Poker Game Specifications

Program structure

```
├──── README.md
├──── card_types.json
├──── cards.json
├──── gameEngine.m
├──── main.m
├──── player.m
├──── pokerGameUI.mlapp
├──── pokerRule.m
├──── resource
|     ├──── audio
|     |     ├──── bg_game.ogg
|     |     ├──── bg_room.mp3
|     |     ├──── deal.mp3
|     |     ├──── end_lose.mp3
|     |     └──── end_win.mp3
|     ├──── audio_mat
|     |     ├──── bg_game.mat
|     |     ├──── bg_room.mat
|     |     ├──── deal.mat
|     |     └──── end_win.mat
|     ├──── cards
|     |     ├──── clubs10.png
|     |     ├──── spadesQ.png
|     |     └──── wbJoker.png
|     ├──── doudizhu.png
|     ├──── icon_default.png
|     ├──── icon_farmer.png
|     ├──── icon_landlord.png
|     ├──── pass.png
|     ├──── poker_back.jpg
|     ├──── ready.png
|     └──── shot.png
└──── rule.json
```

Class: pokerRule

```
classdef pokerRule < handle
    % properties of poker game rule
    properties
        % 'rocket', 'bomb',
        cardType = ["single", "pair", "trio", "trio_pair", "trio_single",
        cardRule = jsondecode(fileread('rule.json'));
        compare_result % -3 -> found; -2 -> not found; -1 -> unkown type;
        gameEngine
    end

    % methods of poker game rule
    methods
        % find the indx, which represents magnitude, and cards type
        function index_of(pkRule, typeName, ele)...
        % determine the cards type
        function card_type(pkRule, cards)...
        % determine cards value: return cards type and value
        function cards_value(pkRule, cards)...
        % compare the poker hands and set compare value
        function compare_poker(pkRule, preCards, selectedCards)...
    end
end
```

Properties

1. **cardType**: 储存所有的斗地主牌型的数组，内容为下：

   "single", "pair", "trio", "trio_pair", "trio_single",
   "seq_single5", "seq_single6", "seq_single7", "seq_single8", "seq_single9", "seq_single10", "seq_single11","seq_single12",
   "seq_pair3", "seq_pair4", "seq_pair5", "seq_pair6", "seq_pair7", "seq_pair8", "seq_pair9", "seq_pair10",
   "seq_trio2", "seq_trio3", "seq_trio4", "seq_trio5", "seq_trio6",
   "seq_trio_pair2", "seq_trio_pair3", "seq_trio_pair4", "seq_trio_pair5",
   "seq_trio_single2", "seq_trio_single3", "seq_trio_single4", "seq_trio_single5",
   “bomb_pair”,”bomb_single"

2. **cardRule:** 由 **rule.json** 导入。

   遍历所有可能斗地主打出手牌的可能性，以牌型归类。同一类别内，手牌按规则从小到大排列，即某一手牌在某一手牌类型中的Index代表了该手牌的大小，Index 越大，则手牌越大。

3. **Compare_result:** 储存当前两组牌型对比结果

   -3：found，已在**cardRule**找到;
   -2： not found，未在**cardRule**找到;
   -1：unkown type，非法牌型;

0： not bigger，当前牌组小于上一轮牌组;
>0： bigger，当前牌组大于上一轮牌组;

4. **gameEngine:** 用于接收当前牌组及信息同步。

Methods

---

function index_of(pkRule, typeName, ele)

输入对应牌型名称，待查找 牌组字符串。若查找到，将当前牌组代表的加权值及牌型赋值给 *gameEngine.cards_value_selected, gameEngine.cards_type_selected,* 设置 *Compare_result*；若未查找到，*Compare_result* 赋对应的值，返回函数。

---

function card_type(pkRule, cards)

调用 **function index_of(pkRule, typeName, ele)** ，判断 Compare_result 的值，若找到，函数结束；若未找到，在相应玩家的 UI 的 UnknownTypeLabel 显示相应的系统信息，并且报错（该报错为主动吊起，非程序崩溃），在命令框显示当前牌型。

---

function cards_value(pkRule, cards)

