REPORT

基于智能合约的分布式大富翁游戏

June 19, 2020

姓名: 杨晨宇、蓝宇霆
学号: 517030910386、517030910376

Contents

1 项目介绍 2
1.1 背景 2
1.2 相关工作 2
2 项目环境 2
3 项目实现 3
3.1 合约实现 3
3.1.1 数据结构 3
3.1.2 合约事件 5
3.1.3 主要函数 7
3.2 前端交互 9
4 项目难点 11
5 后续工作 11
6 分工细节 12
7 参考资料 12
8 完整代码 12
8.1 附录：流程 UML 图 12

1 项目介绍

1.1 背景

智能合约是一种通过区块链实现的新技术。它是数字化的，存储在区块链中，并使用加密代码强制执行协议。正是由于它的强制性和安全性，利用智能合约开发一些简单的小游戏，可以有效地防止玩家篡改游戏数据，无需第三方监管的同时，有效维护游戏的公平性。

本次项目中,我们将利用智能合约开发一个分布式大富翁游戏，它支持多人、多对局同时进行，并将整个游戏逻辑、判定机制嵌入合约内部，玩家通过调用合约的函数推动游戏的进行，函数内部也会对玩家的身份进行验证，从而确保了安全性。另一方面，我们尽可能简化了代码逻辑，在保证游戏完备的情况下减少了开销，节约游戏成本。

1.2 相关工作

在实现该项目的过程中，我们主要参考了 Truffle 官方文档 [1] 和 Web3.js 官方文档 [2]。此外，由于编写合约的 Solidity 及与合约的交互过程对我们而言都较为陌生，因此我们也阅读了许多开源项目的源代码，其中包括官方样例 Pet-Shop[3] 以及非常有名的入门项目僵尸工厂 [4] 等等。虽然这些项目都与我们要做的大富翁有一定区别，但通过它们我们对于智能合约的应用有了更深刻的了解。

此外，我们还参考了 github 上一个利用智能合约实现五子棋的项目 [5]。该项目与我们的目的较为接近——事实上，我们的房间创建/加入的想法就脱胎于此，并在其基础上从双人扩展到多人。但该项目版本过于古老，且游戏机制远不如大富翁复杂，因此也只是作为参考，并未用于实际实现中。

我们的主体框架是在官方样例 webpack-box[6] 的基础上搭建的。该项目原本仅用来展示虚拟货币的支付过程。我们采用了它的目录结构，而其他部分，包括前端设计，页面交互，事件广播及合约逻辑最终全都由我们自己实现。

2 项目环境

本项目中我们以 Truffle 作为框架，采用 Solidity 编写合约，并通过 Remix IDE 对合约进行调试。前端方面主要使用 JS+HTML+CSS 的方式完成，利用 web3.js 实现与合约的交互，并通过 webpack 对项目进行整合打包。前端部分的编译主要采用了 WebStorm IDE。此外，我们使用了 Ganache 来搭建私链并检测区块状态。

