Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

Высшего образования

«Алтайский государственный технический университет им. И. И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчет защищен с оценкой

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. М. Старолетов

(подпись) (и. о., фамилия)

« » 2017 г.

дата

Комплексный отчет

по дисциплине «Архитектурное проектирование и паттерны программирования»

ЛР 09.03.04.26.000О

Студент группы ПИ-42 И. С. Хышов

(и. о., фамилия)

Преподаватель доцент, к.ф.-м.н. С. М. Старолетов

должность, ученое звание (и. о., фамилия)

Барнаул 2017

# Постановка задачи

## Предметная область программы

Приложение представляет собой карточную игру с разными типами карт. Участвуют два игрока. Каждый игрок поочерёдно вводят свои карты на поле либо используют выведенные на поле карты для атаки карты противника или помощи своей карте. Каждая карта имеет свои характеристики, будь то набор способностей, различающиеся по силе и направленности действия.

Выигрывает тот, кто под конец игры последним остаётся с какими-либо картами.

# Используемые паттерны

## Делегирование

**Назначение:**

Класс карты (ActorCard или ActionCard) делегирует задачу воздействия на другую карту классу действия, наследующемуся от интерфейса IEffect. Также, если таких действий несколько (такое возможно у объектов класса Actor Card), предполагается использование класса выборщика действий, наследованного от интерфейса IEffectSelector.

**Диаграмма классов:**

****

**Исходный код:**

**card.h**

#pragma once

#ifndef ICARD\_H

#define ICARD\_H

#include <QObject>

#include "defs.h"

#include "effect.h"

#include "effectselector.h"

#include "namegen.h"

#include "transmitters.h"

#include "util.h"

#include "gamesession.h"

// интерфейс карты

class ICard : public QObject

{

Q\_OBJECT

public:

ICard(QObject \*parent = 0);

// действие

virtual void act(QList<ICard \*> cardsToAffect) = 0;

...

// get и set для некоторых числовых данных

virtual int getStat(QString statName) = 0;

virtual void setStat(QString statName, int statVal, bool isInc = false) = 0;

// get и set для списка действий

virtual void setEffects(QList<IEffect \*> effects) = 0;

virtual QList<IEffect \*> getEffects() = 0;

// обновление статистики

virtual void updateStats() = 0;

...

virtual void acceptStat(QString statName, int statVal, bool isInc = false) = 0;

signals:

void disposeMe();

void updateDrawable(listItem);

void callActionSelect();

void sendMsg(QString msg);

};

// карта-персонаж

// пока позволяют "здоровье" и "энергия", может действовать сколько угодно раз

// может иметь несколько действий

class ActorCard : public ICard

{

Q\_OBJECT

private:

QString name, uid;

int health, healthInit;

int energy, energyInit;

int selectedAction;

QColor bg;

QList<IEffect \*> actionEffect;

IEffectSelector \*defaultSelector;

CardInfoObserver \*cio;

public:

ActorCard(QObject \*parent = 0);

void act(QList<ICard \*> cardsToAffect);

...

int getStat(QString statName);

void setStat(QString statName, int statVal, bool isInc = false);

void setEffects(QList<IEffect \*> effects);

QList<IEffect \*> getEffects();

void setEffectSelector(IEffectSelector \*selector);

IEffectSelector \* getEffectSelector();

void updateStats();

...

public slots:

void makeActionSelection(int id);

void acceptStat(QString statName, int statVal, bool isInc = false);

};

// карта-действие

// имеет только одно действие

// действует только один раз, после чего уходит с поля

// зато неуязвима

class ActionCard : public ICard

{

Q\_OBJECT

private:

QString name, uid;

QColor bg;

IEffect \*actionEffect;

CardInfoObserver \*cio;

public:

ActionCard(QObject \*parent = 0);

void act(QList<ICard \*> cardsToAffect);

...

int getStat(QString statName);

void setStat(QString statName, int statVal, bool isInc = false);

void setEffects(QList<IEffect \*> effects);

QList<IEffect \*> getEffects();

void updateStats();

...

public slots:

void makeActionSelection(int id);

void acceptStat(QString statName, int statVal, bool isInc = false);

};

...

#endif // ICARD\_H

**card.cpp**

#include "card.h"

ICard::ICard(QObject \*parent):QObject(parent)

{

}

ActorCard::ActorCard(QObject \*parent):ICard(parent)

{

cio = NULL;

NameGen \*ng = new NameGen();

bg = QColor(Random::randInt(0, 255),

Random::randInt(0, 255),

Random::randInt(0, 255));

healthInit = health = 100;

energyInit = energy = 100;

selectedAction = -1;

uid = ng->randomUID();

delete ng;

}

void ActorCard::act(QList<ICard \*> cardsToAffect)

{

GameSession \*gs = GameSession::getInstance();

if (energy <= 0)

return;

QVector<StatsTransmitter \*> transmitters;

QMap< StatsTransmitter \*, ICard \* > transMapping;

StatsTransmitter \*srcTrans = new StatsTransmitter();

for (int i = 0; i < cardsToAffect.size(); i++)

{

StatsTransmitter \*st = new StatsTransmitter();

connect(st, SIGNAL(sendStat(QString,int,bool)), cardsToAffect[i], SLOT(acceptStat(QString,int,bool)));

transmitters.push\_back(st);

transMapping[st] = cardsToAffect[i];

}

connect(srcTrans, SIGNAL(sendStat(QString,int,bool)), this, SLOT(acceptStat(QString,int,bool)));

int effectsNum = actionEffect.size();

if (effectsNum == 1)

