C 语言程序设计:*凤囚凰*

摘要

我们的这款游戏 凤囚凰（ALL FOR LOVE）是根据电视剧《凤囚凰》的故事背景进行设计的。我们小组中的每一个成员对于这个电视剧的情节都有不一样的看法，所以我们决定设计制作这款游戏，让更多的玩家进行抉择与判断，迸发出思想的火花。

对于每一个玩家来说，在真正开始游戏之前，他们都会先了解游戏的背景介绍，再结合自己对不同人物的不同的感情，进行游戏模式的选择。

在游戏的编程中，我们认为需要解决一些核心问题。第一个问题是，在我们构建游戏的过程中，图形表面是非常重要的，那么我们该怎么把背景，人物等一系列图片加到代码中。然后我们提出了该怎么实现人物与人物之间最短路径的计算的问题。之后我们遇到了控制敌人和拯救者的移动，跳跃，捡拾物品等问题。当然，在游戏的形成过程中还有很多大大小小的问题，这三个问题是最基本的。

在我们的这款游戏中，会有两个地图，也即两个模式供玩家尝试。尝试结束后我们还可以根据玩家的游戏结果提供给玩家不同的选择。

1. 简介与问题描述

电子游戏曾经给我们的童年带来无数的欢乐，项目开始之初，我们小组决定用 C 语言编写一款躲避与追捕的游戏。游戏剧情中，国王被敌人俘获，而同时爱慕着国王的王后和女将军都打算深入敌营，只身救出国王。开场动画带入以后，玩家通过选择不同的角色（王后或女将军）进入不同的模式。游戏过程中，玩家需要通过键盘操作控制玩家移动，通过躲避，攻击敌人，拿到钥匙后成功开启宝箱，游戏即判定胜利；如果敌人与玩家重合，游戏即判定角色死亡，游戏失败。

除了剧情与游戏规则，我们也希望能够制作出友好美观的用户交互界面，但是图形化的界面设计方面，一直属于 C 语言的弱项。结合实际的项目要求，经过多方面比对，我们选择了通过使用游戏引擎的方式完成交互界面与地图的设计，并制作了精美的开场动画，以将玩家更好的代入游戏之中。

此外，人物与敌人的控制也是我们所面临的问题之一，通过利用游戏引擎中的部分函数，并查找资料，参考学习源代码，以及对算法的使用，我们成功实现了对玩家移动的控制，以及敌人的自动寻路与移动等功能。

2．分组划分

名字	学号	学院	工作百分比
李成午	11170521	物理	40%
郝新培	13170202	生物	30%
王鹤翔	12180426	化学	30%

1）李成午

李成午是小组的组长，安排了项目的分工，建立了项目的框架并给出了程序的设计方案。李成午主要负责这个程序中的控制功能，包括程序的流程以及控制游戏中各个物品与角色的代码编写。最后，他整合了其他小组成员编写的代码。

2）郝新培

郝新培主要负责制作这款游戏的辅助引擎的选择（如鹏引擎）以及对游戏界面的设计，主要是人物图片，背景图片的搜索、处理以及插入转换和对地图的设计、绘制与整合。

3）王鹤翔

王鹤翔提供一个寻找最短路径的算法，用以规划敌人的路径运动，完成了流程图的制作与后期调试。

3.分析

3.1 游戏逻辑与游戏规则

首先，我们需要确定游戏逻辑与游戏规则。游戏的目标十分简单，即玩家利用键盘的“↑”“←”“→”键（或者“W”“A”“D”“J”键）操控游戏人物移动，躲避追杀，或者按下“K”键发射子弹消灭敌人。最终取得钥匙，打开宝箱，使游戏取得胜利。另外，还要对地图中不同方块的类型与合法性作出规定，比如，因为游戏中使用了“爬楼梯”类型的地图，玩家和敌人跳跃时不能在空中停留过长时间，不能通过障碍物与边界方块等等。

概括游戏规则如下:

(1)键盘控制人物移动与子弹的发射

(2)玩家需要得到钥匙，开启宝箱，赢得胜利

(3)敌人（僵尸）能够察觉到玩家并进行追捕

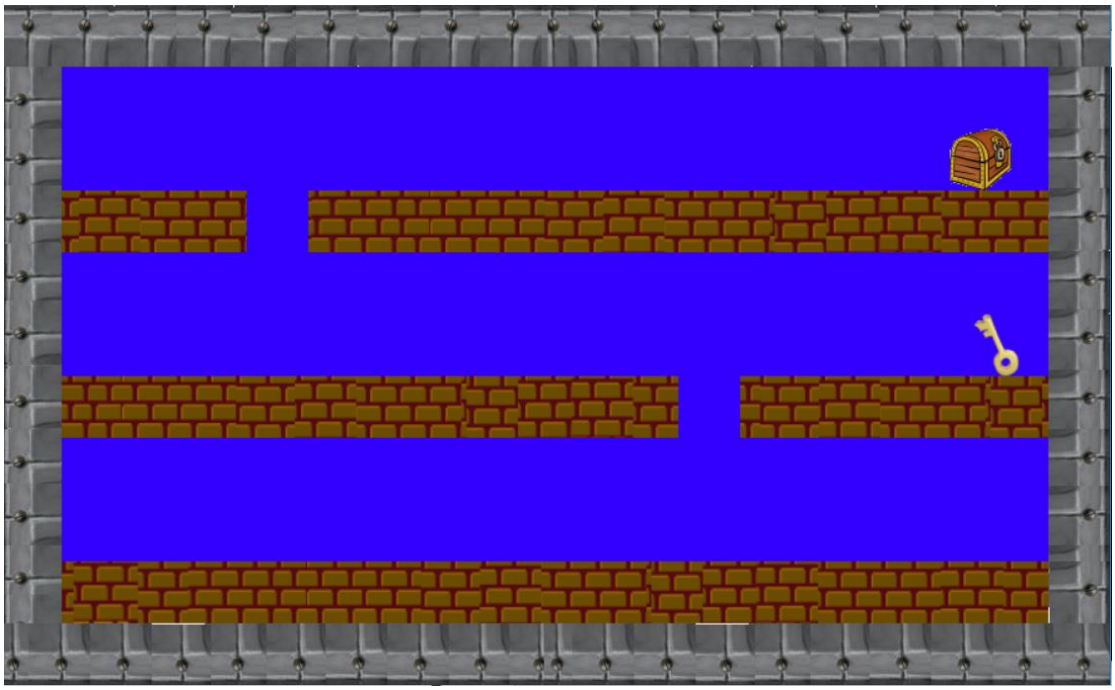
(4)玩家与敌人接触即失败

(5)宝箱打开、游戏胜利

3.2 界面与游戏地图绘制

建立游戏界面，绘制人物、地图、背景。作为一个游戏，它当然需要一个图形化的表面来与玩家互动。更重要的是,游戏地图必须完全显示在地图上。因此，我们进行了对地图的设计和绘制，然后进行网络搜集，搜索我们所需要的素材如墙体、边框、人物、背景等等进行加工处理，之后再利用如鹏引擎实现对图片的插入和利用。我们也进行了界面坐标的设计，每一个图片都有其对应的代号以及坐标，我们通过它们的坐标控制界面上的一切。

界面大小为 900×550，我们将其网格化为一张 18×11 的网格，每个网格填充一张 50×50 的图片。图片在地图上的坐标（以下简称坐标）为由其左下角的顶点在网格中的坐标。值得注意的是玩家的 x 坐标与其他图片的坐标不同，是一个 0-899 的整数,经过((int)player.x/LENGTH)换算之后的坐标才是与前述统一的地图坐标。这么做是为了提升 player 在左右移动时的流畅度。左下角为原点(0,0)，向右、向上方向分别为 x、y 轴的正方向。在使用游戏引擎加载图片的时候，只需要将图片的坐标乘以 50 即为图片在界面上的坐标。



(图 1 游戏界面：地图)

3.3 控制

游戏的控制是程序组成的基本部分，在这方面，我们面临的主要问题是实现对玩家与敌人移动的控制。首先，针对不同的方块类型，比如地图边界、障碍物、空中、可行走的路径等等，我们利用定义全局变量与明示常量的方式加以分类。玩家控制方面，我们需要从键盘读取输入，实现对玩家的移动，尤其是在如何实现玩家跳跃的重力效果（即在空中会下落，但可以在地面上停留）需要解决一系列问题。此外，玩家还应该可以射击子弹，与敌人对抗。子弹的速度应该适宜，人物应当可以捡拾钥匙，打开宝箱。

对于敌人来说，最重要的功能是实现针对玩家的追逐，因为使用网格化地图，在这里我们对地图中每一个点的位置与类型进行标注，并使用了 A*算法，以实现两点之间最短路径的获取，并通过特定的函数根据返回的坐标值控制人物移动。除此之外，敌人受到攻击之后还会减少生命值，同时也会随机恢复，当生命值降低到 0 时敌人死亡。