- 18.04.2-Ubuntu x86_64 GNU/Linux

- node.js v12.17.0
- npm 6.14.4
- truffle v5.1.28
- webpack 4.41
- web3 1.2.9

3 项目实现

3.1 合约实现

3.1.1 数据结构

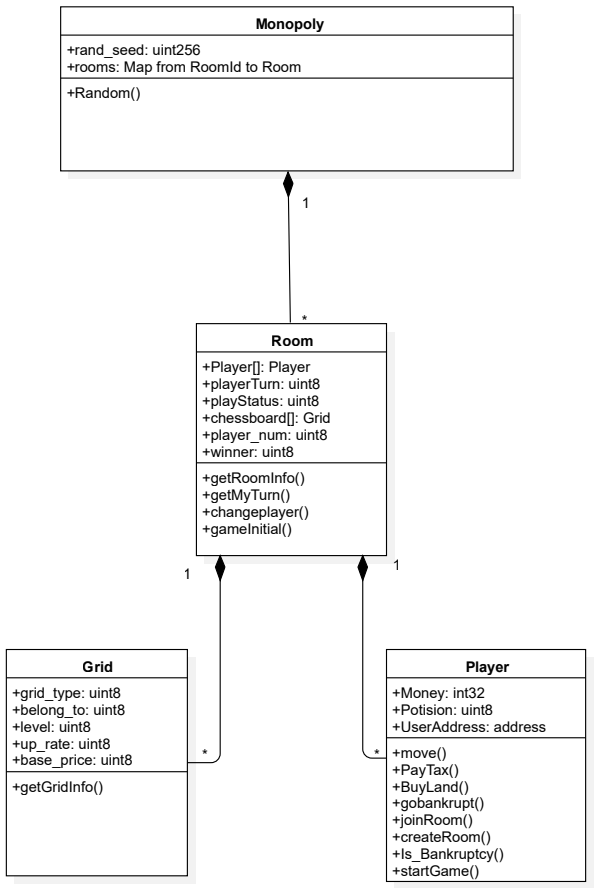

Figure 1: Class Diagram

- **Monopoly**

Attrubutes
(a) rand_seed: 用来实现高并发情况下的随机数
(b) rooms：一个关于房间 id 到房间的映射

Operations
(a) Random()：用来获得全局随机数的方法。

- **Grid**

Attrubutes
(a) grid_type: 用来表示土地的类型，0 表示不能购买，1 表示可以购买的土地。
(b) belong_to：对应游戏角色编号来表示土地所有权，默认是 0
(c) level：土地的等级，游戏角色可以购买别人的土地或者升级自己的土地。最高不超过三级，如果达到了 3 级土地将会变得不可购买。
(d) up_rate：土地价格的增长率，用来进行税务的计算
(e) base_price：土地的初始价格

对于土地的价格和税务，我们根据以上参数给出了相应定义

price = base_price * (level+1)

tax = base_price * level * (1 + up_rate/10 * level)

- **Room**

Attrubutes
(a) Player[4]: 记录房间中的游戏角色的信息。
(b) playerTurn：记录轮到哪个游戏角色进行交互。
(c) playStatus：记录游戏状态（等待/进行）。
(d) chessboard：保存该房间的游戏棋盘。
(e) player_num 记录该房间游戏开始时游戏角色的个数。
(f) living_num：实时记录存活的游戏玩家。
(g) winner：记录胜利者

Operations
(a) gameInitial()：初始化游戏
(b) changeplayer(): 用来切换到下一个行动的角色

- **Player**

Attrubutes
(a) Money 游戏角色的金钱，可以用来购买土地和支付税务；
(b) Potision: 玩家当前处于棋盘上第几格；
(c) address：玩家的地址，用来进行身份验证。

Operations
(a) move()：游戏角色进行移动操作，并且会触发支付税，买土地等事件。
(b) PayTax()：游戏角色走到了特殊的土地上，就会触发向另外的游戏角色，或者银行支付税务。
(c) BuyLand()：游戏角色行支付税务后，可以选择购买别人的土地升级，或者直接给自己的土地升级。
(d) gobankrupt()：清空该游戏角色的所有的资产。
(e) joinRoom()：加入房间
(f) createRoom()：创建房间
(g) Is_Bankruptcy()：判断是否破产
(h) startGame()：游戏房间的创建者可以决定是否开始游戏

3.1.2 合约事件

本小节中我们将介绍合约中用到的事件。每当合约内部数据发生变化时，前端通过监听这些触发事件来及时获取并更新相关信息，保证不同玩家间界面的一致。这里仅介绍这些事件的含义及参数，具体的触发过程将在 3.2 节中详细描述。

```
event GameStart(uint32 roomId);
```

- **roomId**: 对局进入开始状态的房间号；

通知用户某一房间游戏开始。

```
event PlayerChange(uint32 roomId);
```

- **roomId**: 人员发生变动的房间号；

（游戏开始前）当某一房间人员变动时，提醒房间内其他玩家更新参与者状态；

```
event OneStep(uint32 roomId, uint8 step, uint8 thisTurn, uint8 nextTurn);
```

- **roomId**: 当前房间号；
- **step**: 当前玩家移动的距离；
- **thisTurn**: 当前玩家的轮次；
- **nextTurn**: 接下来行动的玩家的轮次；

（游戏过程中）广播当前玩家的移动信息，标志着该玩家的轮次已经结束，并提醒下个玩家开始行动。

```
event BuyGrid(uint32 roomId, uint8 step, int32 cost, uint8 thisTurn,uint8 grid_pos);
```

