C++ Project

Machikoro 骰子街

团队成员

黄浩 邱奕博 李毅聪 陈诺 肖文婷

2022/11/25

目录

一、 项目概括

我们成功设计出了 MachiKoro 的标准版玩法和其扩展包玩法，完成了项目文件中的大部分要求，能够实现 2-5 名玩家，通过在控制台的输入，进行游戏的游玩。该项目使用了标准 STL 库中的 Vector 及其迭代器，通过灵活使用指针，保证了项目中基础配置（牌库、玩家、银行等数值设置）的稳定性；项目中引用了单例模式，保证了 Class Game 只能产生一个实例，并向整个系统提供；我们引入了游戏牌库 Card，并根据其颜色属性进一步分成了四类属性牌库，易于添加新扩展包的卡牌；最后，我们为程序加入了 AI 玩家，并实现了对于标准玩法和扩展包玩法的自由选择，并且控制台界面上显示了卡牌、点数、钱币的详细信息；此外，我们通过 Qt 部分实现了本项目的可视化界面。

已实现功能：玩家人数自由添加、AI 玩家、标准版与 DLC 玩法切换与实现

待优化与实现功能：更多 DLC 版本附加卡牌的实现；紫色重要建筑物卡牌的功能封装

二、 项目流程

1. 小组成员了解该桌游规则，体验游戏
2. 配置协同工作环境，利用 Git 代码版本管理平台和 gitee 远程库进行版本管理
3. 根据桌游规则和玩法，来构想程序的大致框架，列出所需编写的类，分工进行编程，并设定相关时间节点，保证项目按时完成
4. 画出项目的 UML 图，在程序中声明与实现类中的属性和方法
5. 整合小组成员的编程内容，并进行运行调试，对于程序进行漏洞修补
6. 实现游戏的成功运行，撰写项目报告

三、 代码实现

本项目主要分为以下四大部分，分别为

Card：卡牌，包含所有的卡牌属性及具体功能

Deck：牌桌，主服务于扩展包玩法的随机抽牌机制，负责从 Card 中抽牌到牌桌上

Bank：银行，负责钱币发放和扣款,管理玩家钱币

Game：游戏主体流程，负责完整流程的正常运行，包含了玩家（真人/AI）的设置，游戏玩法选择，以及游戏掷骰子、购买扣款、游戏胜负判定等主要操作的触发

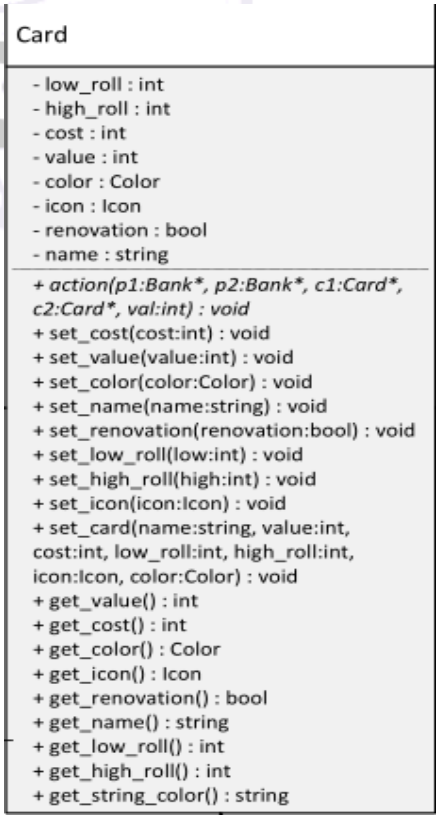
接下来，将基于以上四部分及其他重要功能，结合 UML 图进一步阐述本项目的框架与可扩展性

