

Minecraft

——基于LWJGL 3 图形库

Minecraft

- 1.项目介绍:
- 2.背景介绍:
- 3.代码架构:
- 4.运行情况

作者:

[关键词]: 开放世界 光学 物理 计算几何 java LWJGL

1.项目介绍:

- 本项目借助**轻量级JAVA图形库LWJGL**, 通过三维建模等方式实现了一个仿Minecraft的开放世界3D沙盒游戏。我们实现了块、地图、植物、动物、人物控制、物理模拟、光学模拟、渲染等一系列基本功能。玩家可在开放世界内自由移动、跳跃、飞行（探索模式）、击碎、收集、放置各种块。整个地图包括多种复杂地形，且对光线、重力等物理情景进行了很好的模拟。
- **使用方法和功能介绍:**

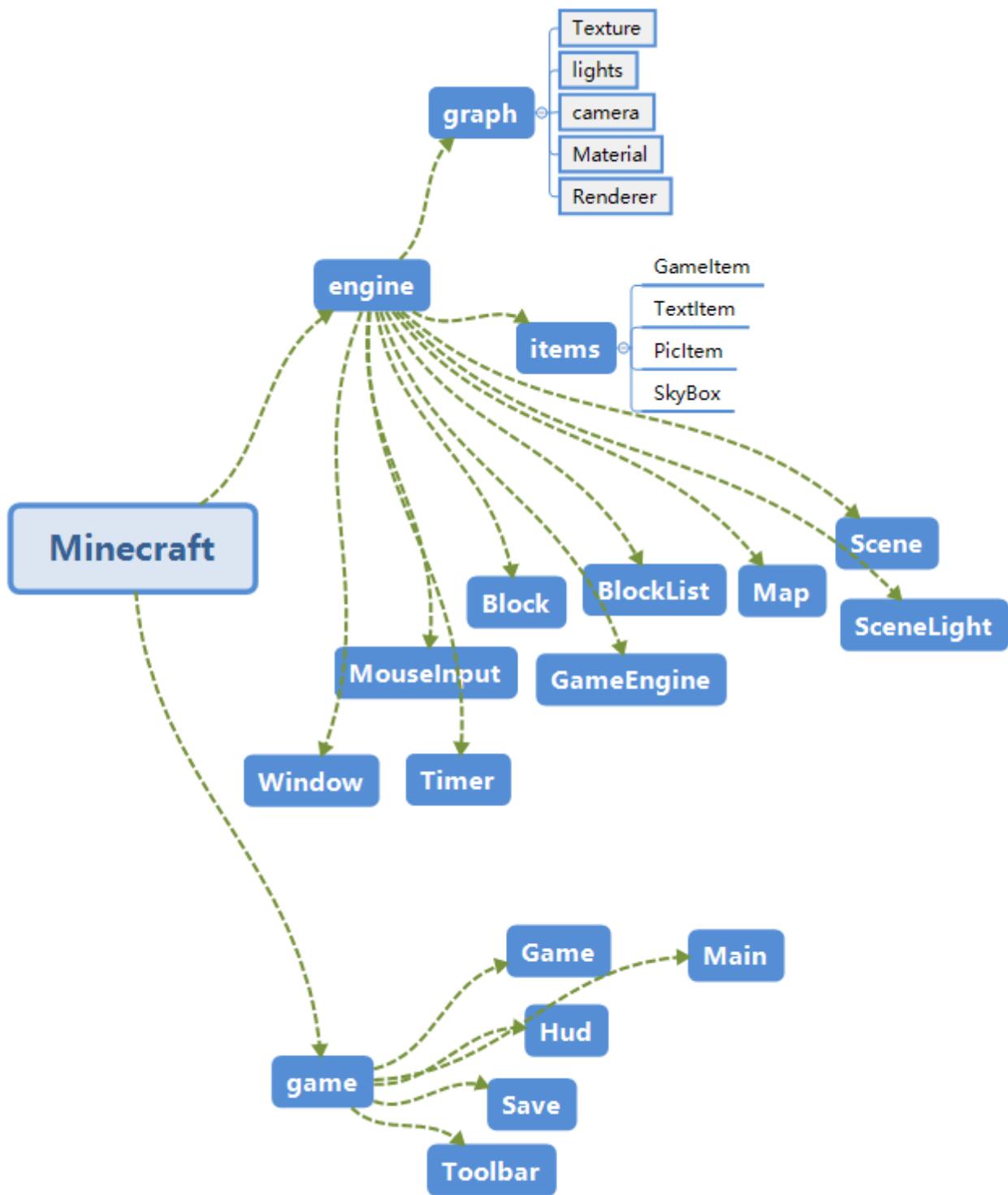
2.背景介绍:

LWJGL库, 全称是Light Weight Java Game Library, 是跨平台的Java库, 封装了对图形接口 (OpenGL) 、音频接口 (OpenAL) 、并行计算接 (OpenCL) 操作方法。封装的方式是直接并且高性能, 并且更安全和友好对于开发者, 更加适合Java开发。它不是框架并且不提供高级别的的工具相比被封装的库提供的方法。著名的游戏《Minecraft》（我的世界）, 就是用LWJGL开发的。

- **GLFW:** 用于创建OpenGL上下文, 以及操作窗口的第三方库。在项目中我们使用**GLFW**的部分函数实现窗口生成以及游戏运行基本逻辑gameLoop的控制。
- **JOML:** 即Java OpenGL Math Library, 用于向量处理等一系列数学计算的工具库。在项目中我们用来处理坐标变换、三维建模等一系列数学计算。
- **MemoryUtil:** 内存memory等的处理工具。在项目中, 配合OpenGL等工具实现较好性能的渲染和优化。
- and so on...

