中国科学: 信息科学

SCIENTIA SINICA Informationis

论文

Developing a Java Game from Scratch

蔡鸿彬[1]

1. 南京大学, 南京 210046

E-mail: pure.r.junko@gmail.com

收稿日期: 2021-12-31; 接受日期: 2021-12-31; 网络出版日期: 2021-12-31

国家信息科学基金 (批准号: 0000000, 0000000, 00000000)

摘要 本文探讨了如何从零开始设计一个游戏和使用 JAVA 语言对其进行实现,游戏整体基于 LibGdx 引擎进行开发，使用 Maven 进行管理，通过 Kryo 来实现序列化，并基于 Reactor 模式来实现网络通信功能；主要从功能实现和性能优化两大方面入手，由浅入深地对游戏的各层次设计进行分析，并展示最终的实现成果。

关键词 JAVA, 游戏设计, LibGdx, Kryo, 并发, 网络通信, Reactor 模式

1 游戏设计

1.1 从想法开始

在战斗类游戏中造成伤害是必不可少的，攻击敌人与受到攻击都往往离不开伤害的计算，尤其是一些较为复杂的游戏中，技能、人物和装备对伤害造成的影响都会叠加计算并最后返回一个最终值，如何搭配这些组合使得造成伤害最大化自然也是这些游戏的一部分，也有许多游戏爱好者热衷于探究更好的搭配，那么这是否可以作为一个可以拓展出一系列玩法的点呢？

如果把这个计算过程变得透明可控，把技能装备等对伤害的影响提炼为一个个独立的公式，使玩家能够手动控制这些公式的组合，面对不同类型的敌人打出不同的伤害（正负、导数、甚至到后期的矩阵等），并且在联网后与队友一起组合打出叠加的效果，将这一部分挖掘扩展为游戏的核心玩法，加之简单的随机生成的地图闯关，便设计好了一个游戏的玩法部分。

1.2 实现的构想

世界运行的基础——逻辑 要实现一个游戏，首先就要创造一个代码里的世界，这个世界由生物、环境以及运行的规则构成，生物在环境中遵循运行的规则与环境以及其他生物互动，那么我们现在要实现的就有：生物、环境与对世界全局行动的控制器。

引用格式: 蔡鸿彬. Developing a Java Game from Scratch. 中国科学: 信息科学, 在审文章
Cai Hongbin. Developing a Java Game from Scratch (in Chinese). Sci Sin Inform, for review

对于生物，一个生物的生命周期从出生到生存（做各种行为）再到死亡，那么它的出生显然由一个专门的工厂来生成会更好，而生物的行为则交给独立的线程去运行，那么这独立的线程便成为了生物的管理器。

而对于环境，在一个闯关游戏中，环境更多被叫做关卡地图，在地图上不仅承载着生物，还有它的地形、一些物件等，例如对玩家造成阻碍的陷阱和障碍物，对玩家产生帮助的道具、装备，一些装饰用的建筑，和远处的背景等，那么环境便包含了地图、道具、障碍物与陷阱。

而全局的管理，例如切换地图、将死亡生物移除、创造生物等行动则由一个类（Screen 类, 至于为什么后面会讲）来处理。

画面的展示——渲染 这些完善之后，游戏的底层逻辑基本就实现了，可是画面要如何显示给玩家呢？这就需要使用到游戏引擎的渲染功能了，在 LibGdx 中 Screen 就是可渲染的一种类，因此我们需要在 render 函数中渲染各个物体，但渲染物体首先需要图像资源，音乐也是如此，想要从外部加载这些资源，就要一个管理资源加载的管理器。

游戏画面可以显示后，我们还需要给玩家创造一个可以观看角色状态和各种游戏统计状态的图像界面，也就是 HUD(抬头显示)，作为 Screen 的成员实现。

序列化 为了将游戏整个状态序列化，最好的办法是创建一个用于将游戏状态存储成序列信息，并能从序列信息中读取状态复现到游戏中的模块，但基于历史原因和性能原因，我选择了对不同游戏物件进行不同的存储处理，因此只创造一个管理序列化的类，在其中实现对各种游戏物件的序列化与反序列化。

