myruguelike learning

————from the day one

12/7版 未完待续

191220059 林龙杰

一、游戏构思

在jw04的作业中，了解到ruguelike的历史背景。以及一些经典的地牢算法，比如地牢的随机生成算法————通过多次smooth地砖与墙壁来随机构造地牢，还有一些其他知识也引起了我很大的兴趣。在jw05中将要求实现一个地牢游戏，这对于我来说是一个大挑战。所以我初步构想，先沿用jw04提供的地牢框架，来简易实现一按键触发的爷爷救葫芦娃的rugulike。然后进行优化，从单一作战方式到近远程多武器作战；从单一道具拾取到多种类；从单一敌人模式到多种类。在jw06,07中分别加入进度储存和测试。

1.规则构思

我的想法是，爷爷作为主角，在洞穴中寻找七个葫芦娃，找到七个葫芦娃并打败最终boss蝎子精即可获胜。在寻找途中将可以进行跟怪物战斗，拾取加血道具，拾取装备等操作。

-暂时只实现了一关卡，所以找到楼层中的葫芦娃即可获得胜利，击败敌人可以提高得分。

2.框架构思

我暂时准备实现一个视角无限，可以挖掘墙壁，可以战斗，有状态栏，队伍成员栏，有一定关卡的地牢。就利用asciiPanel提供的图标绘制简易的界面。怪物的一些功能将会使用学到的多线程技术实现，比如射箭就准备用多线程来实现。怪物的移动也将采取一定的寻路算法。游戏将随着开发从按键刷屏制转化为实时对战游戏。

二、展示与细节

展示视频发布在bilibili：https://www.bilibili.com/video/BV18Z4y197vr?share_source=copy_web

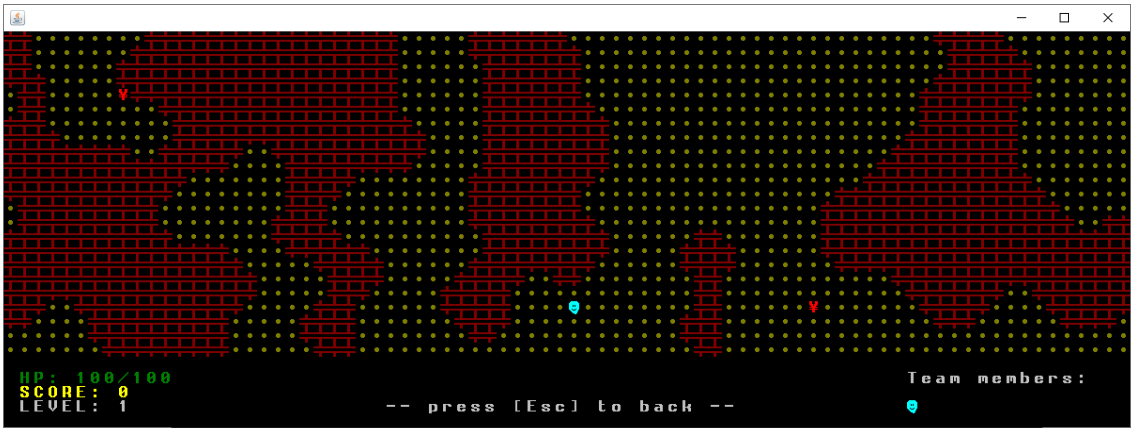
展示描述：

- 首先进入游戏，BGM响起，眼前的是PlayScreen欢迎你来到"爷爷救葫芦娃"这个游戏，"WELCOME TO THE GAME"，此时可以选择进行游戏，查看手册，或者退出游戏。
- 

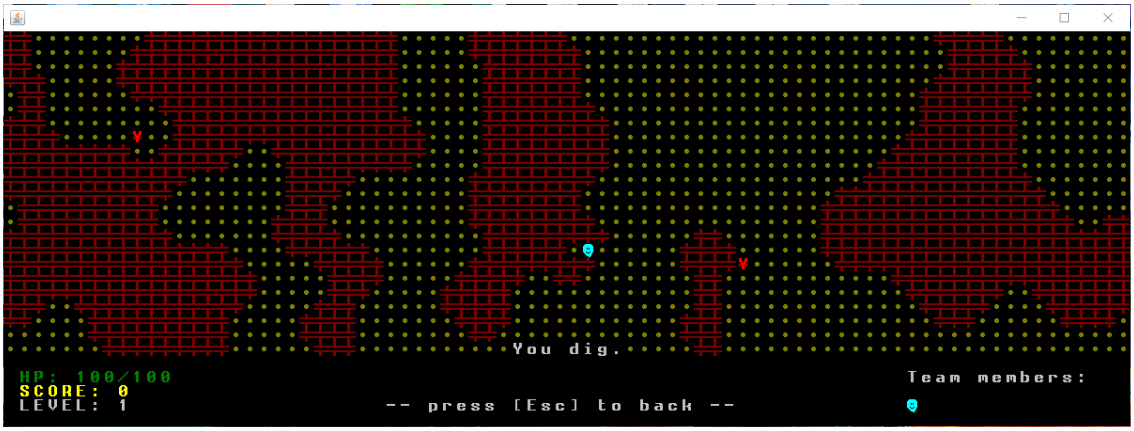
- 如果选择查看手册则进入手册界面，你可以形象的了解到游戏操作方法，怪物种类和规则。

