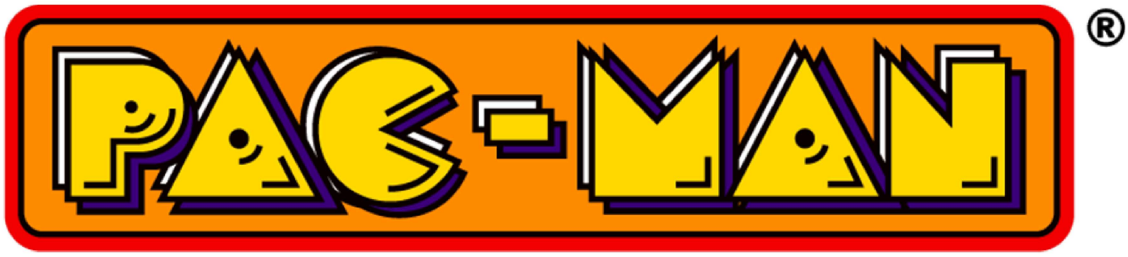
Hong Kong University of Science and Technology

2018-2019 Fall semester

COMP2012h-L01

Object-Oriented Programming and Data Structure

Programming Project:



To: Desmond Tsoi

From Au Chun Man Group Leader 20533728

Chan Chung Yin 20533481

Email auchunman.607.887@gmail.com

barry149149@gmail.com

Objective

Pac-man was a classic arcade video game developed by Namco in 1980. It was immensely popular since the day it was released. Its popularity is the worldwide level. In 1990, Pac-man has generated more than 2.5 billion in US quarters (Wolf, 2008). And throughout the years after, varies versions of Pac-man has been released, which, according to GameRevolution.com, has totally generate an amount of profit of 12 billion, after calculating the inflation.

This game is a great example of a successful game, but it playing is rather simple compared to the games nowadays. Pac-man consists of five characters, Pacman and four ghosts in different colours. The winning condition of Pacman is the Pacman eat up all the dots, which are separated on every part of the maze. However, the Pacman cannot collide with any moving ghosts, the collision will cause the deduction of the final score of the game as well as the life of Pacman. If the life of Pacman, started from three, has been deducted to zero, then the player will lose the game. Although the game seems simple, the four ghosts are actually chasing Pacman from behind, the players are required to eating the dots while escaping the ghosts which has greatly increased the difficulty of the games.

Our project is aimed to reform the games using C++ programming language with the GUI provided by using Qt Creator. It is expected that this game will have all the features of the original one while some additional features are added to the game.

Project Management

Developing Schedule

The development period is expected to be from 19th to 30th November. The table of schedule is below:

	No. of Day					
Activity	19th-20th	21th-22th	23th-24th	25th-26th	26th-27th	28th-29th
Design classes and algorithm						
Implementation of the game						
Design Test and Test						
Documentation						

Job Allocation

Au Chun Man: Data structure and Algorithm

Chan Chung Yin: Interface design and Map design

Tested Running Environment

OS for testing

- Window 10 (Family Edition) with MinGW
- MacOS Mojave

IDE

- Qt Creator 4.7.1 (Enterprise)

Qt

The Pacman game will be developed in the IDE provided in Qt Creator. Qt is a cross-platform developer which utilizing the C++ language. Qt is usually used for the development of the program with GUI, as well as those without. Qt is widely used worldwide, some big company like Samsung, Panasonic also using the Qt platform for their software developments.

Another feature of Qt is cross-platforming. Since our group members are using different kinds of OS (Window and MacOS). The platform chosen for development are required to be used in all these OS. Qt, by the way, support varies kind of platform which included Window, MacOS, Linux etc. It seems that Qt will be the most suitable IDE under consideration of using it in varies OS.

System design

Overall description

The program should be clear and required no command utilities. Every method of classes will be called based on the button in the UI. For the control of during the game, the character should be controllable via the direction keys on the keyboard and the game should not be disturbed even if the players have mistakenly pressed other buttons.

Window

The program initially is designed to be separated into two windows, the one handling the game and the one handling the starting of the game. However, since most of the game functions are not going to be dealt with the mouse and, in addition, it is preferred to have a higher response time as the time spent to finish the game is related to the score, the final decision is to combine both windows into one and separate it with different scenes. This design will provide a better transition between the start window and main game. Also, the KeyPress Event for the game control should also be set in the Window class for simplicity.

Map

Map of the game was one of the biggest issues in this game. While this game will execute under animations, the map of the game needs to support the smooth transition. And also it should be able to handle varies kinds of algorithms for seeking the short path. There Also on the map, it is necessary to place the dot and the Powerball on the map.