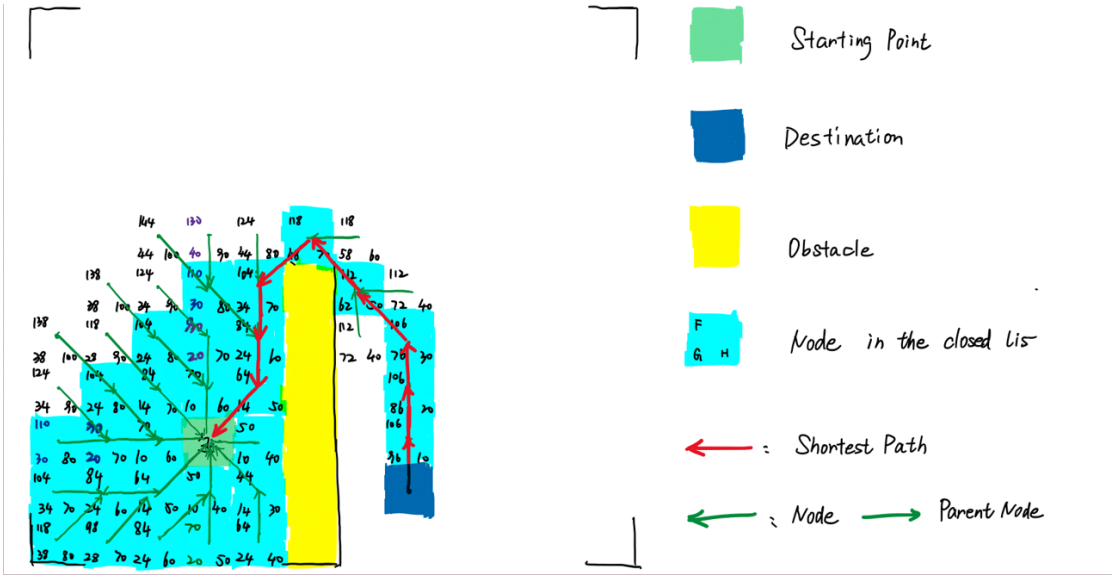


图 2　A*算法示例

3.4 游戏胜负判断

当玩家成功拿到钥匙打开宝箱时即判定游戏成功，并播放成功动画，并给出玩家选项，是在玩一次还是返回菜单。当玩家被敌人碰到时，玩家失去生命，游戏结束，播放结局动画。

图 3 游戏结束场景：左-胜利 右-失败

4.设计

4.1 游戏逻辑

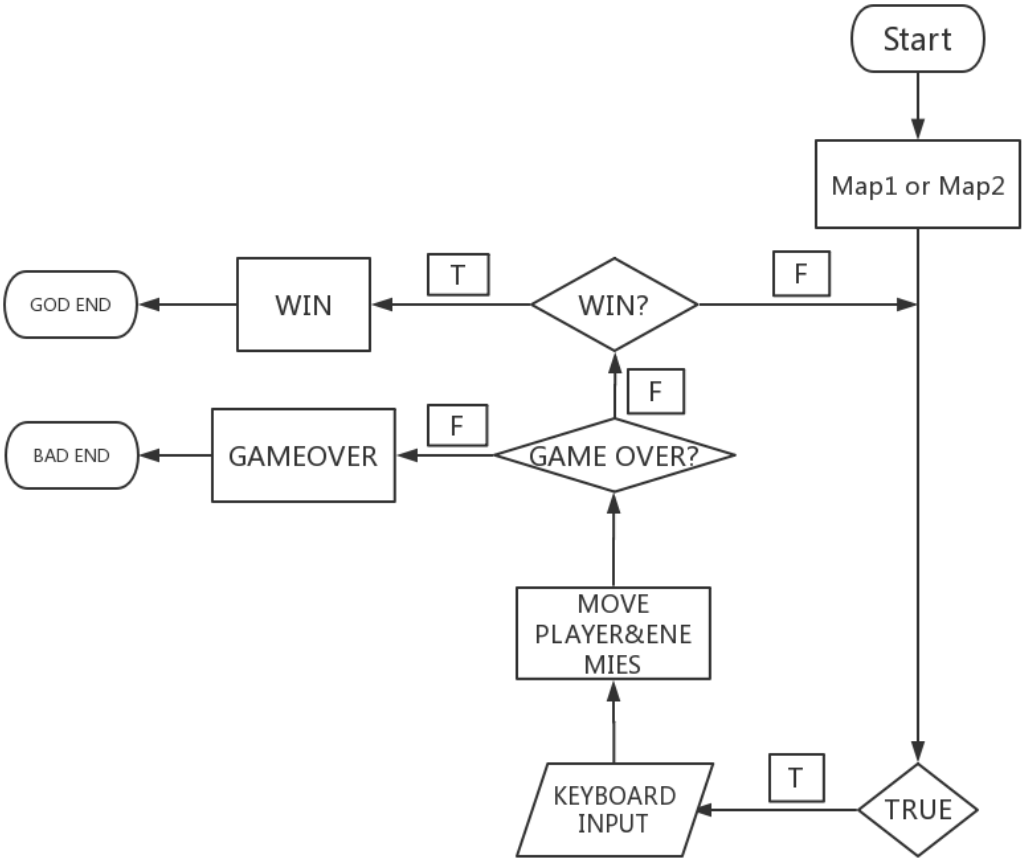
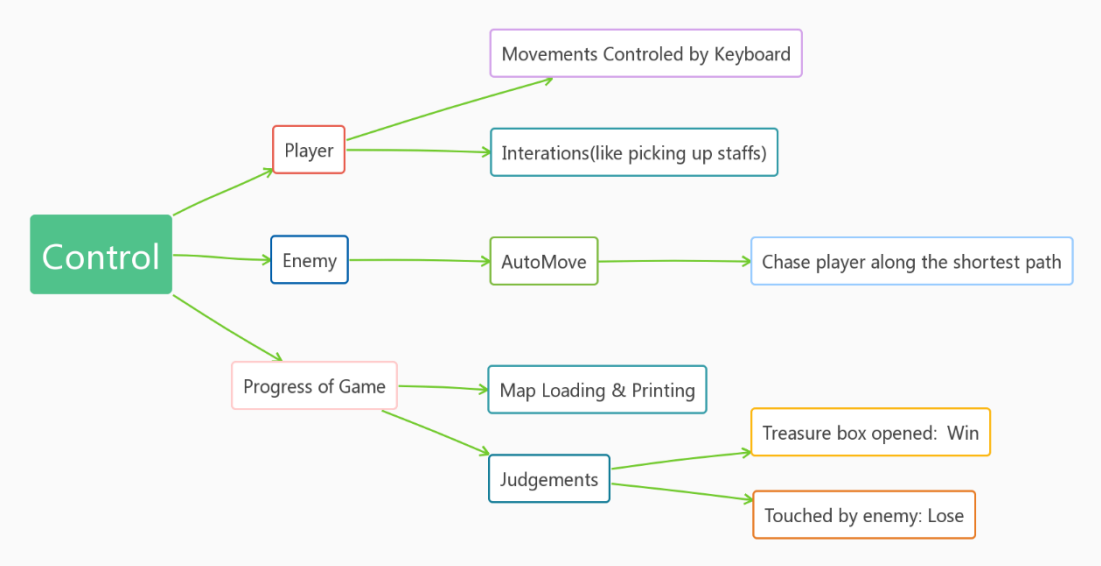


图 4 游戏总流程图

游戏是通过单线程顺序结构运行的。进入选择界面后，根据玩家的不同选择，进入不同的模式。然后，读取键盘输入，并根据输入控制玩家的移动、跳跃和射击。之后进行敌人到玩家最短路径的寻找并按照路径移动敌人。然后游戏进入胜负判定阶段，如果敌人与玩家发生接触（两者坐标重合），那么游戏即为失败，跳出循环体，否则判断是否游戏胜利，如果胜利则跳出循环体，播放胜利动画，否则，继续循环，这个过程不断重复，直到两个判定条件有一个成立为止。这便是整个游戏的大概运行过程。

4.2 游戏控制

关于控制部分，大致模块如下图：



从总体上看，游戏控制方面主要由针对玩家、敌人、游戏进程的控制三部分组成。各部分应当实现的功能已在分析部分中提及，不必赘述。这里主要讲一讲具体功能的设计思路与解决办法。

4.2.1 程序进程控制

首先播放程序的开始界面，根据操作者的选择进入游戏的不同模式。在进入游戏循环之前，首先创建并初始化所有变量并创建一个敌人的链表。从一个“.txt”文本文件中读取地图数组，并填充到游戏界面中。