- 选择继续游戏，进入Play界面：

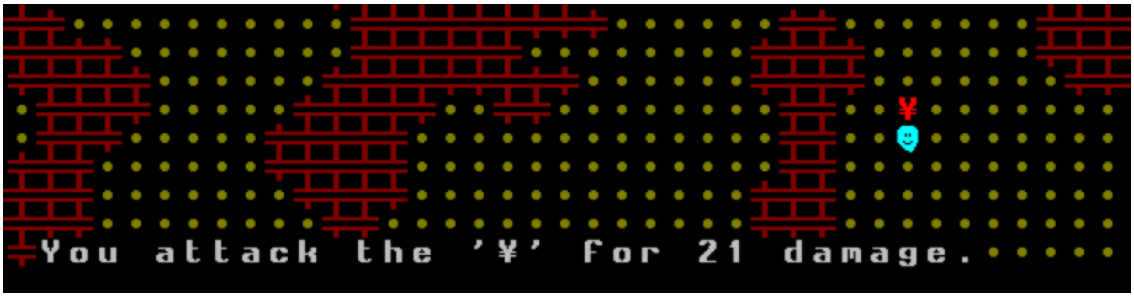
  蓝色小人为主角爷爷，下方左侧状态栏显示你的HP，得分，以及关卡难度（LEVEL），右侧显示队伍成员信息，暂时只有爷爷在队伍中，下方是提示信息。

