Splendor 游戏说明

For Girl Hackathon Season V

游戏规则

游戏规则详见桌游内附说明书，电子版地址：

http://www.gokids.com.tw/tsaiss/Asmodee/Rules/Splendor_0627Chinese.pdf

注意，相比原始游戏规则，我们为此次 Hackathon 做了以下几处修改：

1. 始终采用三人对局的设定，即，每种宝石初始数量为 **5 个**，每局对战使用贵族卡数量为 **4 张**
2. 如果玩家的某一步操作会使得自己手中的宝石和黄金总数>10，我们会认定这步操作**不合法**，玩家直接失去这步操作，而不会给玩家放回部分的选择
3. 玩家在可以想要获得贵族卡且满足条件时必须主动要求指定贵族卡，如果**不指定，当作玩家放弃贵族卡**，而不是原规则中的自动获得且无法拒绝
4. 每个玩家的保留卡都是**所有玩家可见**的，而不是原规则中描述的“从牌堆里拿的保留卡不需要让其他玩家看见”
5. 玩家可以购买手中**任意一张保留卡**，不需要是前一回合保留的（此处中文说明书表述有问题）

程序和评测

最终评测时，每组选手上交自己的程序代码，由裁判组编译和运行（裁判组机器为 Linux 系统）。

程序运行

简而言之，选手程序需要做的是：对于输入的当前局面进行处理，计算得出玩家当前要做的操作，并输出。即，运行一次程序只有**一次输入**和**一次输出**，只决策当前的**一个操作**。所有的输入、输出都是**标准输入输出(stdin 和 stdout)**。输入输出格式详见下一章。

评测机运行过程

评测机是由裁判组提供的，可以模拟游戏运行，得到游戏结果的程序，选手可以利用评测机验证自己的程序是否可以正确运行、及测试程序的策略和性能。请阅读评测机文件夹中的 README

文件了解如何启动评测机。评测机运行一次是一局游戏，过程大致如下：

- 开始游戏，生成初始局面
- 每一轮，按照给定的三个玩家的顺序，依次运行如下部分（假设当前玩家为 A）：
    - 将当前局面整合成指定格式的输入数据
    - 调用 A 的可执行文件，把输入数据传进去
    - 等待 A 运行完毕，读出它的输出数据（如果运行超时，当作 A 放弃操作）
    - 验证 A 的输出数据合法性，及它操作的合法性，如果有不合法，则当作 A 放弃操作
    - 如果 A 操作合法，则根据 A 的操作更新局面状态
    - 对下一个玩家重复该过程
- 当有一个玩家达到 **15 分**后，这轮三个玩家操作结束后结束游戏，记录最后的排名到 log
- 注意，如果轮数超过 **150 轮**，还没有达到结束条件，评测机强制结束对局，以结束时的排名作为最终排名

评测机整个运行过程中都有 log 记录，选手可以参考 log 看自己程序的问题或者调整策略。

battle 流程和分数

任意三个队伍都会进行一场比赛，每场比赛进行**三局**（例如 ABC 三支队伍，分别以 A-B-C, B-C-A, C-A-B 顺序对局），保证公平。每局结束后按照排名，**第一名积 4 分，第二名积 2 分，第三名积 1 分。**

最后根据每个队伍的总积分，从高到低排名，换算成 battle 部分的分数：前七名分别为 10、9、...4 分，其余队伍 3 分。

输入/输出数据格式

所有的输入、输出数据都采用 JSON 格式，各语言都有处理 JSON 的库和工具（tips：使用 C++ 的选手可以使用 JSONcpp 库）。我们定义了一套完整的表示游戏局面、操作的规则。

基本结构

首先，所有的颜色都用 string 表示，共六种：green, white, blue, black, red, gold（**都必须是小写**）；其中 gold 是黄金，其余所有颜色都可以表示宝石或红利。

发展卡

一张发展卡的 JSON 表示里包括以下字段：level（等级，int），score（分数，int，如果该字段不存在则为 0），color（红利颜色，string），costs（花费，是一个 list）；

其中，costs 中每个元素都是有两个字段：color（需要的宝石颜色，string），count（需要该颜色宝石的个数，int）；

例如，下图这张发展卡（假如它是等级二的）：

它的 JSON 表示就是：

```
{
 "level": 2,
 "score": 2,
 "color": "red",
 "costs": [{
   "color": "white",
   "count": 1
 }, {
   "color": "blue",
   "count": 4
 }, {
   "color": "green",
   "count": 2
 }]
}
```