游戏开始进入循环以后，利用一个变量控制游戏进行的速度，接着依次执行人物与敌人的控制函数，并刷新地图上的物品的状态。在每次游戏的有效循环末尾对游戏的胜利条件进行判断，判断是否进入游戏的结束界面。

4.2.2 人物控制

创建人物的结构体，在有效循环中每隔一定的循环次数读取一次键盘输入，并根据键盘输入调用不同的函数。

键盘输入的数据被分为以下四类：

1. 对玩家进行左右移动的操作
2. 对玩家进行跳跃的操作
3. 创建一枚子弹
4. 退出游戏

其中 2.3.类利用如鹏游戏引擎实现。

在左右移动的函数中首先根据输入判断移动方向，检查移动方向上是否有障碍物，再更改玩家与界面中玩家图片的坐标。

跳跃函数也是先检查向上的方向上是否有障碍，将玩家的坐标和图片向上移动，这样的过程将重复几次，使玩家可以跳跃到一定高度。再使玩家下降，直到碰到地面为止。

创建子弹的函数只是根据玩家当前的朝向，在玩家前方创建一枚子弹，不过这时子弹**还没有**出现在地图界面中，在之后的子弹刷新函数中逐个刷新并判断子弹的状态。

按下 ESC 键可以直接退出游戏。

除此之外还有一些与人物有关的函数：

1.上下左右四个方向的四个检查函数：将玩家的地图坐标代入地图数组，分别检查上下左右四个方向上是否有障碍物。

2.碰撞函数（玩家）：检查玩家是否与地图上的物品，某个敌人的坐标重回。

3.重力函数：确保玩家站在地面上，否则就把玩家及玩家图片的纵坐标减一。

4.子弹的飞行与碰撞：

对每个存在于地图上的子弹操作。此时有两种情况：新创建的子弹与之前就存在于地图上的子弹。

分成两种情况的原因是对于新创建的子弹来说，子弹并不存在与地图上，它的位置上是否有障碍是未知的；而对于不是新创建的子弹来说，移动之前子弹就已经在地图上，需要先将子弹向子弹的方向移动，再进行判断位置上是否有障碍。

先把已经存在于地图上的子弹的坐标根据其移动方向递增；把新创建的子弹标记为旧的子弹。再对其现在的位置进行障碍判断（注意此时地图上的子弹图片还没有或者还在原来位置）：将其地图上的图片移动到当前位置或者执行碰撞函数，清除其在地图上的图片。

子弹的碰撞函数：判断子弹的位置的东西并进行相应操作

4.2.3 敌人的控制

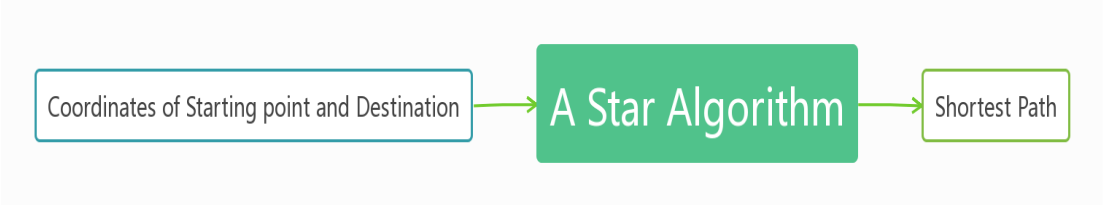
创建一个链表储存敌人的信息。在有效的游戏循环中使用 A*算法得出移动路径并进行移动，再每次刷新敌人的生命值，消除生命值为 0 的敌人。

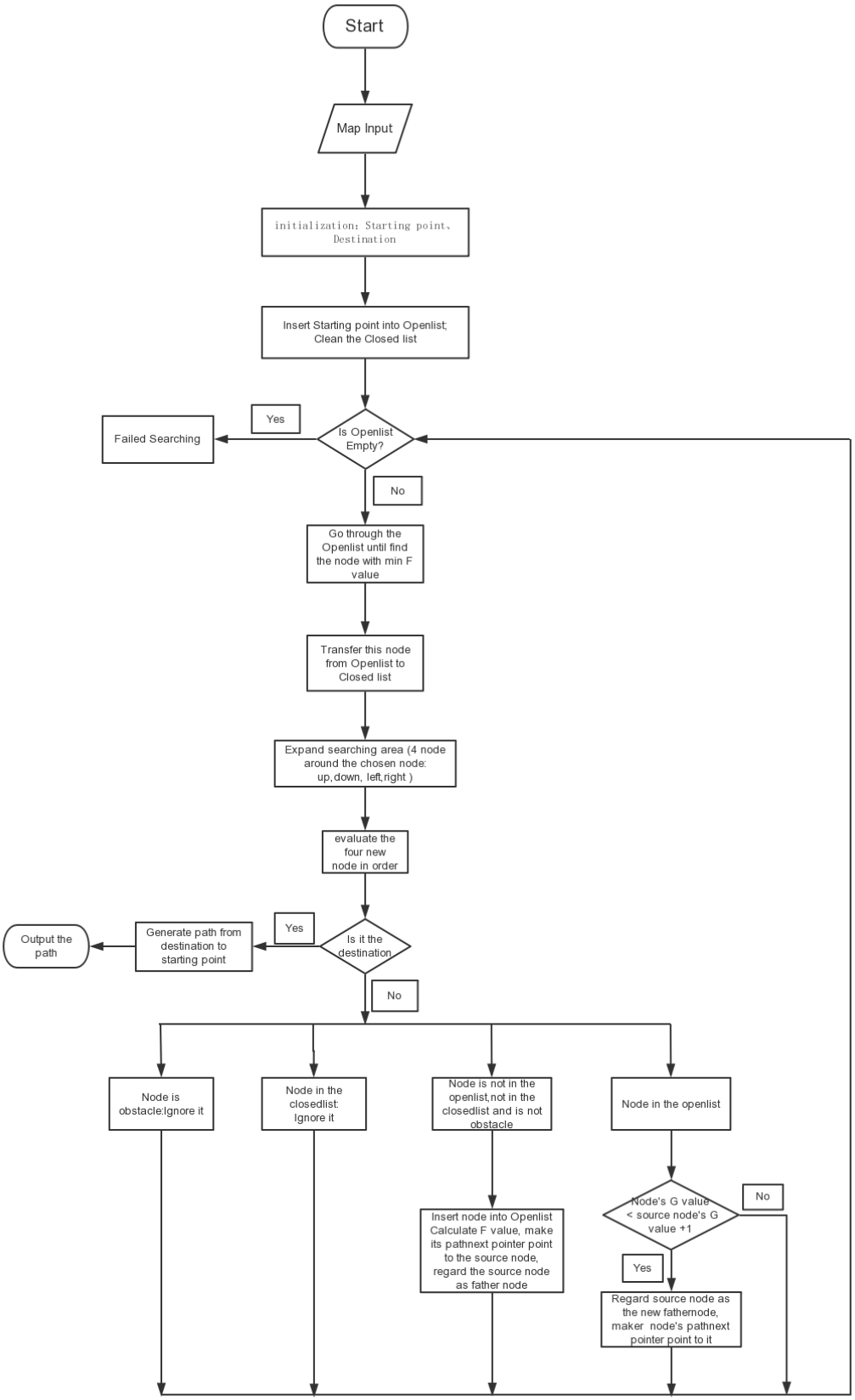
4.2.4 A*算法：最短路径的寻找

游戏过程中，我们想要实现敌人对玩家的高效率追击，就需要敌人有能力找到自己与玩家之间的，合法的最短路径，并根据路径进行移动到达玩家位置。基于这样的一种想法，我们开始寻找适合我们小组使用的一种算法，来帮助我们达成目的。A*算法作为一种经典的，启发式搜索寻路算法已广泛地被人们使用。他同时避免了只考虑局部最优解而无法取得正确最短路径，以及深度或广度优先算法搜索全图时间较长的问题，算法流程图在下页.