- **roomId**: 当前房间号；
- **step**: 当前玩家移动的距离；
- **cost**: 土地的价格；
- **thisTurn**: 当前玩家的轮次；
- **grid_pos**: 可购买土地的编号；

（游戏过程中）广播当前玩家的移动信息，同时通知当前玩家该土地可以购买。

```
event BuyInfo(uint32 roomId, int32 cost, uint8 player_turn, uint8 grid_pos);
```

- **roomId**: 当前房间号；
- **cost**: 土地的价格；
- **player_turn**: 当前玩家的轮次；
- **grid_pos**: 可购买土地的编号；

（游戏过程中）当某玩家选择购买土地后，向房间内所有玩家广播交易信息。

```
event BankRupt(uint32 roomId, uint8 player_turn, uint8 step, uint8 nextTurn, uint8
    left_player_num);
```

- **roomId**: 当前房间号；
- **player_turn**: 当前（破产）玩家的轮次；
- **step**: 当前（破产）玩家最近一次移动的距离；
- **nextTurn**: 接下来行动的玩家的轮次；
- **left_player_num**: 场上剩余人数；

当某玩家资产小于 0 后，向房间内所有玩家宣告其已破产及剩余人数，如果此时剩余人数为 1，则自动触发结算程序，游戏结束；否则下一名玩家继续掷骰。

3.1.3 主要函数

```
function createRoom(uint32 roomId) public payable ;
```

- **roomId**: 想要创建的房间的房间号

玩家输入想要创建的房间的房间号，然后由服务端检查房间是否存在后，房间号是否符合规则，即可成功创建房间。

```
function joinRoom(uint32 roomId) public payable ;
```

- **roomId**: 想要加入的房间的房间号

玩家输入想要加入的房间的房间号，服务端检查房间是否存在，已满，游戏是否正在进行。若符合条件，成功加入，并且通过 PlayerChange 事件进行广播告知人员变动。

```
function PayTax(uint32 _roomId) private
```

- **roomId**: 正在进行支付税务的房间的房间号

若游戏角色走到了需要支付税务的土地上，则触发该函数，由触发者向土地所有者支付税务。

```
function gameInitial(uint32 _roomId) public  payable
```

- **roomId**: 正要进行初始化的房间的房间号

在该函数中，我们会对游戏的地图进行初始化，调用随机函数设置土地的类型，基础价格，增长率。初始化每个游戏角色的初始位置和启动资金。最后我们通过 GameStart 事件进行全局广播游戏开始。

```
function random() public returns (uint)
```

- 获取随机数

我们设计的大富翁游戏中，要求短时间内大量调用随机函数。我们的设计方法是把区块的 timestamp, difficulty，和调用的次数乘以时间，这三个参数输入 keccak256 函数中取余数获得。这样利用了大富翁游戏的高并发性的特点，获得了可靠性、随机性极强的随机数。

```
function buy(uint32 _roomId, uint8 player_turn, bool will_buy) public
```

- **roomId**: 正在进行购买土地的房间的房间号
- **player_turn**: 正在进行购买土地的游戏角色编号
- **will_buy**: 该游戏角色是否选择购买

在该函数中，游戏角色选择不购买，就会切换下一个玩家进行投掷，并且通过 OneStep 事件进行广播。如果游戏角色选择购买，我们就会减少和增加相应角色身上的钱，修改土地的所有权，并且进行升级。

```
function goBankrupt(uint32 _roomId, uint8 player_turn, uint8 step, uint8
    next_player_turn) private
```

- **roomId**: 正在进行破产操作的房间的房间号
- **player_turn**: 正在进行破产的游戏角色编号
- **step**: 该游戏角色是否选择购买
- **next_player_turn**: 下一个游戏角色编号

服务端会把该游戏角色的所有资产的所有权归还给银行，把土地等级清空，并且通过 BankRupt 事件进行广播。

```
unction move(uint32 _roomId, uint8 player_turn) public
```

- **roomId**: 正在进行破产操作的房间的房间号
- **player_turn**: 正在进行破产的游戏角色编号
- **step**: 该游戏角色是否选择购买
- **next_player_turn**: 下一个游戏角色编号