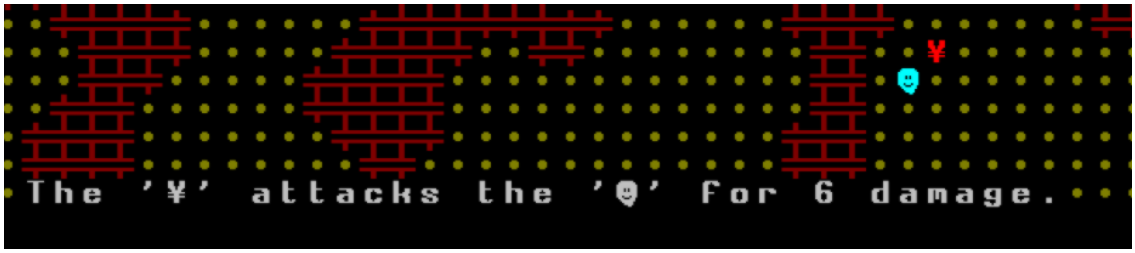


- 你可以进行挖掘：



- 蝙蝠兵¥在你靠近时（设置的半径为6）将会主动攻击你，在远离时则随机游走。你可以攻击它们，他们也会攻击你：
-

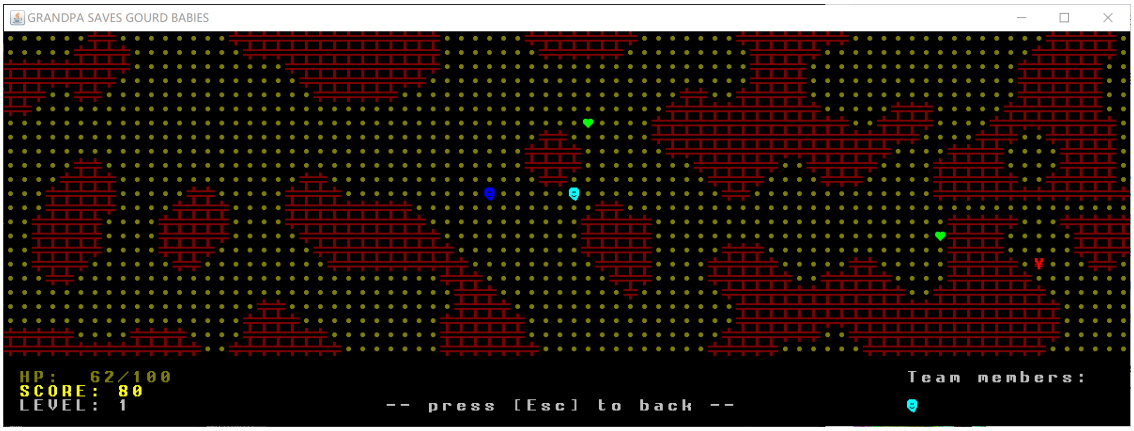




- 随时注意你的状态。绿色血量表示你还比较强壮，黄色则表示你可能要注意一下血量了，红色则代表你已经很虚弱了。

- 在杀死敌方单位后你会获得金币，而被地方杀死则会来到Lost界面，当然，你可以选择重头再来。

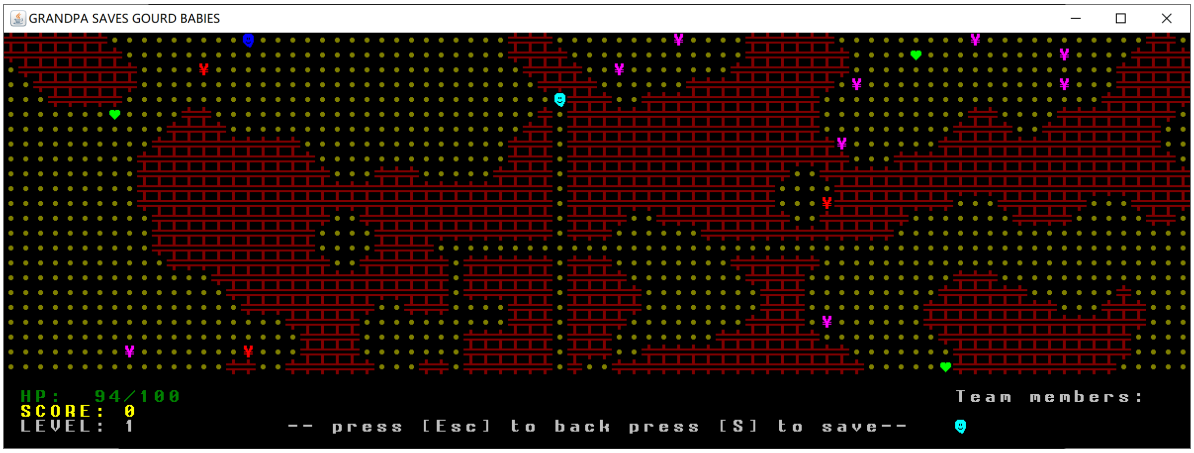
- 在找到本层的葫芦娃后会跳转到下一层（暂未实现关卡）



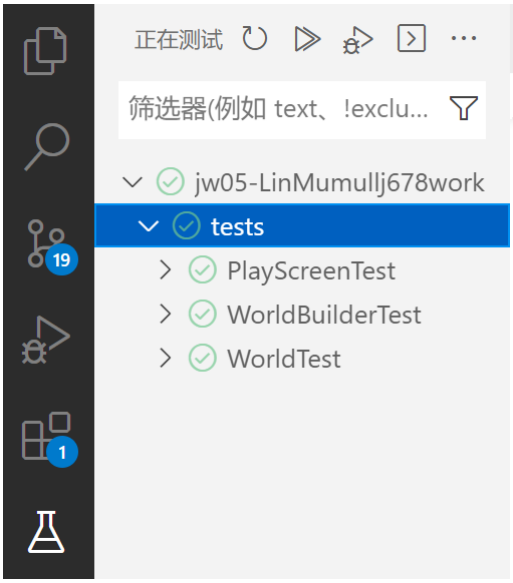
12/31版本更新：

- 增加进度存储功能，可以在退出游戏后进入之前保存的存档。

在游戏界面点击“S"即可存档。进入游戏时选择”L"选项即可继续上次的游戏。



- 游戏代码可测试



三、思考与总结

2021/12/6版本：

在这个阶段。实现了基本流畅的游戏运行，和较为好看的游戏界面。但由于时间和个人原因（最开始拖延了，最近一周事情又突然变多），很多东西待完善，这里列出几点：

1.**敌方的多线程**。在研究我用的这套rugelike框架之前，我以为多线程非常好实现：首先将敌方单位的class实现runnable或者callable，来创建敌方单位。然后在界面里创建线程，或者用线程池来运行创建敌方任务的线程即可。但写到能运行的时候发现，不知道用哪个类来实现runnable了，而且run中的任务也没想明白。

所以准确的来说并不能算是多线程优化，只是在Ai类里实现了Runnable，在工厂创建时调用线程来创建新的Ai：大概是这样：

```
public Creature newBat(PlayScreen plsc){
    Creature bat = new Creature(world, (char)157, AsciiPanel.brightRed, 100, 5,
0,plsc);
    world.addAtEmptyLocation(bat);
    Thread t=new Thread(new BatAi(bat, plsc));
    t.start();
    //new BatAi(bat, plsc);
    return bat;
}

public Creature newEBat(Creature player,PlayScreen plsc){
    Creature bat = new Creature(world, (char)157, AsciiPanel.brightMagenta, 150,
10, 0,plsc);
    world.addAtEmptyLocation(bat);
    Thread t=new Thread(new EBatAi(bat, player, plsc));
    t.start();
    //new EBatAi(bat, player, plsc);
    return bat;
}
```

2.**关卡跳转**。虽然没能实现但有一个大概的思路。就是用playscreen中关卡标志来记录当前难度，在生成怪物或者涉及攻击防御等战斗因素时，用关卡难度来区别每一关。在救到葫芦娃之后，将救到的葫芦娃加入Team队列，然后更新关卡难度，重刷界面。再重刷界面时界面中的“LEVEL"和“Team members”也相应随记录的关卡标志和Team队列更新。

3.**敌方移动的智能算法。**目前的寻路算法就是在距离较远时进行游荡，在玩家靠近时，向玩家方向移动。

最大的收获：

在这次游戏的debug中，没想到最大的收获感觉是对于类中函数参数的理解加深了。打个比方。在我用的jw04的这个ruguelike框架中，最外层是入口类Begin继承JFrame，其中包含界面screen和terminal。在对screen进行操作时，是每次将terminal作为参数传入的，所以如果在screen中想对terminal进行操作只能通过相应函数。虽然这体现了面向对象的封装但有的时候感觉还是好麻烦。果然方便和安全性是不可兼得的。

再比如Creature类和CreatureAi类是互相绑定的，在他们的构造函数中他们就通过传入参数绑定了彼此，这种互相绑定就可以是他们作为彼此的成员，互相调用，简化了好多操作。

2021/12/31版本：

这个版本的学习中感受很多：

1.游戏进度储存读取功能√：

在实现游戏进度储存的功能之前，我以前接触过的存取代码差不多要追溯到大一程设实验写文件存取的时候，当时用c语言来实现文件内容的存取，用到文件指针来操控存取位置，还要按照一定格式来存字符串，读，拆分字符串。总之非常麻烦。

但在保存游戏进度的时候，用到java中的序列化，使得游戏全局的保存非常简单。java序列化的使用，只需要将需要序列化的类以及类中的成员类都实现Serializable接口，在进行类似下方的IO操作即可实现序列化：

```
public void save()throws Exception
    {
        FileOutputStream fos=new FileOutputStream("save.ser");
        ObjectOutputStream oos=new ObjectOutputStream(fos);

        //开始序列化
        PlayScreen tmp=new
PlayScreen(world,player,screenWidth,screenHeight,messages,score,level,iswin,ispa
use);
        oos.writeObject(tmp);
        oos.close();

    }
```

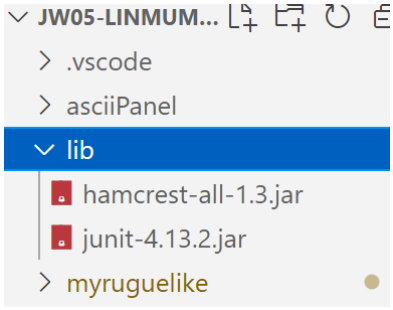
反序列化也非常简便：

```
public Screen loadsave() throws Exception
    {
        FileInputStream fis=new FileInputStream("save.ser");
        ObjectInputStream ois=new ObjectInputStream(fis);

        //读取
        PlayScreen sc= (PlayScreen)ois.readObject();
        ois.close();
        //System.out.println(sc.toString());
        return sc;
    }
```

