贪吃蛇开发文档

作者：王浩天

学号：519021910685

日期：12/17/2020

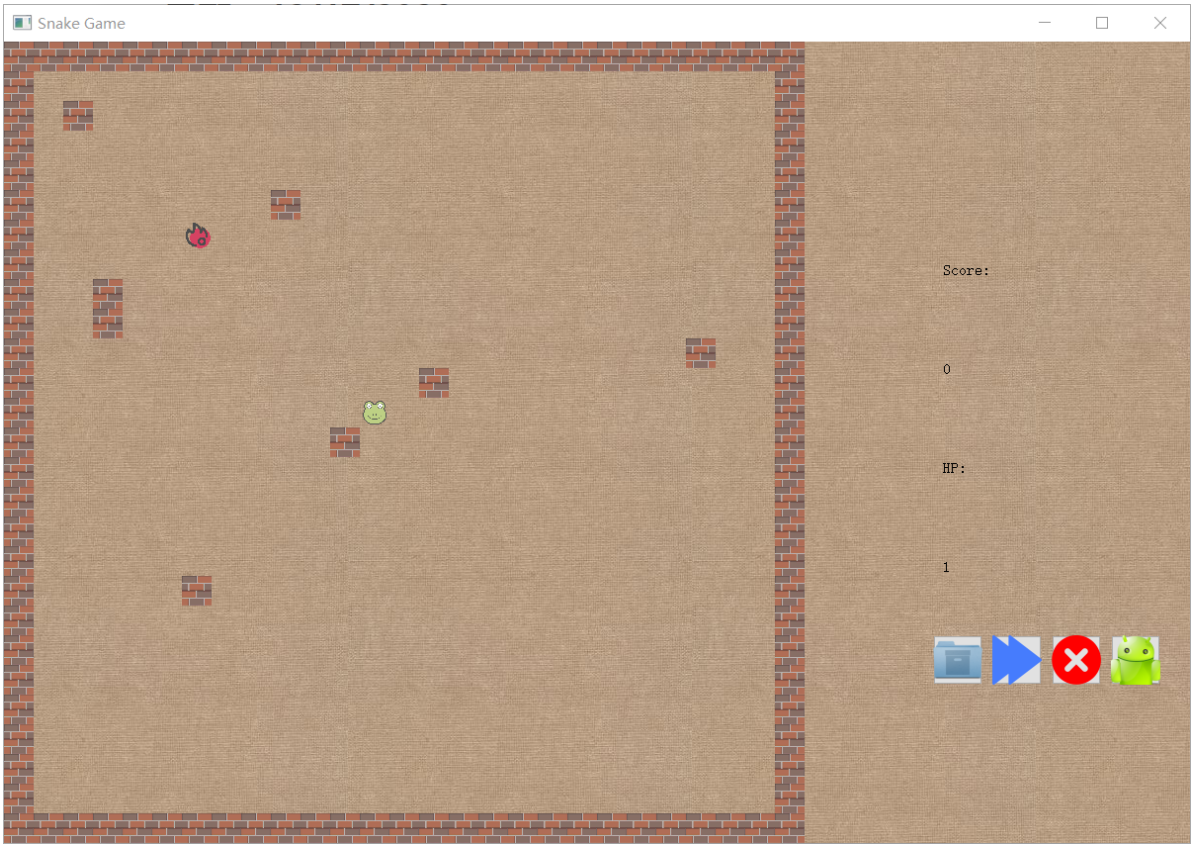
1. 游戏玩法

进入开始界面，共五个按钮

其中前两个按钮分别为单人和双人的开始新游戏，第三第四个按钮分别为单人和双人的继续游戏，最后一个按钮为退出游戏

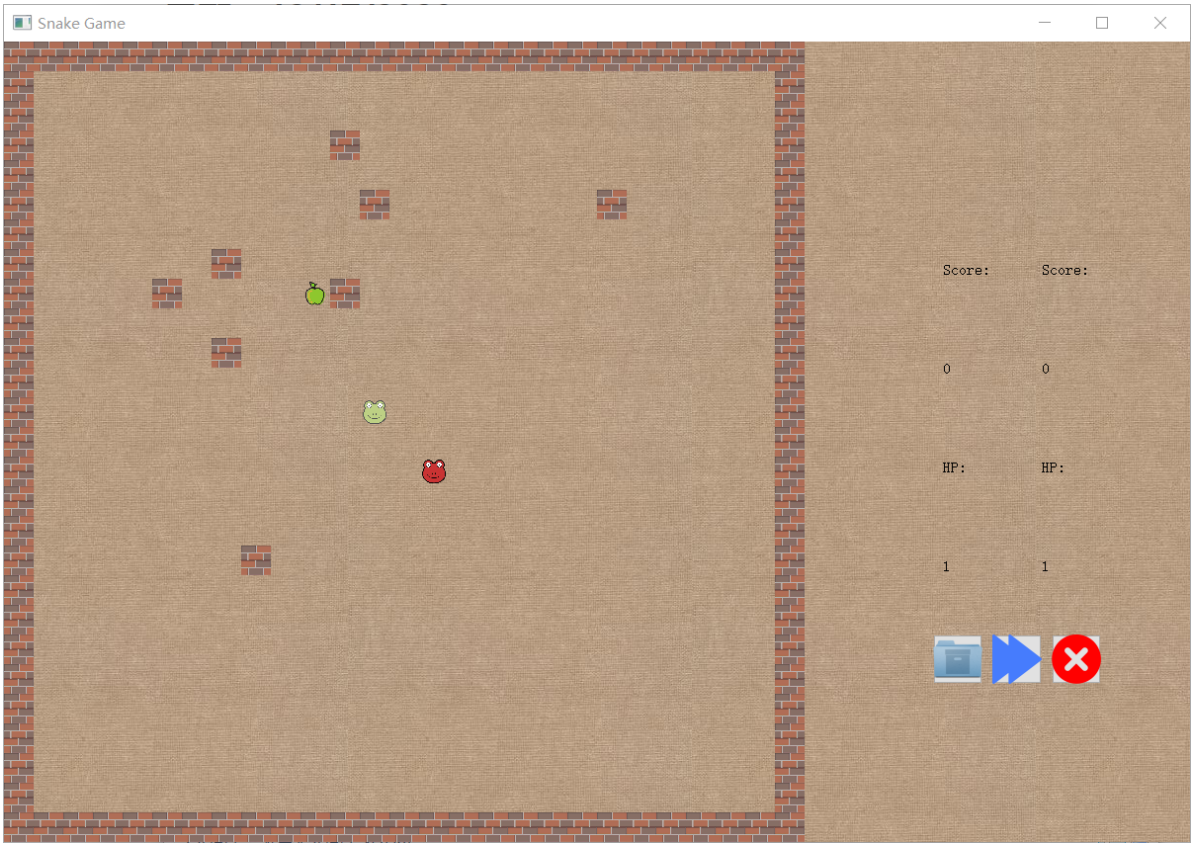
1.1 单人模式

第一个按钮为存档，第二个按钮为开始/继续游戏，第三个按钮为退出游戏，第四个按钮为自动游戏。



1.2 双人模式

第一个按钮为存档，第二个按钮为开始/继续游戏，第三个按钮为退出游戏。



1.3 按键和鼠标

`W A S D`和`I J K L`控制蛇的移动

在游戏暂停时可以点击障碍更改障碍的位置

在游戏暂停时可以点击食物更改食物的种类

普通食物仅增加得分

四种特殊食物分别实现加速、减速、双倍得分、增加生命

2. 头文件及类声明

2.1 Class Widget

```
class Widget : public QWidget
{
    Q_OBJECT

public:
    QPushButton *OneStartButton;
    QPushButton *TwoStartButton;
    QPushButton *OneContinueButton;
    QPushButton *TwoContinueButton;
    QPushButton *exitButton;

    Widget(QWidget *parent = 0);
    ~Widget();

private:
    SubWidget *gamewidget;

private slots:
    void OnePlayerStart();
    void TwoPlayerStart();
    void OnePlayerContinue();
    void TwoPlayerContinue();
    void mainExit();
};
```

Widget类声明了主窗口，会在进入游戏时打开。类内含有五个`QPushButton`成员变量，实现了五个按钮，通过声明的五个不同用途的槽函数定义五个按钮的具体行为。每个按钮的行为可通过变量名得知。