Pacman

Pacman is the main and the only controllable character inside the game. There are two major tasks for this class. First, it should store the information of the coordinate, and handle the change in information by the main gain system. Second, it has to handle the animation of itself which is the open and close of the mouth.

Ghost

Ghost class functions are similar to those of Pacman. The difference is the ghost also need to handle determine the colour of the character as well as there are four different colour ghosts.

Internal Game Control

The internal game will handle most of the action taken in the game. Inside this, there should be at least two Qtimers to handle the frequency of movements. And also, it will have to contain all the information about the game in order to pass it for the characters to calculate its moves. On the other hand, the score and other necessary data will be stored to provide players with some information they have to know.

Implementation

Common functions in the classes

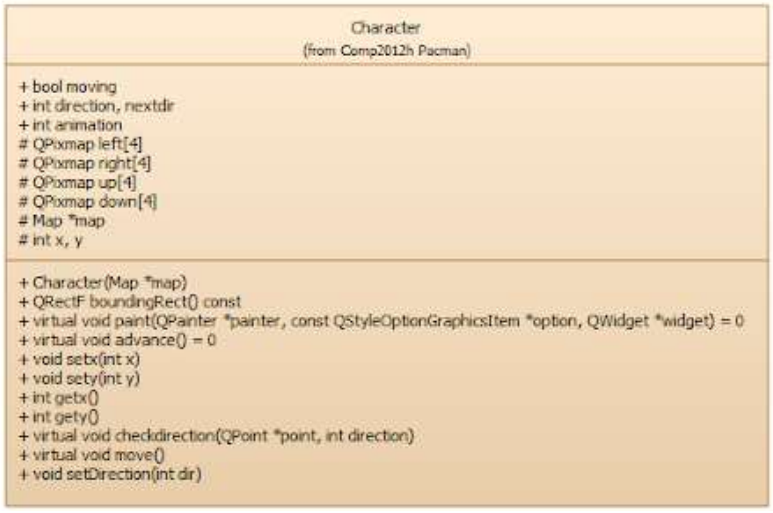
`QRectF boundingRect() const`: return QRect in corresponding position of fixed size

`void changeGeometry()`: call the QGraphicItem::prepareGeometryChange()

`void paint()`: paint the Pixmap on the corresponding position on the map.

`void advance()`: override the advance() function of the QGraphicItem. Will change the corresponding variable if animaiton is needed.

Character



Character class is the base class of two kinds of characters inside the game. The function of the variables are stated as below:

- `bool moving`: true when the character is moving, false when not
- `int direction`: the direction is saved as int, each value refer to one direction (assume no other value will be set into this variable)
    - 1: left
    - 2: up
    - 3: down
    - 4: right
- `int nextdir`: used to save the next direction same as the direction
- `int animation`: As the image of the character will change over time, this variable is set to help to choose the corresponding Pixmap
- `QPixmap left[], right[], up[], down[]` : these arrays are used to store the image of all image of the characters
- `Map* map`: the map is needed for the character to make the decision for the next move
- `int x, y`: store information of coordinate of the characters

Since Characters' variable value is distinct among each derived class, all the int and bool will be initialized as 0 in the constructor. But the pointer to the map will initialize at this point.

`setX(), setY(), getX(), getY(),setdirection()`

These function will return and set the corresponding variable.

```
void checkdirection(QPoint *point,int direction)
```

This function will determine and change the coordinate of the accepted point by using the direction accepted.

`void move()`

This function is the function for the character to change its coordinate. It will change to the variable `direction` to the value of `nextdir`. And the special case of moving will be handled in this function. For Instance, when the characters passing through the tunnel in the middle of the map, the move function will adjust the coordinate to another side of the tunnel. Also, it will stop character if the direction it moves has a wall. One special case is as the Pacman's next direction inputted by users may somehow be earlier, in order to accept this error, the next direction inputted will be stored until the character can move in that direction.

Pacman

`Pacman` class is the derived class of `Character` class. The constructor of Pacman class will set the coordinate and the direction to the suitable value. Also, images for the animation of Pacman will also be loaded in the constructor.

`void paint()`

The `paint()` works as the description above, but the image selected will vary with the variable animation.

`void advance()`

The `advance()` will loop the variable animation between 0-5.

Ghost

Ghost class is the another derived class from Character class. There is more variable in this class as the ghost in this game will have more texture than Pacman, and some variable is set for handling the algorithm of the AI. The constructor of the class will load the image differently, determined by the Color accepted.

`QPixmap scareb[], QPixmap scarew[]:`

These two arrays of `QPixmap` is to save the image of the ghost when the Pacman eat the power pellets. And the game will swap the image between two arrays in order to achieve animation.