为当前赋加权值（gameEngine.cards_value_selected），用于与上一句牌组比较。

A. 王炸，单独提出，赋值： 2000；
B. 炸弹，单独提出，赋值：1000 + index（即该组排在炸弹数组中的位置）
C. 其余牌型通过调用 **function card_type(pkRule, cards)** 设置加权值

---

function compare_poker(pkRule, preCards, selectedCards)

比较主函数，比较输入牌组与上一局牌组的大小。

i) 调用 ***function cards_value(pkRule, cards)*** 确定牌型及牌组加权值
ii) 若当前牌组和上一局牌组任一为空，即可判断孰大孰小
iii) 若为相同牌型，则比较结果为输入牌型加权值与一局牌组加权值之差
iv) 其余情况皆为输入牌型加权值小于一局牌组加权值之差，在相应玩家的 UI 的 UnknownTypeLabel 显示相应的系统信息。

Class: gameEngine

Properties

```
    properties
        % All players and thier apps
        player_0
        player_0_UI
        player_1
        player_1_UI
        player_2
        player_2_UI

        rule            % the instance of poker game rule
        landlord = -1   % -1 -> defalut
        whoseTurn = -1; % -1 -> defalut
        passNum = 0;    % passNum should not more than 2
        cardNum = 54;   % Total card Num = 54
        winner = -1;    % winner=-1 -> have not determined winner
        isEnd = false;  % determine whether to end the game
        isStart = false; % determine whether to start the game
        isBGM = true;   % determine if play BGM

        % In order to sychronize, Distribute cards as soon as one player
        % push 'Ready'. Then create cards compoents and invisible them
        % until game started.
        isDistribute = false;

        % import cards data
        cardsData = transpose(struct2cell(jsondecode(fileread('cards.json'))));

        % store the cards that has shotted: 0 -> last turn
        % row 1: str num % row 2: lable
        % row 3: img source % row 4 num
        cards_shotted_0 = {};
        cards_selected = {};
        % used to compare selected cards with last turn's
        cards_type_0;
        cards_value_0;
        cards_type_selected;
        cards_value_selected;

        % for bgm
        bg_room = load('./resourse/audio_mat/bg_room.mat');
        bg_game = load('./resourse/audio_mat/bg_game.mat');
        deal = load('./resourse/audio_mat/deal.mat');
        end_win = load('./resourse/audio_mat/end_win.mat');
        player;         % used to play bgm
    end
```

1. **player_#, player_#_UI**

   三位游戏玩家及其游戏界面的实例的handle。

2. **rule**

   pokerRule 的实例的handle，用于实现斗地主规则判断。

3. **landlord { -1, 0, 1, 2 }**

   储存地主的玩家编号， **-1**为缺省值，**{0, 1, 2}** 代表对应的玩家编号。

4. **whoseTurn { -1, 0, 1, 2 }**

   储存当前出牌玩家编号， **-1**为缺省值，**{0, 1, 2}** 代表对应的玩家编号。

5. **winner { -1, 0, 1, 2 }**

   储存获胜玩家编号， **-1**为缺省值，**{0, 1, 2}** 代表对应的玩家编号。

6. **isEnd { true, false }**

   表示游戏是否结束，false 为缺省值（游戏为开始），true 代表游戏结束。

7. **isStart { true, false }**

   表示游戏是否开始，false 为缺省值（游戏未开始），true 代表游戏开始。

8. **isBGM { true, false }**

   表示是否播放背景音乐（音效），true 为缺省值（播放音效），false 代表暂停音效。

9. **isDistribute { true, false }**

   表示是否已经发牌，false 为缺省值（未发牌），true 代表已经发牌。为实现三个玩家洁面信息同步，当任一玩家点击 **“准备”** 按钮，即发牌，但所有的牌为不可见，知道三个玩家都已经切换至 **“准备”** 状态。

10. **cardsData**

    一副扑克牌的所有信息，从 **cards.json** 文件导入。**cards.json** 文件中储存一副扑克牌中所有纸牌信息，包括其代表的数字（字符），大小（int），图片源文件地址，花色。