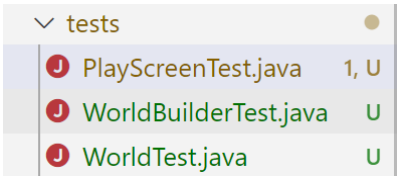
在我的代码中，是将整个PlayScreen保存，里面包含了角色信息，世界信息，游戏状态信息等等，在点击保存按钮时，调用序列化代码，在读取时将存储文件反序列化即可实现游戏进度的保存和继续。

2.代码测试√:

如果要想实现java项目的测试，那么需要用到第三方测试工具，在本项目中我选择了Junit。我在网站上下载了正确版本的junit.jar与hamcrest-core.jar,并且导入到了我的工程中：



只需要编写测试代码对于各个类进行测试即可。比如我暂时编写了三个重要类的测试：



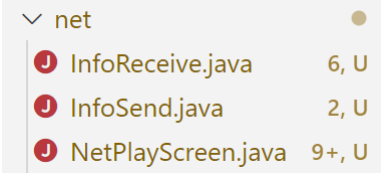

```
//在测试中before用于在测试前初始化一些类别，用expected来测异常，timeout来测性能。
//在World的测试中，对空位加入物品的算法进行正确性验证
public class WorldTest {
    World sample;
    Items a;
    @Before
    public void init()
    {
        sample=new WorldBuilder(5, 5).makeCaves().build();
        a=Items.GOURD;
    }
    @Test
    public void test1()
    {
        sample.addAtEmptyLocation(a);
        for(int i=0;i<5;i++)
        {
            for(int j=0;j<5;j++)
            {
                if(sample.item(i, j)!=null)
                {
                    assertSame(a, sample.item(i, j));
                }
            }
        }
    }
}
......
//比如此处测试构建世界函数的性能:
@Test(timeout = 1000)
    public void test0()
    {
        //smooth函数的超时测试
```

```
        sample=new WorldBuilder(5,5);
    }
......
//此处来测试存save函数在路径错误时会不会抛出正确的异常。
@Test(expected = Exception.class)
    public void test1()throws Exception
    {
        sample.save("wrong path");
    }
```

3.游戏联机功能×：

实现联机功能，写client，server端，其实看了很多遍老师的课件也不懂。这种东西必须靠上网搜一些实践教学，在写代码的过程中一点点悟。所以我在B站看了许多java网络编程的视频。大多涉及到一个网络聊天室的例子，网络聊天室要编写client，server端互相发送信息，多人联机server端还要实现负责更新全体状态等功能。在学习后，我对我游戏的联机想法是：

写一个服务端运行游戏框架，以线程来接收各玩家信息，每次更新游戏状态都向所有玩家发包同步状态。玩家客户端就只用打印地图并案件操作发包就行。但在网上学了一些杂七杂八的NIO selector的知识并不能很会用。所以我就又想的是由于只有两个玩家进行游戏，所以他们分别带着不同端口的参数运行游戏框架，进入客户端的NetPlayScreen：



这个游戏运行界面继承了单机的PlayScreen但在其基础上增加了Payer2。

首先介绍InfoReceive类，是一个线程实现UDP不停读序列化对象的类：

```
public class InfoReceive implements Runnable{
    DatagramSocket socket=null;
    private int port;
    NetPlayScreen t;
    public InfoReceive (int port,NetPlayScreen t)
    {
        this.t=t;
        this.port=port;
        try{
            socket=new DatagramSocket(port);
        }catch(Exception e){
            e.printStackTrace();
        }
    }

    @Override
    public void run() {
        while(true)
        {
            try {
                byte[] data=new byte[1024*60];
                DatagramPacket packet= new DatagramPacket(data,0,data.length);
                //接受包裹
                socket.receive(packet);
                //分析数据
                System.out.println(port+": 收到一个包");
                byte[] datas=packet.getData();
```

```
                int len=packet.getLength();

                ObjectInputStream ois=new ObjectInputStream(new
BufferedInputStream(new ByteArrayInputStream(datas)));
                Object r=ois.readObject();
                if(r instanceof content)
                {
                    content tmp=(content)r;
                    t.iswin=tmp.iswin;
                    t.ispause=tmp.ispause;
                    t.world=tmp.world;
                    t.world.items[t.player2.x][t.player2.y]=null;
                    t.world.items[tmp.player.x][tmp.player.y]=t.player2;
                    t.player2.x=tmp.player.x;
                    t.player2.y=tmp.player.y;
                    t.score=tmp.score;
                    t.level=tmp.level;
                }
                System.out.println(port+": 更新了一次地图");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

    }
}
```

可以看到其实现的功能就是从特定端口不停读取数据，读到的是一个content类，其中是我把地图信息给装进去了，每次读到信息就更新本客户端的地图信息。（这里本来是直接把整个界面给当成包发送的，结果涉及到一个UDP包超过大小限制的Exception，我想不会还要涉及到包的分割吧，所以就精简了发送的内容用content包装）

