# 游戏规则：

跟大家平常玩的大富翁差不多。

有100个城市，掷骰子（自动）产生步数，人物就可以向前走，走到有主人的城市需要交过路费。可以买下自己所在的城市（无论是否有主人，无需经过主人同意，因为主人会在城市价格上涨中受益）。如果有过路费，必须交过过路费才能继续往前走。走完一圈后会自动进入下一圈，如此循环不止。

用户可以针对自己的城市设置logo，img,url等，类似广告位，游戏结束后这些就永久是该用户的了。

没有监狱、机会那些。也没有银行家、破产抵押等规则，所有规则就是上面这些。

**一些设定**：（部分合约处理，部分前端处理）

**合约处理部分**：

初始城市价格1EOS，每位玩家购买后，价格要上升125%。其中110%归上一位玩家，15%进入奖池。

当最后一次购买城市后，如果3天之内没有任何人再次购买城市，则游戏结束，此时运营方开始分发奖池。

奖池分3部分，40%由所有城市的最终主人按当前价格比例分，35%归最后一位买城市的玩家，25%归游戏运营团队

玩家的所得（过路费，卖城市升值部分，奖池分红）记录在其游戏余额中，可以随时提现成EOS

所有逻辑均在合约中，完全去中心化。

**前端处理部分**：

我们有自己发行的代币ELL，游戏过程中玩家可以通过一些操作获得ELL代币，具体包括：

每次有EOS转账（买城市，交租金）时，每1EOS对应发放100ELL（发出5亿后减半），（减半这个操作在发币合约里处理，前端只需调用发币）

每次到8的尾数，发相应数字ELL，例如走到8，就发8ELL，走到18，就发18 ELL。

每走一圈，发放500 ELL

**示例**：

玩家A开始玩游戏，走到城市1，并买下，此时花费1EOS

玩家B此时玩游戏，走到城市1，需交0.1EOS的过路费（不交不能往前走），可以买下该城市，此时售价1.25EOS，其中1.1EOS归玩家A，0.15EOS进入奖池

# 合约介绍

合约测试版本部署在麒麟测试网，monopolygame账号。后续正式版本会部署在主网

包括如下Action：

* **ACTION move(name user,uint8\_t step)**  
  user:执行move的账号  
  step:走的步数  
  含义：user会向前走step步。目前只有100个城市，每次到了100就会再回到0号城市。（合约会自动处理，前端只需输入这两个参数即可。）

User必须是符合eos账户标准的字符串，12位，只包含小写a-z和数字1-5。  
step必须是>=2且<=12的数字，因为只有两个骰子。

* **ACTION reset()**含义：重置游戏数据。

该接口只能monopolygame账号调用，其他账号调用会返回assert错误。这是后台管理接口，会清空monopolygame内的数据库，包括奖池，所有城市状态。

* **ACTION testterminate()**

含义：结束游戏，计算奖池分配，且修改各个可得到奖池奖励的玩家的余额。  
该接口只能monopolygame账号调用，其他账号调用会返回assert错误，这是后台管理接口。

并不是调用该接口就会结束游戏，而是需要在满足游戏结束的条件下，调用该接口，来计算奖池分配。  
游戏结束条件是：当最后一位玩家买卖城市后，3天内都没有人买城市时，游戏自动结束（合约里控制，此时不能move了，买城市也会失败，但是交租金还是可以的）

* **ACTION setlogo(name user,uint8\_t city\_num,string label,string img,string url)**

User:调用者的EOS账号

City\_num：要设置的城市的号码

Label：要设置的label字符

Img: 要设置的logo的网络地址

url: 要设置的url

含义：该接口供玩家调用，设置自己的城市的logo,label,url，可以让玩家城市作为自己的广告位展示。所有数据会被写入EOS永久保存。其中label的长度限制为32字符，img和url限制64字符（byte）

* **ACTION transfer(const name& from, const name& to, const asset& quantity, const string& memo)**

这是个隐藏接口，用户直接调用会返回失败，实际上该接口是对eosio.token->transfer的监听，当用户调用eosio.token->transfer，且to==monopolygame，即向monopolygame转账EOS币的时候，该接口就会被触发。触发时通过memo来分辨是什么操作，执行对应的处理。

1. Memo=”pay\_rent”, 交租金，玩家付给monopolygame
2. Memo=”buy\_city”,买城市，玩家付给monopolygame
3. Memo=”reveal”,提现，monopolygame付给玩家

如上3个转账都需要由前端触发调用1和2在前端直接触发玩家本地调用就行，因为需要玩家的私钥对交易签名。3需要发一个消息到后端，触发后端执行，因为monopolygame的私钥需要保存在后端。请注意前后端都需要贵司开发完成。后端内容很少，后续会涉及到。

* 游戏数据库介绍

本游戏中有3个数据库：

1. citys, code=monopolygame, scope=monopolygame

数据结构：

city{

uint8\_t city\_num; 城市的号码，0-99

name owner; 当前主人，如果是monopolygame，说明没主人

asset price; 当前价格

asset rent; 当前租金

uint32\_t last\_modified; 上次买卖时间

string label; 用户设置的label

string img; 用户设置的图片

string url; 用户设置的url

}

该表格最多有100条记录，只有一个城市有买卖过后，才会出现在表格中，所以最初查询时其内容应该是空的，随着游戏进行会逐渐增多，最多增加到100.