11. cards_shotted_0

储存上一局打出牌组的所有信息。
1) 第一列：字符
2) 第二列：花色
3) 第三列：图片源文件地址
4) 第四列：大小（int）。

12. **cards_selected**

储存当前玩家尝试打出的牌组的所有信息。
1) 第一列：字符
2) 第二列：花色
3) 第三列：图片源文件地址
4) 第四列：大小（int）。

13. cards_type_0

储存上一局打出牌组的牌组类型。

14. cards_type_selected

储存当前玩家尝试打出的牌组类型。

15. cards_value_0

储存上一局打出牌组的牌组**加权值。**

16. cards_value_selected

储存当前玩家尝试打出的牌组**加权值。**

17. bg_room, bg_game, deal, end_win

保存音效的工作区。分别为进入房间的背景音乐，开始游戏的背景音乐，发牌的背景音乐，游戏结束的背景音乐。

18. Player

背景音乐播放器实例的handle，通过设置采样幅度值、采样频率来控制播放的背景音效。

Methods

```
% methods that game engine has:
methods
    % In 0 stage, random assign Role; -1-defult, 0-landlord, 1-peasant;
    function assignRole(eg)...
    % determine which player is winner
    function determineWinner(eg)...
    % Sort hand cards of players
    function sortCard(eg, player)...
    % Shuffle and distribute cards
    function distributeCards(eg)...
    % update related variables in three players and their apps: lable,
    % display cards
    function update(eg)...
    % display handcards in three UIs
    function displayCard(eg)...
    % Display the shotted cards in other players' UIs
    function dispShotCards(eg)...
    % decide whose turn to shot cards
    function nextTurn(eg)...
    % start game with following process
    function startGame(eg)...
    % End game with following process
    function endGame(eg)...
    % control the BGM
    function bgm(eg)...
end
```

---

function assignRole(eg)

为三个玩家分配游戏角色。随机抽取 [0, 2] 的一个整数，序号为该整数的玩家为“地主”，其余玩家为“平民”角色。

---

function determineWinner(eg)

判定赢家。任一玩家手牌首次达 0 张，则该玩家为赢家，该玩家所代表的一方获胜。为 **eg.winner, eg.isEnd** 设置对应参数，调用 ***function endGame(eg)*** 以结束游戏。

---

function sortCard(eg, player)

整理手牌，输入为 **Player** 实例的handle。通过实现冒泡排序算法，为玩家手牌从小到大排序。

function distributeCards(eg)

为三个玩家分配手牌，一副牌共54张，“地主”手牌为20张，“平民”手牌为17张。
运用 Matlab 随机置换函数 randperm( )，将 1 到 54 的整数随机打乱，输出为数组 order。
将**order[1: 17]** 内容分配给玩家0，将 **order[18: 34]** 内容分配给玩家1，将 **order[35: 51]** 内容分配给玩家2，剩余三张手牌 。每一个 **order** 内容所代表数字为该玩家手牌在 **eg.cardsData** 的索引，据此为所有玩家分配好纸牌。
分配结束后调用 ***function sortCard(eg, player)*** 为每位玩家的手牌排序。

---

function update(eg)

更新玩家头像下方 **Label** 显示的内容。三个玩家界面通过 **gameEngine** 实现状态同步。

1. 若游戏未开始，**Label** 显示为该玩家的“准备”状态
2. 若游戏开始，**Label** 显示为该玩家的手牌数目

主要功能结束后，调用 ***function displayCard(eg)*** 在玩家界面显示手牌， ***function bgm(eg)*** 播放背景音效。

---

function displayCard(eg)

在每个玩家 UI 里显示整理后的手牌，调整手牌位置使手牌位置居中。

1. 计算当前玩家手牌数目的中位数： mid = (card number + 1) / 2
2. 每一张手牌的 y 坐标位置相同，计算 x 坐标。
3. 计算方法：
    1) 565 为界面中点 x 坐标值
    2) 565 +（手牌 x 坐标值为该手牌到中点的矢量距离）* （牌与牌之间的间隔）
    3) x = 565 + dist * 32

---

function dispShotCards(eg)