3.代码架构:

主体代码结构如下:



整体代码包括**控制**和**块**两大部分，具体按功能可分为：游戏逻辑控制、人物控制及场景更新、块、渲染、Hud、光影、地形生成等，接下来将按功能逐一详解：

- 游戏逻辑控制：

由于LWJGL基本可以看成是一个图形库，因此在游戏逻辑方面需要自己实现部分代码，相比于jME3等成熟框架增加了些许难度。基本的逻辑框架如下所示：

- GameEngine：

实现了初始化和gameLoop等游戏基本逻辑：

```
protected void gameLoop() {
    float elapsedTime;
    float accumulator = 0f;
    float interval = 1f / TARGET_UPS; //最小间隔时间，限定了帧率，避免了过于灵敏
    boolean running = true;
    while (running && !window.windowShouldClose()) {
        elapsedTime = timer.getElapsedTime();
        accumulator += elapsedTime;
        input();
        while (accumulator >= interval) { //更新
            update(interval);
            accumulator -= interval;
        }
        render(); //渲染
        if (!window.isVSync()) {
            sync();
        }
    }
}
```

在每个gameLoop中，输入input、场景更新update、渲染render交替进行，达到人物与场景实时交互的功能；通过UPS限定，来处理帧率等问题。

- Window:

使用gfw库函数解决窗口初始化和生成的问题，以及包括鼠标隐藏等细节问题。

- Timer:

处理时间相关的问题，辅助后续过程。

- Main:

定义主界面，并确定函数入口。本程序中图形界面主要依赖于javafx.swing和java.awt下的有关包。主界面是一个JFrame类作为载体将背景图片和有关的按钮加载其中。背景是将一个ImageIcon放入一个JLabel类中生成。图中有三个JButton类，对于三个主界面中央的按钮。为不遮盖背景图片，将setContentAreaFilled设置为false。对于每个JButton添加监听对象，当三个按钮的动作被监听时就产生对应的动作。

```
final JFrame frame = new JFrame();
final JButton button1 = new JButton("开始游戏");
final JButton button2 = new JButton("载入存档");
final JButton button3 = new JButton("退出游戏");
```

- Save:

实现存档功能和读入存档等辅助功能。

通过将人物位置和方块位置以文本的形式存储在record文件夹下实现存档。读取文档是直接在record的文件夹下读取有关的文本从而实现独挡。玩家在电脑键盘上键入G和退出时会自动存档。读档在主菜单部分通过按钮实现。

- 人物控制及场景更新:

- Camera:

摄像机视角，即用来控制人物视角和人物位置。

通过旋转角度来确定前方是什么方向，存储在front里：

```
public void setFrontFromRotation(){  
    front.x = (float)Math.cos(Math.toRadians(rotation.y - 90)) *  
    (float)Math.cos(Math.toRadians(-rotation.x));  
    front.y = (float)Math.sin(Math.toRadians(-rotation.x));  
    front.z = (float)Math.sin(Math.toRadians(rotation.y - 90)) *  
    (float)Math.cos(Math.toRadians(-rotation.x));  
}
```

移动（涉及矩阵运算）：

```
public void movePosition(float offsetX, float offsetY, float offsetz) {  
    if (offsetz != 0) {  
        position.x += (float)Math.sin(Math.toRadians(rotation.y)) * -1.0f *  
        offsetz;  
        position.z += (float)Math.cos(Math.toRadians(rotation.y)) * offsetz;  
    }  
    if (offsetX != 0) {  
        position.x += (float)Math.sin(Math.toRadians(rotation.y - 90)) * -1.0f *  
        offsetX;  
        position.z += (float)Math.cos(Math.toRadians(rotation.y - 90)) *  
        offsetX;  
    }  
    position.y += offsetY; //y是上下，不受旋转影响  
}
```

旋转（并更新front）：

```
public void moveRotation(float offsetX, float offsetY, float offsetz) {  
    rotation.x += offsetX;  
    rotation.y += offsetY;  
    rotation.z += offsetz;  
    setFrontFromRotation();  
}
```

- Game:

包含一系列的更新操作，包括：选择方块、放置方块、摧毁方块、鼠标自由移动视角、行走或奔跑、跳跃（完全模拟重力环境）、背包选择方块以及飞行模式等等。同时在每个gameLoop中对渲染进行优化，更新HUD信息。

选择方块核心代码：

获得面向的方向，再选择前方的一个块

```

for (float dist = SELECT_SENSITIVITY * 2; dist <= LIMIT_SELECT; dist += SELECT_SENSITIVITY * 2) {
    selectPosition.add(camera.getFront().normalize().mul(SELECT_SENSITIVITY * 2));
    Block block = map.getBlock((int) (Math.rint(selectPosition.x)), (int) (Math.rint(selectPosition.y)), (int) (Math.rint(selectPosition.z)));
    //省略部分代码
}