城市并不记录城市名，只有个编号，城市名是前端写的。

每次有城市买卖时就会插入/修改该表的记录。调用setlogo也会修改表格记录。

1. status, code=monopolygame, scope=monopolygame

数据结构：

stats{

name account; 无意义，总是monopolygame

asset pool; 当前奖池

uint32\_t last\_modified; 最后一次买卖城市发生的时间。与citys表里的一致，只记录最后一次的。是从1970年到当前的**秒数**

}

该表格只是记录pool和last\_modified,其中last\_modified是供判断游戏是否结束用的。每当发生城市买卖时，该表格会变化。

1. account\_info，code=monopolygame，scope=玩家的EOS账号

数据结构：

account\_info{

name account; 玩家的账户名

uint8\_t pos; 当前所处位置

asset rent\_quantity; 当前欠租金数量

name rent\_owner; 欠谁的租金

asset to\_reveal; 可提现金额

};

该表格记录在每个玩家账户中，记录玩家自身的一些状态。当执行动作时会变化。如move,买卖城市、交租金、提现等。Testterminate也会改变其to\_reveal的值（如果用户可以分到奖池的话）

# 游戏执行调用过程

通过游戏中用户的各个状态来说明，可能更清晰。

* 游戏中，用户一开始是处于未开始状态。这可以通过查询account\_info表知道。  
  此时用户唯一可做的是move向前进。（调用合约的move）
* 处于某个城市时，可能的状态和操作：  
  有无租金？  
   有： 交租金  
   买城市

提现（如有余额）

无： 交租金

买城市

前进

提现（如有余额）

* 游戏结束时  
  如果有余额，可以提现

在游戏执行过程中还需要注意按照游戏介绍里的方法发放ELL代币。

# 各个操作的实现方法

* Move：·本地产生前进的步数，（可以直接用随机数发生器，需在2-12）.调用monopolygame->move，传入玩家账号，step即可
* 交租金：  
  调用eosio.token->transfer，即转账，  
  from: 玩家的eos账号  
  to: monopolygame  
  quantity: 该玩家的account\_info的rent\_quantity  
  memo: pay\_rent
* 买城市：  
  调用eosio.token->transfer，即转账，  
  from: 玩家的eos账号  
  to: monopolygame  
  quantity: 玩家account\_info里查到pos，即所在位置，再在citys表中查出该pos的城市信息，从中读取price。  
  memo: buy\_city
* 提现  
  调用eosio.token->transfer，即转账，  
  from: monopolygame  
  to: 玩家eos账号  
  quantity: 该玩家的account\_info的to\_reveal  
  memo: reveal  
  注意该操作发生在服务器后端，而不是前端。需要由前端把to,quantity,memo发送到后端，由后端调用。后端上保存monopolygame的私钥，对操作签名发送。
* 服务器后端需执行testterminate，该操作最好自动执行。可以在每次发生城市买卖时，设定一个3天的计时器，当有新的城市买卖发生时，就取消上一个计时器，直到计时器时间到（即达到游戏结束条件）。此时执行monopolygame->testterminate。这时候就会计算奖池，并将计算结果加到各个玩家的to\_reveal，玩家就可以提现了。
* 服务器后端执行reset。这个只有在发生意外时才可以执行，正常状态下不能执行。建议前后端都不要添加触发入口。如需执行，我们会自己处理。
* 如何查询数据：  
  直接调用EOS的rpc接口get\_table\_rows即可。Code和scope上面都有写。注意查询citys时需要填写limit=100，否则可能无法一次性拿到所有数据。
* 如何发放ELL  
  后续会给出更详细的发放方法。
* 当前monopolygame的合约部署在麒麟测试网上，可以直接进行测试。麒麟测试网rpc接口地址：[https://api-kylin.eosasia.one](https://api-kylin.eosasia.one/)  
  部署的账号：monopolygame

# 需要贵司做的

* UI设计
* 前端开发（界面和相应逻辑，如查询，发送action，发给后端的action）
* 后端处理（提现动作处理，testterminate调用，发放ELL）
* 钱包接入（一般来说是scatter，但是需要看要接入的钱包的要求）
* 集成测试
* 可能会同时开展多个游戏，如世界地图的，欧洲的，美国的，中国的，亚洲的，等，但各个地图的只是前端

# 参考资料

我自己开发了一个安卓的客户端，实现了所有功能，可以作为参考。  
<https://github.com/maodaishan/monotest>

# UI设计参考

我考虑可以有如下界面和功能，仅供参考

* 初始界面  
  提供玩法介绍，当前游戏状态（如奖池，是否结束等）  
  用户可以输入EOS账号名，并点击开始，此时查询账号数据，城市数据，stats表等，并进入平常状态
* 平常状态界面  
  当前所处位置信息展示，如城市号码，城市名，城市label,img,url  
  账号信息展示，如是否有租金，余额等  
  奖池、查看全部城市等系统信息展示  
  根据账号状态显示可以执行的操作：掷骰子，交租金，买城市，提现。
* 如果无租金，当点击掷骰子后，  
  显示骰子动画及结果，开车/飞机等动画，到达新城市，然后刷出“平常状态动画”
* 点击付租金、买城市时  
  出现当前动作的提示界面（需计算出要付给的EOS数量），玩家确认后，本地发起转账操作
* 点击提现  
  出现提现提示界面，用户确认后，向后台发出提现转账请求，后台接收到之后向rpc接口发出转账操作。结果刷新给用户。
* 点击奖池详情  
  出现当前奖池情况和奖池分配规则
* 点击查看所有城市信息

出现所有城市的列表及其详情

* 帮助  
  玩法规则和公司介绍等。