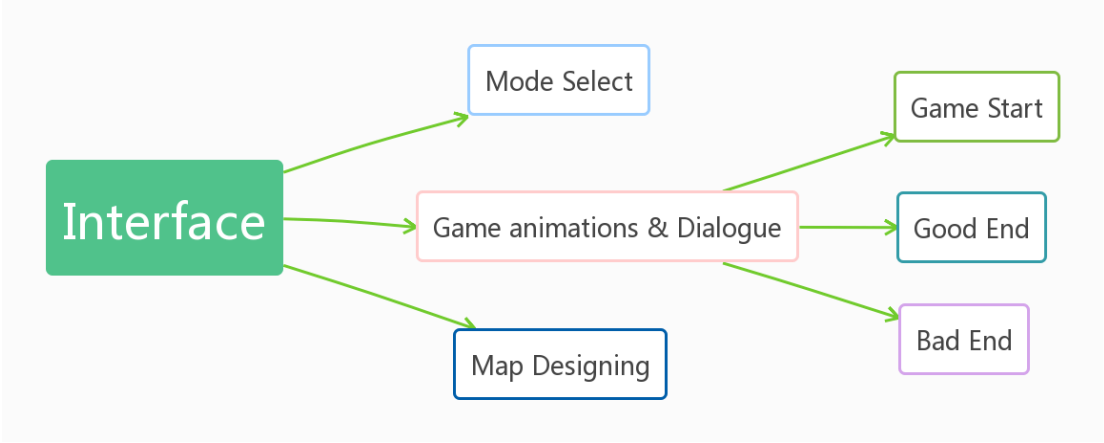
我们小组在使用 A*算法时，输入敌人与玩家的坐标值，分别作为终点与起点（因为 A*算法通过父指针给出终点到起点的路径，想要得到实际上敌人到玩家的路径，需要将敌人和玩家分别认为是终点和起点），而后，算法仅返回敌人运动完第一步所在位置坐标，并传入相关函数使敌人移动。如此重复，最终使敌人移动到玩家坐标，同时，玩家与敌人的坐标是实时更新的。

对于起点到终点移动耗费（G 值）的估价问题，我们使用了经典而简单的曼哈顿法（街区法），即在计算当前节点时，不考虑障碍物，不存在“斜着走”的情况，估算起点到终点最短距离，以此数值加上已经起点到当前节点的移动耗费(F 值，此处因为敌人在空中时从实际角度出发，不存在斜着下落的可能性，因此规定只搜索上下左右四个方格)，二者求和作为判断最优解的指标。


Coordinates of Starting point and Destination
A Star Algorithm
Shortest Path


Start
Map Input
initialization: Starting point、Destination
Insert Starting point into Openlist; Clean the Closed list
Is Openlist Empty?
Yes
Failed Searching
No
Go through the Openlist until find the node with min F value
Transfer this node from Openlist to Closed list
Expand searching area (4 node around the chosen node: up, down, left, right)
evaluate the four new node in order
Is it the destination
Yes
Generate path from destination to starting point
Output the path
No
Node is obstacle:Ignore it
Node in the closedlist: Ignore it
Node is not in the openlist,not in the closedlist and is not obstacle
Node in the openlist
Insert node into Openlist Calculate F value, make its pathnext pointer point to the source node, regard the source node as father node
Node's G value < source node's G value +1
No
Yes
Regard source node as the new fathernode, maker node's pathnext pointer point to it

4.3 界面设计



在界面设计方面，我们运用了如鹏游戏引擎，完成了素材收集，文字生成，图片的放置以及鼠标指针等功能。

使用游戏引擎对每一张图片进行编码，在执行程序过程中依次播放与隐藏图片。利用游戏引擎获取鼠标点击信息，当点击到按钮区域时执行相应的操作。

5.实施

本程序共分为界面，控制与 A*算法三个主要部分，每位组员负责一个部分。其中界面部分包含开始界面与结束界面；控制部分又分为对游戏进程，对玩家，对敌人，对子弹，对地图及物品几个部分的控制。

程序有 3 个头文件，其中头文件 control.h 是整个游戏的核心头文件，包含游戏中主要的声明；头文件 Apath.h 负责与 A*算法有关的声明；list.h 是与“敌人链表”有关的声明。

整个程序的代码分散在 8 个“.c”文件中，每一个文件负责储存相应部分的代码。其中 Apath.c 中的代码为 A*算法的代码；control.c 以及 main.c 中为游戏进程控制部分及游戏界面部分的代码；control_bullet.c 中为与子弹有关的代码；control_enemy.c 中为与敌人有关的代码；list.c 为“敌人链表”的代码；control_map.c 为与地图及图中物品有关的代码；control_player.c 中为与 player 有关的代码。共计约 2300 行。

游戏的地图储存在 map.txt 文件中。

在这里，针对上文设计中提到的部分重要的功能，我们将展示部分代码并加以分析。

5.1 界面

在 main()中使用如鹏游戏引擎的 rpInit()函数调用界面函数，在界面函数中根据选择调用模式 1 和模式 2 的函数。

使用游戏引擎中的 createImage(int num, char* filePath)，setImagePosition(int num, int x, int y)，hideImage(int num)，isMouseLeftButtonDown()，getMousePositionX()，getMousePositionY()等函数实现图片的导入以及鼠标状态的获取。

5.2 游戏进程控制

```
void gamelMain(void) //hard
{
    int control_game = 0;
    int control_bullet = 0;
    setGameTitle("All for love");
    setGameSize(900, 550);
    setBgColor(0.2, 0, 1);

    wantMap();
    Initialize();
    jump_shoot();

    while (1)
    {
        if (hasfailed || hassuccess)
        {
            break;
        }

        if (control_game++ % SPEED ==0)
        {
            if (speed++ % TIME == 0)
            {
                playerMove(keyboard());  //获取键盘输
            }
            if (control_bullet++ % BULLETTIME == 0)
            {
                moveBullet();
            }
            gravity();
            updateEnemyImportantDate();
            check_intersect();
            check_win();
        }
```

在游戏的循环开始之前首先初始化所有变量，紧接着进入游戏循环。在这里为了降低游戏循环的代码部分（下称有效循环部分）的执行速度，创建了一个变量 control_game，使其在每次循环中递增 1，只有当其增加到某一数的整数倍时才执行游戏的有效循环部分。通过采用这种方式，可以使程序没有那么快。

在之后的代码中也使用了类似的手法，以降低循环中有效部分执行的速度。

在有效循环中依次执行获取键盘输入使玩家移动的函数，刷新子弹的函数，重力函数，刷新并移动敌人的函数，玩家的碰撞判断函数已经游戏的胜利判断函数，这些函数都在其他的控制部分中

5.3 创建地图

从一个 txt 文档读取出一个地图数组。地图数组是一个二维数组，canvas[y 坐标][x 坐标]。

地图数组的每一个值只能取 0 或 5 或 9。0 代表地图上这个位置是空白（什么也没有，意味着可以通过的区域）；5 代表地图上这个位置是地图的边界；9 代表地图上这个位置是地面。。通过使用 ch 这个数组（其中 ch[1]='\0'），将读取的字符储存为一个字符串，再使用 atoi()函数将字符串转换为数字并储存到地图数组中。

```
/*头文件中的代码--地图数组编码*/
#define WALL 5 //边界
#define FLOOR 9 //脚下的土地
#define AIR 0  //什么都没有的位置
#define I_WALL 70000 //图片编号
#define I_FLOOR 80000 //图片编号
#define I_AIR 90000 //图片编号
```

Control_map.c 中的代码

```
FILE *fCanvas;
int canvas[HIGH][WIDTH];
void loadCanvas(void)
{
    char ch[2] = { '\0','\0' };//为了获取一个0-9的数字的一个方法
    int y = 0, x = 0;

    if ((fCanvas = fopen("map.txt", "r+")) == NULL)
    {
        fprintf(stderr, "reverse can't open map");
        exit(EXIT_FAILURE);
    }
    for ( ; ; )
    {
        ch[0] = getc(fCanvas);
        if (ch[0] == '\n')
        {
            y++;
            x = 0;
            continue;
        }
        if (ch[0] != EOF && ch[0] != '\r')
```

```
        {
            canvas[y][x] = atoi(ch);
            x++;
        }
        if (ch[0] == EOF)
            break;
        /*if (y = 10 && x = 18) break;*/
    }
    fclose(fCanvas);
}
```

之后再按照地图数组中的值，使用游戏引擎，将墙和地面的图片填充到界面中。

```
void getCanvas(void)
{
    int x, y;
    for (y = 0; y < HIGH; y++)
    {
        for (x = 0; x < WIDTH; x++)
        {
            switch (canvas[y][x])
            {
            case AIR:
                drawCanvas(AIR, y, x);
                break;
            case WALL:
                drawCanvas(WALL, y, x);
                break;
            case FLOOR:
                drawCanvas(FLOOR, y, x);
                break;
            default:
                exit(EXIT_FAILURE);
                break;
            }
        }
    }
}

void drawCanvas(int input, int y, int x)
{
    switch (input)
    {
    case WALL:
        setImagePosition((I_WALL + x + y * 50), LENGTH * x, LENGTH * y);
```

```
            break;
        case AIR:
            break;
        case FLOOR:
            setImagePosition((I_FLOOR + x + y * 50), LENGTH * x, LENGTH * y);
            break;
        default:
            break;
        }
}
```

