LO02 Rapports de Projet

——Machi-Koro en C++

骰子街游戏

LU CHENJIE

SHAO CHENYU

ZHANG YIFAN

YANG YILIN

SHI XINYI

Groupe 10

目 录

一、 项目概况

我们通过代码实现了 MachiKoro 的基础版本，游戏流程是每位玩家通过掷骰子的方式获得卡牌相对应的金币，并购买新的卡牌，每张卡牌都有自己独特的属性，它们都有可能会对整局游戏的进程产生关键性的影响。为此我们引入了牌库，用来存放每张卡牌的点数、奖励以及剩余数量，以保证游戏的正常进行，此外我们也加入了 AI 玩家，当人数不足预定的 4 人会自动补齐。

稍有不足的是我们未能实现游戏的可视化图形用户界面，整局游戏都会在软件的控制台中实现，因此总体的设计目前还较为粗糙，可能存在新玩家上手难度较高的问题。

二、 项目流程

1. 小组成员搜集卡牌信息，线上组织进行游戏体验

2. 拟定项目整体框架，总结游戏需要实现的功能

3. 画出 UML 图，并根据 UML 图将不同的任务分配给小组成员

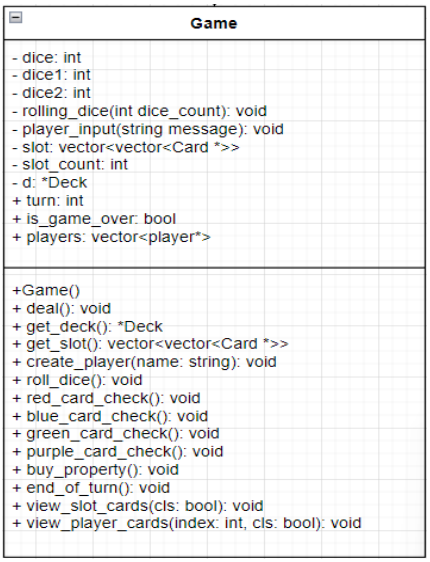
4. 收集每位组员负责的代码并进行整合，完成控制台的界面设计

5. 完成代码的整体调试，修复游戏内出现的漏洞

6. 实现 MachiKoro 基础版的运行

三、 代码实现

1. Game



命名为 Game 的文件负责管理整局游戏的进程，它负责了游戏中：创建玩家、掷骰子、金币交易、检查场上效果牌触发状态、判断玩家的回合是否结束、判断游戏是否结束的功能。

- 首先在游戏的开头，*Game ::create_player()* 函数会首先判断当前玩家的数量 如果人数不够则自动使用简易的 AI 补齐，并且在创建玩家的同时，会分别给予一张麦田、面包房，作为每位玩家的初始手牌。此外每位玩家也会同时得到 4 张黄色的地标牌，它们的状态默认都是未被激活（四张牌同时都被激活则宣布改名玩家取得游戏胜利）

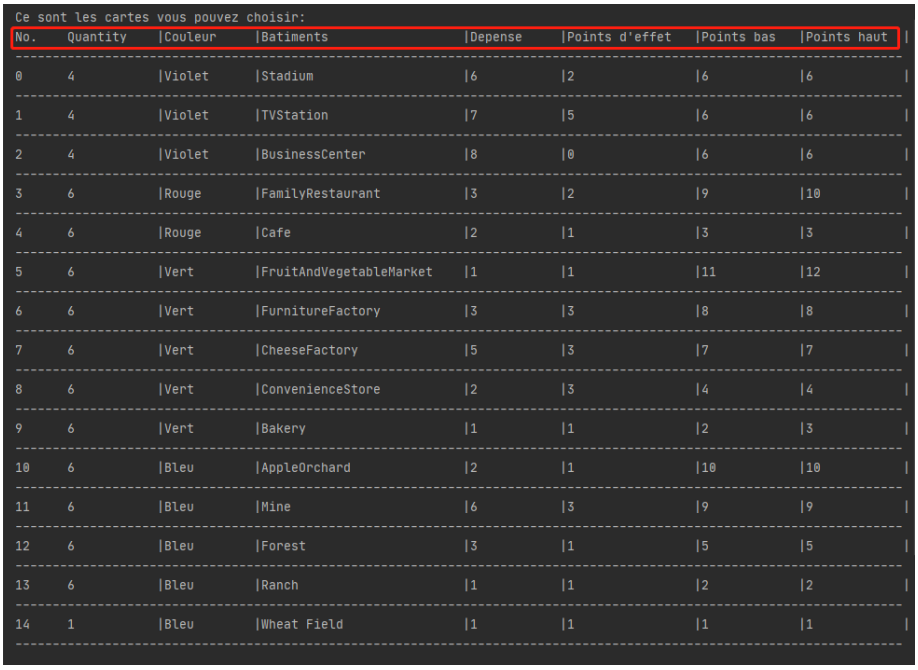
- *Game ::deal()*函数用于负责更新牌库中剩余的卡牌：如果牌库中已经有全部买完的牌，那么在打印牌库信息时，就会将已经买完的卡牌自动移除。