贵族卡

一张贵族卡的 JSON 表示中包含以下几个字段：score（分数，int），requirements（红利需求，是一个 list）；

其中，requirements 中每个元素都有两个字段：color（需要的红利颜色，string），count（需要该颜色红利的点数，int）；

例如，下图这张贵族卡：

它的 JSON 表示就是：

```
{
 "score": 3,
 "requirements": [{
   "color": "green",
   "count": 3
 }, {
   "color": "blue",
   "count": 3
 }, {
   "color": "white",
   "count": 3
 }]
}
```

桌面

即游戏局面中桌面上的内容，包含翻出的发展卡、贵族卡、和宝石。桌面的 JSON 表示中包含以下几个字段：gems（宝石，是一个 list），cards（翻出的发展卡，是一个 list），nobles（贵族卡，是一个 list）；

其中，gems 中每个元素包含两个字段：color（某种颜色，string，如果是 gold 表示黄金），count（这种颜色的宝石数，int）；cards 中每个元素就是一张发展卡的表示；nobles 中每个元素就是一张贵族卡的表示。

例如，以下 JSON 可以表示一个初始情况的桌面：

```
{
   "gems": [{
     "color": "red",
     "count": 5
   }, {
     "color": "gold",
     "count": 5
   }, {
     "color": "green",
     "count": 5
   }, {
     "color": "blue",
     "count": 5
   }, {
     "color": "white",
     "count": 5
   }, {
     "color": "black",
     "count": 5
   }],
   "cards": [{
     "level": 3,
     "score": 3,
     "color": "green",
     "costs": [{
       "color": "white",
       "count": 5
     }, {
       "color": "blue",
       "count": 3
     }, {
       "color": "red",
       "count": 3
     }, {
       "color": "black",
```

```
      "count": 3
    }]
  }, (省略一些发展卡表示) ],
  "nobles": [{
    "score": 3,
    "requirements": [{
      "color": "red",
      "count": 4
    }, {
      "color": "green",
      "count": 4
    }]
  }, (省略一些贵族卡表示) ]
}
```

玩家

一个玩家的 JSON 表示包含以下几个字段：name（玩家名字，string），score（当前分数，int，如果没有这个字段则为 0），gems（拥有的宝石和黄金，是一个 list，如果没有这个字段则玩家没有宝石），purchased_cards（已经购买的发展卡，是一个 list，如果没有这个字段则玩家还没有购买卡），reserved_cards（玩家保留的卡，是一个 list，如果没有这个字段则玩家没有保留卡），nobles（玩家获得的贵族卡，是一个 list，如果没有这个字段则玩家没有获得过贵族卡）；

其中，gems 中每个元素包含两个字段：color（某种颜色，string，如果是 gold 表示黄金），count（玩家拥有的这种颜色的宝石数，int）；purchased_cards 和 reserved_cards 中每个元素都是一个发展卡的表示；nobles 中每个元素是一个贵族卡的表示；

例如，以下 JSON 可以表示一个玩家的当前状态：

```
{
  "name": "player1",
  "score": 3
```

```
"gems": [{
  "color": "red",
  "count": 1
}, {
  "color": "blue",
  "count": 1
}, {
  "color": "white",
  "count": 1
}],
"purchased_cards": [{
  "level": 1,
  "color": "white",
  "costs": [{
    "color": "blue",
    "count": 2
  }, {
    "color": "green",
    "count": 2
  }, {
    "color": "black",
    "count": 1
  }]
}, {
  "level": 2,
  "score": 3,
  "color": "white",
  "costs": [{
    "color": "white",
    "count": 6
  }]
}],
"reserved_cards": [{
  "level": 2,
  "score": 2,
  "color": "black",
  "costs": [{
    "color": "black",
```

```
        "count": 5
      }]
    }]
}
```

玩家名字为 player1；当前有红色、蓝色、白色宝石各 1 个；买过两张发展卡，分别是 level1 和 level2 的；手里保留了一张 level2 的发展卡；还没有获得过贵族卡；由于已购买的 level2 的发展卡具有 3 分，level1 的发展卡没有分，所以他的总分数是 3 分。

游戏局面（输入）

选手提交的程序的输入是当前的游戏局面，包含了以下几个字段：round（当前的轮数，int），player_name（当前轮到的玩家的名字，string），table（桌面，即一个桌面的表示），players（玩家列表，一个 list，其中每个元素都是一个玩家的表示）；

这里有一个完整的 JSON 格式的输入样例。

玩家操作（输出）

选手程序的输出是当前局面下的操作，可以包含以下五种操作之一。注意，如果输出的 JSON 中包含了多个操作，评测机只会从中**挑选一个执行**（玩家无法预测挑选的是哪个）。