服务端会把该游戏角色的所有资产的所有权归还给银行，把土地等级清空，并且通过 BankRupt 事件进行广播。

```
function move(uint32 _roomId, uint8 player_turn) public
```

- **roomId**: 正在进行游戏角色移动操作的房间的房间号
- **player_turn**: 正在进行游戏角色移动游戏角色编号

Move 函数主要在玩家 roll 过后执行，游戏角色会走到随机出来的位置后进行支付税务，破产，买地等一些操作，触发事件。

3.2 前端交互

在启动 Ganache 服务（端口 9545）以及服务器后，打开网页（这里以 http://172.17.0.1 为例），我们就进入了初始页面（如2所示）。在这里，用户可以选择创建房间或是加入房间。创建房间会随机生成一个六位数的房间号，并自动跳转到房间页面。其他玩家可以通过输入已存在的房间号来加入该房间，加入的房间要求不为空且人数未满。此外，如果该房间正在进行游戏，那么新玩家同样无法加入房间。

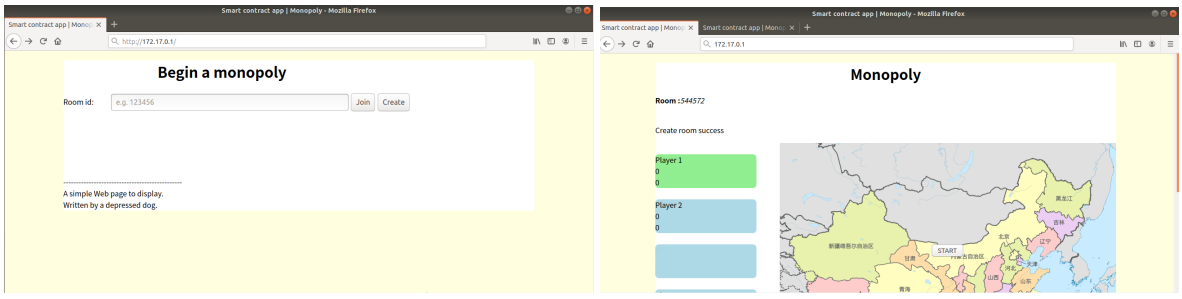

Figure 2: Start page and room page

在创建房间/加入房间以后，根据当前房间人数，玩家会被分配一个座位号。考虑到实际测试是在私链上进行，风险较小，同时每个窗口对应一名玩家可以简化测试过程，我们利用房间号 + 座位号的方式唯一确定一名玩家的身份，并利用他的地址作为验证。

房间页面如2所示，左边是房间内玩家的信息，分别为 ID，当前位置，当前金钱。由于游戏尚未开始，后两项都为 0。绿色的框代表自己，蓝色代表其他玩家。当有新玩家加入时，合约会向其他玩家广播，其他玩家收到后重新获取当前房间内玩家视角（图中为 P1 视角，可以看到玩家 P2 加入后信息已实时更新）。

当任一玩家点击 START 按钮，游戏立即开始合约会初始化玩家的金钱（每人初始 20000 元），并随机初始化每一格的初始价格（1000-4000）和税率（0-0.9）。初始化完成后向所有玩家广播 GameStart，前端收到后从合约获取地图信息并更新页面. 接着玩家按顺序行动，玩家的每一步行动都会被广播给所有其他玩家并更新在界面上，轮到某玩家轮次时，界面会出现 ROLL 按钮（如3），通过点击该按钮，玩家可以向前随机移动一定步数。为了防止作弊，该随机数生成过程也是在合约内部实现的。当玩家到达已有归属的地皮时，合约会自动扣除相应税务并支付给所有者。如果到达的地皮可以购买（不属于公共区域以及未达到满级），合约会广播一个 BuyGrid。该事件对应的玩家可以选择是否购买，如果选择购买，则合约会自动扣费、更改产权，并广播一个 BuyInfo 事件告知所有玩家。当一名玩家的所有行动都完成后，合约广播一个 OneStep 事件，提醒下一名玩家开始掷骰。