```

跳跃(模拟重力)：

给定初速度，随时间步衰减重力加速度

```

if (window.isKeyPressed(GLFW_KEY_SPACE)) {
    if (!flying && isOnBlock(camera.getPosition())) {
        cameraInc.y = jump_initial_speed;
    }
}
//省略部分代码
cameraInc.y += grav_acc * interval;

```

背包选择方块：

对toolbar进行操作，实现背包更新或更改方块种类的功能

```

if (window.isKeyPressed(GLFW_KEY_R) && System.currentTimeMillis() - lasselGroup > 400) {
    lasselGroup = System.currentTimeMillis();
    int t = toolbar.getSelGroup();
    t = (t + 1) % toolbar.groupCount;
    toolbar.setSelGroup(t);
}

```

• 块：

大部分物品为方块。读取立方体3D对象和贴图包中绘制的对应三视图贴图构建这类物品。花、草、蘑菇等植物为非方块物品。绘制两个交叉的垂直平面，并对四个面分别贴图，构建这类物品。使用Q键选择放置上一个可放置方块，E键选择放置下一个可放置方块。初始可放置方块为草方块。有些方块被设置为不能使用的，基岩被设置为不能摧毁的。树叶以及花、草、仙人掌等植物都使用透明贴图带有透明部分。为减轻透明部分与实际环境不对应的问题，渲染时先渲染不透明方块，再渲染透明方块。

◦ Blocklist:

Blocklist类用来存储目前支持的所有物品信息。静态的Blocklist类包含公开列表Bdic，维护所有类型物品。静态initblocks方法中，所有物品类型被初始化，并添加到Bdic中。每添加新的物品种类，都要向贴图文件中添加新的贴图，并向initblocks中加入新的初始化语句。Blocklist类还提供了一些实用的静态变量和方法。Bmap维护了从物品名字到编号的倒排字典，便于编程时利用told方法直接用名字代表方块。switchplace方法可以遍历Bdic列表，找到目前物品对性的下一个或上一个可放置物品。getMesh方法可以由给定物品种类找到对应模型与贴图，生成对应的Mesh对象。类Blocklist类的实例本身是物品类型类型，包含是方块还是植物等非方块类型、能否被放置、能否被破坏、编号、名称等属性。目前的问题主要是类没有被封装，物品类型没有从文件读取而是有源代码指定不利于扩展。

现阶段实现的块的种类：

```
Bdic.add(new Blocklist(true, false, false, false, "grass"));
Bdic.add(new Blocklist(true, false, false, false, "stone"));
Bdic.add(new Blocklist(true, false, false, false, "tnt"));
Bdic.add(new Blocklist(true, false, false, false, "sand"));
Bdic.add(new Blocklist(true, false, false, false, "oak"));
Bdic.add(new Blocklist(true, false, false, false, "plank"));
Bdic.add(new Blocklist(true, false, false, false, "dirt"));
Bdic.add(new Blocklist(true, false, false, false, "brick"));
Bdic.add(new Blocklist(true, false, false, false, "diamondblock"));
/* miserable blocks */
Bdic.add(new Blocklist(true, true, true, false, "bedrock"));

/* ores*/
Bdic.add(new Blocklist(true, true, false, false, "diamond"));
Bdic.add(new Blocklist(true, true, false, false, "coal"));
Bdic.add(new Blocklist(true, true, false, false, "iron"));

/* tree and leaf */
Bdic.add(new Blocklist(true, true, false, true, "leaf"));
Bdic.add(new Blocklist(false, true, false, true, "deadtree"));
Bdic.add(new Blocklist(true, true, false, false, "spruce"));
Bdic.add(new Blocklist(true, true, false, true, "sleaf"));
Bdic.add(new Blocklist(true, true, false, false, "jungle"));
Bdic.add(new Blocklist(true, true, false, true, "jleaf"));

/* ??? */
Bdic.add(new Blocklist(true, false, false, true, "cactus"));
Bdic.add(new Blocklist(false, true, false, true, "tallgrass"));
/* flowers */
Bdic.add(new Blocklist(false, false, false, true, "rose"));
Bdic.add(new Blocklist(false, true, false, true, "orchid"));
Bdic.add(new Blocklist(false, true, false, true, "tulip"));
Bdic.add(new Blocklist(false, true, false, true, "paeonia"));
Bdic.add(new Blocklist(false, false, false, true, "mushroom"));
Bdic.add(new Blocklist(false, true, false, true, "mushbrown"));
```

- **渲染：**

- Renderer:

- 处理三部分内容：

- 对天空的渲染renderSkyBox()
 - 对场景的渲染renderScene()
 - 对Hud的渲染renderHud()

- 并在每次渲染步骤内进行优化

```
public void render(Window window, Camera camera, Scene scene, IHud huds[]) {
    clear();
```

```

    if (window.isResized()) {
        glviewport(0, 0, window.getWidth(), window.getHeight());
        window.setResized(false);
    }

    // Update projection and view matrices once per render cycle
    transformation.updateProjectionMatrix(FOV, window.getWidth(),
    window.getHeight(), Z_NEAR, Z_FAR);
    transformation.updateViewMatrix(camera);
    renderScene(window, camera, scene);
    renderSkyBox(window, camera, scene);
    renderHud(window, huds);
}

