```

```
void Gh0st::loadGraphics()
{
	green[0][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/gu1.png");
	green[0][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/gu2.png");
	green[1][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/gd1.png");
	green[1][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/gd2.png");
	green[2][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/gl1.png");
	green[2][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/gl2.png");
	green[3][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/gr1.png");
	green[3][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/gr2.png");

	red[0][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/ru1.png");
	red[0][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/ru2.png");
	red[1][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/rd1.png");
	red[1][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/rd2.png");
	red[2][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/rl1.png");
```

```
        red[2][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/rl2.png");
        red[3][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/rr1.png");
        red[3][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/rr2.png");

        orange[0][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/ou1.png");
        orange[0][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/ou2.png");
        orange[1][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/od1.png");
        orange[1][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/od2.png");
        orange[2][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/ol1.png");
        orange[2][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/ol2.png");
        orange[3][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/or1.png");
        orange[3][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/or2.png");

        pink[0][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/pu1.png");
        pink[0][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/pu2.png");
        pink[1][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/pd1.png");
```

```
	pink[1][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/pd2.png");
	pink[2][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/pl1.png");
	pink[2][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/pl2.png");
	pink[3][0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/pr1.png");
	pink[3][1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/pr2.png");

	peacful[0].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/peacful1.png");
	peacful[1].load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/peacful2.png");

	eyes.load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Ghosts/eyes.png");
}

void Gh0st::setGhostPic()
{
	if(condition==ghostCondition::Normal)
	{
	if(color==ghostColor::Green)
	{
	if (dir == direction::Up) {
		setPixmap(green[Up][currentframe]);
	} else if (dir == direction::Down) {
		setPixmap(green[Down][currentframe]);
```

```
} else if (dir == direction::Left) {
        setPixmap(green[Left][currentframe]);
} else if (dir == direction::Right) {
        setPixmap(green[Right][currentframe]);
}
}
else if(color==ghostColor::Red)
{
if (dir == direction::Up) {
        setPixmap(red[Up][currentframe]);
} else if (dir == direction::Down) {
        setPixmap(red[Down][currentframe]);
} else if (dir == direction::Left) {
        setPixmap(red[Left][currentframe]);
} else if (dir == direction::Right) {
        setPixmap(red[Right][currentframe]);
}
}
else if(color==ghostColor::Pink)
{
if (dir == direction::Up) {
        setPixmap(pink[Up][currentframe]);
} else if (dir == direction::Down) {
        setPixmap(pink[Down][currentframe]);
} else if (dir == direction::Left) {
        setPixmap(pink[Left][currentframe]);
} else if (dir == direction::Right) {
        setPixmap(pink[Right][currentframe]);
}
}
```

```
        else if(color==ghostColor::Orange)
        {
        if (dir == direction::Up) {
                setPixmap(orange[Up][currentframe]);
        } else if (dir == direction::Down) {
                setPixmap(orange[Down][currentframe]);
        } else if (dir == direction::Left) {
                setPixmap(orange[Left][currentframe]);
        } else if (dir == direction::Right) {
                setPixmap(orange[Right][currentframe]);
        }
        }
        }else if(condition==ghostCondition::Panic)
        {
        setPixmap(peacful[currentframe]);
        }
        else if(condition==ghostCondition::Running)
        {
        setPixmap(eyes);
        }
}

void Gh0st::animateGhost()
{
        currentframe = (currentframe + 1) % 2;
        setGhostPic();
}

void Gh0st::moveUp()
{
```

```
	dir=direction::Up;
	ycor--;
	setPos(xcor * GameObject::Width, ycor * GameObject::Width);
	animateGhost();
}

void Gh0st::moveDown()
{
	dir=direction::Down;
	ycor++;
	setPos(xcor * GameObject::Width, ycor * GameObject::Width);
	animateGhost();
}

void Gh0st::moveLeft()
{
	dir=direction::Left;
	xcor--;
	setPos(xcor * GameObject::Width, ycor * GameObject::Width);
	animateGhost();
}

void Gh0st::moveRight()
{
	dir=direction::Right;
	xcor++;
	setPos(xcor * GameObject::Width, ycor * GameObject::Width);
	animateGhost();
}
```

game header File

```
#include<QMainWindow>
#include"gameobject.h"
#include<QGraphicsView>
#include<QTimer>
#ifndef GAME_H
#define GAME_H

#define ballScore 10
#define powerBallScore 20
#define ghostScore 30


struct Point {
        int x;
        int y;
};


class mapGraph{
public:
        mapGraph(int r,int c);
        int rows,cols;
        std::vector<std::vector<Cell>> Graph;
        bool isValidCell(int x, int y) const ;
        const Cell& getCell(int x, int y) const;
        bool areAdjacent(const Cell& cell1, const Cell& cell2) const;
        std::vector<Cell> getNeighbors(const Cell& cell) const ;
```

```
        std::vector<Cell> findShortestPath(const Cell& start, const Cell&
goal) const;
        int heuristic(const Cell& a, const Cell& b) const ;
        std::vector<Cell> dijkstra(const Cell &start, const Cell &goal)
const;
        Point findFarthestPoint(int x0, int y0) ;
};


class Game: public QGraphicsScene
{
        Q_OBJECT
public:
        static const int MapRows = 25;
        static const int MapCols = 30;
        Game();
        ~Game();
        friend class mapGraph;
private:
        Point p;
        int powerBalls;
        QPixmap gameover,gamewon;
        QGraphicsPixmapItem *gameOver;

        QPixmap start;
        QGraphicsPixmapItem *mainMenu;


        QPixmap menu;
        QGraphicsPixmapItem *escapeMenu;
```

```
        bool paused;
        bool gameStarted;