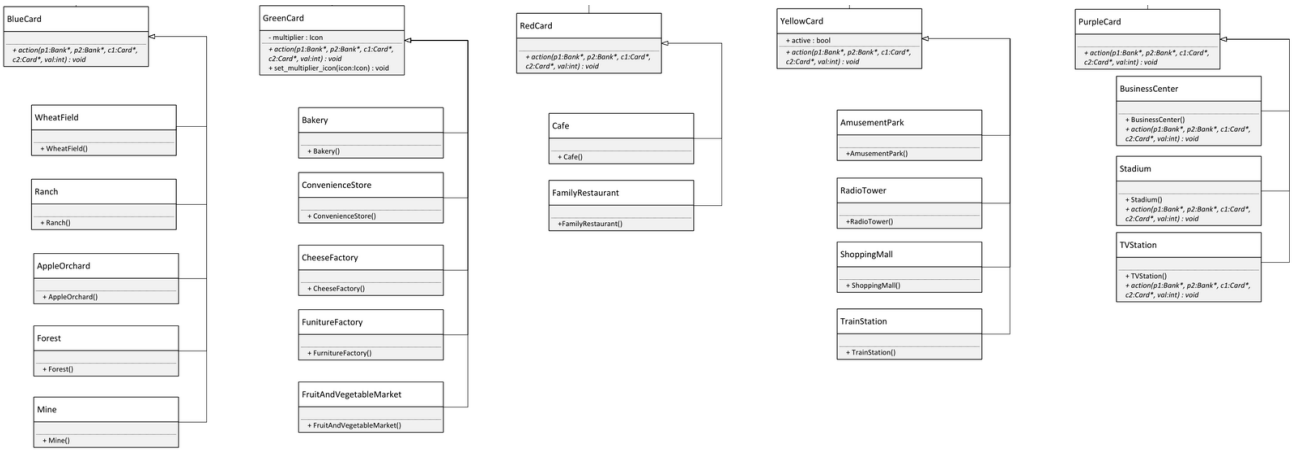
1. Card

Card 文件中定义了游戏中所有需要的卡牌及其属性（包括其名字，颜色，名字，颜色，价格，触发点数范围，可获得钱币数量等），为了便于管理和进行后续卡牌的扩展，我们将 Card 类定义为抽象类，仅作为接口使用。

之后由五类不同属性的卡牌（以颜色作为区分，分别为 Red，Blue，Green，Purple，Yellow）进行继承，最后为每一种具体的建筑物卡牌撰写一个类并继承其对应的颜色属性类，进行卡牌属性的写入。

其 Card 相关的 UML 图下图和右图所示





Card 类主要为后面的子类提供了卡牌属性和功能（action()为纯虚函数）的写入接口

除了紫色和黄色卡牌以外，红，绿，蓝三种卡牌都各自具有共通的功能，故在撰写这三个派生类时，可直接重载 action 函数，传入 player 指针和 bank 指针，通过多态来实现当前玩家钱币扣款、拿钱操作。

而紫色和黄色卡牌，由于自身的特异性，在此处仅仅是预留了 action 接口，并未在处说明实现方式，而是由具体的派生类卡牌继承之后，在具体的卡牌名中进行重载实现。

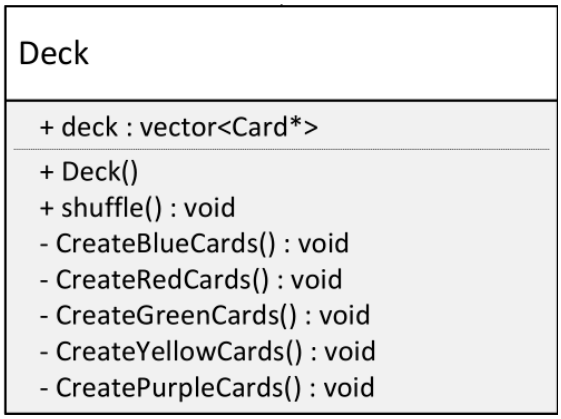
在具体的游戏进程中，Deck 将加载入 Card 中的卡牌，完成牌桌上的卡牌设置，之后游戏主进程 Game 类将对 Deck 的生命周期负责，代表玩家的 Player 类中会有五种颜色的 vector 指针数组来对应存储各类卡牌，并通过 vector 自身的迭代器实现游玩进程中卡牌的增减，并根据骰子点数和卡牌情况调用对应的 action 函数，实现玩家钱币的增减或其他效果 的触发