{

gs->msgRepeater(this->getName() + " использует " + actionEffect[0]->getName());

actionEffect[0]->act(transmitters, srcTrans);

}

else

{

if (selectedAction != -1)

{

actionEffect[selectedAction]->act(transmitters, srcTrans);

selectedAction = -1;

}

else

{

QList<IEffect \*> pick = defaultSelector->selectEffect(actionEffect);

for (int i = 0; i < pick.size(); i++)

{

gs->msgRepeater(this->getName() + " использует " + pick[i]->getName());

}

QVector<StatsTransmitter \*> selectedTrans;

for (int j = 0; j < transmitters.size(); j++)

{

if ((transMapping[transmitters[j]])->getStat("health") > 0)

{

gs->msgRepeater(this->getName() + " воздействует на " + transMapping[transmitters[j]]->getName());

selectedTrans.push\_back(transmitters[j]);

}

}

EffectChainlink \*efc = new EffectChainlink(pick.takeLast());

while (!pick.empty())

efc = new EffectChainlink(pick.takeLast(), efc);

efc->act(selectedTrans, srcTrans);

//pick[i]->act(selectedTrans, srcTrans);

delete efc;

}

}

while (!transmitters.empty())

delete transmitters.takeAt(0);

delete srcTrans;

}

int ActorCard::getStat(QString statName)

{

if (statName == "health")

return health;

else if (statName == "energy")

return energy;

else if (statName == "healthInit")

return healthInit;

else if (statName == "energyInit")

return energyInit;

else if (statName == "actionNum")

return actionEffect.size();

else return -1;

}

void ActorCard::setStat(QString statName, int statVal, bool isInc)

{

if (statName == "health")

{

if (isInc)

health += statVal;

else

health = statVal;

}

else if (statName == "energy")

{

if (isInc)

energy += statVal;

else

energy = statVal;

}

else if (statName == "healthInit")

{

if (isInc)

healthInit += statVal;

else

healthInit = statVal;

}

else if (statName == "energyInit")

{

if (isInc)

energyInit += statVal;

else

energyInit = statVal;

}

else if (statName == "selectedAction")

selectedAction = statVal;

updateStats();

}

void ActorCard::setEffects(QList<IEffect \*> effects)

{

actionEffect = effects;

}

QList<IEffect \*> ActorCard::getEffects()

{

return actionEffect;

}

void ActorCard::setEffectSelector(IEffectSelector \*selector)

{

defaultSelector = selector;

}

IEffectSelector \*ActorCard::getEffectSelector()

{

return defaultSelector;

}

void ActorCard::updateStats()

{

if (health <= 0)

{

disposeMe();

if (cio)

cio->dispose(this);

}

else

{

if (cio)

cio->notify(listItem(this->getDrawable(), this->getPseudoDrawable(), this->name, this->uid));

}

}

void ActorCard::makeActionSelection(int id)

{

selectedAction = id;

}

void ActorCard::acceptStat(QString statName, int statVal, bool isInc)

{

this->setStat(statName, statVal, isInc);

}

ActionCard::ActionCard(QObject \*parent):ICard(parent)

{

cio = NULL;

NameGen \*ng = new NameGen();

bg = QColor(Random::randInt(0, 255),

Random::randInt(0, 255),

Random::randInt(0, 255));

uid = ng->randomUID();

delete ng;

}

void ActionCard::act(QList<ICard \*> cardsToAffect)

{

GameSession \*gs = GameSession::getInstance();

QVector<StatsTransmitter \*> transmitters;

StatsTransmitter \*srcTrans = new StatsTransmitter();

gs->msgRepeater(this->getName() + " использует " + actionEffect->getName());

for (int i = 0; i < cardsToAffect.size(); i++)

{

gs->msgRepeater(this->getName() + " воздействует на " + cardsToAffect[i]->getName());

StatsTransmitter \*st = new StatsTransmitter();

connect(st, SIGNAL(sendStat(QString,int,bool)), cardsToAffect[i], SLOT(acceptStat(QString,int,bool)));

transmitters.push\_back(st);

}

connect(srcTrans, SIGNAL(sendStat(QString,int,bool)), this, SLOT(acceptStat(QString,int,bool)));

actionEffect->act(transmitters, srcTrans);

while (!transmitters.empty())

delete transmitters.takeAt(0);

delete srcTrans;

disposeMe();

if (cio != NULL)

cio->dispose(this);

}

int ActionCard::getStat(QString statName)

{

if (statName == "actionNum")

return 1;

else if (statName == "health" || statName == "energy")

return 999;

return 0;

}

void ActionCard::setStat(QString statName, int statVal, bool isInc)

{

return;

}

void ActionCard::setEffects(QList<IEffect \*> effects)

{

actionEffect = effects[0];

}

QList<IEffect \*> ActionCard::getEffects()

{

return {actionEffect};

}

void ActionCard::updateStats()

{

}

QList<effectInfoItem> ActionCard::getEffectsInfo()

{

return {actionEffect->getInfo()};

}

void ActionCard::makeActionSelection(int id)

{

return;

}

void ActionCard::acceptStat(QString statName, int statVal, bool isInc)

{

this->setStat(statName, statVal, isInc);

}

**effect.h**

#pragma once

#ifndef IEFFECT\_H

#define IEFFECT\_H

#include <QObject>

#include "defs.h"

#include "transmitters.h"

const QList<QString> effectDict = {"AttackEffect",

"HelpEffect"};

class ICard;

// интерфейс действия карты

class IEffect : public QObject

{

Q\_OBJECT

public:

IEffect(QObject \*parent = 0);

virtual QString getName() = 0;

virtual void setName(QString effectName) = 0;

virtual int getStat(QString statName) = 0;

virtual void setStat(QString statName, int statVal) = 0;

virtual void act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src) = 0;

virtual QString getFullInfo() = 0;

virtual effectInfoItem getInfo() = 0;

virtual IEffect \* clone() = 0;

// загрузка и сохранение

virtual void loadFromStream(QDataStream \*stream) = 0;

virtual void saveToStream(QDataStream \*stream) = 0;

signals:

void callCardSelect();

void done();

void sendMsg(QString msg);

};

// атака на другую карту (урон другой и трата "энергии" у своей)

class AttackEffect : public IEffect

{

Q\_OBJECT

private:

QString name; // назавние

int power; // сила

int cost; // затраты

public:

AttackEffect(QObject \*parent = 0);

QString getName();

void setName(QString effectName);

int getStat(QString statName);

void setStat(QString statName, int statVal);

void act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src);

QString getFullInfo();

effectInfoItem getInfo();

IEffect \* clone();

void loadFromStream(QDataStream \*stream);

void saveToStream(QDataStream \*stream);

};

// помощь "своей" карте

class HelpEffect : public IEffect

{

Q\_OBJECT

private:

QString name;

int amount;

int helpType;

int cardsLimit;

public:

HelpEffect(QObject \*parent = 0);

QString getName();

void setName(QString effectName);

int getStat(QString statName);

void setStat(QString statName, int statVal);

void act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src);

QString getFullInfo();

effectInfoItem getInfo();

IEffect \* clone();

void loadFromStream(QDataStream \*stream);

void saveToStream(QDataStream \*stream);

};

#endif // IEFFECT\_H

**effect.cpp**

#include "effect.h"

IEffect::IEffect(QObject \*parent):QObject(parent)

{

}

AttackEffect::AttackEffect(QObject \*parent):IEffect(parent)

{

}

QString AttackEffect::getName()

{

return name;

}

void AttackEffect::setName(QString effectName)

{

name = effectName;

}

int AttackEffect::getStat(QString statName)

{

if (statName == "power")

return power;

else if (statName == "cost")

return cost;

else if (statName == "effectType")

return 0;

return 0;

}

void AttackEffect::setStat(QString statName, int statVal)

{

if (statName == "power")

power = statVal;

else if (statName == "cost")

cost = statVal;

}

void AttackEffect::act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src)

{

for (int i = 0; i < cardsToAffect.size(); i++)

{

src->setStat("energy", -1 \* cost, true);

cardsToAffect[i]->setStat("health", -1 \* power, true);

}

}

QString AttackEffect::getFullInfo()

{

return name + " (сила "

+ QString::number(power)

+ ", стоимость"

+ QString::number(cost) + ");";

}

effectInfoItem AttackEffect::getInfo()

{

effectInfoItem result;

result.info = getFullInfo();

result.canAffectEnemy = true;

result.canAffectPlayer = false;

result.effectType = 0;

return result;

}

HelpEffect::HelpEffect(QObject \*parent):IEffect(parent)

{

}

QString HelpEffect::getName()

{

return name;

}

void HelpEffect::setName(QString effectName)

{

name = effectName;

}

int HelpEffect::getStat(QString statName)

{

if (statName == "amount")

return amount;

else if (statName == "helptype")

return helpType;

else if (statName == "limit")

return cardsLimit;

else if (statName == "effectType")

return 1;

return 0;

}

void HelpEffect::setStat(QString statName, int statVal)

{

if (statName == "amount")

amount = statVal;

else if (statName == "helptype")

helpType = statVal;

else if (statName == "limit")

cardsLimit = statVal;

}

void HelpEffect::act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src)

{

for (int i = 0; i < cardsToAffect.size(); i++)

{

if (helpType == 0)

cardsToAffect[i]->setStat("health", amount, true);

else

cardsToAffect[i]->setStat("energy", amount, true);

}

}

QString HelpEffect::getFullInfo()

{

return name + " (уровень "

+ QString::number(amount)

+ ");";

}

effectInfoItem HelpEffect::getInfo()

{

effectInfoItem result;

result.info = getFullInfo();

result.canAffectEnemy = false;

result.canAffectPlayer = true;

result.effectType = 1;

return result;

}

**effectselector.h**

#ifndef EFFECTSELECTOR\_H

#define EFFECTSELECTOR\_H

#include <QObject>

#include "defs.h"

#include "effect.h"

// интерфейс "выборщика" действий

class IEffectSelector : public QObject

{

Q\_OBJECT

public:

explicit IEffectSelector(QObject \*parent = 0);

virtual QList<IEffect \*> selectEffect(QList<IEffect \*> effs) = 0;

virtual int getType() = 0;

};

// берет одно случайное действие

class EffectSelectorRandom : public IEffectSelector

{

Q\_OBJECT

public:

explicit EffectSelectorRandom(QObject \*parent = 0);

QList<IEffect \*> selectEffect(QList<IEffect \*> effs);

int getType();

};

// берет все и перемешивает их

class EffectSelectorAll : public IEffectSelector

{

Q\_OBJECT

public:

explicit EffectSelectorAll(QObject \*parent = 0);

QList<IEffect \*> selectEffect(QList<IEffect \*> effs);

int getType();

};

#endif // EFFECTSELECTOR\_H

**effectselector.cpp**

#include "effectselector.h"

IEffectSelector::IEffectSelector(QObject \*parent) : QObject(parent)

{

}

EffectSelectorRandom::EffectSelectorRandom(QObject \*parent) : IEffectSelector(parent)

{

}

QList<IEffect \*> EffectSelectorRandom::selectEffect(QList<IEffect \*> effs)