        GameObject ***gameMap;
        mapGraph *g;
        PacMan *Pac;
        Gh0st *Gho1,*Gho2,*Gho3,*Gho4;

        QTimer
*animationTimer,*collisionTimer,*releaseTimer,*spawnTimer;

        QGraphicsTextItem
*scoreDisplay,*highScoreDisplay,*livesDisplay,*escdisplay;
        QPixmap blank,heart;
        QGraphicsPixmapItem *h1,*h2,*h3;

        void keyPressEvent(QKeyEvent *ev);
        void mousePressEvent(QMouseEvent *ev);

        void moveGhost(Gh0st *Gho1,int tarx,int tary,int time);
        void updateScore();
        void startPanicMode(Gh0st *Gho);
        void startRunMode(Gh0st *Gho);
        void generateGame();
        void pauseGame();
        void resumeGame();
        void deleteTimer(QTimer *&timer);
        void restartGame();
        void releaseGhosts();
```

```
        bool gameWon();
        void spawnPowerUp();
private slots:

        void chasePacman(Gh0st *Gho,int xcor,int ycor);
        void escapefromPacman(Gh0st *Gho);
        void  backToCage(Gh0st *Gho);

        void endPanicMode(Gh0st *Gho);
        void endRunMode(Gh0st *Gho);

        void movePacman();
        void collision();
};


#endif // GAME_H
```

game implementation File

```
#include "game.h"
#include "qapplication.h"
#include<QFile>
#include<QKeyEvent>
#include<queue>
#include <QtGlobal>
Game::Game()
```

```
{
    gameStarted=false;
    powerBalls=4;
    p.x=0;p.y=0;
    g=new mapGraph(MapRows,MapCols);
    gamewon.load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/gamewon.jpg");
    gameover.load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/gameOver.jpg");
    menu.load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/menu.jpg");
    start.load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/start.png");

    generateGame();
    releaseTimer=new QTimer(this);
    collisionTimer=new QTimer(this);

    Pac->pacTimer=new QTimer(this);
    animationTimer=new QTimer(this);

    spawnTimer=new QTimer(this);

    Gho1->chaseTimer = new QTimer(this);
    Gho2->chaseTimer = new QTimer(this);
    Gho3->chaseTimer = new QTimer(this);
    Gho4->chaseTimer = new QTimer(this);

    Gho1->runTimer = new QTimer(this);
    Gho2->runTimer = new QTimer(this);
```

```
    Gho3->runTimer = new QTimer(this);
    Gho4->runTimer = new QTimer(this);

    Gho1->runMode = new QTimer(this);
    Gho2->runMode = new QTimer(this);
    Gho3->runMode = new QTimer(this);
    Gho4->runMode = new QTimer(this);

    Gho1->escapeTimer = new QTimer(this);
    Gho2->escapeTimer = new QTimer(this);
    Gho3->escapeTimer = new QTimer(this);
    Gho4->escapeTimer = new QTimer(this);

    Gho1->panicModeTimer = new QTimer(this);
    Gho2->panicModeTimer = new QTimer(this);
    Gho3->panicModeTimer = new QTimer(this);
    Gho4->panicModeTimer = new QTimer(this);



    connect(Gho1->panicModeTimer, &QTimer::timeout, [this]()
{endPanicMode(Gho1);});
    connect(Gho2->panicModeTimer, &QTimer::timeout, [this]()
{endPanicMode(Gho2);});
    connect(Gho3->panicModeTimer, &QTimer::timeout, [this]()
{endPanicMode(Gho3);});
    connect(Gho4->panicModeTimer, &QTimer::timeout, [this]()
{endPanicMode(Gho4);});
```

```
        connect(Gho1->escapeTimer, &QTimer::timeout, [this]()
{escapefromPacman(Gho1);});
        connect(Gho2->escapeTimer, &QTimer::timeout, [this]()
{escapefromPacman(Gho2);});
        connect(Gho3->escapeTimer, &QTimer::timeout, [this]()
{escapefromPacman(Gho3);});
        connect(Gho4->escapeTimer, &QTimer::timeout, [this]()
{escapefromPacman(Gho4);});

        connect(Pac->pacTimer,&QTimer::timeout,
this,&Game::movePacman);
        connect(animationTimer, &QTimer::timeout, [this]()
{Pac->animate();});

        connect(releaseTimer,&QTimer::timeout,
this,&Game::releaseGhosts);

        connect(spawnTimer,&QTimer::timeout,
this,&Game::spawnPowerUp);

        connect(Gho1->chaseTimer, &QTimer::timeout, [this]()
{chasePacman(Gho1,Pac->xcor,Pac->ycor);});
        connect(Gho2->chaseTimer, &QTimer::timeout, [this]()
{chasePacman(Gho2,Pac->xcor+5,Pac->ycor);});
        connect(Gho3->chaseTimer, &QTimer::timeout, [this]()
{chasePacman(Gho3,Pac->xcor-5,Pac->ycor);});
        connect(Gho4->chaseTimer, &QTimer::timeout, [this]()
{chasePacman(Gho4,Pac->xcor,Pac->ycor+5);});
```