网络通信 想要联机就需要网络通信，想要多端口间网络通信就可以用服务端和客户端来实现，基于 Reactor 模式设计客户端和服务端，从而更好地处理不同的事件，而不同的事件其实就是不同的协议，对于每种协议规定其对于消息数据包的内容，在输出时序列化，接受后反序列化。

2 游戏实现

2.1 整体结构

游戏是基于 LibGdx 开发的，那么首先就得知道 LibGdx 应用的生命周期，如1所示，游戏的基类是 Game，拥有一个当前屏幕成员，在每次调用渲染、重定尺寸、暂停和继续等操作时都会对其当前屏幕调用相同的函数，也可以通过切换屏幕函数切换当前屏幕。而总的来说，LibGdx 各类之间的调用关系都是基于 render 实现的，上级在 render 中渲染自己的画面部分，更新状态并调用下一级成员的 render 函数。

而在我的游戏中如2所示, 渲染层级为:MadMath->Screen->UI/Map/Stage->UI:Stage/Map:Stage。

2.2 游戏主体

2.2.1 地图

因为美术资源为 16x16 的 Tile，则在地图中包含一个 GdxLib 的 TileMap 来放置 Tile，TileMap 里面实际上只有一个 Tile 的集合，而在地图上显示的所有 Tile 都是引用自该 Tile 集，这样做的好处在于不用重复创建 Tile 且动画帧的计算也只用计算一次;Tile 分为 StaticTile 和 AnimTile,StaticTile 中放置的是静止的 Tile,AnimTile 包含一个 StaticTile 数组会随着帧的变化给出不一样的 Tile 以达到动画效果；在设计中有些 Tile 是不可通行的 (如坍陷的洞等)，而有些 Tile 是陷阱（如尖刺陷阱，当尖刺露出时，玩家站在上面会掉血)。这些格子构成了地图的最底层，在其之上的是障碍物与生物，地图负责给出生物和障碍物的出生位置，同时也负责生成障碍物（调用工厂)，因为要做成 roguelike，所以 Tile 与障碍物的种类、数量和位置都是在地图创建时随机生成的。

2.2.2 生物

生物在地图中实际的模型是一张贴图和一个碰撞箱，生物的移动是通过矢量速度与矢量加速度完成的（自己写的一个模拟惯性系统)，当生物的碰撞箱接触到其他碰撞箱（如其他生物、障碍物）或者是不可通行的 Tile 时，生物在该方向上的速度会变为零。生物的动画往往是最复杂的，因为生物静止是一个动画，行走是一个动画，冲刺、攻击、受伤、死亡等等都需要动画来展示，所以一个生物带有一系列的动画贴图，需要根据生物的不同状态去显示不同的动画，因此在生物中设置一个状态枚举类，让生物每次渲染时根据自身的状态来选择相应的动画显示；当然只有动画是不够的，例如受击时的僵直、击退以及无敌时间的表示等等，对这些效果的实现不仅要用到我自己的惯性系统，还需要借助 stage 的 Action 功能为生物添加计时器触发事件。

2.2.3 道具

道具有拾取、抛弃和使用的功能，当玩家按下拾取键时会自动选取拾取范围内最近的道具进行获取，装备也是道具的一部分。目前实现的就是道具中的武器装备部分，在拾取装备，角色可以通过使用功能让武器挥舞，武器在挥舞过程中并不是通过计算武器本身和敌人碰撞先是否有重叠来判断攻击（这样太耗性能了)，而是通过对碰撞箱四点坐标和武器坐标的计算来得出距离和角度，只有距离在攻击距离内，角度也符合当前武器挥舞角度的敌人才会被判定攻击到。

2.3 Screen

Screen 是游戏主要显示的画面，也持有游戏中绝大部分控件，包括背景、窗口、按键等等。

MainMenuScreen 负责游戏开始界面，显示游戏开始时的动画，处理按钮事件，可以跳转至 SelectScreen、GameScreen 和直接离开游戏。

SelectScreen 负责游戏模式的选择和游戏难度的设置，游戏的难度设为 4 个等级，模式分为单人模式与多人模式，多人模式分为客户端和服务端，这之间的按键跳转就由 SelectScreen 来处理，最

后跳转到 GameScreen。

GameScreen 负责显示游戏主体的画面和所有全局性的逻辑，例如进入下一关卡时如何清除当前地图和资源，如何设置新建地图的配置，又或者是如何判断当前游戏状态，是否该结束或是暂停。而且 GameScreen 还有一个 Camera 负责将地图的哪一部分展示出来（也就是视角的移动和缩放），这些都是要在 GameScreen 中处理的。