```

- Material:

材质渲染，并定义光的放射比，在下面的光影中用来处理反射情景等等

每种材质的属性包括

◦ ambientColour diffuseColour specularColour texture reflectance

- Mesh:

在确定位置、渲染方式等一系列属性后对其进行渲染，在这个类中还实现了一定程度上的内存上的优化，即对释放掉的块进行cleanup操作

- Scene:

场景处理，对场景内的块进行增添或删除操作，并进行相应的渲染处理

- FontTexture/PicTexture:

分别对Hud中的文字和图片进行渲染操作，由此引申得到的TextItem和PicItem用于具体的辅助块的实现

- **Hud:** 平视显示器(Head Up Display)，通过修改渲染方式，实现以下功能：

- Toolbar:

◦ 背包栏，可以放置各种物品，总计27种方块或透明植物



- helpText:

◦ 帮助信息，用来显示功能

```
W/A/S/D - Move  
L Click - Place  
R Click - Destroy  
F - Toggle flying  
Space - Jump/Up  
Shift - Down  
1~9 - Switch item  
R - Switch toolbar  
ESC - Exit
```

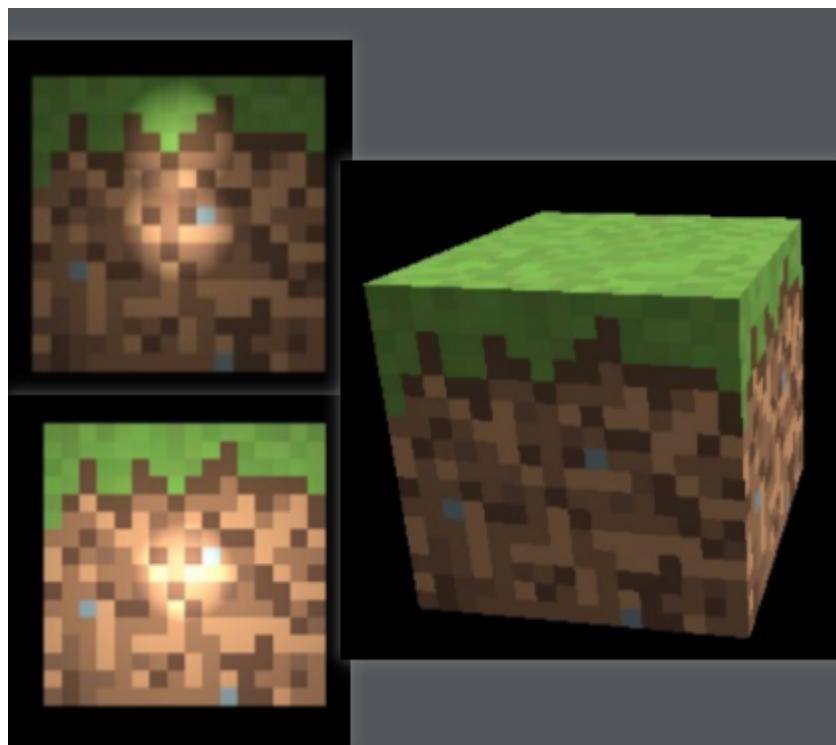
- statusText:

显示当前坐标，当前太阳角度，是否飞行模式

```
x = 0.00, y = 14.41, z = 0.00  
lightAngle = 120.30  
flying = false
```

- 光影:

- Light: 模拟光学环境，定义了多种反射模式



我们在这里定义了两种光：

- DirectionalLight
- PointLight

具体实现细节上：

采用 Phong 光照模型计算光照，将一个点受到的光照分为三部分考虑：

1. 环境光 A：在整个空间中均匀分布的光照

2. 漫反射 D: 光在物体表面向各个方向反射的光

3. 镜面反射 S: 平行的向一个方向反射的光

在一个点受到的光照可由三个分量求出, 为 $L = A + D + S$

每种光照根据材质不同有不同的颜色: $L = A * \text{textureColour} + D * \text{textureColour} + S * \text{textureColour}$

受篇幅限制, 各分量的详细计算不展开阐述

- SceneLight:

对场景进行光照模拟

实现的函数如下:

```
public Vector3f getAmbientLight() {
    return ambientLight;
}
public void setAmbientLight(Vector3f ambientLight) {
    this.ambientLight = ambientLight;
}
public PointLight[] getPointLightList() {
    return pointLightList;
}
public void setPointLightList(PointLight[] pointLightList) {
    this.pointLightList = pointLightList;
}
public DirectionalLight getDirectionalLight() {
    return directionalLight;
}
public void setDirectionalLight(DirectionalLight directionalLight) {
    this.directionalLight = directionalLight;
}
public Vector3f getSkyBoxLight() {
    return skyBoxLight;
}
public void setSkyBoxLight(Vector3f skyBoxLight) {
    this.skyBoxLight = skyBoxLight;
}
```

- 地形生成:

- Noise:

*PrimitiveNoise*类用来生成随机二维噪声。

Noise类使用Perlin Noise方法。其中的PrimitiveNoise方法得到一个随机排列, 对应原始噪声图像。使用getXY方法, 通过指定附近整点随机梯度与线性插值, 得到随机噪声图像在某一实数坐标点的取值。这样, 我们就得到了一个256*256的随机图像, 可以使用getXY得到任意一点的值。