- *Game ::rolling_dice()*函数负责给每位玩家掷骰子：dice1、dice2 分别代表两颗骰子，（默认只使用 dice1）只有当玩家激活了 TrainStation 黄牌，并且确认要掷两个骰子时，才会使用 dice2 存放第二个骰子的数值，最后 dice 会将两个骰子的点数相加，并返回给 roll_dice()

- *Game ::roll_dice()*负责判断当前玩家是否拥有 TrainStation 黄牌，并且是否选择要同时掷两颗骰子，同时由于 RadioTower 黄牌提供给了玩家重新投掷骰子的机会，因此此函数也会负责判断当前玩家是否想要重新投掷骰子 (y/n)：如果选择为 “y” 则重新投掷点数。

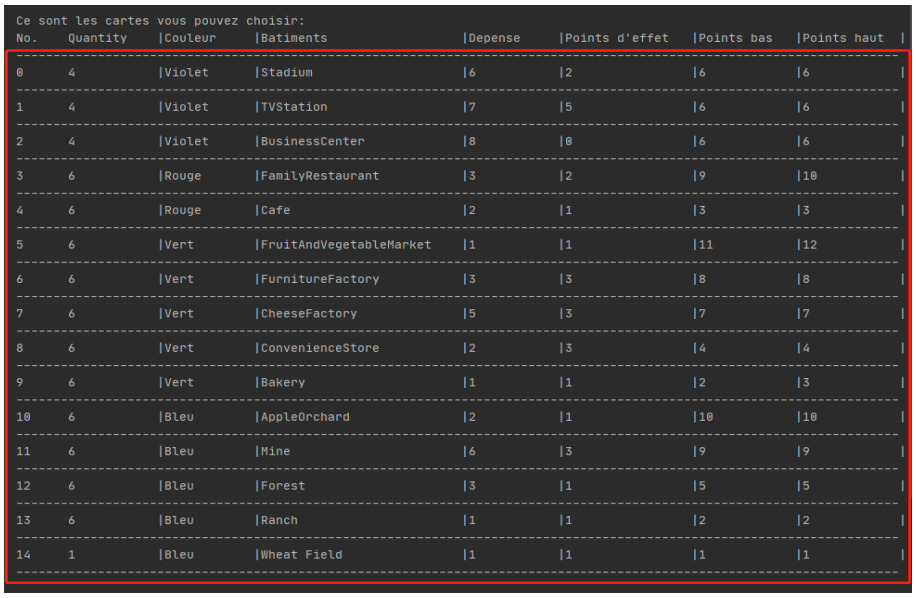
- 在 Game 中，我们也实现了对于当前回合信息状态的打印 此功能由：*print_card()* ， *print_card_heading()* ， *view_slot_cards()* ， 以及 *view_player_cards()* 共同组成