这种设计有利于我们日后添加新的建筑卡牌，只需要在其对应的颜色属性文件中写入 public 继承并通过函数 set_card 进行属性设置即可添加完毕，而不需要大改游戏的主体运行架构。

2. Deck

为了使代码最大程度地贴合实际玩法，这里引入了个 Deck 类作为牌堆，负责所有卡牌的生成。注意这里只负责生成，因为这些生成的牌最后会被转移至 Game 类中的 slot 卡槽中。Deck 里的函数不多，只有 shuffle()一个，通过 std::random_shuffle 将 deck 所存储的所有 card 随机打乱，生成顺序随机的牌

堆。



3. AI

在 player 类中添加 bool 变量 isBot，在 player 构造函数中添加参数 isBot 默认为 false，需要玩家作为 AI 时则赋真。在买卡阶段，通过 random 函数，随机选择购买某种卡牌。为了避伪随机数在短时间内大量重复导致刷屏的问题，这里用一个计数器和 if 语句，让 AI 在第一次购买失败后选择不购买退出循环。

4. Bank

Bank 类与 Player 的关系是 Composition。Player 有 Bank*对象作为 Player 的成员。并且由 Player 全权管理其生命周期，当 Player 结束，它会带着 Bank 一起结束。

```
player* p = new player();
p->bank = new Bank();
```

可以理解为每生成一个玩家就会有一个他对应的银行生成，用于管理这名玩家这局游戏的 coins（金币）。当该玩家游戏结束，其银行会自动随其一起被销毁。

Bank 类中将 coins 设为私有 private，并在 public 中提供了 int get_coins() 的方法来获取 coins。此外还定义了两个函数：

```
void deposit（int val）;
int withdraw (int val);
```

两个函数分别起到获取金币以及扣除金币的作用。

①获取金币：当调用 deposit 函数时，在该玩家 coins 基础上加上传入的 val。

```
this->coins += val;
```

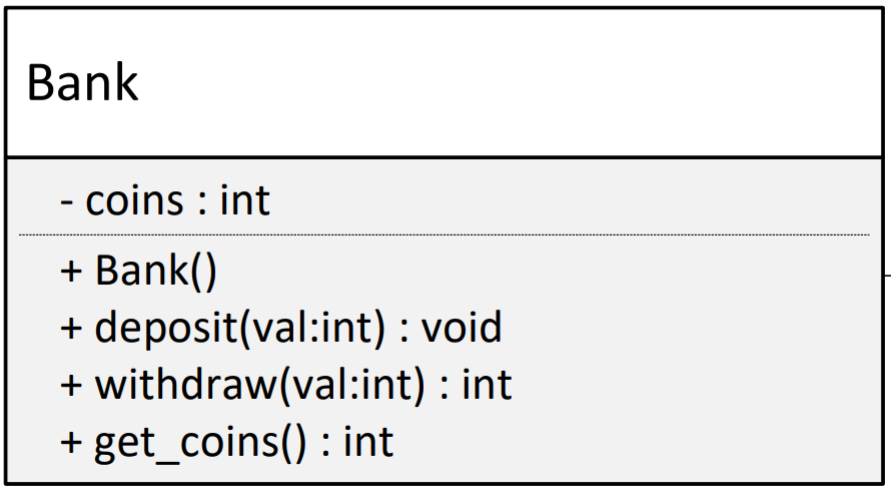
②扣除金币：先将传入的 val 与 coins 作比较（用到 if）。如果需要扣除的金币（val）大于玩家所拥有的金币（coins），则玩家金币（coins）变为 0，并返回。（这里是 withdraw 不能简单写为 void 与 deposit 类似的原因，因为玩家的金币数量必然不为负数，当玩家需要向别的玩家支付金币（coins）的时候，如果玩家的金币（coins）不足够，将被视为 0）

```
int temp = this->coins;
this->coins = 0;
return this->coins;
```