```
        connect(Gho1->runTimer, &QTimer::timeout, [this]()
{backToCage(Gho1);});
        connect(Gho2->runTimer, &QTimer::timeout, [this]()
{backToCage(Gho2);});
        connect(Gho3->runTimer, &QTimer::timeout, [this]()
{backToCage(Gho3);});
        connect(Gho4->runTimer, &QTimer::timeout, [this]()
{backToCage(Gho4);});

        connect(Gho1->runMode, &QTimer::timeout, [this]()
{endRunMode(Gho1);});
        connect(Gho2->runMode, &QTimer::timeout, [this]()
{endRunMode(Gho2);});
        connect(Gho3->runMode, &QTimer::timeout, [this]()
{endRunMode(Gho3);});
        connect(Gho4->runMode, &QTimer::timeout, [this]()
{endRunMode(Gho4);});

        connect(collisionTimer, &QTimer::timeout, this,
&Game::collision);
        releaseTimer->start(5000);
        spawnTimer->start(25000);
        animationTimer->start(100);
        collisionTimer->start(0);
        pauseGame();
}

Game::~Game()
{
        deleteTimer(Pac->pacTimer);
```

```
deleteTimer(animationTimer);
deleteTimer(Gho1->chaseTimer);
deleteTimer(Gho2->chaseTimer);
deleteTimer(Gho3->chaseTimer);
deleteTimer(Gho4->chaseTimer);
deleteTimer(Gho1->runTimer);
deleteTimer(Gho2->runTimer);
deleteTimer(Gho3->runTimer);
deleteTimer(Gho4->runTimer);
deleteTimer(Gho1->runMode);
deleteTimer(Gho2->runMode);
deleteTimer(Gho3->runMode);
deleteTimer(Gho4->runMode);
deleteTimer(Gho1->escapeTimer);
deleteTimer(Gho2->escapeTimer);
deleteTimer(Gho3->escapeTimer);
deleteTimer(Gho4->escapeTimer);
deleteTimer(Gho1->panicModeTimer);
deleteTimer(Gho2->panicModeTimer);
deleteTimer(Gho3->panicModeTimer);
deleteTimer(Gho4->panicModeTimer);

for (int i = 0; i < MapRows; ++i) {
for (int j = 0; j < MapCols; ++j) {
delete[] gameMap[i][j];
}

delete[] gameMap[i];
}
delete[] gameMap;
```

```
        delete [] Pac;
        delete [] Gho1;
        delete [] Gho2;
        delete [] Gho3;
        delete [] Gho4;
        delete [] scoreDisplay;
        delete [] highScoreDisplay;
        delete [] h1;
        delete [] h2;
        delete [] h3;
        delete [] livesDisplay;
        delete [] escapeMenu;
}

void Game::generateGame()
{
        int ghostNum=0;
        int x=0,y=0,mapWidth=30,mapHeight=25;
        QPixmap Wall("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/wall.png");
        QPixmap Gate("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/gate.png");
        QPixmap Ball("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/ball.png");
        QPixmap
PowerBall("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall 2023/Applied
Data Structures/PAC/Objects/powerBall.png");
        QPixmap Pacpix("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/pacman/r3.png");
```

```
gameMap = new GameObject**[mapHeight];
for (int i = 0; i < mapHeight; i++) {
gameMap[i] = new GameObject*[mapWidth];
for (int j = 0; j < mapWidth; j++)
gameMap[i][j] = nullptr;
}
QFile file("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/map.txt");
if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
qDebug() << "Failed to open the text file";
return;
}
QTextStream in(&file);

for(int i=0;i<mapHeight;i++)
{
QString line = in.readLine();
for(int j=0;j<mapWidth;j++)
{
int X=x+(j*20);
int Y=y+(i*20);
if(line[j]=='0')
{

    gameMap[i][j]=new GameObject(type::Ball,Ball);
    gameMap[i][j]->setPos(X,Y);
    addItem(gameMap[i][j]);
}
else if(line[j]=='1')
```

```
        {
                gameMap[i][j]=new GameObject(type::Wall,Wall);
                gameMap[i][j]->setPos(X,Y);
                addItem(gameMap[i][j]);
        }
        else if(line[j]=='2')
        {
                gameMap[i][j]=new GameObject(type::Gate,Gate);
                gameMap[i][j]->setPos(X,Y);
                addItem(gameMap[i][j]);
        }
        else if(line[j]=='3')
        {
                gameMap[i][j]=new GameObject(type::Blank,blank);
                gameMap[i][j]->setPos(X,Y);
                addItem(gameMap[i][j]);
        }
        else if(line[j]=='4')
        {
                gameMap[i][j]=new
GameObject(type::PowerBall,PowerBall);
                gameMap[i][j]->setPos(X,Y);
                addItem(gameMap[i][j]);



        }
        else if(line[j]=='p')
        {
                Pac=new PacMan(Pacpix);
                Pac->setPos(X,Y);
```

```
        Pac->xcor=j;
        Pac->ycor=i;
        Pac->initx=j;
        Pac->inity=i;
        addItem(Pac);
        gameMap[i][j]=Pac;

}
else if(line[j]=='g')
{
        if(ghostNum==0)
        {
        Gho1=new Gh0st(ghostColor::Red,blank);
        Gho1->setPos(X,Y);
        Gho1->xcor=j;
        Gho1->ycor=i;
        Gho1->initx=j;
        Gho1->inity=i;
        addItem(Gho1);
        gameMap[i][j]=Gho1;
        }
        else if(ghostNum==1)
        {
        Gho2=new Gh0st(ghostColor::Orange,blank);
        Gho2->setPos(X,Y);
        Gho2->xcor=j;
        Gho2->ycor=i;
        Gho2->initx=j;
        Gho2->inity=i;
        addItem(Gho2);
```