2.2 Class SubWidget

```
class SubWidget : public QWidget
{
    Q_OBJECT

    friend int Manhattan (int iniX, int iniY, int tarX, int tarY);
    friend int F(int x, int y, int tarX, int tarY, int G);

private:
    int player; // number of player
    Food *food_obj; // food object
```

```
    Snake *Snake_obj[2]; // snake object, if player = 1,there is only one snake.
    Obstacle *obstacle; // obstacle
    QImage imageWall; // image of obstacle and wall
    QImage imageWallLight; // image of selected obstacle

    bool pauseConnected; // a flag to judge if it is paused
    bool autoRunning; // a flag to judge if it is auto-running mode

public:
    std::vector<coordinate> open; // open list and closed list
    std::vector<coordinate> closed; // used in A-Star algorithm to implement auto-running mode
    std::stack<pathNode> path; // store autp-running mode's path data

    explicit SubWidget(int num, QWidget *parent = 0);
    explicit SubWidget(std::string &filename, QWidget *parent = 0);
    ~SubWidget();

    void keyPressEvent(QKeyEvent *k);
    void paintEvent(QPaintEvent *);
    void mousePressEvent(QMouseEvent *m);

    bool SnakeFree(int x, int y); // x,y exists snake or not
    bool ObstacleFree(int x, int y); // x,y exist obstacle or not
    bool FoodFree(int x, int y); // x,y exists food or not
    void SnakeReborn(int num); // snake will be reborn
    bool recursion(node *&tmp, int x, int y,
                   std::stack<int> &xstack, std::stack<int> &ystack);
    // called in function SnakeReborn
    // this is a DFS algorrthm to find a proper location

    // auto-running mode
    // this is a AStar algorithm of static version,
    // so when the snake is too long, it cannot find any path
    void AStar();
    bool valid(int x, int y); // x,y is vacant

    QLabel *ScoreTitle[2];
    QLabel *Score[2];
    QLabel *HPTitle[2];
    QLabel *HP[2];
    QTimer *Timer;
    QPushButton *buttonStart;
    QPushButton *buttonExit;
    QPushButton *buttonSave;
    QPushButton *buttonAuto;
private slots:
    void runSnake();
    void exitSnake();
    void pauseSnake();
    void saveSnake();
    void autoSnake();
};
```

SubWidget声明了子窗口，在主窗口点击按钮产生对应的事件后会打开对应的子窗口，在子窗口实现具体的贪吃蛇行为。具体行为见代码注释。

2.3 Class Snake

```
enum direction {up, right, down, left};

struct node
{
    int x;          // x coordinate
    int y;          // y coordinate
    node *next;     // next node's pointer
    QImage image;

    node(int x_tmp, int y_tmp, node *&p):x(x_tmp), y(y_tmp), next(p){};
    // for body and head
    node(int x_tmp, int y_tmp):x(x_tmp), y(y_tmp){next = nullptr;}
    // for tail
};

class Snake
{
    friend class Food;
    friend class SubWidget;
    friend class Obstacle;
private:
    int number;         // this number determines whether it's green or red
    node *head;         // snake's head
    node *tail;         // snake's tail
    node *pretail;      // snake's pre-tail
                        // (record tail's previous location to make it grow)
    direction dir;      // moving direction
    int HP;             // health point
    int score;          // your score
    int speed;          // moving speed
    //the greater the value of speed is, the slower the snake moves
    int interval;       // to control two snakes move;

    /* Special Food Attribute */
    void speedUp();     // speed Up
    void slowDown();    // slow down

    /* Judgement */
    // judge if the snake grows longer and do corresponding operation
    bool ifGrow(Food *&food);
    // judge if the snake is alive
    bool isAlive(Snake *&mysnake, Obstacle &obs, int flag);
public:
    Snake(int number = 0);
    Snake(int health, int numbner);
    Snake(int number, int length, std::vector<int> snake_data,int px, int
py,
          int DIR, int health, int SCORE, int SPEED); // for file input
    ~Snake();

    /* Operation */
    void toUp();        // shift to up
    void toRight();     // shift to right
    void toDown();      // shift to down
    void toLeft();      // shift to left
```

```
55
56      void move();        // move
57      void updateTail();  // update tail's image
58      bool reborn();      // reborn after death if HP>1
59  };
```