{

return {effs[Random::randInt(0, effs.size() - 1)]};

}

int EffectSelectorRandom::getType()

{

return 0;

}

EffectSelectorAll::EffectSelectorAll(QObject \*parent) : IEffectSelector(parent)

{

}

QList<IEffect \*> EffectSelectorAll::selectEffect(QList<IEffect \*> effs)

{

QList<IEffect \*> result, src = effs;

while (!src.empty())

result.push\_back(src.takeAt(Random::randInt(0, src.size() - 1)));

return result;

}

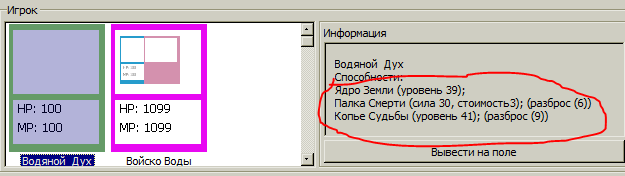
int EffectSelectorAll::getType()

{

return 1;

}

**Пример:**

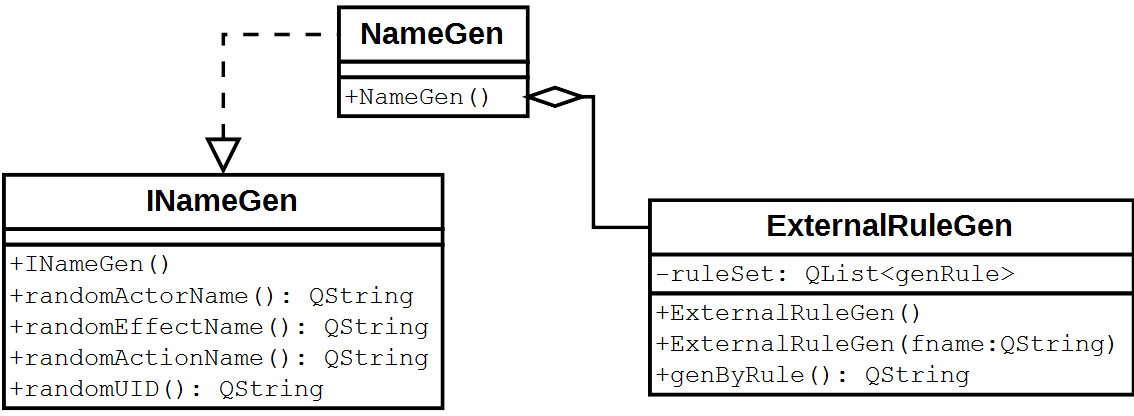


## Адаптер

**Назначение:**

Для генерации имён карт используется класс NameGen, который адаптирует для использования класс с аналогичным назначением ExternalRuleGen, использующий для генерации подгружаемые из файла правила.

**Диаграмма классов:**

**Исходный код:**

**namegen.h**

#ifndef NAMEGEN\_H

#define NAMEGEN\_H

#include "defs.h"

struct genRule

{

QString ruleName;

QStringList ruleData;

};

class ExternalRuleGen

{

private:

QList<genRule> ruleSet;

public:

ExternalRuleGen();

ExternalRuleGen(QString ruleSetName);

QString genByRule();

};

class NameGenRuleFactory

{

private:

static NameGenRuleFactory \*instance;

QMap< QString, ExternalRuleGen\* > ruleGenSets;

NameGenRuleFactory();

public:

static NameGenRuleFactory \* getInstance();

ExternalRuleGen \*getRuleGenerator(QString ruleSetName);

};

// генератор имен

class INameGen

{

public:

INameGen();

virtual QString randomActorName() = 0;

virtual QString randomEffectName() = 0;

virtual QString randomActionName() = 0;

virtual QString randomTeamName() = 0;

virtual QString randomUID() = 0;

~INameGen();

};

class NameGen : public INameGen

{

private:

ExternalRuleGen \*currentGenerator;

NameGenRuleFactory \*usedFactory;

public:

NameGen();

QString randomActorName();

QString randomEffectName();

QString randomActionName();

QString randomTeamName();

QString randomUID();

~NameGen();

};

#endif // NAMEGEN\_H

**namegen.cpp**

#include "namegen.h"

NameGenRuleFactory \* NameGenRuleFactory::instance = NULL;

NameGen::NameGen()

{

}

QString NameGen::randomActorName()

{

currentGenerator = NameGenRuleFactory::getInstance()->getRuleGenerator("actornames");

return currentGenerator->genByRule();

}

QString NameGen::randomActionName()

{

currentGenerator = NameGenRuleFactory::getInstance()->getRuleGenerator("actionnames");

return currentGenerator->genByRule();

}

QString NameGen::randomTeamName()

{

currentGenerator = NameGenRuleFactory::getInstance()->getRuleGenerator("teamnames");

return currentGenerator->genByRule();

}

QString NameGen::randomUID()

{

QString result;

// 32 16-ичные цифры

for (int i = 0; i < 32; i++)

{

result += QString::number(Random::randInt(0, 15), 16);

}

return result;

}

NameGen::~NameGen()

{

}

QString NameGen::randomEffectName()

{

currentGenerator = NameGenRuleFactory::getInstance()->getRuleGenerator("effectnames");

return currentGenerator->genByRule();

}

ExternalRuleGen::ExternalRuleGen()

{

}

ExternalRuleGen::ExternalRuleGen(QString ruleSetName)

{

int sizeInt = 0;

QFile f("data/" + ruleSetName + ".dat");

f.open(QFile::ReadOnly);

QList<genRule> rules;