如果需要扣除的金币（val）小于拥有的金币（coins），则返回扣除了 val 的 coins。

```
this->coins -= val;
return val;
```

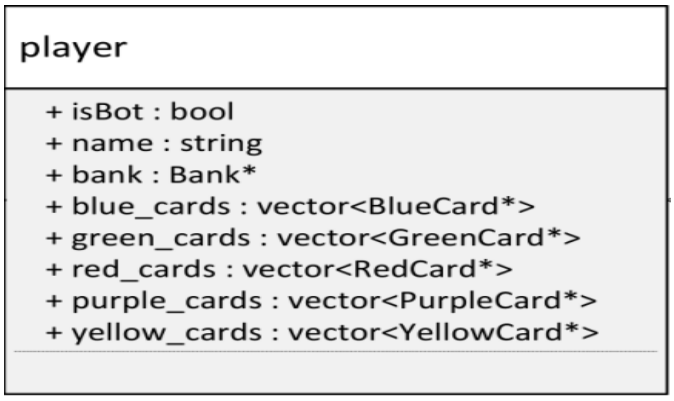
因为 Bank 由 Player 生成，所以每次调用 Bank 时，都需要由 Player 来调用。例如：“this->players[this->turn]->bank;”。



5. Game

Game.h 文件中总共包含两个类，一个是 Player 类，一个是 Game 类。他们分别起到管理玩家（player）以及管理游戏进程（game progress）的作用。

1. player 类



player 类中包含玩家的游戏内的基础信息

① name（玩家名）

② bank（生成玩家对应的银行 bank）

③ 五种卡牌的容器（例如：vector<BlueCard*> blue_cards;）

生成玩家后，相应的为玩家开辟一个银行的内存，同时生成五中卡牌的容器，玩家获取的所有卡牌会放在相应的卡牌容器中，方便游戏主程序中的遍历

在此基础上我们增加了 AI 玩家（isBot）通过 bool 的方式来决定 AI 是否进入游戏生成一名玩家。使得单人也可以进行游戏。

2. Game 类

Game 类起到管理游戏进程的作用。

私有成员（private）包含了一些重要的信息，包骰子（dice）、摇骰子（rolling_dice），规定玩家 input 的信息（player_input）、卡槽（slot）、牌堆（Deck）等重要且不能被类外访问的信息，保证游戏合理进行。公有成员（public）

游戏开始，Game 会分别通过 choose_game()、deal()来选择游戏模式以及生成数据，并通过 get 的方式用于获取数据。之后通过 create_player(string name, bool bot=false)生成玩家，通过 bool 方式决定 AI 是否运作。后面包含 roll_dice()用于进行投骰子操作，多个 check 函数，判断玩家库中是否存在某种颜色的卡片并根据卡的不同颜色进行相应操作。

bool choose_game()

选择游戏模式

void deal()

生成基本数据

void create_player(string name, bool bot=false)

生成玩家，通过 bool 决定 AI 是否运作

void roll_dice()

进行投骰子操作

void colour_card_check()

判断玩家库中是否存在某种颜色的卡片并根据卡的不同颜色进行相应操作

void buy_propery()

进行买卡操作

void end_of_turn()

对是否结束进行判定，并且进入下一玩家回合

void view_slot_cards(bool cls)

显示卡槽目前卡片

void view_player_cards(int index, bool cls)