2.4 Class Obstacle

```
1   class Obstacle
2   {
3       friend class Snake;
4       friend class SubWidget;
5       friend class Food;
6   private:
7       int arrayX[8]; // there are eight obstacles in total
8       int arrayY[8];
9       int chosen;    // when editing obstacle, store the index of selected one
10  public:
11      Obstacle();
12      Obstacle(int *x, int *y);
13  };
```

2.5 Class Food

```
1   class Food
2   {
3       friend class Snake;
4       friend class SubWidget;
5   public:
6       Food(Snake **mySnake, Obstacle &obs, int num);
7       Food(int x, int y, int type);
8
9       void reset(Snake **mySnake, Obstacle &obs, int num); // reset location
    and type randomly
10      void changeFood(); // reset type of food
11      QImage image;
12  private:
13      int x;
14      int y;
15      foodType ftp;
16
17      bool SnakeFree(int x, int y, Snake **p, int num) const;
18      bool ObstacleFree(int x, int y, Obstacle &obs) const;
19  };
```

3. 游戏实现思路

3.1 蛇的实现

蛇的实现采取单链表的设计，每一个结点存储位置信息，并且根据该结点在蛇中的具体位置和方向加载不同图片，达到较好的可视化效果。Class Snake中存储蛇的状态（方向、速度、得分、生命）和一些辅助调控组件，并且声明了许多函数控制蛇的运动。

3.2 游戏开始菜单

通过 Class Widget 和 Class SubWidget 实现，Widget 是主窗口，提供不同的游戏模式和退出事件，通过QT事件和 QPushButton 实现，在点击特定按钮后会出现对应的子窗口和游戏模式（即 SubWidget）。而 SubWidget 定义了对应游戏模式的构造函数和成员变量，实现贪吃蛇的主要功能。

3.3 蛇的显示和移动

显示：蛇身体的每个结点有自己对应状态的 QImage 成员变量，通过调用 QPaintEvent 绘图绘制每一条蛇。

移动：在 Snake 类内有 move 函数，通过调用 move 函数，将蛇的每个结点坐标向目标位置移动。move 函数在runSnake中被调用，runSnake 为槽函数，于计时器信号相连，每经过一定时间，计时器发出 timeout 信号，runSnake 判断是否蛇是否移动（注意，计时器间隔固定，由于蛇的速度会产生变化，故在Snake 中增加了 interval 变量，每次 timeout 会使 interval 自增100，只有当 interval 等于 speed 时蛇才进行移动操作），若蛇移动，就调用 move 函数。

3.4 砖块和食物的显示与判定

砖块采取数组存储，定长为8，减少了代码编写的难度，在 QPaintEvent 遍历砖块数组，在每个相应位置绘图以显示障碍物。

食物在任意时刻有且仅有一个，故 food 类直接存储位置坐标并在 QPaintEvent 中绘图显示。

3.5 暂停、存档、读档与重新开始

暂停：停止计时器

存档：采用C++文件流，将所有信息以固定格式存储在 .txt 文件中，暂时只能存最近一次的游戏记录。

读档：存档的反向操作，将 .txt 文件中的特定信息解码，调用特定构造函数实现继续游戏操作。

重新开始：由于按钮过多会造成界面冗杂，重新开始的操作需要用户退出子界面，然后再次创建新的游戏即可重新开始。

3.6 单机多人游戏

目前仅支持双人游戏，在开始界面，点击双人游戏模式对应按钮即可进入双人游戏，在有已有双人游戏记录时，也可以点击继续双人游戏按钮，继续上一次的双人游戏。WASD 和 IJKL 分别控制绿蛇和红蛇的方向。