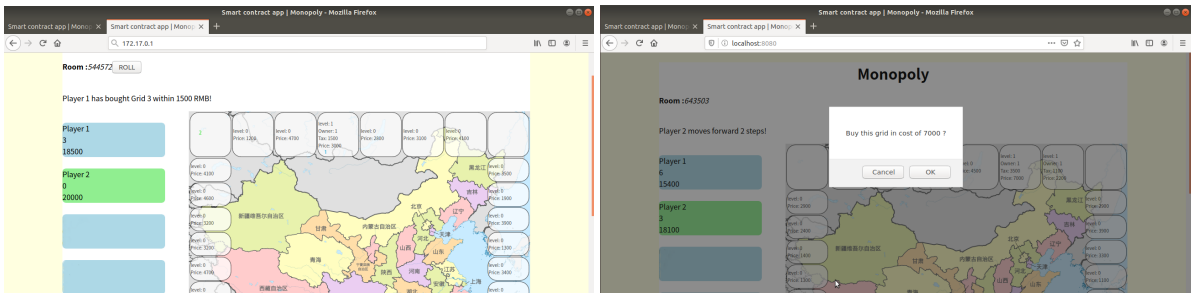

Figure 3: Roll and buy

在某一次行动中，一旦玩家资产小于 0，即宣告破产。破产信息以及当前剩余人数会随着一个 BankRupt 事件告知其他玩家（如4）。由于我们这里采用的是二人对战的形式，当一人破产后，游戏结束，系统会宣告玩家的胜利。如果剩余人数大于等于 2，游戏还将继续，此时 BankRupt 除了宣告破产以外，还将代替 OneStep，提醒下一位玩家掷骰。游戏结束后，合约会清除房间信息，同时所有玩家回到主页面。（完整流程图详见附录）

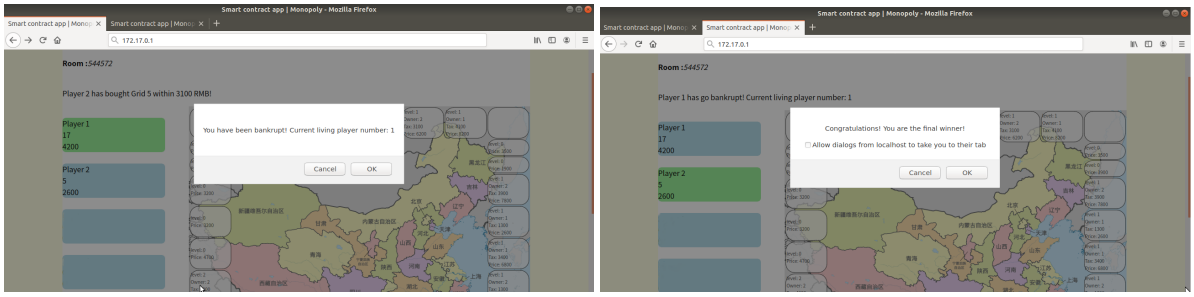

Figure 4: Game ending

4 项目难点

1. **随机数生成;**
   因为我们可能会在同一区块内多次调用生成随机数，而同一区块的时间戳等都是相同的。我们无法简单的根据区块的信息生成不同的随机数，于是我们设置了一个全局变量 rand_seed，每次调用 random 函数该 seed 都会发生变化，把这个参数加入到随机数生成的因子中，从而在单个区块中获得多个可靠的随机数。

2. **玩家间信息同步;**
   不同于一些智能合约实现的放置类手游，棋牌游戏的每一步都需要告知所有参与玩家，因此我们设计了一些必要的合约事件，并采用了能够支持订阅的 WebSocket-Provider。通过这些事件，前端能够及时得知合约内部数据的变化，从合约中获取更新后的数据，从而维持玩家间页面的同步。

3. **节省开销;**
   由于区块的每一次上链都需要一定的 gas，且代码逻辑越复杂，需要的 gas 越多。因此我们在合约中尽可能减少了诸如循环之类代码逻辑的使用。同时把一些不必要的操作（如 getPlayerInfo）封装成另一个 view 函数，分两次执行。减小上链过程中的代码量。

5 后续工作

1. **完善房间机制，添加掉线重连等功能;**
   目前游戏房间添加了基础的功能，但是在实际情况中，网络波动而产生的延迟，掉线情况是十分容易产生的，我们暂时还没有进行这些异常情况的处理。