```
        gameMap[i][j]=Gho2;
        }
        else if(ghostNum==2)
        {
        Gho3=new Gh0st(ghostColor::Pink,blank);
        Gho3->setPos(X,Y);
        Gho3->xcor=j;
        Gho3->ycor=i;
        Gho3->initx=j;
        Gho3->inity=i;
        addItem(Gho3);
        gameMap[i][j]=Gho3;
        }
        else if(ghostNum==3)
        {
        Gho4=new Gh0st(ghostColor::Green,blank);
        Gho4->setPos(X,Y);
        Gho4->xcor=j;
        Gho4->ycor=i;
        Gho4->initx=j;
        Gho4->inity=i;
        addItem(Gho4);
        gameMap[i][j]=Gho4;
        }
        ghostNum++;

}
if (gameMap[i][j]) {
        gameMap[i][j]->xcor = j;
        gameMap[i][j]->ycor = i;
```

```
        }
        }
        }
        file.close();
        scoreDisplay = new QGraphicsTextItem();
        highScoreDisplay = new QGraphicsTextItem();
        scoreDisplay->setPos(620, 0);
        highScoreDisplay->setPos(600, 20);
        updateScore(); // Helper function to update the score text
        addItem(scoreDisplay);
        addItem(highScoreDisplay);
        h1=new QGraphicsPixmapItem;
        h2=new QGraphicsPixmapItem;
        h3=new QGraphicsPixmapItem;
        heart.load("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/Heart.png");
        h1->setPixmap(heart);
        h2->setPixmap(heart);
        h3->setPixmap(heart);
        h1->setPos(645,50);h2->setPos(675,50);h3->setPos(705,50);
        addItem(h1);addItem(h2);addItem(h3);
        livesDisplay=new QGraphicsTextItem();
        livesDisplay->setPos(600,47);
        livesDisplay->setPlainText(QString("Lives:"));
        QFont font("Comic Sans MS", 12, QFont::Bold);  // Adjust the
font size as needed
        livesDisplay->setFont(font);
        livesDisplay->setDefaultTextColor(QColor(Qt::white));
        addItem(livesDisplay);
        escapeMenu=new QGraphicsPixmapItem(blank);
```

```
    escapeMenu->setPos(200,190);
    addItem(escapeMenu);
    escdisplay=new QGraphicsTextItem();
    escdisplay->setPos(200,500);
    escdisplay->setPlainText(QString("Press [ESC] to Pause the
Game."));
    escdisplay->setFont(font);
    escdisplay->setDefaultTextColor(QColor(Qt::white));
    addItem(escdisplay);
    gameOver=new QGraphicsPixmapItem(blank);
    gameOver->setPos(0,0);
    addItem(gameOver);
    mainMenu=new QGraphicsPixmapItem(start);
    mainMenu->setPos(0,0);
    addItem(mainMenu);
}


void Game::pauseGame()
{
    escapeMenu->setPixmap(menu);
    paused=true;

    Pac->pacTimer->stop();
    animationTimer->stop();


    collisionTimer->stop();

    Gho1->chaseTimer->stop();
```

```
        Gho2->chaseTimer->stop();
        Gho3->chaseTimer->stop();
        Gho4->chaseTimer->stop();

        Gho1->runTimer->stop();
        Gho2->runTimer->stop();
        Gho3->runTimer->stop();
        Gho4->runTimer->stop();

        Gho1->runMode->stop();
        Gho2->runMode->stop();
        Gho3->runMode->stop();
        Gho4->runMode->stop();

        Gho1->escapeTimer->stop();
        Gho2->escapeTimer->stop();
        Gho3->escapeTimer->stop();
        Gho4->escapeTimer->stop();

        Gho1->panicModeTimer->stop();
        Gho2->panicModeTimer->stop();
        Gho3->panicModeTimer->stop();
        Gho4->panicModeTimer->stop();
}

void Game::resumeGame()
{
        escapeMenu->setPixmap(blank);
        paused=false;
        Pac->pacTimer->start();
```

```
animationTimer->start();
collisionTimer->start();
if(Gho1->condition==ghostCondition::Normal)
{
Gho1->chaseTimer->start();
}
if(Gho2->condition==ghostCondition::Normal)
{
Gho2->chaseTimer->start();
}
if(Gho3->condition==ghostCondition::Normal)
{
Gho3->chaseTimer->start();
}
if(Gho4->condition==ghostCondition::Normal)
{
Gho4->chaseTimer->start();
}
if(Gho1->condition==ghostCondition::Running)
{
Gho1->runTimer->start();
Gho1->runMode->start();
}
if(Gho2->condition==ghostCondition::Running)
{
Gho2->runTimer->start();
Gho2->runMode->start();
}
if(Gho3->condition==ghostCondition::Running)
{
```

```
        Gho3->runTimer->start();
        Gho3->runMode->start();
        }
        if(Gho4->condition==ghostCondition::Running)
        {
        Gho4->runTimer->start();
        Gho4->runMode->start();
        }
        if(Gho1->condition==ghostCondition::Panic)
        {
        Gho1->escapeTimer->start();
        Gho1->panicModeTimer->start();
        }
        if(Gho2->condition==ghostCondition::Panic)
        {
        Gho2->escapeTimer->start();
        Gho2->panicModeTimer->start();
        }
        if(Gho3->condition==ghostCondition::Panic)
        {
        Gho3->escapeTimer->start();
        Gho3->panicModeTimer->start();
        }
        if(Gho4->condition==ghostCondition::Panic)
        {
        Gho4->escapeTimer->start();
        Gho4->panicModeTimer->start();
        }
}
```

```
void Game::restartGame()
{


    pauseGame();
    paused=false;

    escapeMenu->setPixmap(blank);
    Pac->lives=3;
    Pac->dead=false;
    Pac->won=false;
    h1->setPixmap(heart);
    h2->setPixmap(heart);
    h3->setPixmap(heart);
    Pac->score=0;
    updateScore();

    animationTimer->start(100);
    collisionTimer->start(0);