显示玩家拥有卡片

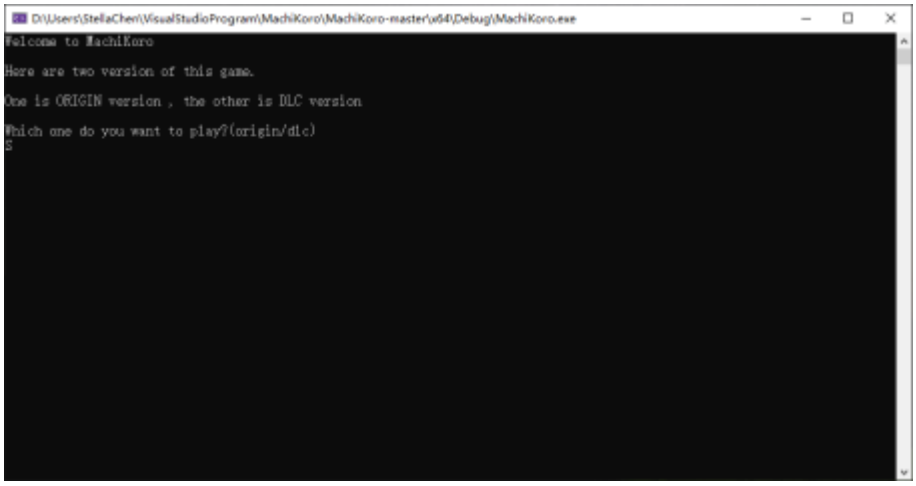
class iterator :public vector<vector<Card*>>::iterator()

定义迭代器方便遍历

几个关键函数

1. choose_game()

放置于主程序中，作为游戏运行时的开始界面，用来选择模式。在 Game 类中增加布尔变量 version_old，通过正则（std::regex）控制输入，若输入为 dlc，则将 version_old 赋假，进入 DLC 模式。



2.deal()

deal 函数紧接着 choose_game()，用于将牌堆中的牌放入卡槽。DLC 扩展包相比于原版，除了牌的种类有了扩充，最大的改变是卡槽功能。这里创建一个 slot，这是 card*的 vector 的 vector，整合所有卡槽。通过 Deck.back()将牌从牌堆底抽出，然后通过迭代器设计模式（iterator），遍历 slot 已有的牌种类进行对比，如果有同种就通过 pushback 叠加在该种卡槽上，如果没有就开一个新的卡槽，每次加卡，不创建新卡，而是从 deck 类中将牌指针转移到卡槽内，并从 deck 里 popback 销毁这张牌，达到发牌的功能。

3.buy_propery()函数

玩家选择要买的卡号，并根据颜色对卡片进行分类加入玩家手中，并使卡槽中的卡数量-1。其中紫卡进行判定是否已经拥有来决定是否可以购买（黄卡独立在该函数之外）

4.end_of_turn()函数

对玩家是否结束本回合进行判定，并且进入下一玩家回合。并通过判断玩家是否已经将四张黄卡全部买完进而判断玩家是否获胜。

5.关于黄卡 YellowCard

YellowCard* c;

例如：c = new TrainStation(); p->yellow_cards.push_back(c);

不同于其他颜色卡，黄色卡在玩家创建时已经加入到人物卡包里面，但激活状态默认为 false。之后通过 bool 来对对它是否 active 进行判断来进行相应操作。

6. print/view 函数

通过操作符<<重载、setw 控制输出格式以及 iterator 的遍历，实现各种 print 和 view 的函数，包括 print_card / print_card_heading / view_slot_cards / view_player_cards 等等