那么回到NetPlayScreen的游戏界面，初始化就运行不断抓包的线程：

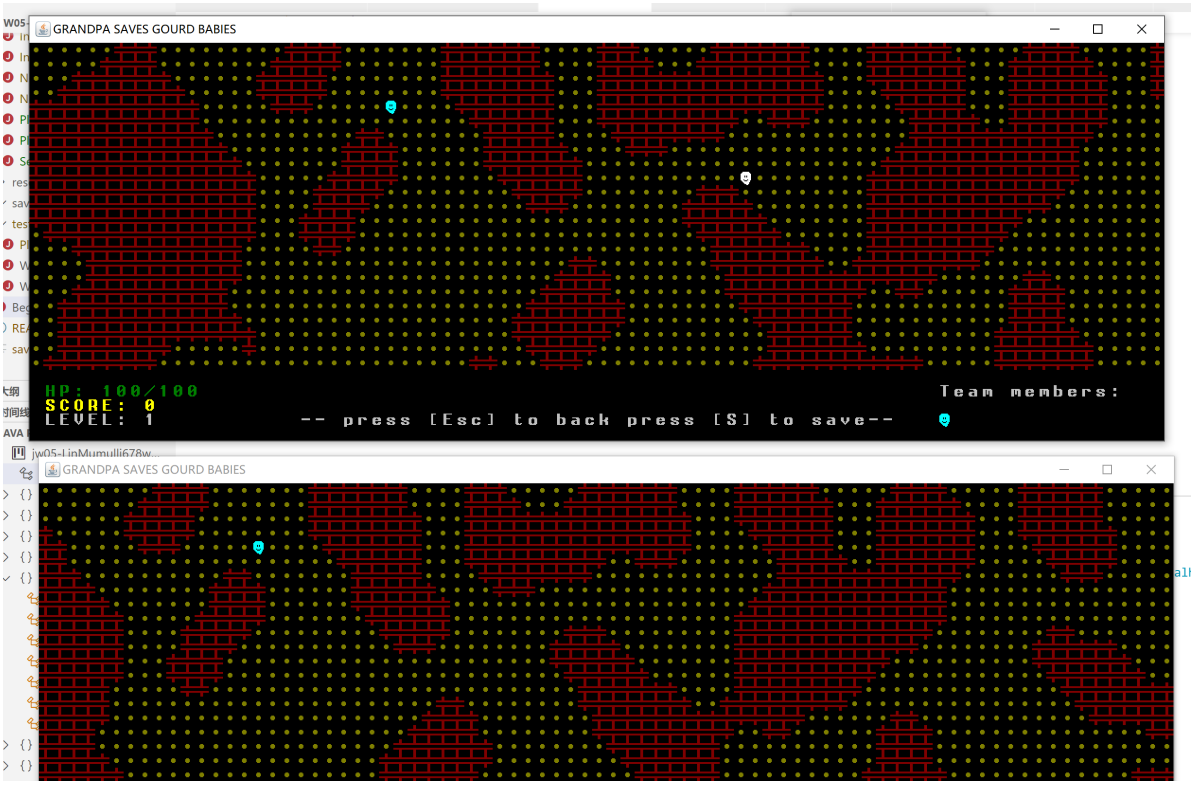
```
public NetPlayScreen(int port,int sendport,int tarport){
        super();
        //Net
        this.sendport=sendport;
        this.port=port;
        this.tarport=tarport;
        //
        this.player2=Items.PLAYER2;
        bos=null;
        oos=null;
        //
        try {
            //socket =new DatagramSocket(port);
            bos=new ByteArrayOutputStream();
            oos=new ObjectOutputStream(new BufferedOutputStream(bos));
        } catch (Exception e) {
            e.printStackTrace();
        }

        //开线程接包
        new Thread(new InfoReceive(port,this)).start();
    }
```

然后在玩家每次按键后，进行当前地图状态的发送：

```
//UDP发送当前状态
        //1、把所有信息保存起来
        content tmp=new content(world, player, score, level, iswin, ispause);
        //2、通过socket发送
        try {
            socket=new DatagramSocket(sendport);
            oos.writeObject(tmp);
            oos.flush();
            byte[] datas2=bos.toByteArray();
            DatagramPacket packet2 =new
DatagramPacket(datas2,0,datas2.length,new
InetSocketAddress("localhost",tarport));
            //发送包裹
            socket.send(packet2);

            System.out.println(sendport+"向"+tarport+"发了一个包");
        } catch (Exception e) {
            e.printStackTrace();
        }
        //释放资源
        if(socket!=null)
            socket.close();

        //
        System.out.println("玩家一发出信息");
        //world.update();
        //
        if(iswin==true)
            return new WinScreen();
        if (player.hp() < 1)
            return new LoseScreen();

        return this;
```

Player1的各种功能跟之前单机版本一样，而Player2我想的是在某个玩家的界面就把对方玩家看做一个Item，只是在自己的界面显示对方就可以了，但只能操作自己。所以游戏的步骤大概就是两个玩家分别对接上对方的端口号，不停接受对方的信息，每次按键更新打印整个世界，并发送最新的世界状态给对方。

但几经debug，最后虽然能实现地图的同步，但会出现bug：



具体表现为：虽然能够正确发送信息（下两图为两个客户端收发信息的记录，都正常）：

但玩家位置更新会出现问题，并且随着多次互相通信，依然会报下列错：

```
java.net.SocketException: The message is larger than the maximum supported by the underlying transport: Datagram send failed
```

因为个人时间原因，这个功能算是寄了，虽然不能保质量按时完成作业了，但会在后续更新中，打倒重来推出一个比较合理的游戏版本。

四、主要类的介绍（待完善）

0.入口类

集成JFrame框架，重写了repaint等主要函数，学习了一下播放音乐的方法用musicStuff类包含，main函数中设置JFrame的窗口名字，窗口位置等等参数。进入游戏。

```
public class Begin extends JFrame implements KeyListener {
    private static final long serialVersionUID = 191220059L;

    private AsciiPanel terminal;
    private Screen screen;

    //实现背景音乐播放
    public class musicStuff {
        void playMusic(String musicLocation)
        {
            try
            {
                File musicPath = new File(musicLocation);

                if(musicPath.exists())
                {
                    AudioInputStream audioInput =
AudioSystem.getAudioInputStream(musicPath);
                    Clip clip = AudioSystem.getClip();
                    clip.open(audioInput);
                    clip.start();
                    clip.loop(Clip.LOOP_CONTINUOUSLY);
                }
                else
                {

                }
            }
            catch(Exception ex)
            {
                ex.printStackTrace();
            }
        }

    }
    musicStuff musicObject;
    public Begin(){
        super();
        terminal = new AsciiPanel(80,28,AsciiFont.TALRYTH_15_15);
        add(terminal);
        pack();
        screen = new StartScreen();
        addKeyListener(this);
        repaint();
        musicObject = new musicStuff();
    }


    @Override
    public void repaint(){
        terminal.clear();
        screen.displayOutput(terminal);
```

```
        super.repaint();
    }

    @Override
    public void keyPressed(KeyEvent e) {
        screen = screen.respondToUserInput(e);
        if(screen == null)
        {
            this.dispose();
            System.exit(0);
        }
        repaint();
    }

    @Override
    public void keyReleased(KeyEvent e) { }

    @Override
    public void keyTyped(KeyEvent e) { }

    public static void main(String[] args) {
        Begin app = new Begin();
        app.setTitle("GRANDPA SAVES GOURD BABIES");
        app.setLocationRelativeTo(null);
        app.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        app.setVisible(true);
        // playMusic();
        String filepath = "resources/Title.wav";
        app.musicObject.playMusic(filepath);
    }
}
```