->*print_card_heading()*：用来打印表格的第一行，用来展示当前列所展示的信息



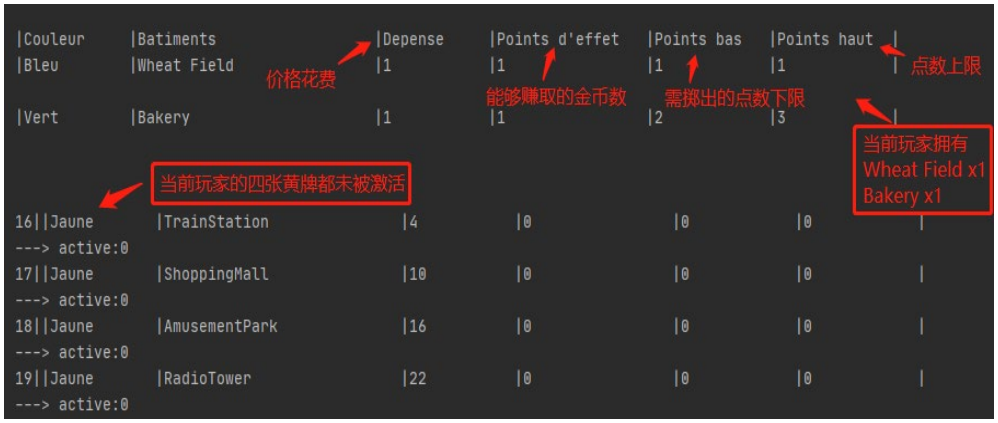
```
Ce sont les cartes vous pouvez choisir:
No.   Quantity   |Couleur    |Batiments                |Depense    |Points d'effet   |Points bas   |Points haut  |
-------------------------------------------------------------------------------------------------------------------
0     4          |Violet     |Stadium                  |6          |2                |6            |6            |
-------------------------------------------------------------------------------------------------------------------
1     4          |Violet     |TVStation                |7          |5                |6            |6            |
-------------------------------------------------------------------------------------------------------------------
2     4          |Violet     |BusinessCenter           |8          |0                |6            |6            |
-------------------------------------------------------------------------------------------------------------------
3     6          |Rouge      |FamilyRestaurant         |3          |2                |9            |10           |
-------------------------------------------------------------------------------------------------------------------
4     6          |Rouge      |Cafe                     |2          |1                |3            |3            |
-------------------------------------------------------------------------------------------------------------------
5     6          |Vert       |FruitAndVegetableMarket  |1          |1                |11           |12           |
-------------------------------------------------------------------------------------------------------------------
6     6          |Vert       |FurnitureFactory         |3          |3                |8            |8            |
-------------------------------------------------------------------------------------------------------------------
7     6          |Vert       |CheeseFactory            |5          |3                |7            |7            |
-------------------------------------------------------------------------------------------------------------------
8     6          |Vert       |ConvenienceStore         |2          |3                |4            |4            |
-------------------------------------------------------------------------------------------------------------------
9     6          |Vert       |Bakery                   |1          |1                |2            |3            |
-------------------------------------------------------------------------------------------------------------------
10    6          |Bleu       |AppleOrchard             |2          |1                |10           |10           |
-------------------------------------------------------------------------------------------------------------------
11    6          |Bleu       |Mine                     |6          |3                |9            |9            |
-------------------------------------------------------------------------------------------------------------------
12    6          |Bleu       |Forest                   |3          |1                |5            |5            |
-------------------------------------------------------------------------------------------------------------------
13    6          |Bleu       |Ranch                    |1          |1                |2            |2            |
-------------------------------------------------------------------------------------------------------------------
14    1          |Bleu       |Wheat Field              |1          |1                |1            |1            |
-------------------------------------------------------------------------------------------------------------------
```