    QPixmap ball("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/ball.png");
    QPixmap
PowerBall("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall 2023/Applied
Data Structures/PAC/Objects/powerBall.png");
    int x=0,y=0,mapWidth=30,mapHeight=25;
    QFile file("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/map.txt");
```

```
if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
qDebug() << "Failed to open the text file";
return;
}
QTextStream in(&file);

for(int i=0;i<mapHeight;i++)
{
QString line = in.readLine();
for(int j=0;j<mapWidth;j++)
{
int X=x+(j*20);
int Y=y+(i*20);
if(line[j]=='0')
{

        gameMap[i][j]->objectType=type::Ball;
        gameMap[i][j]->setPixmap(ball);
        gameMap[i][j]->setPos(X,Y);
}
else if(line[j]=='4')
{
        gameMap[i][j]->objectType=type::PowerBall;
        gameMap[i][j]->setPixmap(PowerBall);
        gameMap[i][j]->setPos(X,Y);
}
}
}
file.close();
Gho1->condition=ghostCondition::Normal;
```

```
        Gho2->condition=ghostCondition::Normal;
        Gho3->condition=ghostCondition::Normal;
        Gho4->condition=ghostCondition::Normal;
        Gho1->setGhostPic();
        Gho2->setGhostPic();
        Gho3->setGhostPic();
        Gho4->setGhostPic();
        gameOver->setPixmap(blank);
        escdisplay->setPlainText(QString("Press [ESC] to Pause the
Game."));
        livesDisplay->setPlainText(QString("Lives:"));
        Gho1->released=false;
        Gho2->released=false;
        Gho3->released=false;
        Gho4->released=false;

        Pac->xcor=Pac->initx;
        Pac->ycor=Pac->inity;
        Pac->setPos(Pac->xcor*20,Pac->ycor*20);

        Gho1->xcor=Gho1->initx;
        Gho1->ycor=Gho1->inity;
        Gho1->setPos(Gho1->xcor*20,Gho1->ycor*20);

        Gho2->xcor=Gho2->initx;
        Gho2->ycor=Gho2->inity;
        Gho2->setPos(Gho2->xcor*20,Gho2->ycor*20);

        Gho3->xcor=Gho3->initx;
        Gho3->ycor=Gho3->inity;
```

```
        Gho3->setPos(Gho3->xcor*20,Gho3->ycor*20);

        Gho4->xcor=Gho4->initx;
        Gho4->ycor=Gho4->inity;
        Gho4->setPos(Gho4->xcor*20,Gho4->ycor*20);
}

void Game::releaseGhosts()
{

        if(!Gho1->released)
        {
        Gho1->chaseTimer->start(150);
        Gho1->released=true;
        }
        else if(!Gho2->released)
        {
        Gho2->chaseTimer->start(150);
        Gho2->released=true;
        }
        else if(!Gho3->released)
        {
        Gho3->chaseTimer->start(150);
        Gho3->released=true;
        }
        else if(!Gho4->released)
        {
        Gho4->chaseTimer->start(150);
        Gho4->released=true;
        }
```

```
}

bool Game::gameWon()
{
	for(int i=0;i<MapRows;i++)
	{
	for(int j=0;j<MapCols;j++)
	{
	if(gameMap[i][j]->objectType== type::Ball ||
gameMap[i][j]->objectType== type::PowerBall)
	{
		return false;
	}
	}
	}
	return true;
}

void Game::spawnPowerUp()
{
	if(powerBalls<4)
	{
	QPixmap
PowerBall("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall 2023/Applied
Data Structures/PAC/Objects/powerBall.png");
	int x=0,y=0;
	do{
	x = rand() % MapCols;
```

```
        y = rand() % MapRows;
        if(gameMap[y][x]->objectType==type::Blank)
        {
                gameMap[y][x]->objectType=type::PowerBall;
                gameMap[y][x]->setPixmap(PowerBall);
                powerBalls++;
                break;
        }
        }while(true);
        }

}

void Game::deleteTimer(QTimer *&timer)
{
        delete timer;
        timer = nullptr;
}

void Game::startPanicMode(Gh0st *Gho) {
        Gho->chaseTimer->stop();
        Gho->runTimer->stop();
        Gho->runMode->stop();
        Gho->condition=ghostCondition::Panic;
        Gho->panicModeTimer->start(5000);
        Gho->escapeTimer->start(150);
}
void Game::endPanicMode(Gh0st *Gho) {

        Gho->panicModeTimer->stop();
```

```
        Gho->condition=ghostCondition::Normal;
        Gho->escapeTimer->stop();
        Gho->chaseTimer->start();
}

void Game::startRunMode(Gh0st *g) {
        g->panicModeTimer->stop();
        g->escapeTimer->stop();
        g->runMode->start(8000);
        g->chaseTimer->stop();
        g->runTimer->start(150);
        g->condition=ghostCondition::Running;
}

void Game::endRunMode(Gh0st *Gho) {
        Gho->runTimer->stop();
        Gho->runMode->stop();
        Gho->condition=ghostCondition::Normal;
        Gho->chaseTimer->start();
}

void Game::keyPressEvent(QKeyEvent *ev)
{
        if(ev->key()==Qt::Key_Escape)
        {
        if(gameStarted)
        {
        if(!paused)
                pauseGame();
        else
```