`bool white, bool scared:`

For switch the image between the `QPixamp` array above.

`Color color`

`Color` is the enum created in this class. The value is stated below.

```
enum Color {PINK, ORANGE, RED, BLUE};
```

This variable will store the `color` of the ghost for changing texture as well as for AI to choose the distinct target.

`Pacman* pacman`

The ghosts need the information of the Pacman for the calculation of the path.

`bool corner`

This bool variable is for certain ghost to switch its target.

`void paint()`

The `paint()` works as the description above, but the image selected will vary with the variable animation.

`void advance()`

The `advance()` will loop the variable animation between 0-1.

`void initialmove()`

The function will provide the left and right movement of the ghosts at the beginning of the game.

void chase()

chase() is for the calculation of the next move of the ghost. At the starting point of the project, it is suggested that the ghost should use the BFS algorithm to handle the selection for

the next direction. However, the problem is the BFS will require a large amount of resource for calculation as well as the time for it. These disadvantages cause the frame rate of the game drop from 20fps to 3-4fps which is not playable at all. Therefore, another algorithm is suggested to be used. The principle of this algorithm is to find the closest next possible turning point to the target. The process of the calculation is as the following:

```
initialize the ghost target;
switch (color of ghosts){
     PINK
      Point at ~30pixel in front of the Pacman;
      If it is scared, to the bottom right corner;
     BLUE
      The next turning point of the Pacman;
      If it is scared, to the top left corner
     RED
      The point of Pacman;
      If it is scared, to the top right corner;
     ORANGE
      If it is less 100 unit distance from the Pacman, target
to the corner;
      else to the Pacman;
      If it is scared, to the bottom left corner;
}
when the ghost is not moving || at the turning point
      for any moveable direction
```

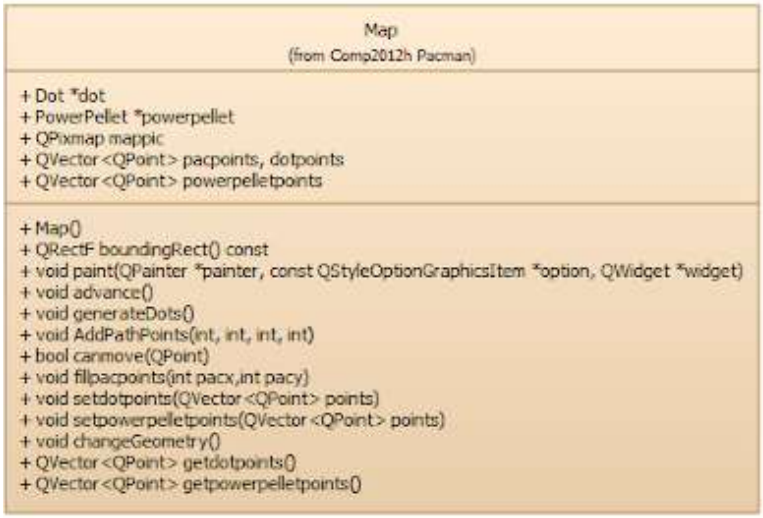
```
                find it next turning point
                calculate the distance from the target and the
turning point
          ban the reverse direction to avoid the looping motion;
          find the next turning point has the shortest distance
                change next direction to that turning point;
          Special case handling to avoid the ghost loop in a cycle;
```

One of the constraints of the algorithm is the path was not actually the shortest path. However, in return to have a smooth game, this disadvantage seems necessary. Also, because it only finds the point which has the shortest direct distance, there might be some action, like circulating among certain path, might happen. For these case, the special case handling will take care of it.

```
int mddistance(int,int,int,int)
```

This function will accept two sets of coordinate and calculation the Manhattan distance between two points.

Map



Map class contain the information on the game map, including the image of the map, all the path and dots as well as the power pellets on the maps.Inside the constructor of Map class, the constructor will initialize all point on the path one by one using AddPathPoint() and which will be stored in the `QVector<QPoint> pacpoints`. Also, it set the dot and the power pellets. For dots, for every 5 points on the path, then there will be a dot on it. And on the four corners on the map, the constructor will also place the power pellets on them.

`boundingRect()`

`boundingRect()` will return the `QRectF` in the size of 450*480. This will be used in the internal game control in other to refresh the map.

`paint()`

`paint()` is similar to all other `paint()` function, it will be called for drawing the map in the corresponding position on the window.

`advance()`

`advance()` has overridden the given function in the library to no use of the `advance()` function.

`generateDots()`

`generateDots()` is for generating the image of the dots and the power pellets on the map by using `QPainter` in every position of dots stored in the `dotpoints` and `powerpelletpoints`.