QStringList data = QString(f.readAll()).replace("\r", "").split('\n');

sizeInt = data.size();

for (int i = 0; i < sizeInt; i++)

{

QStringList rule = QString(data.at(i)).split(" -> ");

genRule r;

r.ruleName = rule.at(0);

r.ruleData = QString(rule.at(1)).split('|');

rules.push\_back(r);

}

ruleSet = rules;

}

QString ExternalRuleGen::genByRule()

{

int sizeInt = 0;

bool ok = true;

QString result = "";

sizeInt = ruleSet.size();

for (int i = 0; i < sizeInt; i++)

{

if (ruleSet[i].ruleName == "<main>")

{

result = ruleSet[i].ruleData.at(Random::randInt(0, ruleSet[i].ruleData.size() - 1));

break;

}

}

do

{

ok = false;

sizeInt = ruleSet.size();

for (int i = 0; i < sizeInt; i++)

{

if (result.contains(ruleSet[i].ruleName))

{

ok = ok || true;

int len = ruleSet[i].ruleName.length();

int pos = result.indexOf(ruleSet[i].ruleName);

result = result.left(pos)

+ ruleSet[i].ruleData.at(Random::randInt(0, ruleSet[i].ruleData.size() - 1))

+ result.right(result.length() - len - pos);

}

}

} while (ok);

return result;

}

INameGen::INameGen()

{

}

INameGen::~INameGen()

{

}

ExternalRuleGen \* NameGenRuleFactory::getRuleGenerator(QString ruleSetName)

{

ExternalRuleGen \*erg;

if (ruleGenSets.keys().indexOf(ruleSetName) != -1)

{

erg = ruleGenSets[ruleSetName];

}

else

{

erg = new ExternalRuleGen(ruleSetName);

ruleGenSets[ruleSetName] = erg;

}

return erg;

}

NameGenRuleFactory::NameGenRuleFactory()

{

}

NameGenRuleFactory \*NameGenRuleFactory::getInstance()

{

if (instance == NULL)

instance = new NameGenRuleFactory;

return instance;

}

**Пример:**

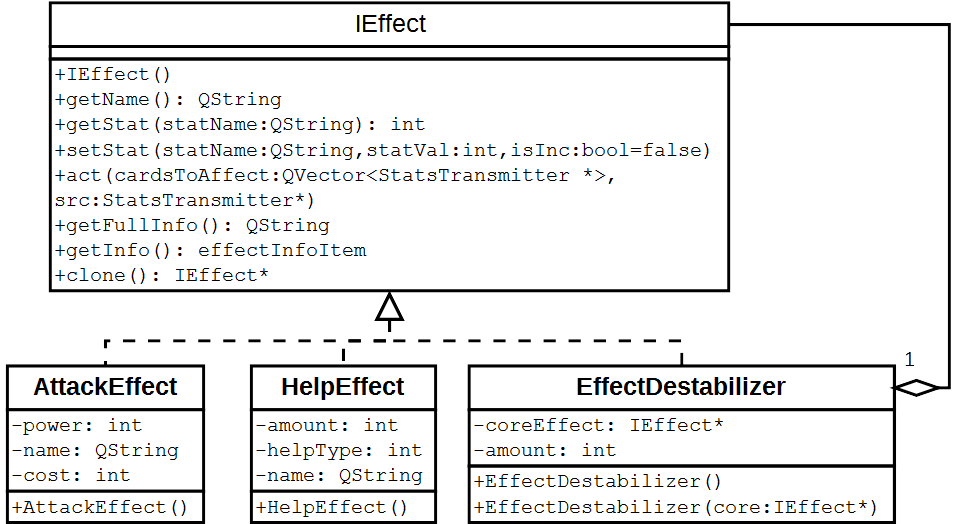
(заглушка)

## Декоратор

**Назначение:**

Для классов действий вводится дестабилизатор (EffectDestabilizer), добавляющий ещё несколько очков к определённой характеристике исходного действия.

**Диаграмма классов:**

**Исходный код:**

**effect.h**

// дестабилизатор

// после срабатывания основного действия докидывает еще несколько очков

class EffectDestabilizer : public IEffect

{

Q\_OBJECT

private:

IEffect \*coreEffect;

QString name;

int amount;

public:

EffectDestabilizer(QObject \*parent = 0);

EffectDestabilizer(IEffect \*core, QObject \*parent = 0);

QString getName();

void setName(QString effectName);

int getStat(QString statName);

void setStat(QString statName, int statVal);

void act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src);

...

};

**effect.cpp**

EffectDestabilizer::EffectDestabilizer(QObject \*parent):IEffect(parent)

{

}

EffectDestabilizer::EffectDestabilizer(IEffect \*core, QObject \*parent):EffectDestabilizer(parent)

{

coreEffect = core;

}

QString EffectDestabilizer::getName()

{

return (coreEffect->getName() + " (" + name + ")");

}

void EffectDestabilizer::setName(QString effectName)

{

name = effectName;

}

int EffectDestabilizer::getStat(QString statName)

{

if (statName == "distAmount")

return amount;

else

return coreEffect->getStat(statName);

}

void EffectDestabilizer::setStat(QString statName, int statVal)

{

if (statName == "distAmount")

amount = statVal;

else

coreEffect->setStat(statName, statVal);

}

void EffectDestabilizer::act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src)

{

coreEffect->act(cardsToAffect, src);

int effType = coreEffect->getStat("effectType");

switch (effType)

{

case 0:

{

for (int i = 0; i < cardsToAffect.size(); i++)

cardsToAffect[i]->setStat("health", -1 \* (Random::randInt(1, amount)), true);

break;

}

case 1:

{

for (int i = 0; i < cardsToAffect.size(); i++)

{

if (coreEffect->getStat("helpType") == 0)

cardsToAffect[i]->setStat("health", amount, true);

else

cardsToAffect[i]->setStat("energy", amount, true);

}

}

}

}

QString EffectDestabilizer::getFullInfo()

{

return coreEffect->getFullInfo() + " (" + name + " (" + QString::number(amount) + "))";

}

effectInfoItem EffectDestabilizer::getInfo()

{

effectInfoItem result = coreEffect->getInfo();

result.info = getFullInfo();

return result;

}

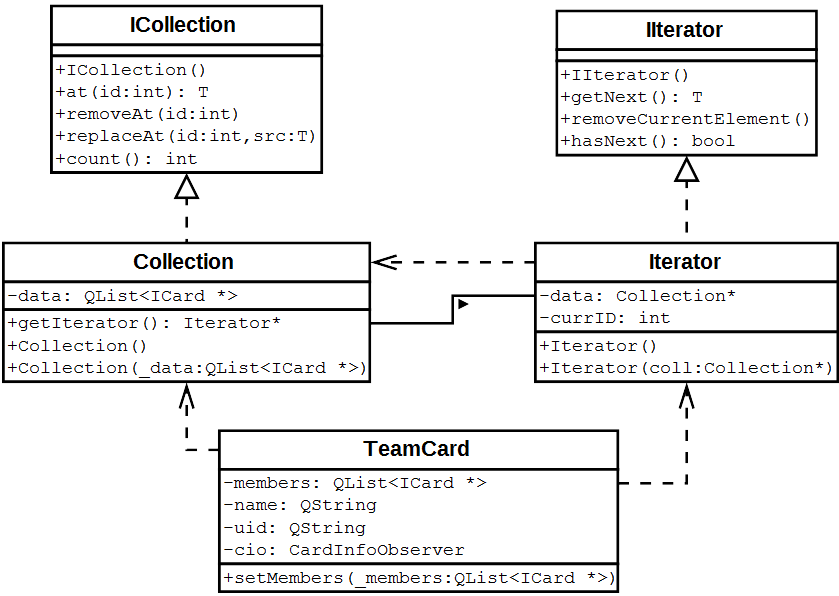
**Пример:**

## Итератор

**Назначение:**

Используется для последовательного выполнения операций с объектами (в данном примере — с картами).

**Диаграмма классов:**



**Исходный код:**

**util.h**

template <class T> class ICollection

{

public:

ICollection();

ICollection(QList<T> \_data);

virtual T at(int id) = 0;

virtual void removeAt(int id) = 0;

virtual void replaceAt(int id, T src) = 0;

virtual int count() = 0;

};

template <class T> class IIterator

{

public:

IIterator();

virtual T getNext() = 0;

virtual void removeCurrentElement() = 0;

virtual bool hasNext() = 0;

};

**util.cpp**

template<class T>

ICollection<T>::ICollection()

{

}

template<class T>

ICollection<T>::ICollection(QList<T> \_data)

{

}

template<class T>

IIterator<T>::IIterator()

{

}

**card.h**

class Iterator : public IIterator<ICard \*>

{

private:

ICollection<ICard \*> \*data;

int currID;

public:

Iterator();

Iterator(ICollection<ICard \*> \*coll);

ICard \* getNext();

void removeCurrentElement();

bool hasNext();

};

class Collection : public ICollection<ICard \*>

{

private:

QList<ICard \*> data;

public:

friend class Iterator;

Collection();

Collection(QList<ICard \*> \_data);

ICard \* at(int id);

void removeAt(int id);

void replaceAt(int id, ICard \*src);

int count();

Iterator \* getIterator();

};

class TeamCard : public ICard

{

Q\_OBJECT

private:

QString name, uid;

QColor bg;

QList<ICard \*> members, membersBeaten;

CardInfoObserver \*cio;

public:

TeamCard(QObject \*parent = 0);

...

QImage getDrawable(int height = 128);

QString getPseudoDrawable();

...

};

**card.cpp**

template <typename T>

ICollection<T>::ICollection()

{

}

Collection::Collection()

{

}

Collection::Collection(QList<ICard \*> \_data):Collection()

{

data = \_data;

}

ICard \* Collection::at(int id)

{

return data[id];

}

void Collection::removeAt(int id)

{

data.removeAt(id);

}

void Collection::replaceAt(int id, ICard \*src)

{

if (id < data.size())

data[id] = src;

}

int Collection::count()

{

return data.size();

}

Iterator \*Collection::getIterator()

{

return new Iterator(this);

}

template <typename T>

IIterator<T>::IIterator()

{

}

Iterator::Iterator()

{

}

Iterator::Iterator(ICollection<ICard \*> \*coll):Iterator()

{

data = coll;

currID = -1;

}

ICard \* Iterator::getNext()

{

return data->at(++currID);

}

void Iterator::removeCurrentElement()

{

data->removeAt(currID);

}

bool Iterator::hasNext()

{

return ((currID + 1) < data->count());

}

QImage TeamCard::getDrawable(int height)

{

QList<QImage> memberDrawables;

QImage img = QImage(height \* 0.75, height, QImage::Format\_RGBA8888);

QPainter pntr(&img);

pntr.fillRect(0, 0, height \* 0.75, height, QBrush(bg));

pntr.fillRect(height \* 0.05 \* 0.75, height \* 0.05, height \* 0.9 \* 0.75, height \* 0.5, QBrush(Qt::white));

Collection \*col = new Collection(members);

Iterator \*it = new Iterator(col);

int health = 0, energy = 0;

while (it->hasNext())