在每个玩家 UI 里显示打出的手牌，三个玩家界面通过 **gameEngine** 实现状态同步。

1. 每个玩家UI打牌区域已经创建好 20 个 **Image Component**，其图片源缺省值为空
2. 将玩家UI打牌区域所有 **Image Component** 设置为不可见
3. 打出手牌的每一张位置的算法与 ***function displayCard(eg)*** 类似。
    1) 计算当打出手牌数目的中位数： mid = (card number + 1) / 2
    2) 则每一张手牌在打牌区的 Index = fix(10.5 - (距离中点的矢量距离))
    3) 设置对应的 **Image Component** 的图片源
    4) 设置对应的 **Image Component** 为可见

function nextTurn(eg)

决定下一轮出牌的玩家。

1. 若当前玩家出牌，则玩家界面的“出牌”、“不出”按钮可见
2. 若非当前玩家出牌，则玩家界面的“出牌”、“不出” 按钮不可见
3. 初始化 **cards_selected = {}，eg.cards_type_selected = ''，eg.cards_value_selected = -2;**

function startGame(eg)

开始游戏必要的所有设置。

1. 将玩家界面的“出牌”、“不出” 设置为可用（enable
2. 若为“地主”玩家出牌，则玩家界面的“出牌”、“不出”按钮可见
3. 若“平民”玩家出牌，则玩家界面的“出牌”、“不出” 按钮不可见
4. 播放“发牌”背景音效
5. 调用 ***function displayCard(eg)*** 显示玩家界面的所有手牌
6. 调用 ***function update(eg)*** 更新玩家状态
7. 调用 ***function bgm(eg)*** 播放背景音乐

function endGame(eg)

结束游戏必要的所有设置。

1. 打牌区域的所有 **Image Component** 设置为不可见
2. 玩家界面中央 **“斗地主” Image Component** 设置为不可见
3. 所有玩家界面的“出牌”、“不出” 按钮不可见
4. 所有玩家界面显示获胜方信息
    1) 通过 **game engine** 获取获胜方信息
    2) 玩家界面 **winLabel.Text** 设置为相应信息
    3) 玩家界面 **winLabel** 设置为可见

function bgm(eg)

播放“等待游戏（在房间）”、“游戏中”背景音乐（效）。

1. 若游戏未开始，且背景音乐开关键为打开状态，播放“等待游戏（在房间）”背景音乐
2. 若游戏开始，且背景音乐开关键为打开状态，播放“游戏中”背景音乐
3. 播放背景音乐的方法为，将相应的 采样率和信号（提前导入到 workspace）传递给 **Player，**并开始播放

Class: player

```
classdef player < handle

    % properties of poker game engine
    properties (Access = public)
        avatar = 'icon_default.png';
        role = -1;      % -1-defult, 0-landlord, 1-peasant;
        cards = {};     % store the distributed info from *.json
        cardNum = 0;
        selectNum = 0;  % remember to reset
        isActive = false;
        myTurn = false;
        currUI;         % its own UI

        % row 1 stores the components of cards;
        % row 2 stores whether the card has selected;
        % row 3 stores whether the cards has shotted;
        % row 4 stores the str# of cards;
        % row 5 stores the lable
        % row 6 stores the # of cards
        cards_img = {};
    end

    % methods that game engine has:
    methods

    end
end
```

Properties

1. avatar

储存玩家对应的游戏角色头像的图片源。

2. role

储存当前玩家的角色。 -1为缺省值，0 代表地主，1 代表平民。

3. Cards

储存当前玩家从 **cards.json** 分配所得所有手牌的所有信息

4. cardNum

储存当前玩家手牌的数目，缺省值为 **0**。

5. selectNum

储存当前玩家选择的手牌的数目，缺省值为 **0**。

6. myTurn { true, false }

表示是否为当前玩家出牌，**true** 代表是，**false** 代表不是。

7. currUI

储存当前玩家界面实例的handle。

8. cards_img

储存当前玩家所有手牌的所有信息。

1) 第一列：所有手牌 **Image Component** 的handle
2) 第二列：该牌是否被选中
3) 第三列：该牌是否已被打出
4) 第四列：字符
5) 第五列：花色
6) 第六列：大小（int）。