```
            resumeGame();
        }
        }
        else if(ev->key()==Qt::Key_R)
        {
        if(gameStarted)
        {
        if(paused || Pac->dead || Pac->won)
            restartGame();
        }
        }
        else if(ev->key()==Qt::Key_Q)
        {
        if(gameStarted)
        {
        if(paused || Pac->dead || Pac->won)
        {
            mainMenu->setPixmap(start);
            gameStarted=false;
        }
        }
        }
        else if(ev->key() == Qt::Key_Enter || ev->key() ==
Qt::Key_Return)
        {
        if(!gameStarted)
        {
        gameStarted=true;
        mainMenu->setPixmap(blank);
        resumeGame();
```

```
restartGame();
}
}
else if(ev->key()==Qt::Key_End)
{
if(!gameStarted)
{
QCoreApplication::quit();
}
}
if(!paused)
{
if(!Pac->pacmoving)
{
if (ev->key() == Qt::Key_Up) {
        Pac->dir=direction::Up;
} else if (ev->key() == Qt::Key_Down) {
        Pac->dir=direction::Down;
} else if (ev->key() == Qt::Key_Left) {
        Pac->dir=direction::Left;
} else if (ev->key() == Qt::Key_Right) {
        Pac->dir=direction::Right;
}
}
Pac->pacTimer->start(150);
Pac->pacmoving=true;
if(Pac->pacmoving){
Pac->pacTimer->stop();
Pac->pacmoving = false;
if (ev->key() == Qt::Key_Up) {
```

```
            Pac->dir = direction::Up;
        } else if (ev->key() == Qt::Key_Down) {
            Pac->dir = direction::Down;
        } else if (ev->key() == Qt::Key_Left) {
            Pac->dir = direction::Left;
        } else if (ev->key() == Qt::Key_Right) {
            Pac->dir = direction::Right;
        }
        Pac->pacTimer->start(150);
        Pac->pacmoving = true;
        }
        }
}




void Game::chasePacman(Gh0st *Gho,int xcor,int ycor)
{
        if(!Pac->dead)
        {
        moveGhost(Gho,xcor,ycor,160);

        }
}
void Game::backToCage(Gh0st *Gho)
{
        moveGhost(Gho,Gho->initx,Gho->inity,160);
}
void Game::movePacman()
```

```
{

        int newX = Pac->xcor;
        int newY = Pac->ycor;
        direction initial;
        initial=Pac->dir;
        if (Pac->dir == direction::Up) {
        newY--;
        } else if (Pac->dir == direction::Down) {
        newY++;
        } else if (Pac->dir == direction::Left) {
        newX--;
        } else if (Pac->dir == direction::Right) {
        newX++;
        }
        if(initial!=Pac->dir)
        {
        Pac->pacTimer->stop();
        Pac->pacmoving = false;
        }
        else if (newX >= 1 && newX < MapCols-1 && newY >= 1 &&
newY < MapRows-1 &&
gameMap[newY][newX]->objectType!=type::Wall) {
        Pac->setPos(newX * GameObject::Width, newY *
GameObject::Width);
        Gho1->escapepath=true;
        Gho2->escapepath=true;
        Gho3->escapepath=true;
        Gho4->escapepath=true;
        if (Pac->dir == direction::Up) {
```

```
            Pac->moveUp();
        } else if (Pac->dir == direction::Down) {
            Pac->moveDown();
        } else if (Pac->dir == direction::Left) {
            Pac->moveLeft();
        } else if (Pac->dir == direction::Right) {
            Pac->moveRight();
        }
        if(gameMap[newY][newX]->objectType==type::Ball)
        {
        Pac->score+=ballScore;
        updateScore();
        }
        if( gameMap[newY][newX]->objectType==type::Ball ||
gameMap[newY][newX]->objectType==type::PowerBall)
        {
        if(gameMap[newY][newX]->objectType==type::PowerBall)
        {
            Pac->score+=powerBallScore;
            powerBalls--;
            p=g->findFarthestPoint(Pac->xcor,Pac->ycor);
            if(Gho1->condition==Normal)
            {
            if(Gho1->released)
            {
                startPanicMode(Gho1);
            }
            }
            if(Gho2->condition==Normal)
            {
```

```
        if(Gho2->released)
        {
                startPanicMode(Gho2);
        }
        }
        if(Gho3->condition==Normal)
        {
        if(Gho3->released)
        {
                startPanicMode(Gho3);
        }
        }
        if(Gho4->condition==Normal)
        {
        if(Gho4->released)
        {
                startPanicMode(Gho4);
        }
        }

}
gameMap[newY][newX]->setPixmap(blank);
gameMap[newY][newX]->objectType=type::Blank;
}
}
else
{
Pac->pacTimer->stop();
Pac->pacmoving = false;
}
```

```
}

void Game::collision()
{
        if(gameWon())
        {
        Pac->won=true;
        Gho1->chaseTimer->stop();
        Gho2->chaseTimer->stop();
        Gho3->chaseTimer->stop();
        Gho4->chaseTimer->stop();
        Pac->pacTimer->stop();
        animationTimer->stop();

        gameOver->setPixmap(gamewon);
        escdisplay->setPlainText(QString("[R] PLAY AGAIN   [Q] MAIN
MENU"));
        }
        if(Pac->lives==0)
        {
        Pac->dead=true;
        Gho1->chaseTimer->stop();
        Gho2->chaseTimer->stop();
        Gho3->chaseTimer->stop();
        Gho4->chaseTimer->stop();
        Pac->pacTimer->stop();
        animationTimer->stop();
        gameOver->setPixmap(gameover);
        escdisplay->setPlainText(QString("[R] PLAY AGAIN   [Q] MAIN
MENU"));
```