{

ICard \*member;

member = it->getNext();

memberDrawables.push\_back(member->getDrawable(height \* 0.5 \* 0.75));

health += member->getStat("health");

energy += member->getStat("energy");

}

double xOffset = (height \* 0.5 \* 0.75) / (double)(memberDrawables.size());

for (int i = 0; i < memberDrawables.size(); i++)

{

pntr.drawImage(height \* 0.1 \* 0.75 + xOffset \* i, height \* 0.1, memberDrawables[i]);

}

pntr.fillRect(height \* 0.05 \* 0.75, height \* 0.6, height \* 0.9 \* 0.75, height \* 0.35, QBrush(Qt::white));

QFont f = pntr.font();

f.setPixelSize(height \* 0.1);

pntr.setFont(f);

pntr.drawText(height \* 0.75 \* 0.1, height \* 0.7, "HP: " + QString::number(health));

pntr.drawText(height \* 0.75 \* 0.1, height \* 0.85, "MP: " + QString::number(energy));

pntr.end();

return img;

}

QString TeamCard::getPseudoDrawable()

{

QString result = "@{";

Collection \*col = new Collection(members);

Iterator \*it = new Iterator(col);

int health = 0, energy = 0;

while (it->hasNext())

{

ICard \*member;

member = it->getNext();

result += member->getPseudoDrawable();

result += ";";

health += member->getStat("health");

energy += member->getStat("energy");

}

result += "}(HP:" + QString::number(health) + ";MP:" + QString::number(energy) + ")";

return result;

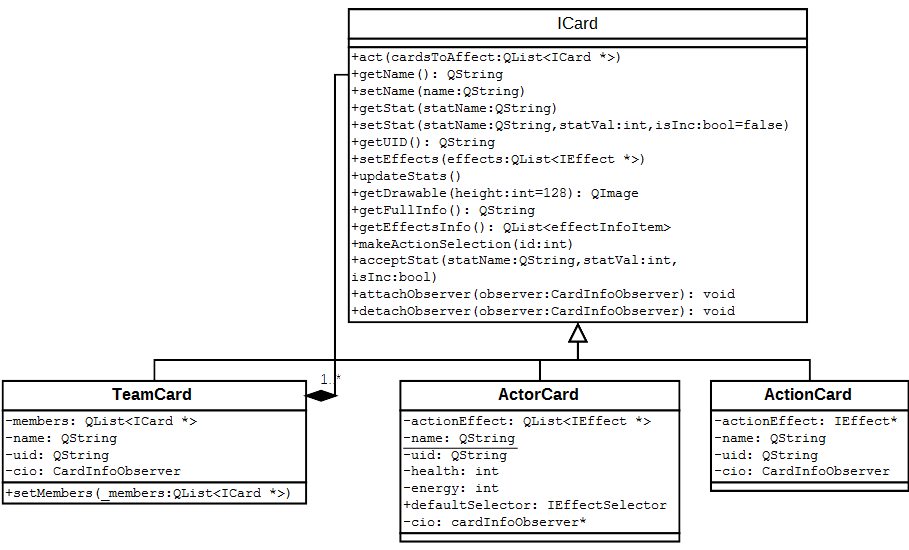
}

## Композит

**Назначение:**

Вводится новый тип карт — карта-команда, состоящая из нескольких карт разного типа. Атакует случайный член команды, принимает удар случайный член команды.

**Диаграмма классов:**



**Исходный код:**

**card.h**

class TeamCard : public ICard

{

Q\_OBJECT

private:

QString name, uid;

QColor bg;

QList<ICard \*> members, membersBeaten;

CardInfoObserver \*cio;

public:

TeamCard(QObject \*parent = 0);

void act(QList<ICard \*> cardsToAffect);

...

int getStat(QString statName);

void setStat(QString statName, int statVal, bool isInc = false);

...

void updateStats();

QImage getDrawable(int height = 128);

QString getPseudoDrawable();

QString getFullInfo();

QList<effectInfoItem> getEffectsInfo();

void setMembers(QList<ICard \*> \_members, bool beaten = false);

QList<ICard \*> getMembers(bool beaten = false);

...

public slots:

void makeActionSelection(int id);

void acceptStat(QString statName, int statVal, bool isInc = false);

void cardDispose();

};

**card.cpp**

TeamCard::TeamCard(QObject \*parent):ICard(parent)

{

cio = NULL;

NameGen \*ng = new NameGen();

bg = QColor(Random::randInt(0, 255),

Random::randInt(0, 255),

Random::randInt(0, 255));

uid = ng->randomUID();

delete ng;

}

void TeamCard::act(QList<ICard \*> cardsToAffect)

{

GameSession \*gs = GameSession::getInstance();

if (this->getStat("energy") == 0)

return;

ICard \*member;

do

{

member = members[Random::randInt(0, members.size() - 1)];

} while (member->getStat("energy") <= 0);

gs->msgRepeater(this->getName() + " выводит " + member->getName());

member->act(cardsToAffect);

updateStats();

}

int TeamCard::getStat(QString statName)

{

int result = 0;

Collection \*col = new Collection(members);

Iterator \*i = new Iterator(col);

while (i->hasNext())

{

result += i->getNext()->getStat(statName);

}

return result;

}

void TeamCard::setStat(QString statName, int statVal, bool isInc)

{

members[Random::randInt(0, members.size() - 1)]->setStat(statName, statVal, isInc);

updateStats();

}

void TeamCard::updateStats()

{

if (this->getStat("health") <= 0)

{

disposeMe();

if (cio)

cio->dispose(this);