`canmove(QPoint)`

`canmove(QPoint)` will return true if the Qpoint exists in the vector `pacpoints`.

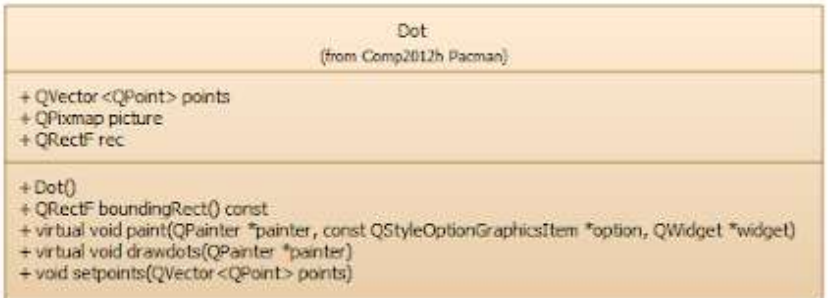
`AddPathPoints(int,int,int,int)`

`AddPathPoints(int,int,int,int)` will accept 2 coordinate's data. And it will then push every point between the two coordinates into the corresponding vector. Because it is assumed that all the path is either horizontal and vertical, the AddPathPoints() will only handle this two cases.

`fillpacpoint(int,int)`

`fillpacpoint(int, int)` is needed for refilling the colour of position of the power pellets after the Pacman eating it.

Dot



Dot class contain the information of all dots on the map, which include the position, and the image. The position will save as the class QPoint and stored in the QVector points. Each time when the dot is eaten by the Pacman, the internal game control will search for the corresponding objects in the QVector and pop it out

PowerPellet

PowerPellet class is the class inherit to Dot class. While it is necessary to separate the dot and power pellets, it is necessary to create one more class than simply identify in the internal system. There is nearly no difference between the two class except the images stored are different.

Log

`Log` Class is the class that handles the message output for informing the players. The constructor of the class will set all bool variable to false, score to 0, and the variable to the corresponding value for the scene refresh and message alignment.

```
void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
```

These function will update the scene as the mentioned above. And it also determines the suitable output message. If the game has not yet started, it will output "PRESS SPACE TO START". If the game is finished, it will out “WIN” or “GAME OVER” determined by whether the player has won the game or not, as well as, the score and the message "PRESS SPACE TO PLAY.".

MainWindow

`MainWindow` class is the class that handling the UI of the game as well as the flow control of the system. Most of the variable of this class is the pointer to the objects that necessary to the game but there is still some exception. The constructor of the class will initialize those variables that are not the pointer to the user-defined class object except the `Log` class. Those pointers, that have not been initialized, will be initialized inside the other functions.

`void checklost()`

`checklost()` is the function that checking the condition of the win or lost condition of the game. It will call the corresponding function for all condition happened on the Pacman.

`void startdelay()`

`startdelay()` will be called for some delay. The program will delay 2 seconds when it is called.

`void startGame()`

`start_Game()` will be called for all starting of the game to initialize all the object that is needed. Once the objects and variables are initialized, the function will start the timer and start the game.

`void end_Game()`

`end_Game()` will be called when the player wins or loses the game. It will remove the character from the window and some variable will be initialized again for the next game. Afterwards, it will refresh the screen to provide the message for the player.

`void set_lcdlife(int life), void set_lcdscore(int score)`

These function will update the corresponding LCD item on the window.

`void updater(), void ghostupdater(), void clock_updater()`

These functions are the updaters for moving of the object in game.

For `updater()`, it will be called when `timeout()` emitted by QTimer timer, each call of updater() will check the condition of the game at the beginning, and update the condition and value of the game by the item collided with the Pacman. The scared mode of the ghosts is also set in this function as well.

For `ghostupdater()`, it will be called when the `timeout()` emitted by `QTimer ghoststimer`, it will call the corresponding move function of each ghost object to do the next move of the ghost.

For `clock_updater()`, it will be simply called for updating the timer in the unit of 0.01 second.

`void KeyPressEvent(QKeyEvent* event)`

This function is for control of the game by the key pressed on the keyboard.

SpaceBar: start the game

Up: change the ghost next direction to up

Down: change the ghost next direction to down

Left: change the ghost next direction to left

Right: change the ghost next direction to right

Reference

Wolf, M. J. (2008). The video game explosion: A history from PONG to Playstation and beyond. Westport, CT: Greenwood Press.

World of Warcraft Leads Industry With Nearly $10 Billion In Revenue. (2017, January 26). Retrieved from https://www.gamerevolution.com/features/13510-world-of-warcraft-leads-industry-with-nearly-10-billion-in-revenue