SelectScreen 负责显示结算画面，可以重新开始游戏或者回到主菜单。

2.4 UI

UI 是游戏世界外的，只有玩家能看到的统计界面，HUD 抬头显示则记录了玩家角色信息和一些必要的游戏状态信息，如角色的武器选择，生命值和当前游戏分数、关卡、剩余怪物数量等；同时也负责菜单功能，点击右上角菜单按钮会显示菜单窗口，同时暂停游戏，然后根据按键的不同选择继续游戏、存档又或是退出。

2.5 多线程

首先 LibGdx 是不支持多线程的，也就是说，关于 LibGdx 的渲染部分必须在主线程进行处理，但这并不意味着我们不可以使用多线程，事实上用独立的线程处理各种生物间的交互和状态的改变，再由主线程负责把它们渲染出来是完全可行的。因此在 GameScreen 初始化线程池，创建生物后，将生物的线程放入执行即可；当然，为一个角色创建一个线程很合理，可为一个随时可生成可被消灭的怪物创建线程则显得有些低效，不妨创建一个怪物总线程，将其想象成一个地下城主在管理手下的怪物，让其收编新生成的怪物，命令怪物行动并遗弃死亡的怪物，这样就能更优雅的处理这些怪物了。

2.6 存档

存档功能其实就是序列化与反序列化，这里我用了 Kryo 来实现，因为 Kryo 的性能[1)]在 JSON 序列化中十分优秀。但由于 LibGdx 不仅不支持多线程，还不允许存储其 Gdx 的上下文, 所以它不能够被序列化（当然高效的存储也不会将整个对象序列化），所以需要具体来说就是在 Kryo 中对每一个要存储的对象的类进行注册，并在注册中为其创建一个序列化器来处理其序列化和反序列化，在游戏中随时存储，而读取时也要等游戏运行起来后在该读取游戏时再读取（为了保证可获取 Gdx 的上下文信息）。因为需要批量化处理的类都实现了许多工厂，因此对于不需要存储个体状态（如障碍物）的类可以直接在序列化时记录名字，在反序列化时使用工厂根据名字生成，但对于需要存储个体状态的类则需要在用工厂生成后再设置其状态。

1) https://github.com/eishay/jvm-serializers/wiki

2.7 网络通信

网络通信功能是使用 KryoNet 实现的，KryoNet 是和 Kryo 绑定的一个网络通信模块（基于 NIO），它将 NIO 封装起来，输入输出封装为 Kryo 的序列化读写，NIO 对事件的响应与处理封装为 Listener 监听事件，因此，只需要自己提前建好通信协议，将对应协议注册到 Kryo, 并创建对应协议事件的 Listener 即可完成网络通信。网络通信的各种处理流程：

客户端连接处理流程： 客户端连接到服务端-> 客户端发送连接协议到服务端-> 服务端根据连接协议中的角色信息在本地创建角色-> 服务端将游戏当前状态封装成游戏初始化协议发送至请求连接客户端并且将创建新角色协议发送给其余客户端-> 各客户端接收数据并处理。

客户端行为处理流程： 客户端输入事件在客户端本地进行处理-> 客户端将输入信息打包成对应协议的数据包-> 客户端将数据包发送给服务端-> 服务端受到数据包后根据数据包的协议类型激活对应的 Listener-> 服务端本地同步该数据包中的信息-> 服务端将该数据包原封不动的广播到其他客户端-> 其他客户端接收数据并处理。

