操作

主界面内按<space>键，即弹出登录窗口。登录窗口内需要填入用户名和密码。对于首次登录的用户，需要勾选上“Register”。 而后会进入游戏界面等待别的玩家登录。方向键控制方向，绿色是食物。按 Esc 键退出游戏界面。点击叉可关掉游戏。按 F11 全屏。

界面设计

主界面是一个简单的文字和贴图。采用点阵式的界面，模仿 gnome-nibbles 的游戏界面。所有界面窗口均可缩放。

类和模块的设计

这曾经是一个我精心设计的项目。但最后草率收工，时间不够了。2.8k 行，最后被客户端毁了，前功尽弃。

我将分模块大致讲一下设计方式（不画 UML 啦）。

游戏的逻辑处理和数据存储

这一部分采用了典型的 MVC 设计模式（包含 Model 和 Controller 的部分）。

- GameState: 一个 Model 类，存放游戏数据。实现了 Serializable 接口，可直接借助网络传输。
  - GameFactory: 一个简陋的工厂类，用于产生初始的游戏状态(GameState)。
  - GameStateObservable: 一个抽象的基类，会产生变动着的 GameState。
  - GameStateObserver: 一个接口，可以处理 GameState 的类，与 GameStateObservable 构成观察者模式。
- PlayerAction: 一个消息类，存储玩家的指令。实现了 Serializable 接口，可直接借助网络传输。
  - PlayerActionConsumer: 一个接口，可以处理 PlayerAction 的类。
- GameController: 一个 Controller 类，操控游戏数据(GameState)，完成游戏的逻辑处理。它继承自 GameStateObservable 类，实现了 PlayerActionConsumer 接口。

界面模块

这一部分是 MVC 设计模式里的 View 的部分。这部分的代码复用了今年科展我写的 Collision 体感游戏的部分代码（觉得 Collision 写得比这个大作业好多了 T^T，心塞，要考试了）。主要是实现了一个栈式的界面管理切换方式。

- View: 一个虚基类，代表一个界面，拥有从 onStart 到 onStop 的一个生命周期。
  - SplashView: 游戏开始的欢迎界面。
  - GameView: 游戏界面，实现了 GameStateObserver 接口，其构造函数接受一个实现了 PlayerActionConsumer 接口的类，并将键盘消息 转化成对应的消息发送至该类。
- Content: 在多个 View 之间传递数据的类。
- ViewManager: 负责实现一个栈式的界面管理，继承自 JComponent，可以 pushView 和 popView。

以及其他的一些类：

- GraphicsWrapper: 封装了常用的做图函数，实现了可以缩放且不影响图像质量的绘图。
- ImageManager: 负责载入图像资源并缓存的类。
- LoginDialog: 登录对话框。
- MainFrame: 客户端界面窗口主类。

网络通信

之所以，搞什么观察者，其实是为了把网络通信完全解耦出去。同样的类也可以用来本地与 AI 对战或是多人对战（当然最后没实现）。

- GamePlayClient: 继承自 GameStateObservable 类，实现了 PlayerActionConsumer 接口。可用于接受 GameState 和发送 PlayerAction（它和 GameController 继承关系很近，实际上正因如此，对于 GameView 并不在意它到底是本地玩还是远程玩）。
- GamePlayServer: 实现了 GameStateObserver 接口。可用于发送 GameState 和接受 PlayerAction （它和 GameView 继承关系很近，实际上正因如此，对于 GameController 并不在意它到底是在处理本地还是在处理远程）。

用户管理

这个游戏本来拥有完整的用户注册、登录、查询游戏列表，创建游戏。甚至可以观战。服务端功能都写好了，客户端硬 coding 了一套逻辑，所以看不出。

- UserInfo: 用于客户端查询用户信息时，传输回去。
- GameInfo: 用于客户端查询游戏信息时，传输回去。
- Users: 处理用户注册，登入，登出等请求。
- Games: 处理游戏创建，结束，加入等请求。
- GameCommand: 一个用于承载网络通讯命令的类。（实际上是一个命令+数据的通信协议）
  - CommandSender: 发送指令，客户端使用。
  - CommandReceiver: 发送指令，服务端使用，并且处理客户端的请求（诸如登入，创建游戏等等）。

相关算法

随机地形生成

`GameFactory` 随机选择出发点后生产一串墙，而后检查连通性，不连通重做。

出身地选择

`GameController` 会寻找离预设出身地点最近的空白区域，并且确保玩家起始方向不是对着墙。

碰撞检测

蛤？这点是散列的，不难。（我软工（1）的物理模拟游戏，多边形碰撞检测 Bug 多炸了，已经噩梦，这次真的不难）

使用的轮子

- Jline v2: 服务器端的命令行自动补全（官网说只能 linux 用，助教可以测试一下 windows 哈）.

协议

游戏按 BSD 协议 发布。