6. QT 可视化界面

在原有代码的基础上，我们使用 qt 对项目进行了简单的可视化处理，使用情况详见视频介绍

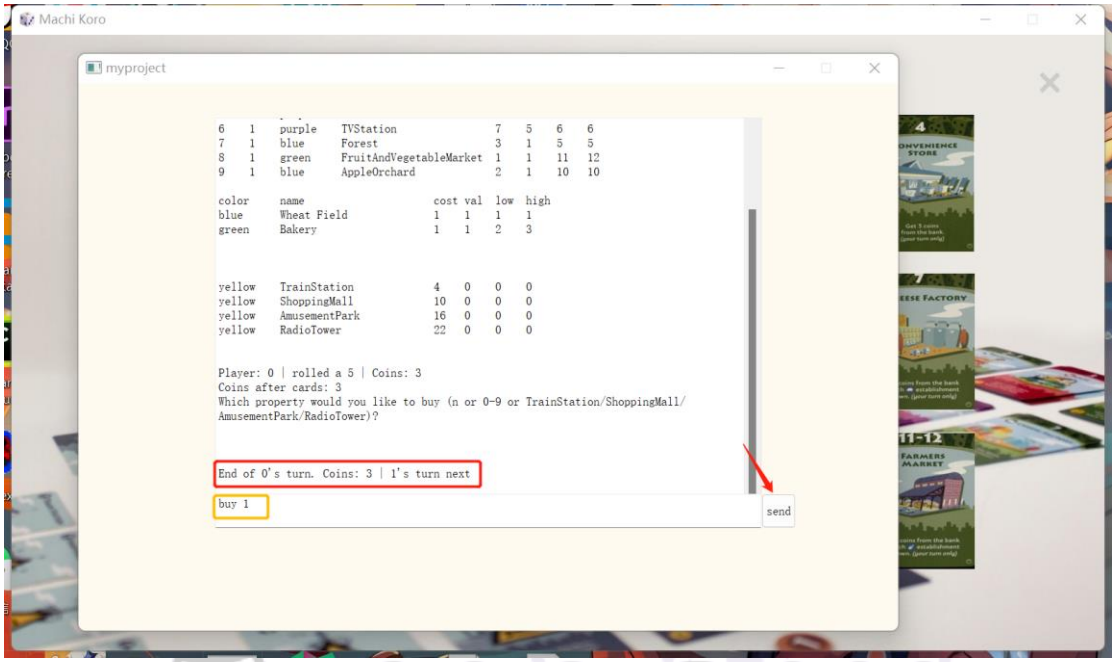
1. 主页面

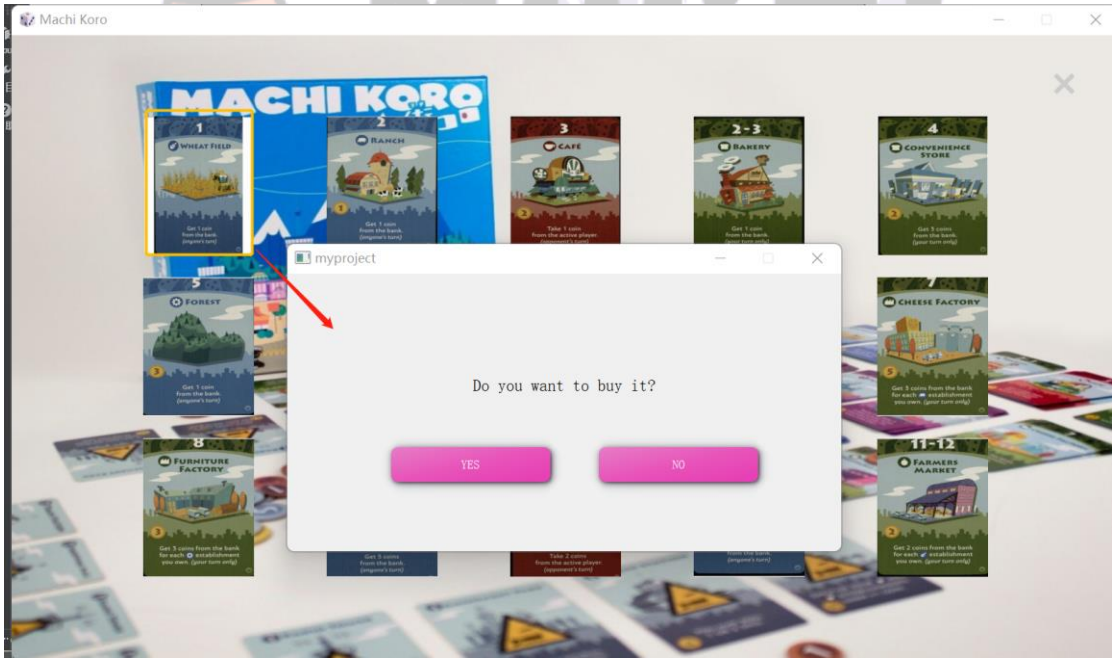
在主页面中，我们提供了两种模式，可供玩家进行自由选择

2. 游戏界面

在游戏界面中我们提供了文本框对游戏实时信息进行打印。

在游戏过程中，我们提供了两种购买模式，一种是输入购买，一种是点击图片购买

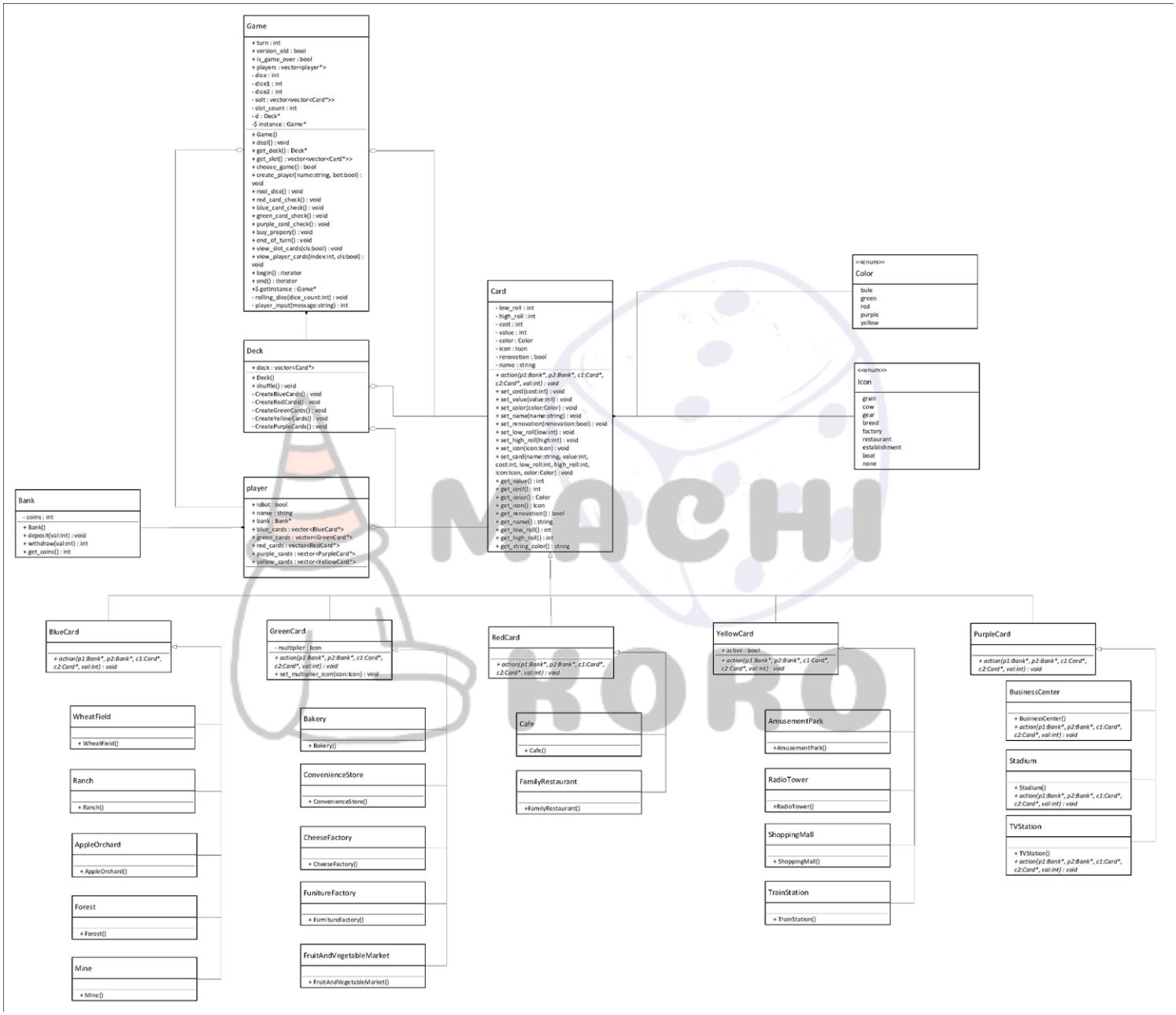




在购买成功之后，更新信息会在文本框中进行打印，进行数据计算，结束该玩家回合，顺位进入下位玩家回合，重复新的循环。

四、 UML

1. UML 图



2. 项目架构设计模式分析

单例设计模式(Singleton）

在 Game 类引入单例设计模式，确保 Game 类在系统中各只产生一个实例，并自行实例化，同时向整个系统提供获取这个唯一实例的接口。

迭代器设计模式(Iterator)

利用 vector 容器的迭代器顺序访问类 slot 中的各个元素而不暴露其内部的表示。

工厂设计模式(Factory)

为方便卡牌管理与灵活扩展，我们引入工厂设计模式。基类 Card 仅负责具体子类必须实现的接口，使系统在后续能够不修改具体工厂角色的同时引进新的扩展。