->*view_slot_cards()*：用来打印当前牌库可供购买的牌（已经买完的卡牌不会出现在表格中）其中每张牌的打印都由 *print_card()*来完成



```
Ce sont les cartes vous pouvez choisir:
No.   Quantity   |Couleur    |Batiments                |Depense    |Points d'effet   |Points bas   |Points haut  |
-------------------------------------------------------------------------------------------------------------------
0     4          |Violet     |Stadium                  |6          |2                |6            |6            |
-------------------------------------------------------------------------------------------------------------------
1     4          |Violet     |TVStation                |7          |5                |6            |6            |
-------------------------------------------------------------------------------------------------------------------
2     4          |Violet     |BusinessCenter           |8          |0                |6            |6            |
-------------------------------------------------------------------------------------------------------------------
3     6          |Rouge      |FamilyRestaurant         |3          |2                |9            |10           |
-------------------------------------------------------------------------------------------------------------------
4     6          |Rouge      |Cafe                     |2          |1                |3            |3            |
-------------------------------------------------------------------------------------------------------------------
5     6          |Vert       |FruitAndVegetableMarket  |1          |1                |11           |12           |
-------------------------------------------------------------------------------------------------------------------
6     6          |Vert       |FurnitureFactory         |3          |3                |8            |8            |
-------------------------------------------------------------------------------------------------------------------
7     6          |Vert       |CheeseFactory            |5          |3                |7            |7            |
-------------------------------------------------------------------------------------------------------------------
8     6          |Vert       |ConvenienceStore         |2          |3                |4            |4            |
-------------------------------------------------------------------------------------------------------------------
9     6          |Vert       |Bakery                   |1          |1                |2            |3            |
-------------------------------------------------------------------------------------------------------------------
10    6          |Bleu       |AppleOrchard             |2          |1                |10           |10           |
-------------------------------------------------------------------------------------------------------------------
11    6          |Bleu       |Mine                     |6          |3                |9            |9            |
-------------------------------------------------------------------------------------------------------------------
12    6          |Bleu       |Forest                   |3          |1                |5            |5            |
-------------------------------------------------------------------------------------------------------------------
13    6          |Bleu       |Ranch                    |1          |1                |2            |2            |
-------------------------------------------------------------------------------------------------------------------
14    1          |Bleu       |Wheat Field              |1          |1                |1            |1            |
-------------------------------------------------------------------------------------------------------------------
```

->*view_player_cards()*：展示当前回合玩家所拥有的手牌



- *player_input()*函数负责识别当前回合玩家的输入指令，玩家可供输入的指令有：

指令	功能
view 玩家序号[1~4]	查看当前序号所对应的玩家手牌
view table	查看当前牌库中的所有卡牌
buy 卡牌序号	购买卡牌序号所对应的卡牌

#如果输入有误，会返回相应的错误原因

Commande inconnue - 未知的指令
Il n’y a pas de joueur - 没有当前的玩家
Tu ne peux pas vous permettre.(Pas assez de monnaies) -没有足够金币购买卡牌
Pas de carte au numero *n* - 没有编号为 *n* 的卡牌

- *red_card_check()*：当前回合玩家掷出骰子后，会判断场上是否有相应的红色卡牌（由于红色卡牌的属性是当别人掷到卡牌对应的点数时，需要支付给自己对应的金币），如果判断场上有相应点数的红牌，则拥有红牌的玩家金币数+n 当前投掷骰子玩家的金币数-n （n 为红色卡牌所对应的金币收益数）

- *blue_card_check()*：判断完红牌之后会继续判断场上的蓝色卡牌，如果场上玩家手牌中有对应的蓝色卡牌，那么所有玩家的金币数都+n

- *green_card_check()* :判断完蓝色卡牌后会判断当前回合玩家手中的绿色卡牌，该函数会统计玩家通过绿色卡牌赢得的金币总数，最后给与当前玩家相应的金币数

- *purple_card_check()* :在判断完绿色卡牌后，最后会判断当前回合玩家的紫色卡牌是否有被触发（紫色卡牌中：Stadium 是从其他所有玩家手中得到 2 个金币 TV Station 是从当前回合玩家手中选择一位玩家拿走 5 个金币 Office 则是选择一名玩家交换一张手牌）

- *buy_property()* ： 首先会打印当前玩家剩余的金币数量，当玩家输入想要购买的卡牌序号时，此函数会进行判断该名玩家是否有条件购买此张卡片，判断条件如下：