如前文所说，键盘输入有四种：左右移动、向上跳跃、射击、退出游戏。其中左右移动与射击的函数比较重要，我们着重说明这两个。

5.4 玩家的左右移动

玩家结构体包括玩家的 x，y 坐标，精灵参数，以及是否拾取黄色的钥匙和宝箱。

```
/*玩家*/
typedef struct player
{
    int sprite;
    int x, y;//坐标
    bool yKey;//yellow key
    bool box;
}Player_t;
```

通过使用 `if((GetAsyncKeyState(VK_LEFT) & 0x8000)||(GetAsyncKeyState(0x41)& 0x8000))` 的判断方式，keyboard()函数将键盘输入的值作为返回值，通过 `playerMove(keyboard());` 的方式传递给 playerMove()函数。playerMove 函数根据传入的值调用移动函数并告诉移动函数移动的方向。

```
int keyboard(void)
{
    int re = 0; //为了return re;
    if ((GetAsyncKeyState(VK_LEFT) & 0x8000) || (GetAsyncKeyState(0x41) & 0x8000)) //左
    {
        position = LEFT;
        re = LEFT;
    }
    else if ((GetAsyncKeyState(VK_RIGHT) & 0x8000) || (GetAsyncKeyState(0x44) &
0x8000)) //右
    {
        position = RIGHT;
        re = RIGHT;
    }
    else if (GetAsyncKeyState(0x1B) & 0x8000)  // Esc键退出程序函数
    {
```

```
            EmptyTheList(&list);
            exit(0);
        }
        else if (GetAsyncKeyState(0xd) & 0x8000)  //回车键
            stop();
        return re;
}

void playerMove(int direction)
{
        switch (direction)
        {
        case LEFT:
        case RIGHT:
            Xmove(direction);
            break;
        case JUMP:
        case DOWN:
        case 0:
            break;
        default:
            exit(EXIT_FAILURE);
            break;
        }

        }
```

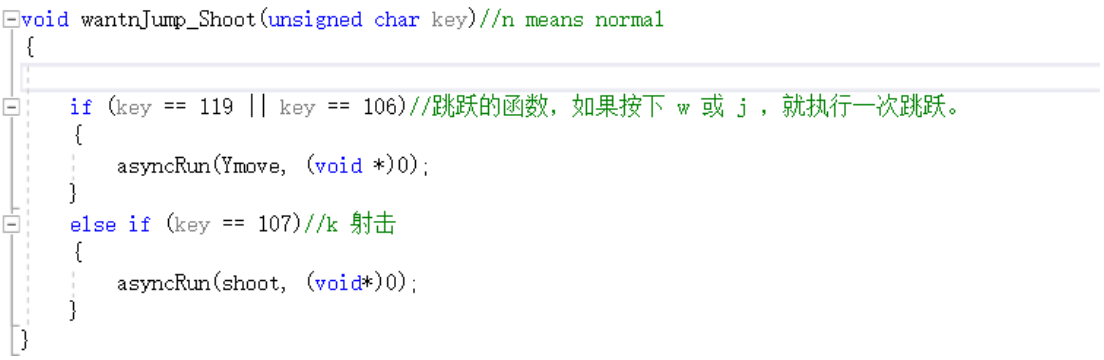
移动函数 Xmove 中先检查移动方向上是否有障碍物，如果有就不改变坐标，如果没有障碍物，就把玩家的 x 坐标增加或减少 1（注意我们前文提到过玩家的 x 坐标的特别之处，它不需要×50 就是实际坐标，这是为了左右移动的流畅性），再把玩家的图片设置到他的坐标位置。

```
void Xmove(int direction) //direction为LEFT或RIGHT
{
        if (check_x_location(direction)) //检查是否可以移动
        {
            if (direction == LEFT)
            {
                player.x--;
                playSpriteAnimate(player.sprite, "LEFT");
                setSpritePosition(player.sprite, player.x, LENGTH * player.y); //这个不乘
50也没关系
                playSpriteAnimate(player.sprite, "LEFT");
            }
            else
            {
```

```
            player.x++;
            playSpriteAnimate(player.sprite, "RIGHT");
            setSpritePosition(player.sprite, player.x, LENGTH * player.y);
            playSpriteAnimate(player.sprite, "RIGHT");
        }
    }
}
```

5.5 子弹与*玩家跳跃*

使用游戏引擎可以在另一个进程中监控键盘按键，当检测到 K 键时调用射击函数，当检测到“W”“J”“↑”键时调用跳跃函数。（图片为一部分代码）



```
void wantnJump_Shoot(unsigned char key)//n means normal
{

    if (key == 119 || key == 106)//跳跃的函数，如果按下 w 或 j ，就执行一次跳跃。
    {
        asyncRun(Ymove, (void *)0);
    }
    else if (key == 107)//k 射击
    {
        asyncRun(shoot, (void*)0);
    }
}
```

跳跃函数同左右移动函数相仿，先判断是否有障碍再移动，只不过需要移动多次，且需要使玩家落下，落到地面上为止。（顺便一提的是 gravity()函数也是这样，判断玩家脚下是否有地面，如果没有就使玩家下降一，直到落到地面为止。）

跳跃代码：（与左右不同的是移动距离及需要下降）

```
void Ymove(void * p)
{
    int i;
    doing_jump = true;
    switch (position)
    {
    case LEFT:
        playSpriteAnimate(player.sprite, "JUMP_L_START");
        break;
    case DOWN:
    case RIGHT:
    default:
```

```
            playSpriteAnimate(player.sprite, "JUMP_R_START");
            break;
        }
        for (i = 0; i < H; i++)
        {
            if (check_y_location_up(JUMP))
            {
                player.y++;
                playSpriteAnimate(player.sprite, "JUMP");
                pauseGame(100);
                setSpritePosition(player.sprite, player.x, LENGTH * player.y);
                playSpriteAnimate(player.sprite, "JUMP");
                pauseGame(100);
            }
            else
                break;
        }
        pauseGame(100);
        for (i = 0; i < H; i++)
        {
            if (check_y_location_down(JUMP))
            {
                player.y--;
                playSpriteAnimate(player.sprite, "JUMP");
                pauseGame(100);
                setSpritePosition(player.sprite, player.x, LENGTH * player.y);
                playSpriteAnimate(player.sprite, "JUMP");
                pauseGame(100);
            }
            else
                break;
        }
        pauseGame(10);
        playSpriteAnimate(player.sprite, "NEW");
        doing_jump = false;
    }
```

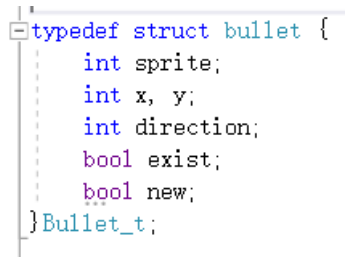
重力函数代码：

```
void gravity(void)
{
    if (!checkLocation() && doing_jump == false)
    {
```

```
        for (player.y; player.y >= 0; player.y--)
        {
            playSpriteAnimate(player.sprite, "DROP");
            setSpritePosition(player.sprite, player.x, LENGTH * player.y);
            playSpriteAnimate(player.sprite, "DROP");
            if (checkLocation())
            {
                playSpriteAnimate(player.sprite, "NEW");
                break;
            }
        }
    }
}
```

子弹

子弹结构体包括子弹的 x，y 坐标，精灵参数，子弹方向，子弹是否还存在以及子弹是否是新创建的。



```
typedef struct bullet {
    int sprite;
    int x, y;
    int direction;
    bool exist;
    bool new;
}Bullet_t;
```

子弹利用子弹数组储存。

每一个子弹都遵循先设置，再刷新的原则，每调用一次射击函数，就会按照当前的子弹编号（也就是数组元素的编号）初始化一个子弹数组的值，根据 player 的状态设置子弹的位置与方向，并标记为初始产生的子弹。每次设置完成后将子弹编号递增，由于地图中的子弹数量是有限的（取为 BULLET_MAX_NUM），当子弹编号超过允许的最大值就把编号归零。

```
void shoot(void * p)
{
    switch (position)
    {
    case LEFT:
        bullet[bul_num].direction = LEFT;
        bullet[bul_num].x = (int)(player.x / LENGTH) - 1;
        bullet[bul_num].y = player.y;
        break;
    case DOWN:
    case RIGHT:
    default:
```

```
            bullet[bul_num].direction = RIGHT;
            bullet[bul_num].x = (int)(player.x / LENGTH) + 1;
            bullet[bul_num].y = player.y;
            break;
        }
        bullet[bul_num].new = true;
        bullet[bul_num].exist = true;
        bul_num++;
        if (bul_num == BULLET_MAX_NUM)
        {
            bul_num = 0;
        }
}
```