Noise类通过对数量由参数level指定的多个原始噪声分形叠加, 得到更加精细的噪声。调用Noise对象的getVal方法, 可以获得任意实数坐标处的噪声值。

- Generator:

Generator类用来生成地图。

Generator的构造函数接收Xsize和Ysize指定地图大小，并有默认、超平坦、森林、沙漠、山脉可选的地图类型。为节省时间，这里没有使用面向对象的方法实现这些子类型地图，而是指定可选字符串参数mode，再根据mode的值决定类的行为。

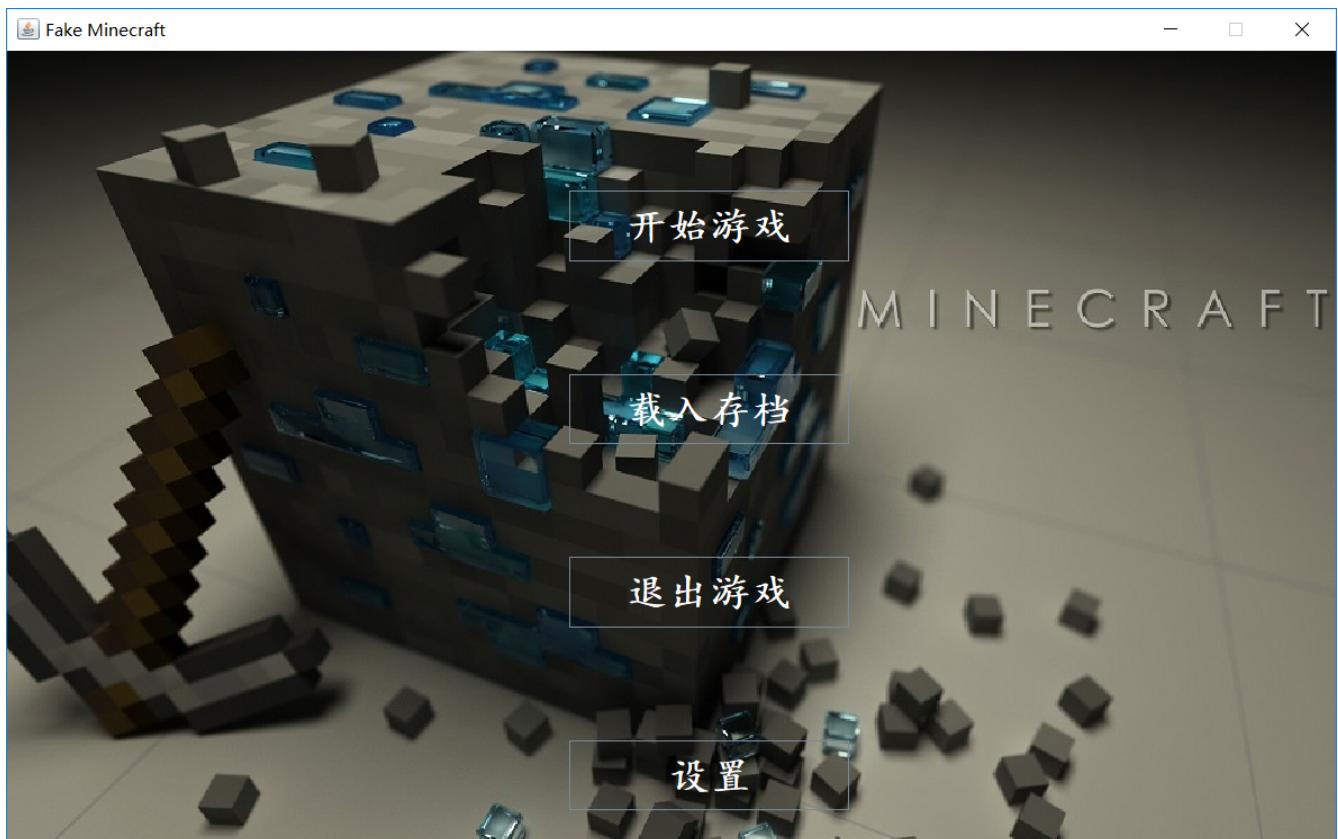
对输入参数，Generator计算偏移值，使得原点总在地图中央。其中，默认模式为草原，在构造函数中首先利用二维噪声生成高度图，然后根据高度图，利用genTerrain方法生成由基岩、石头、矿物、泥土、草方块组成的地形。此后，使用genPlants方法，根据噪声图与简单随机生成植物，包括花、草、蘑菇与树木。树木有随机的高度与叶子分布，植物也有随机类型。

超平坦地图高度固定为2且没有植物；森林地图生成更多的丛林树；沙漠由沙子覆盖且只有随机的枯树和仙人掌；山地噪声更粗糙，地形起伏更大，且树种为云杉。为得到生成地图结果，可以使用getItems方法，过滤过相同位置物品，并得到地图物品列表，这样可以简化生成步骤。