客户端关闭处理流程： 客户端主动关闭/意外断连-> 服务端检测到客户端关闭-> 服务端删除客户端并广播到其他客户端-> 其他客户端接收并处理。

3 遇到问题

3.1 LibGdx

LibGdx 中跟渲染相关的类都需要 Gdx 的上下文信息，而这个信息是运行时产生且仅在 Gdx 应用主线程中存在的，因此带来了许多的问题，例如在多线程中，其他线程无法调用地图中的 tileMap 以及 Stage 里面的事件演员等，又或者是在网络通信中因为反序列化后需要将信息同步到游戏中，那么里面涉及到的各种操作都会引起 Gdx 上下文报错，而且糟糕的是 LibGdx 中很多数据结构都是线程不安全的，这也是的在多线程时需要额外小心，所以这些问题使得 LibGdx 的多线程并发显得比较困难。面对这些问题，我的解决方法是在做设计到 LibGdx 模块相关的操作时将其委托给主线程，让主线程在每次渲染前完成这些操作, 但是在对其单元测试时就没办法了，因为完成这些操作的前提是 Gdx 必须运行起来，而运行 Gdx 的办法只有启动应用，LibGdx 的开发者显然没有考虑这个问题（或者未能实现）而各大社区中对此的资料可以说没有，因此单元测试只能暂时告一段落（虽然可以做，但是若只能设计 LibGdx 完全无关的测试样例，这样做没有多大意义），目前的想法是用 Jmockit 去模拟测试。

3.2 Maven

使用 Maven 管理项目在项目较大时往往是个明智的选择，但当它加上 LibGdx 的组合就显得有点糟糕。LibGdx 并不是一个轻量化模块，它无法直接以模块的形式加入，通用的解决方法是使用开发者给出的软件构造一个 LibGdx 初始项目，那这个软件能构造一个使用 Maven 管理的 LibGdx 项

目吗？不能！对于这个问题，开发者给了一个解决办法，就是给出一个结合了 LibGdx 和 Maven 的初始框架，虽然配置起来有很多问题需要解决，但最终能够使用。

4 游戏介绍

MadMath 是基于 LibGdx 引擎开发的 2D 像素 roguelike 闯关游戏, 如3所示。

4.1 游戏操作

在游戏中按 WASD 或是方向键均可移动，按 Shift 键可以冲刺，Q 键和鼠标滚轮可以切换武器，数字键可以选择武器，按 E 键可以拾取道具（可拾取的道具上方会有箭头显示），鼠标左键点击可以挥舞武器，Z/X 键可以放大/缩小游戏画面

4.2 游戏关卡

玩家初始生成在地图左方，需要前往地图右方的楼梯或梯子处进入下一关卡，目前为无尽模式，关卡等级越高，怪物越多越危险而获得的分数越会更多。

4.3 武器攻击

不同的武器有不同的攻击方式，有刺、砍和圆弧挥舞，每把武器的攻击距离，攻击速度，击退效果和伤害公式都不一样。

4.4 伤害系统

怪物的生命值可正可负，正数生命值要求降至 0 以下死亡，负数生命值要求升至 0 以上死亡，怪物初始累积伤害为 0，被武器攻击后会被武器的伤害公式计算（如被 Add 伤害公式计算后累积伤害会增加一个数值），计算后的累积伤害会被作用在怪物上，如果对怪物产生相对与生命值正负相同的伤害，不会为怪物加血而是变为 0。

4.5 受伤

无论是人物还是怪物，受伤后会被击退，人物被击退后会获得短暂的无敌时间（会闪烁）。

4.6 陷阱

当陷阱处于激活状态时，人物踩上去会受到伤害，而怪物不会。

4.7 存档

当玩家进入下一关时系统会自动保存（但自动保存只有一个槽位，会覆盖先前的存档），玩家也可以在暂停菜单中可以随时选择一个存档槽位保存。在主菜单界面即可加载存档进入游戏。

4.8 多人游戏

在多人游戏中，死亡后会变成灵魂体，半透明、对他人没有碰撞体积且无法做除了移动以外的操作，在队友进入下一关后以 30% 血复活，但当所有玩家死亡后游戏就会结束。

5 课程建议

我个人认为课程十分的有趣，但可以在课上加入一些互动环节或者 OJ 会更有意思。

参考文献

Developing a Java Game from Scratch

Cai Hongbin[1]

1. *NJU, Nanjing* 210046*, China*
E-mail: pure.r.junko@gmail.com

Abstract This paper discusses how to design a game from scratch and implement it in Java language. The whole game is developed based on libgdx engine, managed by maven, serialized through kryo, and realized network communication function based on reactor mode; Mainly from the two aspects of function realization and performance optimization, this paper analyzes the design of all levels of the game from simple to deep, and shows the final achievement.