1.界面类

a.Screen接口类

每一个画面都要完成“打印画面内容”和“响应用户操作”两个操作，所以将这两个功能用Screen接口包含。

```
public interface Screen {
    public void displayOutput(AsciiPanel terminal);

    public Screen respondToUserInput(KeyEvent key);
}
```

b.StartScreen类

运行游戏第一个出现的画面类，负责打印最初的画面，玩家可以进行ENTER进入游戏或者B查看手册操作。

```
public class StartScreen implements Screen {

    @Override
    public void displayOutput(AsciiPanel terminal) {
        terminal.writeCenter("WELCOME TO THE", 2);
        terminal.writeCenter("Grandpa saves gourd baby", 4);
```

```
        terminal.writeCenter("-- press [enter] to start / press [B] to view the
handbook --", 22);
    }

    @Override
    public Screen respondToUserInput(KeyEvent key) {
        switch(key.getKeyCode())
        {
            case KeyEvent.VK_ENTER:
            return new PlayScreen();
            case KeyEvent.VK_B:
            return new TipScreen();
            default:
            return this;
        }
    }
}
```

c.WinScreen/LoseScreen类

显示胜利失败画面。

```
public class WinScreen implements Screen {

    @Override
    public void displayOutput(AsciiPanel terminal) {
        terminal.writeCenter("You won.", 3);
        terminal.writeCenter("-- press [enter] to restart --", 22);
    }

    @Override
    public Screen respondToUserInput(KeyEvent key) {
        return key.getKeyCode() == KeyEvent.VK_ENTER ? new PlayScreen() : this;
    }
}
```

d.PlayScreen类

```
public class PlayScreen implements Screen {
    private World world;
    private Creature player;
    private int screenWidth;
    private int screenHeight;
    private List<String> messages;
    private int score;
    private int level;
    public boolean iswin;
    public boolean ispause;
    public PlayScreen(){
        score=0;
        level=1;
        screenWidth = 80;
        screenHeight = 23;
        iswin=false;
        ispause=false;
        messages = new ArrayList<String>();
        createWorld();
```

```
        CreatureFactory creatureFactory = new CreatureFactory(world);
        createCreatures(creatureFactory);
    }


    //在这里调用工厂来创建你想要的生物
    private void createCreatures(CreatureFactory creatureFactory){
        player = creatureFactory.newPlayer(messages,this);  //将消息列表传递给
playerAI

        for (int i = 0; i < 10; i++){
            creatureFactory.newBat(this);
        }
        for (int i = 0; i < 30; i++){
            creatureFactory.newEBat(player,this);
        }
        for (int i = 0; i < 10; i++){
            creatureFactory.newHeart();
        }
        creatureFactory.newGourd();
    }

    private void createWorld(){
        //90 32
        world = new WorldBuilder(200, 170).makeCaves().build();
        //world = new WorldBuilder(90, 30).makeCaves().build();
    }
    //取打印左上角x，若当前Screen没有碰右下角，且不没有碰左上角，则返回当前Screen左上角x
    public int getScrollX() { return Math.max(0, Math.min(player.x - screenWidth
/ 2, world.width() - screenWidth)); }
    //取打印左上角y，同上
    public int getScrollY() { return Math.max(0, Math.min(player.y -
screenHeight / 2, world.height() - screenHeight)); }


    //在这里打印所有细节，比如得分，难度等级
    @Override
    public void displayOutput(AsciiPanel terminal) {
        int left = getScrollX();
        int top = getScrollY();

        displayTiles(terminal, left, top);
        displayMessages(terminal, messages);

        terminal.writeCenter("-- press [Esc] to back --", 26);

        String stats = String.format("HP: %3d/%3d", player.hp(),
player.maxHp());
        if(player.hp()>70)
        {
            terminal.write(stats, 1, 24,AsciiPanel.green);
        }
        else if(player.hp()<=70&&player.hp()>40)
        {
            terminal.write(stats, 1, 24,AsciiPanel.yellow);
        }
        else
```

```
        {
            terminal.write(stats, 1, 24,AsciiPanel.red);
        }

        score=world.score;
        String stats2 = String.format("SCORE: %d", score);
        terminal.write(stats2,1,25,AsciiPanel.brightYellow);
        String stats3 = String.format("LEVEL: %d", level);
        terminal.write(stats3,1,26);

        terminal.write("Team members: ",64,24);
        terminal.write((char)2,64,26,AsciiPanel.brightCyan);
    }

    private void displayMessages(AsciiPanel terminal, List<String> messages) {
        int top = screenHeight - messages.size();
        for (int i = 0; i < messages.size(); i++){
            //terminal.writeCenter(messages.get(i), top + i);
            terminal.writeCenter(messages.get(i), top);
        }
        messages.clear();
    }


    private void displayTiles(AsciiPanel terminal, int left, int top) {
        for (int x = 0; x < screenWidth; x++){
            for (int y = 0; y < screenHeight; y++){
                int wx = x + left;
                int wy = y + top;

                Creature creature = world.creature(wx, wy);

                //生物存在creature里，非生物（tile，floor）直接可用world中方法获取。
                if (creature != null)
                    terminal.write(creature.glyph(), creature.x - left,
creature.y - top, creature.color());
                else
                    terminal.write(world.glyph(wx, wy), x, y, world.color(wx,
wy));
            }
        }
    }

    //回合制，相应键盘后判断是否胜利，是否死亡。
    @Override
    public Screen respondToUserInput(KeyEvent key) {
        switch (key.getKeyCode()){
        case KeyEvent.VK_ESCAPE: return new StartScreen();

        case KeyEvent.VK_UP:player.moveBy( 0,-1); break;
        case KeyEvent.VK_DOWN:player.moveBy( 0, 1); break;
        case KeyEvent.VK_LEFT:player.moveBy(-1, 0); break;
        case KeyEvent.VK_RIGHT:player.moveBy( 1, 0); break;

        }
        // if(ispause==false&&key.getKeyCode()==KeyEvent.VK_SPACE)
        // {
        // }
```

```
            //每次按键响应后，更新一次世界
            world.update();
            if(iswin==true)
                return new WinScreen();
            if (player.hp() < 1)
                return new LoseScreen();

            return this;
        }
    //检查是否胜利，写了没调。。
        public Screen checkwin()
        {
            if(iswin==true)
                return new WinScreen();
            return this;
        }
    }
```