目前存在的问题主要是生成植物有时过少或过多，不够美观；不能生成动态生物群系；地表太浅，没有原版游戏里的地洞；树的形态不够美观。

4.运行情况

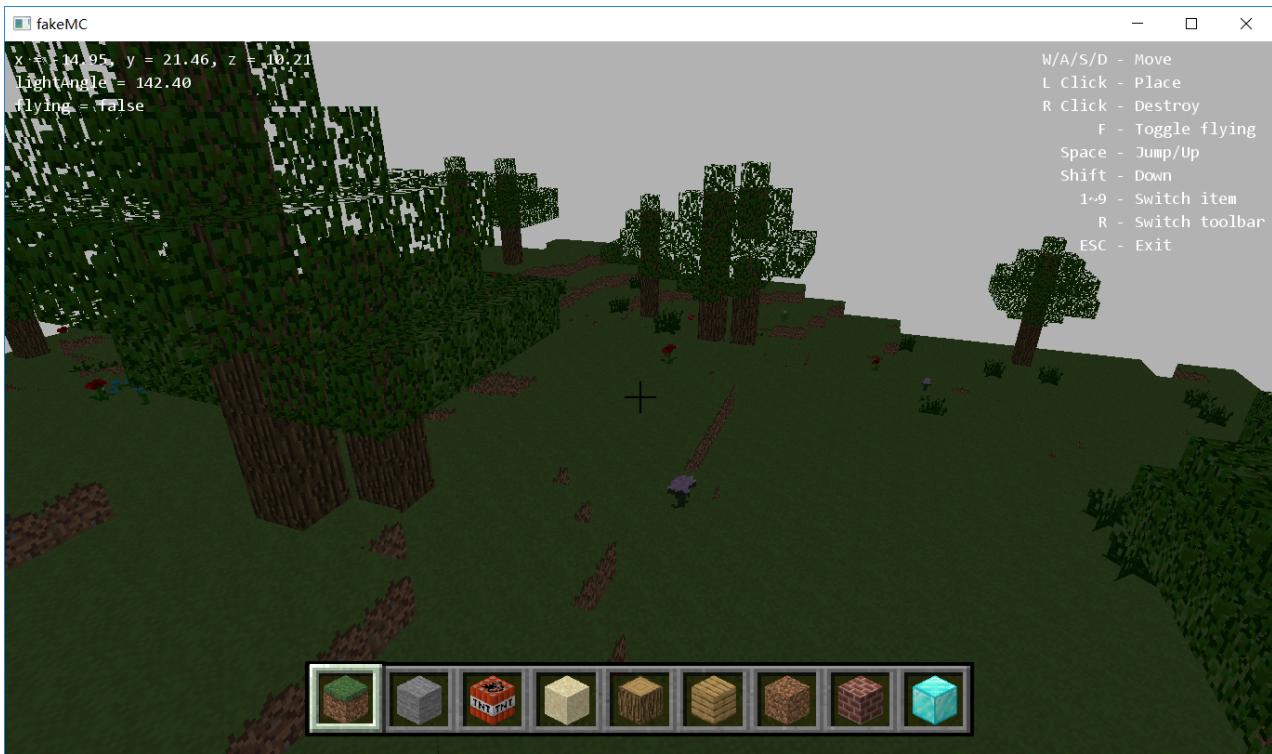
- **开始界面，支持载入存档 和 基本设置**