→当前玩家是否还没有购买过卡牌
→当前玩家所选择的卡牌序号是否存在于牌库之中
→当前玩家是否拥有足够的金币数
→当前玩家是否已经拥有这张卡拍（仅限购买一张的卡牌）

- *end_of_turn()* :此函数首先会判断当前回合的玩家手中的 4 张黄牌是否已经被全部激活（如果回合结束时，玩家已经购买了全部的黄牌——则宣告游戏结束，该名玩家是获胜者，否则继续跳转到下一名玩家的回合）

2. Deck

- 命名为 deck 的文件负责创建整个游戏中所需要的卡牌，通过将卡牌存放在 vector 容器中，一步步新创造卡牌。

```
for (int i = 0; i < 6; i++) { c = new WheatField(); this->deck.push_back(c); }//创建整局游戏
for (int i = 0; i < 6; i++) { c = new Ranch(); this->deck.push_back(c); }
for (int i = 0; i < 6; i++) { c = new Forest(); this->deck.push_back(c); }
for (int i = 0; i < 6; i++) { c = new Mine(); this->deck.push_back(c); }
for (int i = 0; i < 6; i++) { c = new AppleOrchard(); this->deck.push_back(c); }

for (int i = 0; i < 6; i++) { c = new Bakery(); this->deck.push_back(c); }
for (int i = 0; i < 6; i++) { c = new ConvenienceStore(); this->deck.push_back(c); }
for (int i = 0; i < 6; i++) { c = new CheeseFactory(); this->deck.push_back(c); }
for (int i = 0; i < 6; i++) { c = new FurnitureFactory(); this->deck.push_back(c); }
for (int i = 0; i < 6; i++) { c = new FruitAndVegetableMarket(); this->deck.push_back(c); }
```

- *Game::deal()*函数负责检查牌库里是否有已经全部买完的牌，如果有，则不再显示该张卡。
- *static Deck& getstance()*,引用单例模式，保证只能产生一个实例，并面向整个系统提供。

3. Bank

- 命名为 Bank 的文件负责管理玩家的金币数
- 首先在开头，默认构造器 this->coins = 3：每位玩家的初始金币数 3
- Bank::get_coins()函数得到玩家所持有的金币数。
- Bank::deposit(int val)函数负责玩家向里面存放金币，函数返回给存放后玩家持有的总金币数。
- Bank::withdraw(int val)函数负责扣钱，如果当前玩家所持有金币大于所扣的钱，则函数返回扣完前后玩家所持有的金币数；否则，函数返回 0

4. Card

Card 类以及它的继承类 BlueCard、GreenCard、RedCard、PurpleCard 和 YellowCard 等构成了 MachiKoro 游戏的基本牌的类型，它负责构造每一张牌，并且给每一张牌的属性赋值，并且实现了不同种类牌的效果。

- Card 的属性：

  int low_roll;//最低可以掷到的点数

  int high_roll;//最高可以掷到的点数

  int cost;//得到的金币数

  int value;//购买这张牌所需要的点数

  Color color;//卡牌颜色（枚举类型）

  Icon icon;//卡牌类型（枚举类型）

  string name;//卡牌名字（字符串）

  Card 的方法：

- *Set() & Get()* 系列函数达到为卡牌属性赋值的目的和在控制台打印卡牌属性的作用。
- *virtual void action(Bank *p1, Bank *p2, Card *c1, Card *c2, int val) = 0;* 在虚基类里使用纯虚函数来减少代码量的作用，并且可以在子类中进行重载，方便对特殊卡牌的书写。

  ColorandIcon.h 中的类：

- enum class Color 枚举类，存放卡牌的颜色。
- enum class Icon 枚举类，存放卡牌的类型

  BlueCard 及其继承类中的重载：

- p1->deposit(this->get_value()); 购买这张卡牌的玩家可以每个回合从银行收获对应数量的钱币。

PurpleCard 及其继承类中的重载：

- p1->deposit(p2->withdraw(this->get_value())); 购买这张卡牌的玩家可以在自己的回合从其他所有玩家处拿到对应数量的钱币。