3.7 食物系统

食物共有五种，每种食物都会增加蛇身体长度，而食物的特殊效果如下：

普通食物，仅增加得分，无特殊效果；

加速食物，增加得分，增加蛇的移动速度；

减速食物，增加得分，降低蛇的移动速度；

增加生命，增加得分，增加一条生命，在蛇死后可以进行一次复活，复活采用深度优先算法，选择一条可用路径将蛇放入指定位置。

双倍得分，当前得分翻倍。

3.8 地图编辑

在游戏开始前或游戏暂停时，可以进行地图编辑，改变障碍物的位置（不可更改边界）和食物类型，实现思路如下：

通过 QT 的鼠标事件实现，判定当前点击位置是否存在可编辑地图属性（障碍物和食物）。

若存在食物，单击食物改变食物种类（顺序固定），但是食物位置不可更改。

若存在障碍物，单击此障碍物，此障碍物会被选中并呈现高亮状态，再次点击其他可用位置，会将选中的障碍物移动到目标位置，若再次点击的位置无效，则不会进行任何操作。

3.9 简单的AI蛇（A-Star算法实现）

这是我认为本次大作业较为具有挑战性的部分。

为了完成AI蛇，我在最开始尝试选用贪婪法，若当前前进方向的下一个位置无障碍物且蛇和食物的曼哈顿距离减小，就选取当前路径前进，但是在实现过程中发现，贪婪法容易陷入死胡同，故弃用。

二次尝试选取A-Star算法，将可行路径搜索出来，然后存储在 `std::stack<pathNode> path` 中，然后对每一个路径节点出栈，进行相应的转向操作。算法存在的不足是：由于采用静态寻路，坐标点状态随着蛇的运动而没有发生改变，因此虽然在前期可以找到路径，但是后期长度过大时，可能存在动态路径无法找到而报告不存在路径的错误。