五、优点

1. 游戏版本选择

在不改变基础版框架的基础上，通过指针数组与迭代器灵活实现了与 DLC 两种版本游戏玩法的切换，使得游戏完成度更高，游戏趣味性也更高。

2. Card 中的多重继承

考虑到游戏卡牌种类与功能的多样性，我们在撰写牌库的时候，根据卡牌的特点进行了多重继承，每一类建筑物将在继承卡牌基本属性和颜色功能属性后，重载 action()功能函数，这样的做法有两种好处，一是在游戏进程中，简单便捷实现基于颜色顺序的卡牌触发，无需单独将每一种卡牌撰写条件判断语句；二是在程序扩展中，便于直接添加更多的建筑物卡牌，无需对整体游戏程序进行大的修改。

3. AI 实现

游戏中可以实现 AI 与玩家共同进行游戏，也可以完成纯玩家玩法的游玩。

可以实现单人游戏

4. 可视化界面的实现

通过 Qt 我们为本项目撰写了一个可视化界面，相比控制台程序更为美观。

5. 使用容器和迭代器

在程序当中，我们大量使用了 Vector 容器来储存卡牌，包括 Deck 牌堆、Slot 卡槽以及每个玩家类所拥有的卡牌。借助迭代器设计模式，更高效地遍历卡牌，实现洗牌、抽牌以及用牌等效果。

6. 正则

借助 C++标准库的正则，规范玩家的输入，高效地处理输入非法的情况。

六、改进之处

1. 紫色建筑物卡牌的效果代码还有待优化，需要进行进一步的封装，使代码更为简洁高效