Keywords Java, game design, libgdx, kryo, concurrency, network communication, reactor mode

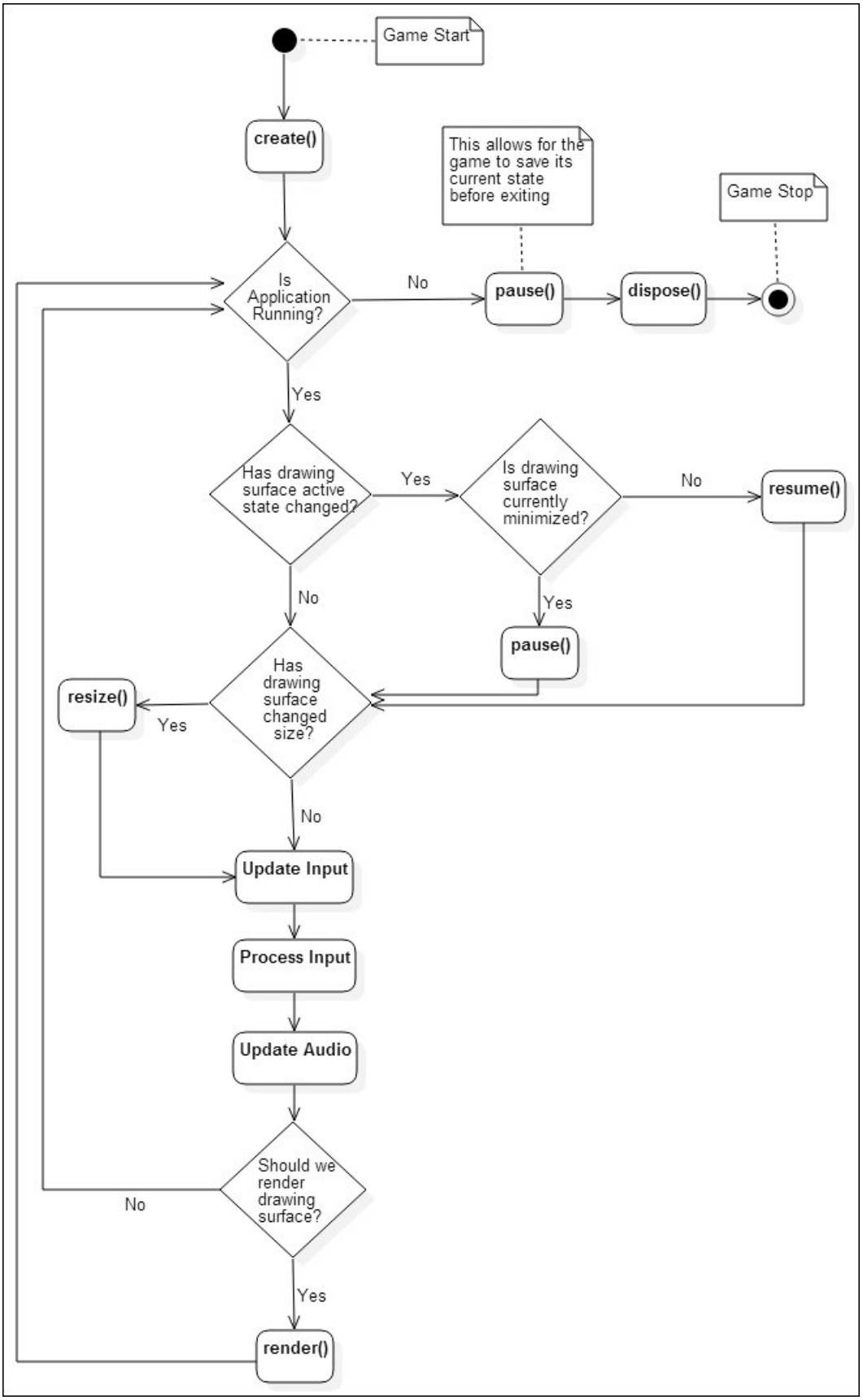

图 1 生命周期