如前文所示（5.2 图），在游戏的循环中会调用 moveBullet();刷新子弹的信息（这里为了减慢刷新的速度用了前文的处理方法）。

对所有的存在于地图上的子弹进行操作，如前文所述，需要分为两种情况：

1. 如果子弹是新创建的，设置这个子弹为不是新创建的，判断子弹位置是否有障碍物。如果有，就执行相应的函数；没有，就再这个位置上导入子弹的图片（利用如鹏游戏引擎）。

2. 如果子弹不是新创建的，先将子弹的坐标进行移动再判断子弹位置是否有障碍物。如果有，就执行相应的函数；没有，就把子弹的图片移动到当前位置（利用如鹏游戏引擎）。

```
void moveBullet(void)
{
    int i;
    int mode;

    for (i = 0; i < BULLET_MAX_NUM; i++)
    {
        if (bullet[i].exist == true)
        {
            if (bullet[i].new == false)
            {
                if ((bullet[i].direction == LEFT && bullet[i].x > 0) ||
(bullet[i].direction == RIGHT && bullet[i].x < WIDTH))
                {
                    switch (bullet[i].direction)
                    {
                    case LEFT:
```

```
                    bullet[i].x--;
                    break;
                case RIGHT:
                    bullet[i].x++;
                    break;
                }
            }
        }
        else
        {
            bullet[i].new = false;
        }

        if (!(mode = checkBulletHit(i)))
        {
            drawBullet(i);
        }
        else
        {
            bullet[i].exist = false;
            hideSprite(bullet[i].sprite);
            BulletHit(i, mode);
        }
    }
}
}

void drawBullet(int num)
{
    showSprite(bullet[num].sprite);
    setSpritePosition(bullet[num].sprite, LENGTH * bullet[num].x, LENGTH *
bullet[num].y);
    playSpriteAnimate(bullet[num].sprite, "NEW");
}
```

子弹的碰撞函数：

如果子弹的位置上是墙或者地面，则清除当前子弹；如果子弹的位置上是敌人，则敌人的生命值减 1，并清除当前子弹。

```
void BulletHit(int num, int mode)
{

    if (mode != 0 && mode != BULLET_MAP)
    {
        Node_t * pTemp;

        num = mode - 20 * BULLET_ENEMY;

        for (pTemp = list; pTemp != NULL; pTemp = pTemp->next)/*enemy[num].life--;*/
        {
            if (pTemp->enemy.number == num)
            {
                pTemp->enemy.life--;
            }
        }
    }
}

int checkBulletHit(int num) //检查子弹碰撞,碰上true
{
    Node_t * pTemp;

    if (canvas[bullet[num].y][bullet[num].x] == WALL ||
canvas[bullet[num].y][bullet[num].x] == FLOOR)
    {
        return BULLET_MAP;
    }

    for (pTemp = list; pTemp != NULL; pTemp = pTemp->next)
    {
        if (pTemp->enemy.exit == false)
        {
            continue;
        }

        if (bullet[num].x == pTemp->enemy.x && bullet[num].y == pTemp->enemy.y)
        {
            return (20 * BULLET_ENEMY + (pTemp->enemy.number));
        }
    }
    return 0;
}
```

5.6 敌人的移动

在 list.c，list.h 文件中建立了定义了一个敌人链表与与之有关的函数，使用 List_t 类型定义一个指向一个敌人链表的变量 list，链表的每一个节点有一个敌人结构体以及一个指针。

```
struct enemy
{
    int sprite;
    int number;
    int x, y;
    int mode;
    int life;
    bool exit;
};
/*一般的声明*/
typedef struct enemy Enemy_t;
typedef struct node
{
    Enemy_t enemy;
    struct node * next;
}Node_t;
typedef Node_t * List_t;
```

敌人结构体包括敌人的 x，y 坐标，精灵参数，敌人编号，敌人的移动模式，敌人的生命值以及是否还存在。

与之相关的函数有：（声明在 list.h 中）

初始化一个链表；确认链表是否为空定义；确认链表是否已满；把一个函数作用于链表的每一项；释放已经分配的内存；在链表的开头添加项；在链表中找出指定项；在链表中删除指定项；确认链表的项数。

首先初始化这个 enemy 链表。使用随机数，随机生成一名敌人的横坐标与生命值。

选择一个离玩家距离最远的敌人，使用 C99 的 hypot()函数求 player 与敌人之间的距离，将距离最远的敌人的移动模式设置为模式 1。

遍历整个链表，找出模式 1 的还存在于地图之上的敌人，使用 A*算法，把它的坐标设置为下一步它的移动位置，再移动，并将移动模式恢复为模式 0。

为了减慢移动速度，使用 if 语句并结合随机数，使得设置移动模式与移动这两个行为都是有一定概率发生的。

【注释】：A*算法能够最快给出的是到达终点前的最后一个位置，把 player 的位置设置为起点，把 enemy 的位置设置为终点。由两点之间的对称性可知，到达终点前的最后一个位置也就是使 enemy 为起点，player 为终点，enemy 需要走的第一个位置。

```
void moveEnemy(void) //mode = 1的链表里的项
{
    Node_t * pTemp;
    coor address = { 0, 0 };

    for (pTemp = list; pTemp != NULL; pTemp = pTemp->next)
    {
```

```
        if (pTemp->enemy.exit == false)
        {
            continue;
        }
        if (pTemp->enemy.mode == 1)
        {//player.x % 50
            A_star_algorithm(mapIn, (int)(player.x / LENGTH), player.y,
pTemp->enemy.x, pTemp->enemy.y, &address);
            pTemp->enemy.x = address.x;
            pTemp->enemy.y = address.y;
            setSpritePosition(pTemp->enemy.sprite, LENGTH * pTemp->enemy.x, LENGTH *
pTemp->enemy.y);
            pTemp->enemy.mode = 0;
        }
    }
}
```

5.7 A*算法

首先我们来看一下算法整体：

```
void A_star_algorithm(int mapn[][18], int startpoint_x, int startpoint_y, int
endpoint_x, int endpoint_y, coor* Coor_next_step)
{
    pnode** mapp = translate_Map(mapn, row, col);//将地图转换为二维数组

    pnode* startpnode = find_start_pnode(mapp, row, col, startpoint_x, startpoint_y);
    //寻找起点

    pnode* endpnode = find_end_pnode(mapp, row, col, endpoint_x, endpoint_y);//寻找终点

    pnode* curpnode = startpnode;//将起点作为当前节点

    curpnode->G = 0;

    count_Pnode_H(curpnode, endpnode);//计算当前节点G值

    count_Pnode_F(curpnode);//F值
```

```
    linklist openlist = (linklist)malloc(sizeof(pnode));//为开启列表申请储存空间初始化
    memset(openlist, 0, sizeof(pnode));
    linklist closelist = (linklist)malloc(sizeof(pnode));//为关闭列表申请储存空间初始化
    memset(closelist, 0, sizeof(pnode));
    insert_openlist_by_asc(openlist, curpnode);//按从小到大顺序插入开始列表

    while (curpnode->is_Endpoint_Here == FALSE)
    {
        curpnode = return_openlist_min_pnode(openlist);//将开启列表中F值最小的节点作为
        当前节点，并从开始列表移除
        insert_into_closelist(curpnode, closelist);//将该节点加入关闭列表
        check_around_curpnode(curpnode, endpnode, openlist, mapp);//检查当前节点上下左
        右四个节点，计算H、F值，将符合条件的加入开启列表
    }
    Coor_next_step->x = (*(endpnode->path_next)).x;//返回下一步移动位置的坐标值
    Coor_next_step->y = (*(endpnode->path_next)).y;

}
```

首先将二维整数数组转化成存储结构体指针的数组，这样子地图的每个个点都变成了结构体指针节点，存储着这个点的相关信息，比如这个点的类型，坐标，是否为起终点，是否在开启或者关闭列表中等。

搜索过程中的两个列表分别用 Openlist、Closelist 两个链表来表示，其中 Closelist 储存不需要再分析计算的节点，插入时头插法插入即可；但是，Openlist 需要返回 F 值最小的节点，插入时需要将节点按从大到小的顺序依次插入，这样在返回时，返回第一个节点即可。Openlist 插入函数如下：

```
void insert_openlist_by_asc(linklist Openlist, pnode* elem)//按照F值从小到大插入
{
    pnode *p, *q;
    p = q = Openlist;
    while (p->next != NULL && p->F < elem->F)
    {
        q = p;
        p = p->next;
    }
```

```
    if (p->F < elem->F) q = p;
    elem->next = q->next;
    q->next = elem;
    elem->open_list = 1;
}
```