取不同颜色的宝石

用 get_different_color_gems 字段表示，该字段是一个 list，每个元素是一个 string，表示要取的宝石的颜色。例如，以下的 JSON 输出：

```
{
  "get_different_color_gems" : [ "red", "green", "blue" ]
}
```

表示玩家取走红、绿、蓝三种颜色宝石各一个。

取同色的宝石

用 get_two_same_color_gems 字段表示，该字段是一个 string，表示要取的两个宝石的颜色。例如，以下的 JSON 输出：

```
{
  "get_two_same_color_gems" : "red"
}
```

表示玩家取走红色宝石两个。

保留发展卡

用 reserve_card 字段表示。有两种情况，首先是保留场上的一张卡，reserve_card 里包含一个 card 字段，是一个发展卡的表示，这张卡必须是桌上已有的。例如，以下的 JSON 输出：

```
{
  "reserve_card" : {
     "card" : {
        "color" : "blue",
        "costs" : [
           {
              "color" : "blue",
              "count" : 5
           }
        ],
        "level" : 2,
        "score" : 2
     }
  }
}
```

表示玩家保留了一张 level2 的 2 分卡，它的花费是 5 个蓝色宝石，红利颜色是蓝色。

另一种方式是从牌库顶端拿一张卡，保留它，此时需要指定从哪个等级拿，reserve_card 里包含 level 字段（int）表示等级。例如，以下的 JSON 输出：

```
{
  "reserve_card" : {
     "level" : 1
  }
}
```

表示玩家从等级一牌库拿一张保留卡。注意，评测机在收到该操作后，不会立即告诉玩家程序保留卡的具体信息，玩家程序在下一轮重新被调用的时候会知道保留卡的具体信息。

如果 reserve_card 中同时包含了 card 和 level 两个字段，则评测机认为玩家这步操作**不合法**。

购买桌上的发展卡

用 purchase_card 字段表示，里面是一个发展卡的表示，这张卡必须是桌上已有的。例如，以下的 JSON 输出：

```
{
  "purchase_card" : {
     "color" : "black",
     "costs" : [
        {
           "color" : "green",
           "count" : 5
        },
        {
           "color" : "red",
           "count" : 3
        }
     ],
     "level" : 2,
     "score" : 2
  }
```

```
}
```

表示玩家购买了一张 level2 的 2 分卡，它的花费是 5 绿 3 红，红利颜色是黑色。注意，在购买卡花费宝石的时候，评测机会先判断玩家的宝石+红利数目是否足够购买这张卡，如果不够，再使用黄金。

购买保留的发展卡

用 purchase_reserved_card 字段表示，其中是一个发展卡的表示，这张卡必须是玩家自己手上的保留卡。例如，以下 JSON 输出：

```
{
  "purchase_reserved_card" : {
     "color" : "black",
     "costs" : [
        {
           "color" : "green",
           "count" : 5
        },
        {
           "color" : "red",
           "count" : 3
        }
     ],
     "level" : 2,
     "score" : 2
  }
}
```

表示玩家买了这张自己保留的 level2 的 2 分卡。

获得贵族卡

除了以上这五种操作之外，在玩家程序输出中还可能出现贵族卡。根据规则，在玩家操作结束后

如果满足贵族卡的要求，玩家可以获得贵族卡。玩家程序自己可以判断出执行自己的操作后是否满足贵族卡的要求，如果满足，就可以在输出中加入一个可以获得的贵族卡，用 noble 字段表示；如果评测机判定玩家可以获得贵族卡，但玩家的输出中没有指定，则认为玩家**主动放弃**贵族卡。

如下的 JSON 输出：

```
{
  "noble" : {
     "requirements" : [
        {
           "color" : "blue",
           "count" : 4
        },
        {
           "color" : "white",
           "count" : 4
        }
     ],
     "score" : 3
  },
  "purchase_reserved_card" : {
     "color" : "blue",
     "costs" : [
        {
           "color" : "white",
           "count" : 1
        },
        {
           "color" : "black",
           "count" : 2
        }
     ],
     "level" : 1,
     "score" : 0
  }
}
```

表示玩家先购买一张自己的保留卡，买完以后将会满足一个贵族卡的获得条件（拥有 4 个蓝色 4

个白色红利），玩家需要在输出中指定这张贵族卡。

评测机 Log

评测机运行过程中输出的 Log 有助于选手发现程序运行的错误，及调整程序策略。评测机会在游戏初始时 log 初始局面；每个玩家操作时 log 具体操作和操作结果，如果操作失败，会写明原因；在一局结束后会 log 本局的最终结果，每个玩家的得分和购买的发展卡数。这里有一个样例 Log 文件，可供参考。选手也可以通过让评测机运行自己的玩家程序来获得更多 Log。