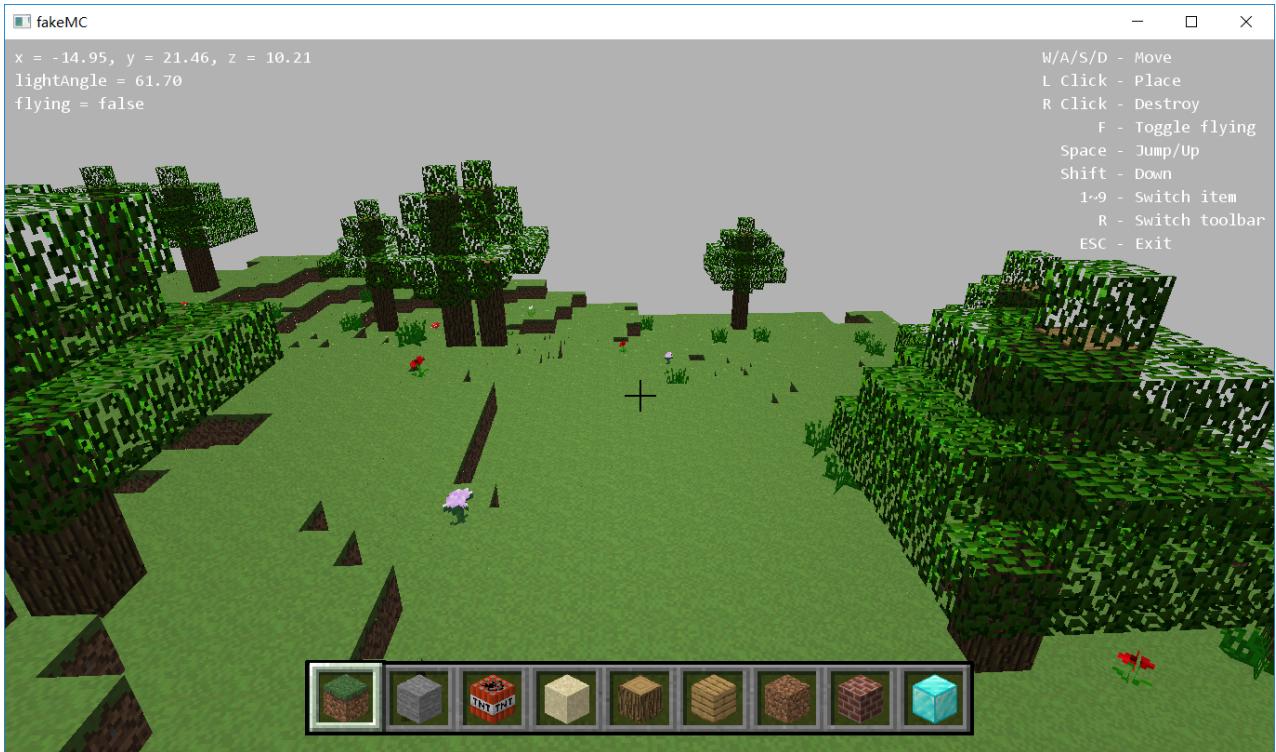
- **设置，可修改分辨率和生成地形**



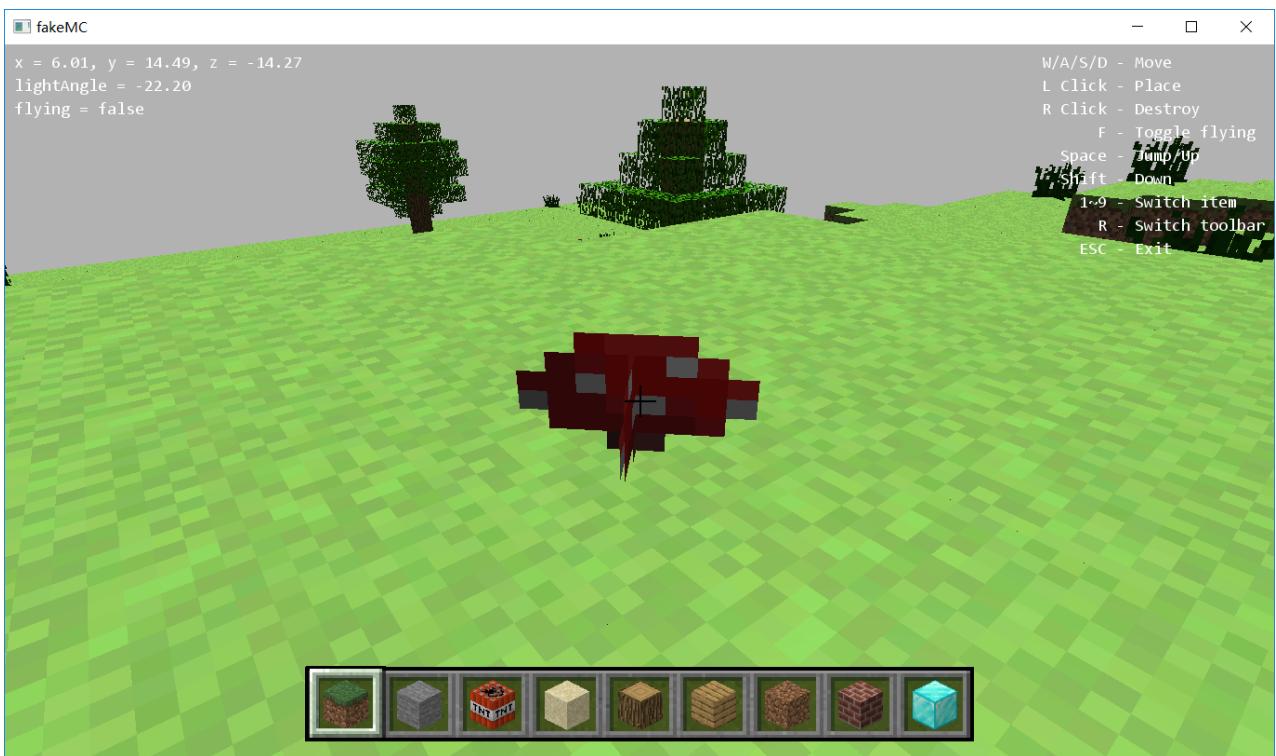
- 默认地形, 天黑