2.世界构建类

a.Tile砖块类

Tile声明为一个枚举类，枚举游戏中基本的砖块类型：墙壁，地板.....

```
public enum Tile {
    FLOOR((char)250, AsciiPanel.yellow),
    WALL((char)177, AsciiPanel.yellow),
    BOUNDS('x', AsciiPanel.brightBlack);

    private char glyph;
    public char glyph() { return glyph; }

    private Color color;
    public Color color() { return color; }

    Tile(char glyph, Color color){
        this.glyph = glyph;
        this.color = color;
    }

    public boolean isGround() {
        return this != WALL && this != BOUNDS;
    }

    public boolean isDiggable() {
        return this == Tile.WALL;
    }
}
```

b.World类

World类来容纳这些砖块，对砖块进行管理。并且对世界中的生物进行管理。其中用二维数组储存Tile，用一个List储存Creature。为了操作方便：

对于Tile：提供了获取特定砖块，挖掘（dig）功能实现函数。

对于Creature：提供了随机生成生物函数，删除生物函数，必要的更新生物状态函数、

```
public class World {
    private Tile[][] tiles;
    private int width;
    public int width() { return width; }

    private int height;
    public int height() { return height; }

    private List<Creature> creatures;

    public World(Tile[][] tiles){
        this.tiles = tiles;
        this.width = tiles.length;
        this.height = tiles[0].length;
        this.creatures = new ArrayList<Creature>();
    }

    public Creature creature(int x, int y){
        for (Creature c : creatures){
            if (c.x == x && c.y == y)
                return c;
        }
        return null;
    }

    public Tile tile(int x, int y){
        if (x < 0 || x >= width || y < 0 || y >= height)
            return Tile.BOUNDS;
        else
            return tiles[x][y];
    }

    public char glyph(int x, int y){
        return tile(x, y).glyph();
    }

    public Color color(int x, int y){
        return tile(x, y).color();
    }

    public void dig(int x, int y) {
        if (tile(x,y).isDiggable())
            tiles[x][y] = Tile.FLOOR;
    }

    public void addAtEmptyLocation(Creature creature){
        int x;
        int y;

        do {
            x = (int)(Math.random() * width);
            y = (int)(Math.random() * height);
        }
```

```
        while (!tile(x,y).isGround() || creature(x,y) != null);

        creature.x = x;
        creature.y = y;
        creatures.add(creature);
    }

    public void update(){
        List<Creature> toUpdate = new ArrayList<Creature>(creatures);
        for (Creature creature : toUpdate){
            creature.update();
        }
    }

    public void remove(Creature other) {
        creatures.remove(other);
    }
}
```

c.WorldBuilder类

构造世界，为了趣味性，这里用到了构造地牢的smooth函数来进行构建。该过程是用洞穴地板和墙壁随机填充该区域，然后通过将大部分相邻墙壁的区域变成墙壁，将大部分相邻地板的区域变成地板来平滑所有内容。重复几次平滑过程，就会得到一个有趣的洞穴墙壁和地板组合。这里涉及到我们的第一次大量更新数据，这样更新经常读取的数据的时候，一定要注意使用双缓存，勿直接修改原数据。

```
public class WorldBuilder {
    private int width;
    private int height;
    private Tile[][] tiles;

    public WorldBuilder(int width, int height) {
        this.width = width;
        this.height = height;
        this.tiles = new Tile[width][height];
    }

    public World build() {
        return new World(tiles);
    }

    private WorldBuilder randomizeTiles() {
        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                tiles[x][y] = Math.random() < 0.5 ? Tile.FLOOR : Tile.WALL;
            }
        }
        return this;
    }

    private WorldBuilder smooth(int times) {
        Tile[][] tiles2 = new Tile[width][height];
        for (int time = 0; time < times; time++) {

            for (int x = 0; x < width; x++) {
                for (int y = 0; y < height; y++) {
                    int floors = 0;
```

```
                    int rocks = 0;

                    for (int ox = -1; ox < 2; ox++) {
                        for (int oy = -1; oy < 2; oy++) {
                            if (x + ox < 0 || x + ox >= width || y + oy < 0
                                    || y + oy >= height)
                                continue;

                            if (tiles[x + ox][y + oy] == Tile.FLOOR)
                                floors++;
                            else
                                rocks++;
                        }
                    }
                    tiles2[x][y] = floors >= rocks ? Tile.FLOOR : Tile.WALL;
                }
            }
            tiles = tiles2;
        }
        return this;
    }

    public WorldBuilder makeCaves() {
        return randomizeTiles().smooth(8);
    }
}
```