2. **丰富游戏内容，使玩家更有参与感;**

目前我们只完成了大富翁移动，购买土地，交税，破产等基础功能。在未来的工作中，我们可以给棋盘添加诸如关进监狱，回到起点，强制缴费等功能增加游戏的可玩性和趣味性。

3. **用户界面优化，提高美观度;**
   我们对前端知识的掌握还不足，游戏界面需要的图像资源也不充分，而且前端设计并不是本次大作业的核心目的，所以我们的界面暂时还不够美观，在未来的工作中希望能对此进行优化和改进。

6 分工细节

杨晨宇主要负责游戏前端开发，框架部署，游戏逻辑及合约 API 的设计。
蓝宇霆主要负责游戏后端智能合约的设计，solidity 编写测试。

7 参考资料

1. Truffle 官网 https://www.trufflesuite.com/docs
2. Web3.js 官网 https://web3js.readthedocs.io/en/v1.2.9/
3. 官方样例 Pet-Shop https://github.com/truffle-box/pet-shop-box
4. 僵尸工厂 https://cryptozombies.io/en/course
5. 利用智能合约实现五子棋对战 https://github.com/Alexygui/Gobang
6. Webpack https://github.com/truffle-box/webpack-box

8 完整代码

https://github.com/Achronferry/SJTU-EE357-project-BlockChain

8.1 附录：流程 UML 图

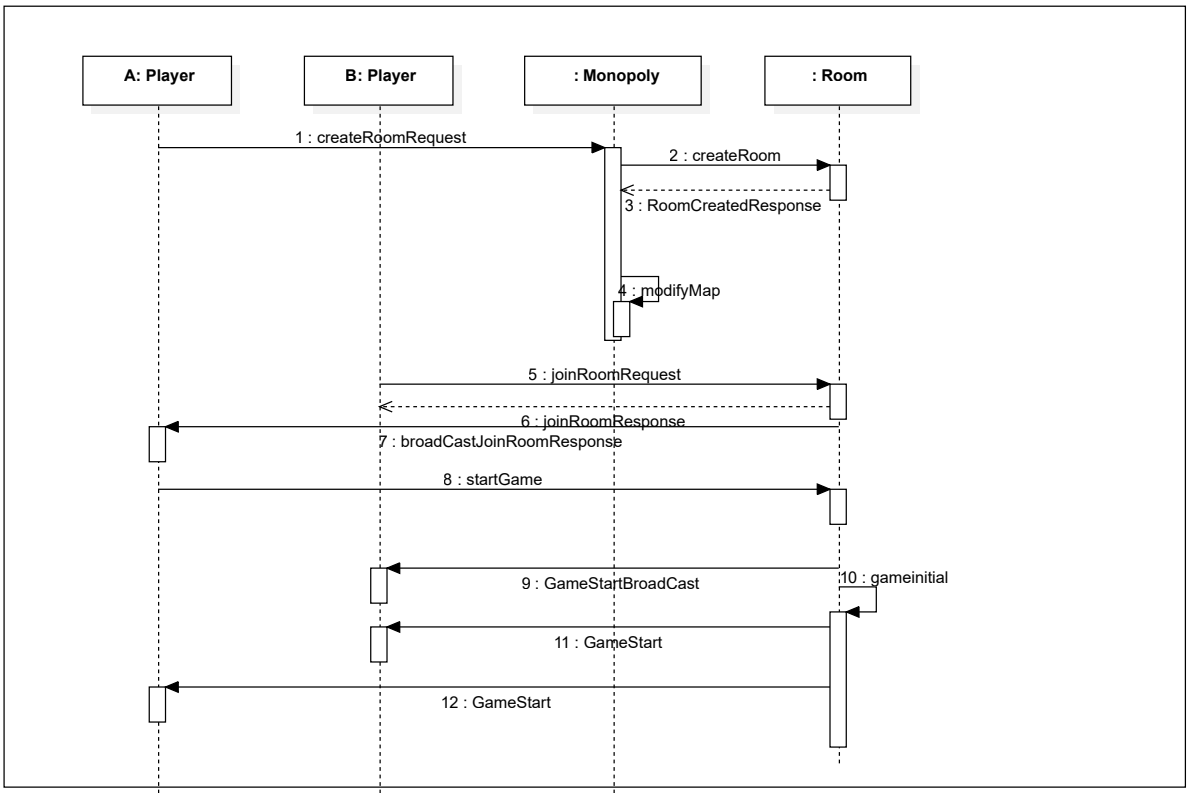

Figure 5: Sequence diagram of CreateRoom, JoinRoom, and StartGame

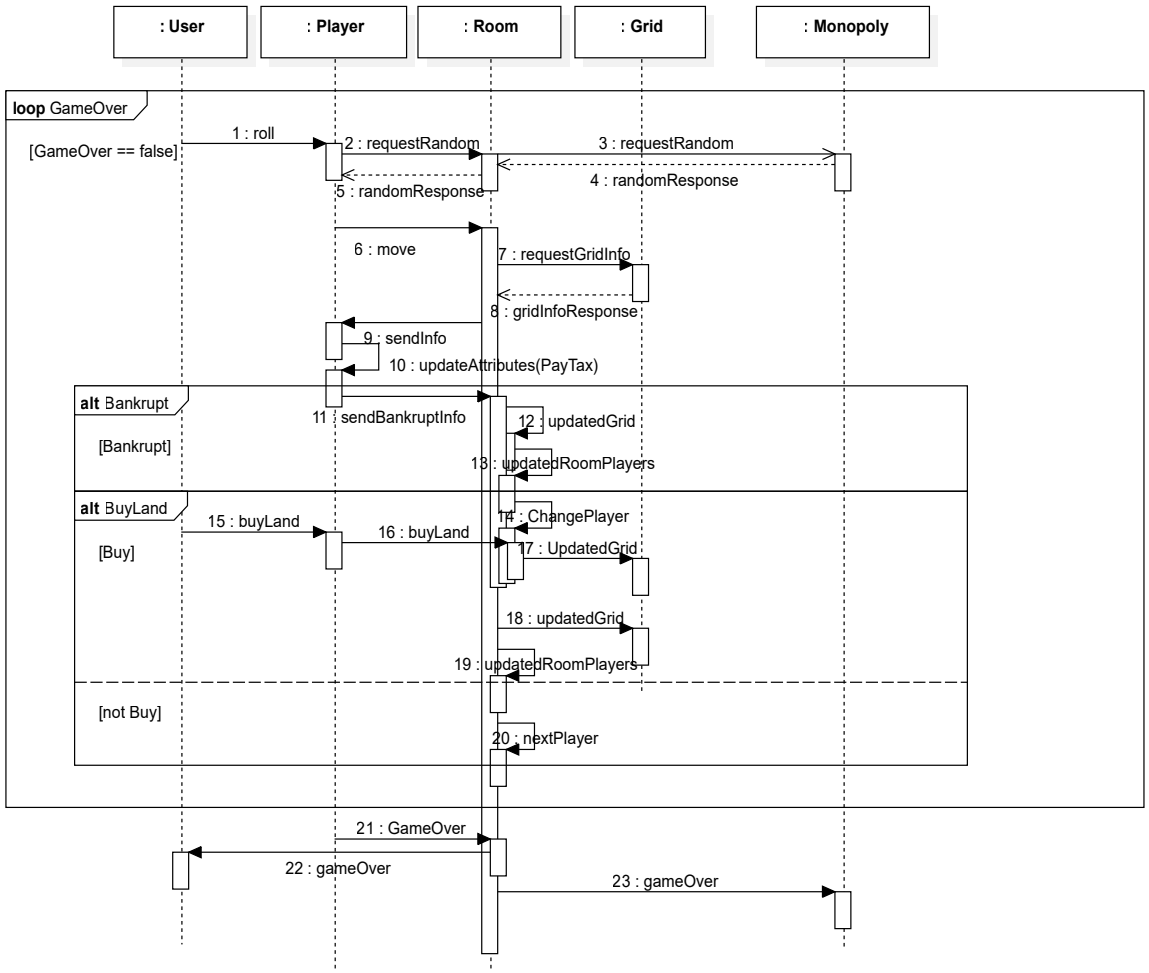

Figure 6: Sequence diagram of GameRunning and GameOver