RedCard 及其继承类中的重载：

- p1->deposit(p2->withdraw(this->get_value()+bonus)); 购买这张卡牌的玩家可以从投骰子的玩家处拿到对应数量的钱币。

GreenCard 及其继承类中的重载：

- void GreenCard::set_multiplier_icon(Icon icon); 根据卡牌类型设置一个 icon。
- void GreenCard::action(Bank *p1, Bank *p2, Card *c1, Card *c2, int val); 根据填写的 icon 类型每个回合从银行拿到对应数量的钱币。

YellowCard 及其继承类中的重载：

- YellowCard 里包含一个 bool 类型的属性 active，购买之后则 this->active = false;则被赋值为 this->active = true;。

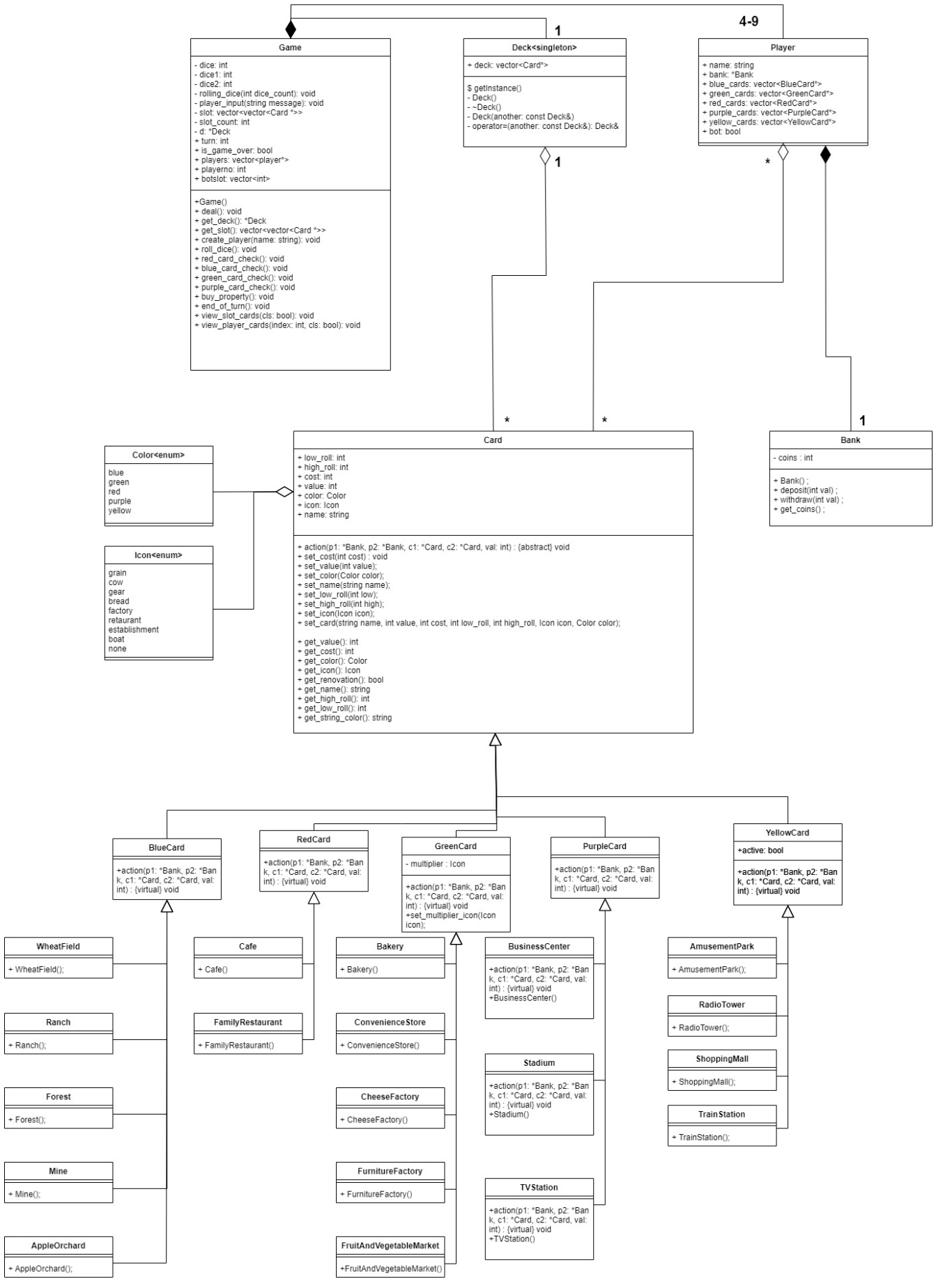
5. MachiKoro

- MachiKoro 中 view_slots(Game *g)函数负责打印出游戏中玩家信息的打印

```
void view_slots(Game *g)
{
    vector<vector<Card*>> s = g->get_slot();
    for (int i = 0; i < s.size(); i++)
    {
        cout << i << ": " << s[i][0]->get_name() << ": " << s[i].size() << endl;
    }
}
```

- 命名为 MachiKoro 的函数负责通过投掷骰子来控制游戏的进程。

四、 UML 图


Game
- dice: int
- dice1: int
- dice2: int
- rolling_dice(int dice_count): void
- player_input(string message): void
- slot: vector<vector<Card *>>
- slot_count: int
- d: *Deck
+ turn: int
+ is_game_over: bool
+ players: vector<player*>
+ playerno: int
+ botslot: vector<int>
+Game()
+ deal(): void
+ get_deck(): *Deck
+ get_slot(): vector<vector<Card *>>
+ create_player(name: string): void
+ roll_dice(): void
+ red_card_check(): void
+ blue_card_check(): void
+ green_card_check(): void
+ purple_card_check(): void
+ buy_property(): void
+ end_of_turn(): void
+ view_slot_cards(cls: bool): void
+ view_player_cards(index: int, cls: bool): void
Deck<singleton>
+ deck: vector<Card*>
$ getInstance()
- Deck()
- ~Deck()
- Deck(another: const Deck&)
- operator=(another: const Deck&): Deck&
Player
+ name: string
+ bank: *Bank
+ blue_cards: vector<BlueCard*>
+ green_cards: vector<GreenCard*>
+ red_cards: vector<RedCard*>
+ purple_cards: vector<PurpleCard*>
+ yellow_cards: vector<YellowCard*>
+ bot: bool
1
4-9
1
*
*
*
1
Color<enum>
blue
green
red
purple
yellow
Icon<enum>
grain
cow
gear
bread
factory
retaurant
establishment
boat
none
Card
+ low_roll: int
+ high_roll: int