3.生物类

生物类是一切生物的基础，不同生物的不同特性区别于他们的Ai不同。

```
public class Creature {
    public World world;
    public PlayScreen playscreen;
    public int x;
    public int y;

    private char glyph;
    public char glyph() { return glyph; }
    ......
    //这里调用Ai中的行走，不同的生物的Ai对于行走有不同的操作，比如player就会拾取当前位置的血包等等。
    public void moveBy(int mx, int my){

        if(mx==0&&my==0)
            return;

        Creature other = world.creature(x+mx, y+my);

        if (other == null)
        {
            if(world.item(x+mx, y+my)==null)
                ai.onEnter(x+mx, y+my, world.tile(x+mx, y+my),world.item(x+mx, y+my));
            else
```

```
            {
                // world.removeitem(x+mx, y+my);
                // if(hp>=90)
                //  hp=100;
                // else
                //  hp+=10;
                ai.onEnter(x+mx, y+my, world.tile(x+mx, y+my),world.item(x+mx,
y+my));
            }
        }
        else
            attack(other);
    }

    ......

    public void attack(Creature other){
        int atkva = Math.max(0, attackValue() - other.defenseValue())+1;
        doAction("attack the '%s' for %d hp", other.glyph, atkva);

        other.changeHp(-atkva);
    }

    public void changeHp(int amount) {
        hp += amount;
        if(hp>100)
            hp=100;
        if (hp < 1) {
            doAction("painfully die");
            world.score+=10;
            world.remove(this);
        }
    }

    public void dig(int wx, int wy) {
        world.dig(wx, wy);
        doAction("dig a hole");
    }

    public void update(){
        ai.onUpdate();
    }

    public boolean canSee(int wx, int wy){
        return ai.canSee(wx, wy);
    }

    public boolean canEnter(int wx, int wy) {
        return world.tile(wx, wy).isGround() && world.creature(wx, wy) == null;
    }

    public void notify(String message, Object ... params){
        ai.onNotify(String.format(message, params));
    }
    //这个函数十分重要，在执行动作时生物需要通知到附近的人，从而在屏幕上打印出动作的信息
    public void doAction(String message, Object ... params){
        int r = 10;  //响应半径
        for (int ox = -r; ox < r+1; ox++){
```

```
            for (int oy = -r; oy < r+1; oy++){
                if (ox*ox + oy*oy > r*r)
                    continue;

                Creature other = world.creature(x+ox, y+oy);

                if (other == null)
                    continue;

                if (other == this)
                    other.notify("You " + message + ".", params);
                else if(other.canSee(x, y))
                    other.notify(String.format("The '%s' %s.", glyph,
makeSecondPerson(message)), params);
            }
        }
    }
    ......
}
```

4.AI类

主要由基类CreatureAi以及他的派生类PlayerAi，BatAi等组成：

其中onEnter为走路函数。playerAi中override它时，实现走路拾取血包加血，拾取葫芦娃过关等操作。

canSee判断是否被地方发现。**只要接近敌方半径为8就被发现。**

wander为地方随机巡逻函数，**向周围的地板随机行走。**

```
public class CreatureAi {
    protected Creature creature;
    PlayScreen playscreen;
    public CreatureAi(Creature creature,PlayScreen plsc){
        this.creature = creature;
        this.creature.setCreatureAi(this);
        this.playscreen=plsc;
    }

    public void onEnter(int x, int y, Tile tile, Items item){
        if (tile.isGround()){
            creature.x = x;
            creature.y = y;
        }
        //
    }

    public void onUpdate(){
    }

    public boolean canSee(int wx, int wy) {

        if ((creature.x-wx)<8&&(creature.y-wy)<8)
            return true;
```

```
        return false;
    }

    public void onNotify(String message){
    }
    public void wander(){
        int mx = (int)(Math.random() * 3) - 1;
        int my = (int)(Math.random() * 3) - 1;
        if (!creature.tile(creature.x+mx, creature.y+my).isGround())
            return;
        else
            creature.moveBy(mx, my);
        // if (creature.world.tile(creature.x+mx, creature.y+my).isGround())
        //  creature.moveBy(mx, my);

    }
}
```

```
//玩家的ONENTER
public void onEnter(int x, int y, Tile tile, Items item){
        if (tile.isGround()){
            if(item == Items.GOURD)
            {
                this.creature.world.removeitem(x, y);
                this.playscreen.iswin=true;
            }
            else if(item == Items.HEART)
            {
                this.creature.changeHp(10);
                this.creature.world.removeitem(x, y);
            }
            creature.x = x;
            creature.y = y;

        } else if (tile.isDiggable()) {
            creature.dig(x, y);
        }
    }
```

4.物品类

物品类是一个枚举类：包括所有可以拾取的东西。

物品类目前主要包括血包❤，与葫芦娃（目前葫芦娃以类似于KEY的形式存在在地牢中，只要爷爷拾取到葫芦娃，即可将其加入队伍）。

```
public enum Items {
    HEART((char)3, AsciiPanel.brightGreen, "heart" ,10),
    GOURD((char)2, AsciiPanel.brightBlue, "gourd1" ,10);


    public PlayScreen playscreen;
    public int x;
```

```
    public int y;
    private char glyph;
    public char glyph() { return glyph; }

    private Color color;
    public Color color() { return color; }

    private String itemname;
    public String getname() { return itemname; }

    private int hpvalue;
    public int getHpvalue() { return hpvalue; }


    Items(char glyph, Color color, String itemname,int hpvalue){
        this.glyph = glyph;
        this.color = color;
        this.itemname = itemname;
        this.hpvalue=hpvalue;
        this.x=0;
        this.y=0;
    }
    public void setScreen(PlayScreen playscreen)
    {
        this.playscreen=playscreen;
    }
}
```