边界上的节点在搜索时，如果不加注意可能出现输入坐标值不合法的问题，为此将搜索区域进行分类，成功解决了这类问题（搜索函数如下）

```
void check_around_curpnode(pnode* cur, pnode* End_pnode, linklist Openlist, pnode**
Mapp)
{
    int x = cur->x;
    int y = cur->y;
    if (y > 0 && y < 11)
    {
        if (x > 0 && x < 18)
        {
            insert_open(Mapp[y] + x - 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y] + x + 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y + 1] + x, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y - 1] + x, cur, End_pnode, Openlist, Mapp);
        }
        if (x == 0)
        {
            insert_open(Mapp[y] + x + 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y + 1] + x, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y - 1] + x, cur, End_pnode, Openlist, Mapp);
        }
        if (x == 18)
        {
            insert_open(Mapp[y] + x - 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y + 1] + x, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y - 1] + x, cur, End_pnode, Openlist, Mapp);
        }
    }

    if (y == 0)
    {
        if (x == 0)
        {
            insert_open(Mapp[y] + x + 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y + 1] + x, cur, End_pnode, Openlist, Mapp);
```

```
        }

        if (x == 18)
        {
            insert_open(Mapp[y] + x - 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y + 1] + x, cur, End_pnode, Openlist, Mapp);
        }
        if (x > 0 && x < 18)
        {
            insert_open(Mapp[y] + x - 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y] + x + 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y + 1] + x, cur, End_pnode, Openlist, Mapp);
        }
    }

    if (y == 11)
    {
        if (x == 0)
        {
            insert_open(Mapp[y] + x + 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y - 1] + x, cur, End_pnode, Openlist, Mapp);
        }
        if (x == 18)
        {
            insert_open(Mapp[y] + x - 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y - 1] + x, cur, End_pnode, Openlist, Mapp);
        }
        if (x > 0 && x < 18)
        {
            insert_open(Mapp[y] + x - 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y] + x + 1, cur, End_pnode, Openlist, Mapp);
            insert_open(Mapp[y - 1] + x, cur, End_pnode, Openlist, Mapp);
        }
    }
}
```