实现代码如下

AStar 主函数

```
void SubWidget::AStar()
{
    open.clear();
    closed.clear();
    while(!path.empty()) path.pop();

    int iniX = Snake_obj[0]->head->x;
    int iniY = Snake_obj[0]->head->y;
    int tarX = food_obj->x;
    int tarY = food_obj->y;

    coordinate initial(iniX, iniY, iniX, iniY, 0);
    initial.F = F(initial.x, initial.y, tarX, tarY, initial.G);
    open.push_back(initial);

    while(true){
        if(open.empty()){
            QMessageBox::information(this, "ExitGame", "Found No Possible Way", QMessageBox::Yes);
            exitSnake();
        }

        std::vector<coordinate>::iterator search = closed.begin();
        for(;search != closed.end();++search){
            if((*search).x == tarX && (*search).y == tarY) break;
        }
        if(search != closed.end()) break;

        std::vector<coordinate>::iterator cur = open.begin();
        int curF = (*cur).F;
        for(std::vector<coordinate>::iterator itr = open.begin();
```

```
            itr != open.end();++itr){
            if((*itr).F < curF){
                curF = (*itr).F;
                cur = itr;
            }
        }

        coordinate curC = *cur; // current coordinate
        closed.push_back(curC);
        open.erase(cur);

        // 1
        if(valid(curC.x + 1, curC.y)){
            // in Closed or not
            std::vector<coordinate>::iterator itr = closed.begin();
            for(;itr != closed.end();++itr){
                if((*itr).x == curC.x + 1 && (*itr).y == curC.y) break;
            }

            // Do the following operation when current
            // coordinate's neighbor is not in Closed
            if(itr == closed.end()){ // not in Closed
                itr = open.begin();
                for(;itr != open.end();++itr){
                    if((*itr).x == curC.x + 1 && (*itr).y == curC.y) break;
                }

                // not in Open
                if(itr == open.end()){
                    coordinate tmp(curC.x + 1, curC.y, curC.x, curC.y,
curC.G + 1);
                    tmp.F = F(tmp.x, tmp.y, tarX, tarY, tmp.G);
                    open.push_back(tmp);
                }else{ // already in open
                    if(curC.G + 1 < (*itr).G){
                        (*itr).parentx = curC.x;
                        (*itr).parenty = curC.y;
                        (*itr).G = curC.G + 1;
                        (*itr).F = F((*itr).x, (*itr).y, (*itr).parentx,
(*itr).parenty, (*itr).G);
                    }
                }
            }
            // if the current coordinate's beighbor is in Closed, skip it
        }

        // 2
        if(valid(curC.x, curC.y + 1)){
            std::vector<coordinate>::iterator itr = closed.begin();
            for(;itr != closed.end();++itr){
                if((*itr).x == curC.x && (*itr).y == curC.y + 1) break;
            }

            if(itr == closed.end()){
                itr = open.begin();
                for(;itr != open.end();++itr){
                    if((*itr).x == curC.x && (*itr).y == curC.y + 1) break;
                }
```

```
87
88                  if(itr == open.end()){
89                      coordinate tmp(curC.x, curC.y + 1, curC.x, curC.y,
    curC.G + 1);
90                      tmp.F = F(tmp.x, tmp.y, tarX, tarY, tmp.G);
91                      open.push_back(tmp);
92                  }else{
93                      if(curC.G + 1 < (*itr).G){
94                          (*itr).parentx = curC.x;
95                          (*itr).parenty = curC.y;
96                          (*itr).G = curC.G + 1;
97                          (*itr).F = F((*itr).x, (*itr).y, (*itr).parentx,
    (*itr).parenty, (*itr).G);
98                      }
99                  }
100             }
101         }
102
103         // 3
104         if(valid(curC.x - 1, curC.y)){
105             std::vector<coordinate>::iterator itr = closed.begin();
106             for(;itr != closed.end();++itr){
107                 if((*itr).x == curC.x - 1 && (*itr).y == curC.y) break;
108             }
109
110             if(itr == closed.end()){
111                 itr = open.begin();
112                 for(;itr != open.end();++itr){
113                     if((*itr).x == curC.x - 1 && (*itr).y == curC.y) break;
114                 }
115
116                 if(itr == open.end()){
117                     coordinate tmp(curC.x - 1, curC.y, curC.x, curC.y,
    curC.G + 1);
118                     tmp.F = F(tmp.x, tmp.y, tarX, tarY, tmp.G);
119                     open.push_back(tmp);
120                 }else{
121                     if(curC.G + 1 < (*itr).G){
122                         (*itr).parentx = curC.x;
123                         (*itr).parenty = curC.y;
124                         (*itr).G = curC.G + 1;
125                         (*itr).F = F((*itr).x, (*itr).y, (*itr).parentx,
    (*itr).parenty, (*itr).G);
126                     }
127                 }
128             }
129         }
130
131         // 4
132         if(valid(curC.x, curC.y - 1)){
133             std::vector<coordinate>::iterator itr = closed.begin();
134             for(;itr != closed.end();++itr){
135                 if((*itr).x == curC.x && (*itr).y == curC.y - 1) break;
136             }
137
138             if(itr == closed.end()){
139                 itr = open.begin();
140                 for(;itr != open.end();++itr){
```

```
                    if((*itr).x == curC.x && (*itr).y == curC.y - 1) break;
                }

                if(itr == open.end()){
                    coordinate tmp(curC.x, curC.y - 1, curC.x, curC.y,
curC.G + 1);
                    tmp.F = F(tmp.x, tmp.y, tarX, tarY, tmp.G);
                    open.push_back(tmp);
                }else{
                    if(curC.G + 1 < (*itr).G){
                        (*itr).parentx = curC.x;
                        (*itr).parenty = curC.y;
                        (*itr).G = curC.G + 1;
                        (*itr).F = F((*itr).x, (*itr).y, (*itr).parentx,
(*itr).parenty, (*itr).G);
                    }
                }
            }
        }
    }
```

计算曼哈顿距离

```
int Manhattan(int x, int y, int tarX, int tarY)
{
    int deltaX = ((x - tarX) >= 0) ? (x - tarX) : (tarX - x);
    int deltaY = ((y - tarY) >= 0) ? (y - tarY) : (tarY - y);
    return (deltaX + deltaY);
}
```

计算代价函数

```
int F(int x, int y, int tarX, int tarY, int G)
{
    return (G + Manhattan(x, y, tarX, tarY));
}
```

coordinate 和 pathNode 结构体

```
struct coordinate{
    int x;
    int y;
    int parentx;
    int parenty;
    int G;
    int F;

    coordinate(int q, int w, int e, int r, int g){
        x = q;
        y = w;
        parentx = e;
        parenty = r;
        G = g;
    }
};

```

```
18 struct pathNode{
19     int x;
20     int y;
21 
22     pathNode(int m, int n):x(m), y(n){};
23 };
```

4. 项目展望

作为个人亲自动手并完全由个人完成的第一个大作业，完成度较高，但是存在一些不足。如程序的鲁棒性不强，代码风格不够简洁，UI界面美化程度较低等。在项目的继续改进过程中可以继续完善和发展。