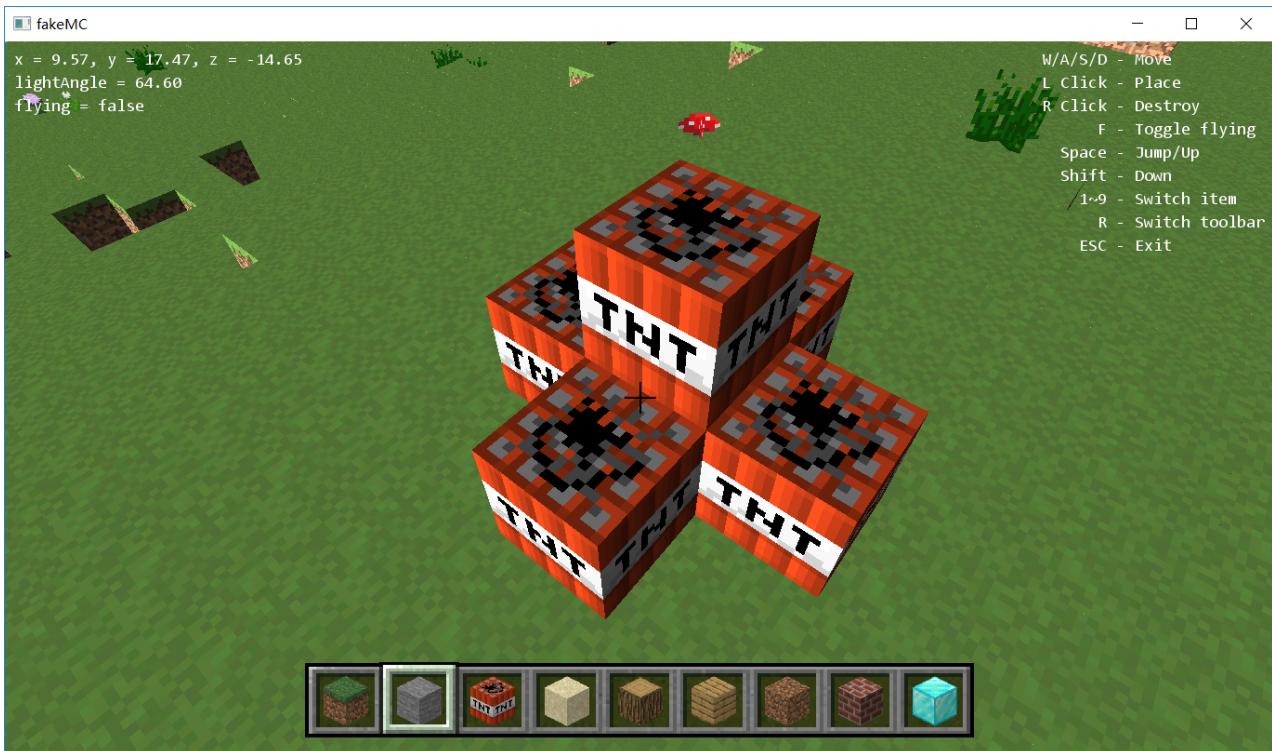
- 默认地形, 天亮



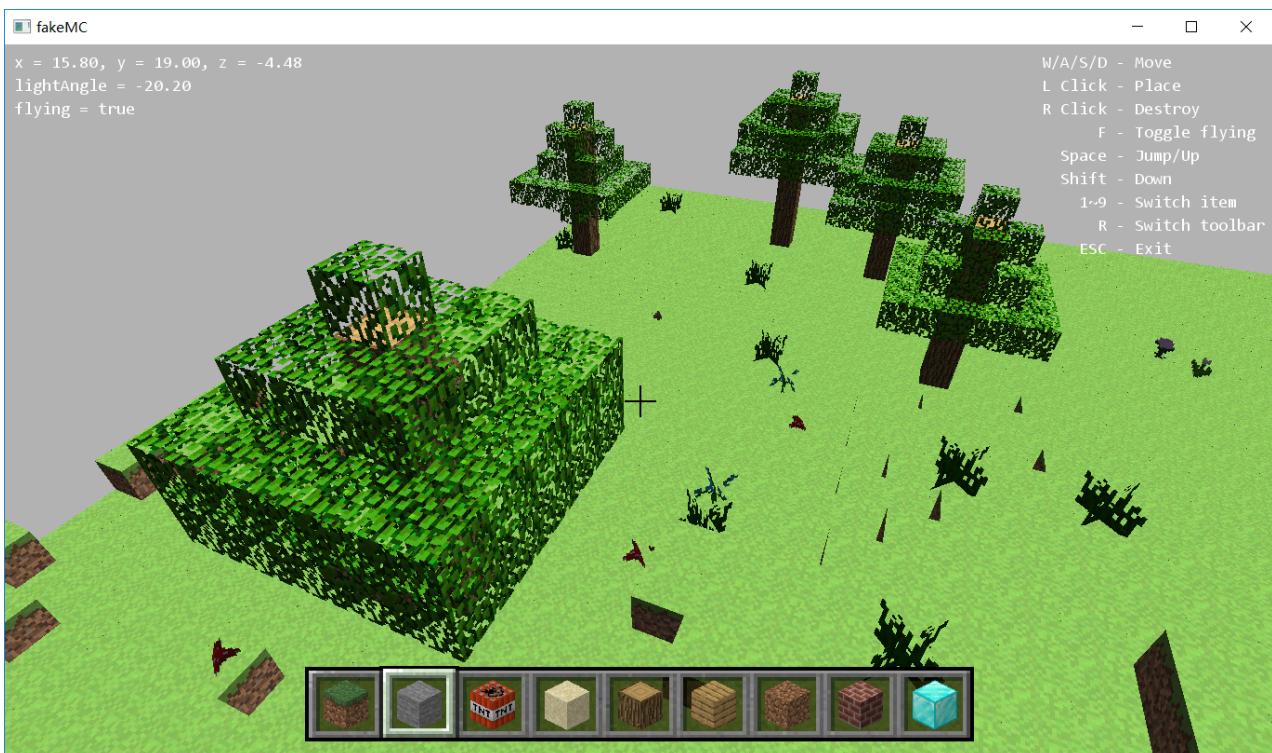
- 透明植物



- 放置的一圈TNT



- 飞行模式



- 切换工具栏