+ cost: int
+ value: int
+ color: Color
+ icon: Icon
+ name: string
+ action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {abstract} void
+ set_cost(int cost) : void
+ set_value(int value);
+ set_color(Color color);
+ set_name(string name);
+ set_low_roll(int low);
+ set_high_roll(int high);
+ set_icon(Icon icon);
+ set_card(string name, int value, int cost, int low_roll, int high_roll, Icon icon, Color color);
+ get_value(): int
+ get_cost(): int
+ get_color(): Color
+ get_icon(): Icon
+ get_renovation(): bool
+ get_name(): string
+ get_high_roll(): int
+ get_low_roll(): int
+ get_string_color(): string
Bank
- coins : int
+ Bank() ;
+ deposit(int val) ;
+ withdraw(int val) ;
+ get_coins() ;
BlueCard
+action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {virtual} void
RedCard
+action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {virtual} void
GreenCard
- multiplier : Icon
+action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {virtual} void
+set_multiplier_icon(Icon icon);
PurpleCard
+action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {virtual} void
YellowCard
+active: bool
+action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {virtual} void
WheatField
+ WheatField();
Ranch
+ Ranch();
Forest
+ Forest();
Mine
+ Mine();
AppleOrchard
+ AppleOrchard();
Cafe
+ Cafe()
FamilyRestaurant
+ FamilyRestaurant()
Bakery
+ Bakery()
ConvenienceStore
+ ConvenienceStore()
CheeseFactory
+ CheeseFactory()
FurnitureFactory
+ FurnitureFactory()
FruitAndVegetableMarket
+FruitAndVegetableMarket()
BusinessCenter
+action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {virtual} void
+BusinessCenter()
Stadium
+action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {virtual} void
+Stadium()
TVStation
+action(p1: *Bank, p2: *Bank, c1: *Card, c2: *Card, val: int) : {virtual} void
+TVStation()
AmusementPark
+ AmusementPark();
RadioTower
+ RadioTower();
ShoppingMall
+ ShoppingMall();
TrainStation
+ TrainStation();