```
        livesDisplay->setPlainText(QString(""));
        }
        else
if((Pac->collidesWithItem(Gho1))||(Pac->collidesWithItem(Gho2))||(Pac
->collidesWithItem(Gho3))||(Pac->collidesWithItem(Gho4)))
        {
        if(Gho1->condition==ghostCondition::Panic &&
Pac->xcor==Gho1->xcor && Pac->ycor==Gho1->ycor)
        {
        Pac->score+=ghostScore;
        startRunMode(Gho1);
        }
        else if(Gho2->condition==ghostCondition::Panic &&
Pac->xcor==Gho2->xcor && Pac->ycor==Gho2->ycor)
        {
        Pac->score+=ghostScore;
        startRunMode(Gho2);
        }
        else if(Gho3->condition==ghostCondition::Panic &&
Pac->xcor==Gho3->xcor && Pac->ycor==Gho3->ycor)
        {
        Pac->score+=ghostScore;
        startRunMode(Gho3);
        }
        else if(Gho4->condition==ghostCondition::Panic &&
Pac->xcor==Gho4->xcor && Pac->ycor==Gho4->ycor)
        {
        Pac->score+=ghostScore;
        startRunMode(Gho4);
        }
```

```
        else if((Gho1->condition==ghostCondition::Normal &&
Pac->collidesWithItem(Gho1)) ||
(Gho2->condition==ghostCondition::Normal &&
Pac->collidesWithItem(Gho2)) ||
(Gho3->condition==ghostCondition::Normal &&
Pac->collidesWithItem(Gho3)) ||
(Gho4->condition==ghostCondition::Normal &&
Pac->collidesWithItem(Gho4)))
        {
        Pac->lives--;
        Pac->xcor=Pac->initx;
        Pac->ycor=Pac->inity;
        Pac->setPos(Pac->initx*20,Pac->inity*20);
        if(Pac->lives==2)
        {
                h3->setPixmap(blank);
        }
        else if(Pac->lives==1)
        {
                h2->setPixmap(blank);
        }
        else if(Pac->lives==0)
        {
                h1->setPixmap(blank);
        }

        }

        }
}
```

```
void Game::escapefromPacman(Gh0st *Gho)
{
	moveGhost(Gho,p.x,p.y,160);
}
void Game::moveGhost(Gh0st *Gho,int tarx,int tary,int time)
{
	Gho->escapepath=false;
	Cell start;start.x=Gho->xcor;start.y=Gho->ycor;
	Cell end;end.x=tarx;end.y=tary;
	QTimer *movementTimer=new QTimer(this);
	int currentMove = 0;
	std::vector<Cell> v1=g->dijkstra(start,end);
	connect(movementTimer, &QTimer::timeout, [=]() mutable{
	movementTimer->setInterval(time);
	if (currentMove < v1.size() && !paused) {

	if (v1[currentMove].x > Gho->xcor) {
	Gho->moveRight();}
	else if (v1[currentMove].x < Gho->xcor) {
	Gho->moveLeft();}
	else if (v1[currentMove].y > Gho->ycor) {
	Gho->moveDown();
		}
	else if (v1[currentMove].y < Gho->ycor) {
	Gho->moveUp();
	}
	++currentMove;
	} else {
	movementTimer->stop();
```

```
        }
        if(Gho->escapepath)
        {
        currentMove = 0;
        v1.clear();
        }
        });
        movementTimer->start();
        Gho->OutofCage=true;
}

void Game::updateScore()
{
        // Set the text for the score display
        if(Pac->score>Pac->highScore)
        {
        Pac->highScore=Pac->score;
        Pac->saveHighScore();
        }
        scoreDisplay->setPlainText(QString("Score:
%1").arg(Pac->score));
        highScoreDisplay->setPlainText(QString("High Score:
%1").arg(Pac->highScore));

        // Set the font and formatting
        QFont font("Comic Sans MS", 12, QFont::Bold);  // Adjust the
font size as needed
        scoreDisplay->setFont(font);
        highScoreDisplay->setFont(font);
        scoreDisplay->setDefaultTextColor(QColor(Qt::white));
```

```
    highScoreDisplay->setDefaultTextColor(QColor(Qt::white));
}


mapGraph::mapGraph(int r, int c)
{

    QFile file("C:/Users/3mrr3/OneDrive/Desktop/AUC/Fall
2023/Applied Data Structures/PAC/Objects/map.txt");
    if (!file.open(QIODevice::ReadOnly | QIODevice::Text)) {
    qDebug() << "Failed to open the text file";
    return;
    }
    QTextStream in(&file);
    rows=r;
    cols=c;
    Graph.resize(r, std::vector<Cell>(c));

    CellType cell;
    for(int i=0;i<rows;i++)
    {
    QString line = in.readLine();
    for(int j=0;j<cols;j++)
    {
    if(line[j]=='1')
    {
        cell=CellType::closed;
    }
    else
    {
```

```
            cell=CellType::open;
        }
        Graph[i][j].x=j;Graph[i][j].y=i;Graph[i][j].type=cell;
        }
        }
        file.close();
}

bool mapGraph::isValidCell(int x, int y) const
{
        return x >= 1 && x < cols-1 && y >= 1 && y < rows-1 &&
Graph[y][x].type!=closed;
}
const Cell &mapGraph::getCell(int x, int y) const
{
        return Graph[x][y];
}
//check starting here
bool mapGraph::areAdjacent(const Cell &cell1, const Cell &cell2) const
{
        return std::abs(cell1.x - cell2.x) + std::abs(cell1.y - cell2.y) == 1;
}

std::vector<Cell> mapGraph::getNeighbors(const Cell &cell) const
{
        std::vector<Cell> neighbors;
        for (int dx = -1; dx <= 1; ++dx) {
        for (int dy = -1; dy <= 1; ++dy) {
        int newX = cell.x + dx;
        int newY = cell.y + dy;
```