Figure 1 LifeCycle

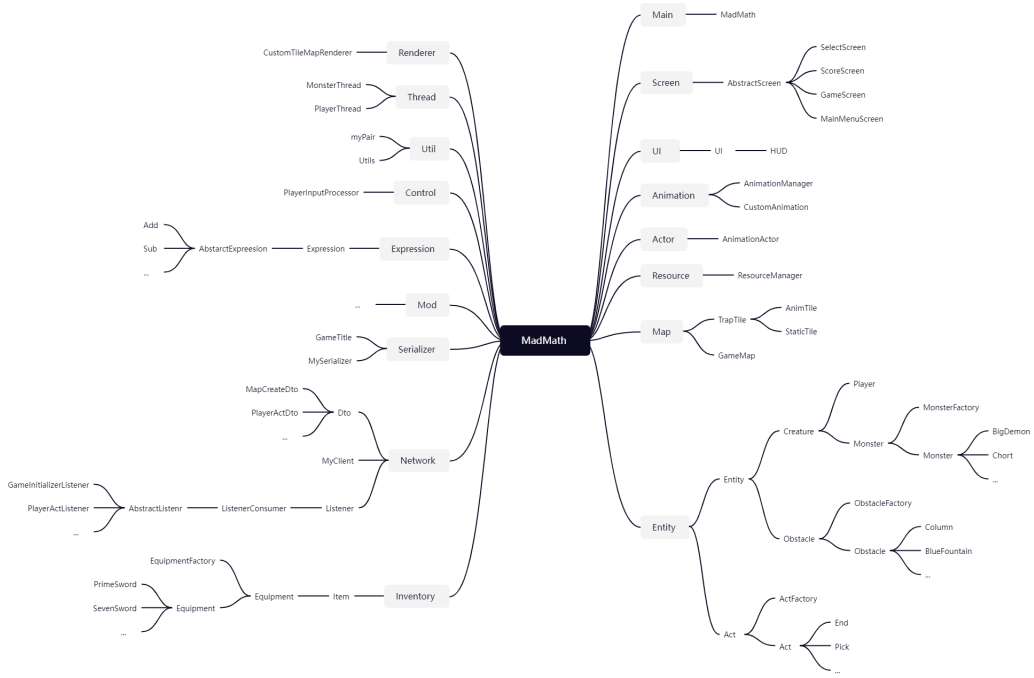

图 2　结构

Figure 2　Structure

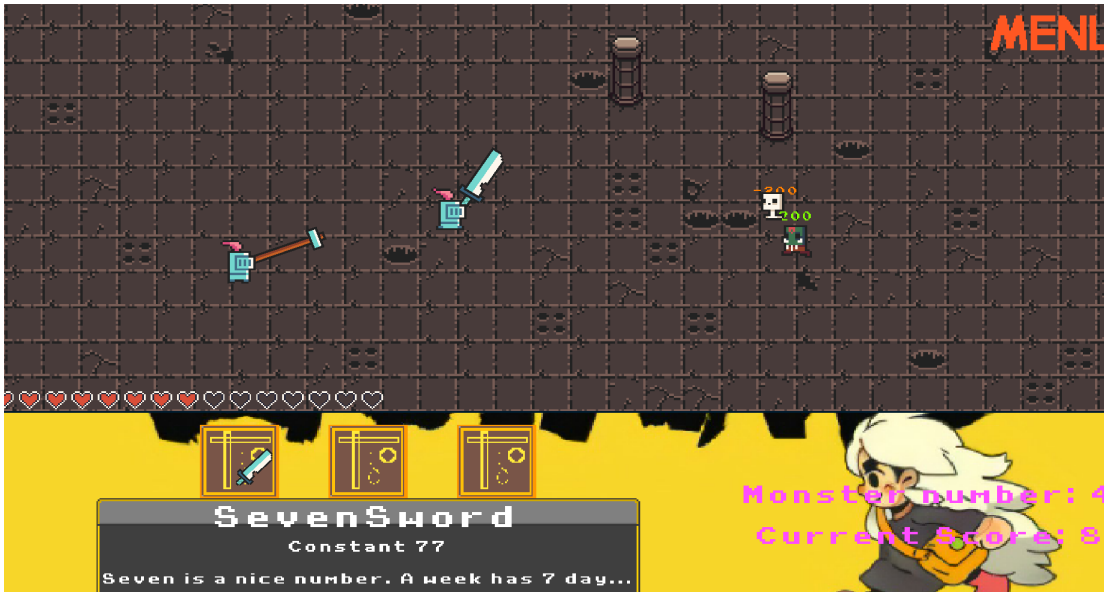

图 3　游戏画面

Figure 3　Game