五、 项目亮点

1.引用工厂模式，通过工厂模式把对象的创建和使用过程分割开来，减少了代码量，易于维护。

2.模块化编程，将整个程序分成 Game、Deck、Carte、Player、Banque 多个模块进行编写，方便修改。

3.实现 AI 和玩家共同进行游戏，可以完成纯玩家玩法，AI 和玩家互战玩法以及 AI 玩法。

4.采用迭代式开发，简化聚合类，增加新的聚合类和迭代器类都很方便，封装性良好。

5.引入牌库，以此保证每张卡牌及其相关数据流的稳定性。

6.编写了 Readme.md 文件作为游戏的说明书，便于更快上手

六、 改进之处

1. 本小组尝试在创建卡牌时使用工厂设计模式，但是在经过一段时间的共同学习与尝试后暂时以失败告终，如果以后时间充裕可以继续学习并尝试。
2. 目前机器人玩家主要是基于随机数购买卡牌，可以进一步修改算法，让机器人行为策略更接近人类玩家。
3. 为了保证控制台输出效果，使用了较多 system(pause)，在游戏过程中需要经常敲回车，带来不好的游戏体验。进一步学习 C++知识可以让系统自动继续或者解决这一问题。
4. 有一些在控制台的输入会在下回合被识别成 commande inconnu。进一步学习计算机原理或者缓存的知识应该可以解决该问题。

七、 成员分工

姓名	学号	负责内容	时长	贡献度
杨翊麟	20124704	**主编文件：** Deck.cpp Deck.h Game.cpp Game.h MachiKoro.cpp MachiKoro.h YellowCard.cpp YellowCard.h **重要功能实现：** 黄色建筑的激活与购买 运用单例设计模式写 Deck 类 **其他工作：** 讨论项目整体流程 部分法语代码修改 UML 草图绘制 代码测试与反馈 部分报告撰写 视频制作与上传 项目提交	35h	22%
邵宸宇	20124701	**主编文件：** Readme.md Game.cpp Game.h MachiKoro.cpp MachiKoro.h PurpleCard.cpp PurpleCard.h **重要功能实现：** 编写 AI 机器人操作 各类卡牌的购买 **其他工作：** 讨论项目整体流程 代码测试与反馈 部分报告撰写	35h	22%
		主编文件： Card.cpp Card.h Bank.cpp Bank.h Game.cpp		

陆臣杰	20124561	Game.h MachiKoro.cpp MachiKoro.h BlueCard.cpp BlueCard.h **其他工作：** 讨论项目整体流程 代码详细记录与注释 代码测试与反馈 部分报告撰写	32h	20%
张一凡	20124702	**主编文件：** Card.cpp Card.h Game.cpp Game.h GreenCard.cpp GreenCard.h ColorAndIcon.h MachiKoro.cpp MachiKoro.h **其他工作：** 讨论项目整体流程 UML 图完整绘制 部分报告撰写 代码测试与反馈	32h	20%
施信宜	20124792	**主编文件：** Card.cpp Card.h Game.cpp Game.h RedCard.cpp RedCard.h MachiKoro.cpp MachiKoro.h **其他工作：** 讨论项目整体流程 美化控制台输出 部分法语代码修改 代码测试与反馈	30h	16%

八、 参考资料

1. https://github.com/CasperCBroeren/machikoro
2. https://github.com/EfdalKislali/MachiKoro
3. https://github.com/jensdenbraber/MachiKoro
4. https://github.com/NoOverflow/MachiKoro
5. https://github.com/jlu7/MachiKoro