```
        if (isValidCell(newX, newY) && (dx == 0 || dy == 0) && !(dx ==
0 && dy == 0)) {
                neighbors.push_back(Graph[newY][newX]);
        }
        }
        }

        return neighbors;

}

std::vector<Cell> mapGraph::findShortestPath(const Cell &start, const
Cell &goal) const
{
        // A* algorithm implementation
        std::priority_queue<std::pair<int, Cell>,std::vector<std::pair<int,
Cell>>, std::greater<>> pq;
        std::vector<std::vector<int>> distance(rows, std::vector<int>(cols,
INT_MAX));
        std::vector<std::vector<Cell>> parent(rows,
std::vector<Cell>(cols, {-1, -1, open}));

        distance[start.x][start.y] = 0;
        pq.push({0, start});

        while (!pq.empty()) {
        auto [dist, current] = pq.top();
        pq.pop();
```

```
        if (current.x == goal.x && current.y == goal.y) {
        // Reconstruct the path
        std::vector<Cell> path;
        while (current.x != -1) {
                path.push_back(current);
                current = parent[current.x][current.y];
        }
        std::reverse(path.begin(), path.end());
        return path;
        }

        for (const auto& neighbor : getNeighbors(current)) {
        int newDist = distance[current.x][current.y] + 1;  // Assuming
uniform cost

        if (newDist < distance[neighbor.x][neighbor.y] && neighbor.type
!= closed) {
                distance[neighbor.x][neighbor.y] = newDist;
                parent[neighbor.x][neighbor.y] = current;
                pq.push({newDist + heuristic(neighbor, goal), neighbor});
        }
        }
        }

        return {};  // No path found
}

int mapGraph::heuristic(const Cell &a, const Cell &b) const
{
        return std::abs(a.x - b.x) + std::abs(a.y - b.y);
```

```
}
std::vector<Cell> mapGraph::dijkstra(const Cell &start, const Cell
&goal) const
{
	// Priority queue to store vertices to be processed, ordered by
distance
	std::priority_queue<std::pair<int, Cell>, std::vector<std::pair<int,
Cell>>, std::greater<>> pq;

	// Vector to store distances from the start cell
	std::vector<std::vector<int>> distance(rows, std::vector<int>(cols,
INT_MAX));

	// Vector to store parent information for reconstructing the path
	std::vector<std::vector<Cell>> parent(rows,
std::vector<Cell>(cols, {-1, -1, open}));

	// Initialize distance for the start cell
	distance[start.y][start.x] = 0;
	pq.push({0, start});

	while (!pq.empty()) {
	auto [dist, current] = pq.top();
	pq.pop();

	if (current.x == goal.x && current.y == goal.y) {
	// Reconstruct the path
	std::vector<Cell> path;
	while (current.x != -1) {
		path.push_back(current);
```

```
            current = parent[current.y][current.x];
        }
        std::reverse(path.begin(), path.end());
        return path;
        }

        for (const auto &neighbor : getNeighbors(current)) {
        int newDist = distance[current.y][current.x] + 1; // Assuming
uniform cost

        if (newDist < distance[neighbor.y][neighbor.x] && neighbor.type
!= closed) {
            distance[neighbor.y][neighbor.x] = newDist;
            parent[neighbor.y][neighbor.x] = current;
            pq.push({newDist, neighbor});
        }
        }
        }

        return {}; // No path found
}

Point mapGraph::findFarthestPoint(int x0, int y0)
{
        Point farthestPoint;
        double maxDistance = 0.0;

        for (int y = 0; y < Graph.size(); ++y) {
        for (int x = 0; x < Graph[y].size(); ++x) {
        // Skip closed cells (walls)
```

```
        if (Graph[y][x].type == closed) {
                continue;
        }
        // Calculate Euclidean distance
        double distance = std::sqrt(std::pow(x - x0, 2) + std::pow(y - y0,
2));

        // Update farthest point if the current point is farther
        if (distance > maxDistance) {
                maxDistance = distance;
                farthestPoint.x = x;
                farthestPoint.y = y;
        }
        }
        }

        return farthestPoint;
}
```

mainwindow header file

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include<QGraphicsView>
#include <QMainWindow>
#include"game.h"
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
```

```
class MainWindow : public QMainWindow
{
	Q_OBJECT

public:
	MainWindow(QWidget *parent = nullptr);
	~MainWindow();

private:
	Ui::MainWindow *ui;
	Game *game;
};
#endif // MAINWINDOW_H
```

mainwindow implementation file

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent)
	: QMainWindow(parent)
	, ui(new Ui::MainWindow)
{
	ui->setupUi(this);
	setWindowTitle(tr("pac-man game by the goats"));
	ui->graphicsView->setStyleSheet("QGraphicsView {border:
none;}");
	ui->graphicsView->setBackgroundBrush(Qt::black);
```

```
        ui->graphicsView->setFocusPolicy(Qt::StrongFocus);
        ui->graphicsView->setGeometry(0,0,28*30,21*25);

ui->graphicsView->setHorizontalScrollBarPolicy(Qt::ScrollBarAlways
Off);

ui->graphicsView->setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff)
;
        game=new Game();
        ui->graphicsView->setScene(game);
        setFocusPolicy(Qt::StrongFocus);
}

MainWindow::~MainWindow()
{
        delete ui;
}
```

main function

```
#include "mainwindow.h"
#include<QLabel>
#include <QApplication>

int main(int argc, char *argv[])
{
        QApplication a(argc, argv);
        MainWindow w;
```

```
    w.setStyleSheet("QMainWindow {background: 'black';}");
    w.show();
    return a.exec();
}
```