6.测试

6.1 模式选择测试（不同按键进入不同模式）

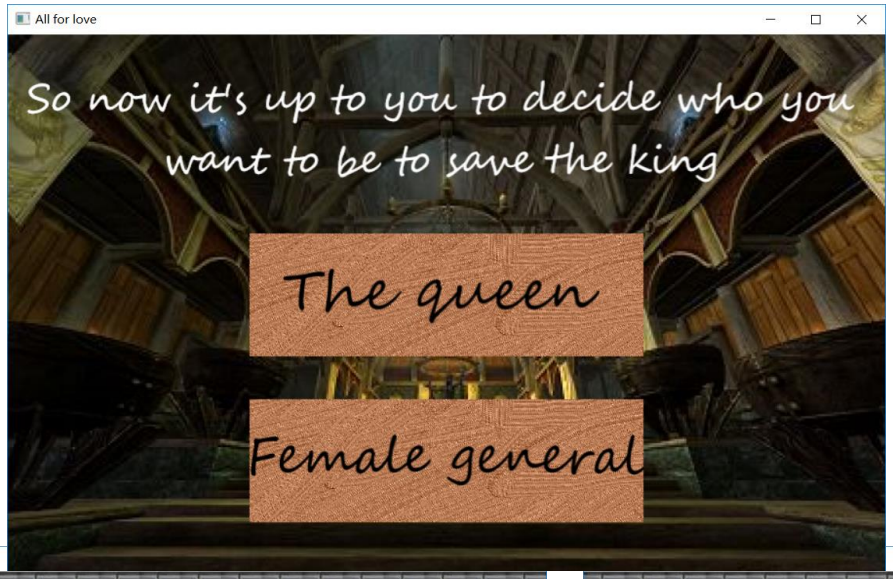


图 按下上面的按钮

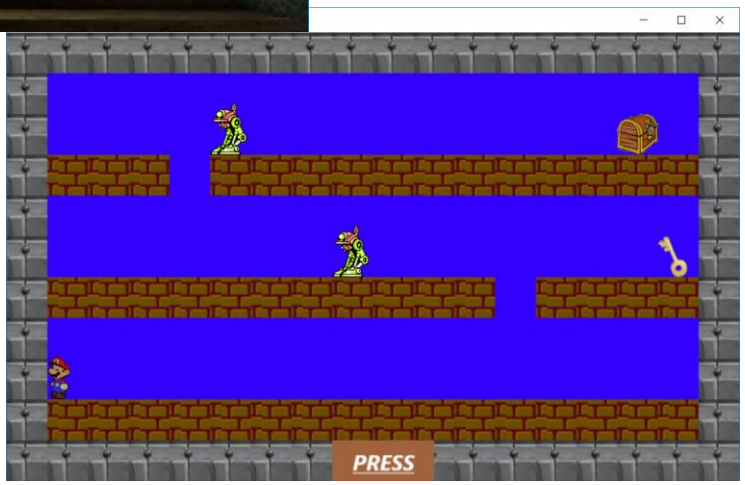


图 按下下面的按钮

6.2 移动与物品交互测试（小人移动去拿钥匙）

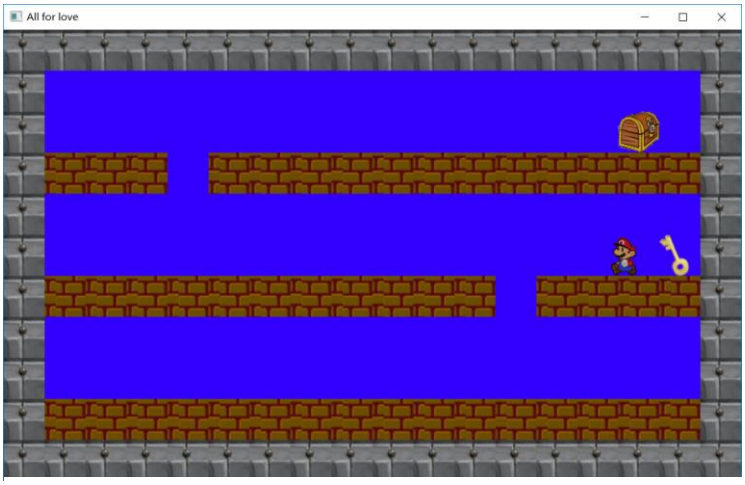


图 拾取前

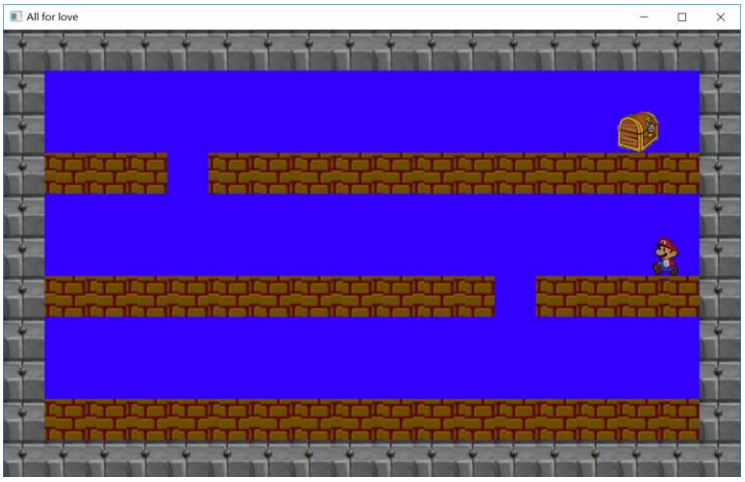


图 拾取后

6.3 子弹发射测试（按下 K 键发射子弹）

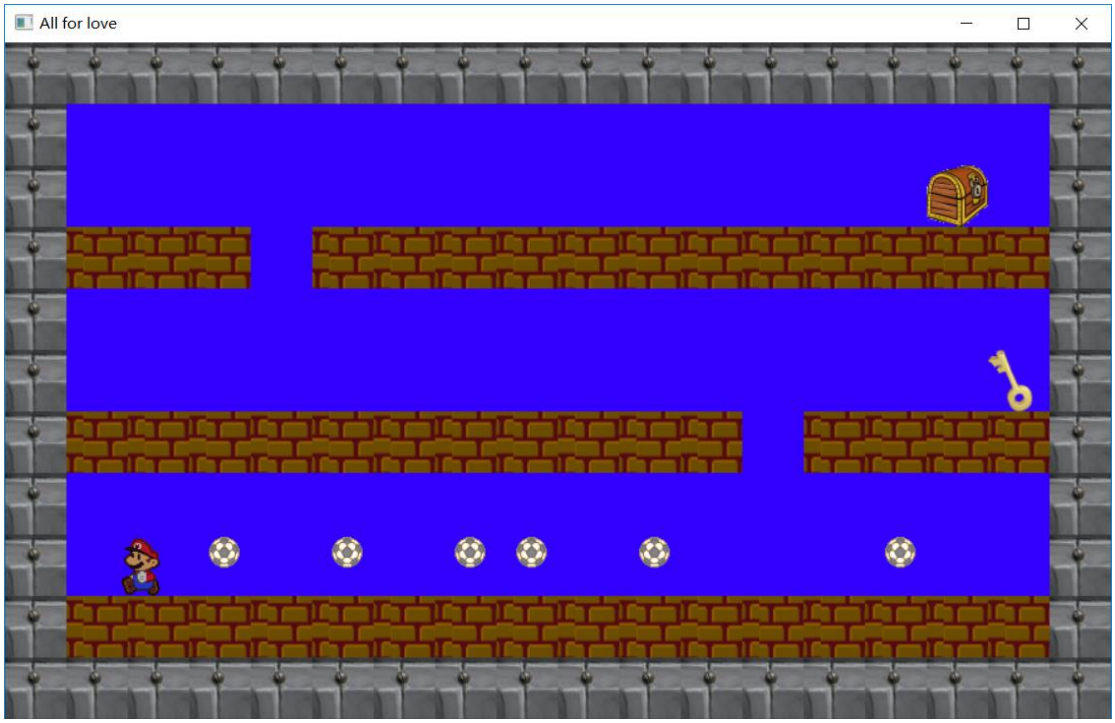
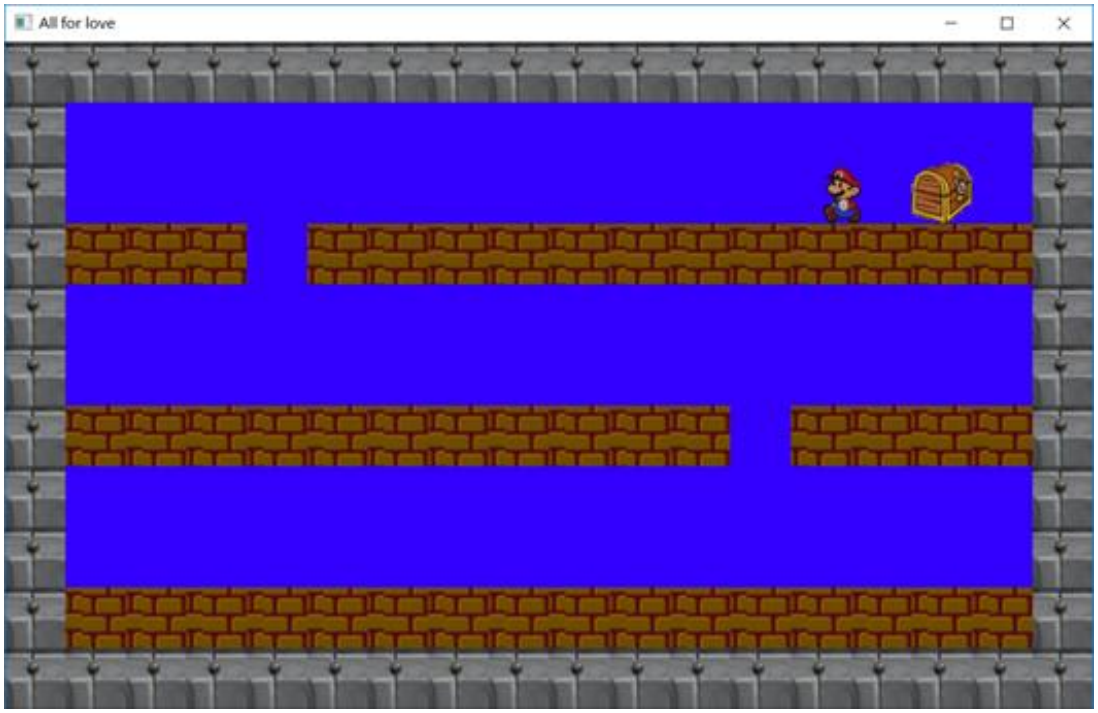


图 发射的子弹

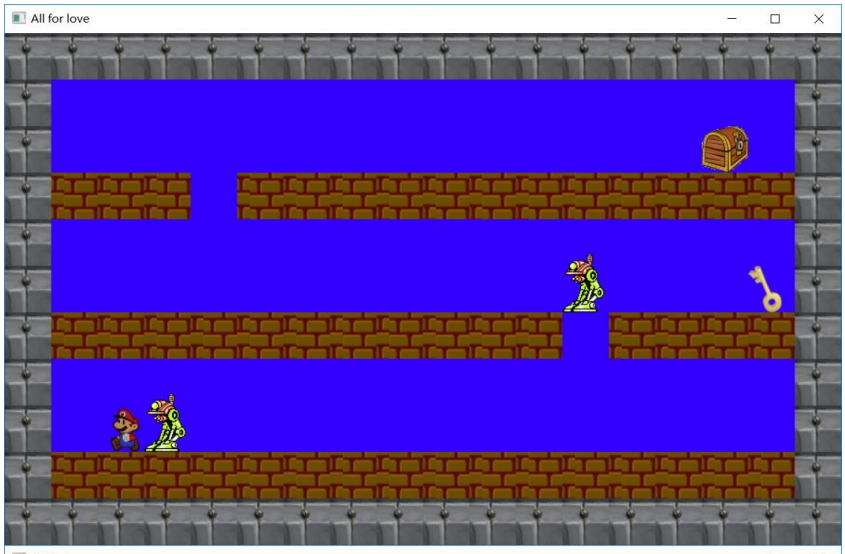
6.4 胜负条件判定测试（胜与负的条件截图）

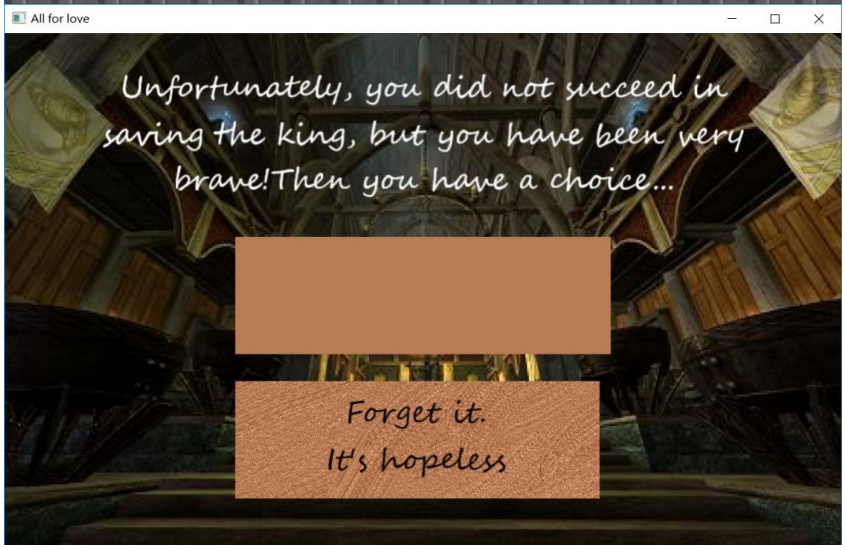
胜利：




All for love
Congratulations on your success in saving the king. You are wonderful!
It's over.
Go see the king

失败：


All for love


All for love
Unfortunately, you did not succeed in saving the king, but you have been very brave!Then you have a choice...
Forget it.
It's hopeless

6.总结

在这段超过三个月的时间里，我们组的成员们都在这款游戏上花了很多时间和精力。我们在这个过程中一起进步，一起学习。一开始，我们甚至不知道我们该做什么项目，游戏？管理系统？或者是最简单的计算器？在我们决定了制作一款游戏后，我们又面临该选择哪种工具，哪种算法，哪种引擎来帮助我们完成对项目的实现，对于这款游戏的完成，每一个人都做出了贡献。我们成功地实现了游戏界面的展示，人物间最短路径的计算，人物的移动、跳跃等一系列功能，玩家控制程序的优化，游戏模式选择的实现，死亡的判断，游戏的结束等等。

从一开始对游戏界面的运行的测试，再到后来对人物的控制的测试时，我们都遇到了很多大大小小的问题，但是我们耐心地一点一点的调试与重整，最终实现了项目的完成。我们能够在这么短的时间内取得进步，是因为我们经常在一起讨论和交流想法。在交流过程中不断更新和成熟我们的程序代码，使我们的游戏运行得更加顺利。现在，我们设计了一款适合所有年龄段的玩家的游戏。它有一个漂亮的界面，自由移动的人物。程序设计中还会有不同的模式提供给玩家选择，不同的游戏结果还会提供给玩家不同的选择。这款游戏的难度适中。在业余时间玩这个游戏是个不错的主意。

总的来说，我们通过这一学期对 C 语言这门学科不间断的学习，掌握了对一些基本功能、算法的利用，实现甚至超越了我们当初对于这款游戏的设想！