2. 生成 Deck 时可以根据卡牌的不同颜色属性对应写不同的方法，使代码的复用性提高
3. 图形化界面还可以进一步的优化和美化

七、个人贡献

姓名	学号	负责内容	时长	贡献度
陈诺	20124686	主编文件： 1. Machikoro 文件 2. Game 文件（卡牌遍历效果） 3. UML 图 其他工作：文档编写、git 库维护、游戏视频录制	5h/week	20%
李毅聪	20124719	主编文件： 1. Deck 文件 2. Game 文件（AI、游戏模式选择、DLC 扩展包卡槽部分、正则） 其他工作：文档编写，代码注释	5h/week	20%
邱奕博	20124695	主编文件： 1. Card 和其衍生文件 2. Game 文件（打印设计，购买条件判断设计，骰子判断设计，单例设计模式，迭代器设计模式） 其他工作：QT 可视化界面，代码注释，文档编写	5h/week	20%
肖文婷	20124787	主编文件： 1. Card 和其衍生文件 2. Game 文件（玩家设置，购买条件判断，卡牌判定） 3. UML 图 其他工作：文档编写、代码注释添加	5h/week	20%

黄浩	20124749	主编文件： 1. Bank 文件以及使用 2. Game 文件中与 coins 有关函数（购买卡牌、扣款、bug 修复...） 其他工作： 视频剪辑、文档编写、代码注释	5h/week	20%