}

else

{

if (cio)

cio->notify(listItem(this->getDrawable(), this->getPseudoDrawable(), this->name, this->uid));

}

}

QImage TeamCard::getDrawable(int height)

{

QList<QImage> memberDrawables;

QImage img = QImage(height \* 0.75, height, QImage::Format\_RGBA8888);

QPainter pntr(&img);

pntr.fillRect(0, 0, height \* 0.75, height, QBrush(bg));

pntr.fillRect(height \* 0.05 \* 0.75, height \* 0.05, height \* 0.9 \* 0.75, height \* 0.5, QBrush(Qt::white));

Collection \*col = new Collection(members);

Iterator \*it = new Iterator(col);

int health = 0, energy = 0;

while (it->hasNext())

{

ICard \*member;

member = it->getNext();

memberDrawables.push\_back(member->getDrawable(height \* 0.5 \* 0.75));

health += member->getStat("health");

energy += member->getStat("energy");

}

double xOffset = (height \* 0.5 \* 0.75) / (double)(memberDrawables.size());

for (int i = 0; i < memberDrawables.size(); i++)

{

pntr.drawImage(height \* 0.1 \* 0.75 + xOffset \* i, height \* 0.1, memberDrawables[i]);

}

pntr.fillRect(height \* 0.05 \* 0.75, height \* 0.6, height \* 0.9 \* 0.75, height \* 0.35, QBrush(Qt::white));

QFont f = pntr.font();

f.setPixelSize(height \* 0.1);

pntr.setFont(f);

pntr.drawText(height \* 0.75 \* 0.1, height \* 0.7, "HP: " + QString::number(health));

pntr.drawText(height \* 0.75 \* 0.1, height \* 0.85, "MP: " + QString::number(energy));

pntr.end();

return img;

}

QString TeamCard::getPseudoDrawable()

{

QString result = "@{";

Collection \*col = new Collection(members);

Iterator \*it = new Iterator(col);

int health = 0, energy = 0;

while (it->hasNext())

{

ICard \*member;

member = it->getNext();

result += member->getPseudoDrawable();

result += ";";

health += member->getStat("health");

energy += member->getStat("energy");

}

result += "}(HP:" + QString::number(health) + ";MP:" + QString::number(energy) + ")";

return result;

}

QString TeamCard::getFullInfo()

{

QString result;

result = name;

result += ":\n";

Collection \*col = new Collection(members);

Iterator \*it = col->getIterator();

while (it->hasNext()) {

result += "{";

result += it->getNext()->getFullInfo();

result += "}\n";

}

return result;

}

QList<effectInfoItem> TeamCard::getEffectsInfo()

{

effectInfoItem result;

result.canAffectEnemy = true;

result.canAffectPlayer = true;

result.effectType = 0;

result.info = "(смешанный)";

return {result};

}

void TeamCard::setMembers(QList<ICard \*> \_members, bool beaten)

{

if (!beaten)

{

members = \_members;

int sizeInt = members.size();

for (int i = 0; i < sizeInt; i++)

connect(members[i], SIGNAL(disposeMe()), this, SLOT(cardDispose()));

}

else

{

membersBeaten = \_members;

}

}

QList<ICard \*> TeamCard::getMembers(bool beaten)

{

return (beaten ? membersBeaten : members);

}

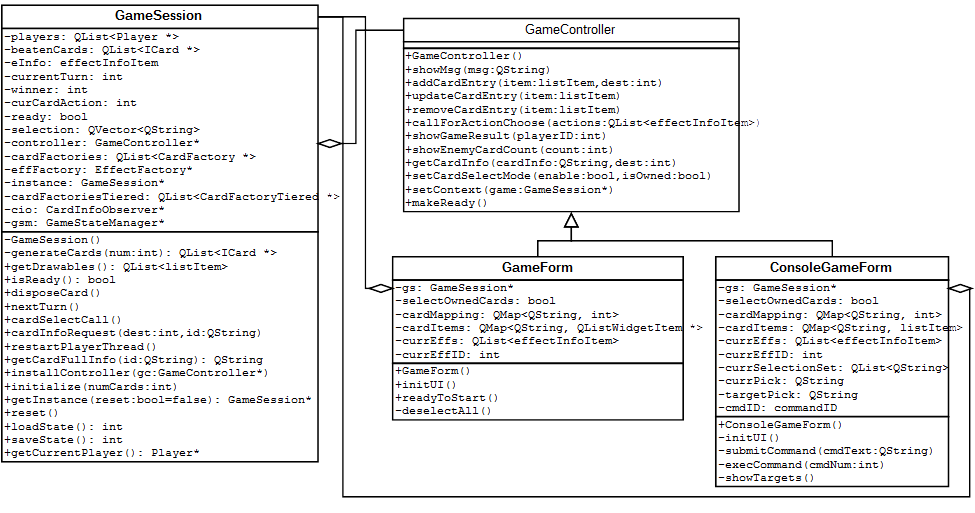
**Пример:**

## Мост

**Назначение:**

Позволяет разделить интерфейс пользователя и модель, что позволяет совершенствовать их независимо друг от друга.

**Диаграмма классов:**

****

**Исходный код:**

**gamecontroller.h**

#ifndef GAMECONTROLLER\_H

#define GAMECONTROLLER\_H

#include "defs.h"

#include "util.h"

class GameSession;

class GameController

{

public:

explicit GameController();

virtual void showMsg(QString msg) = 0;

virtual void addCardEntry(listItem item, int dest) = 0;

virtual void updateCardEntry(listItem item) = 0;

virtual void removeCardEntry(listItem item) = 0;

virtual void callForActionChoose(QList<effectInfoItem> actions) = 0;

virtual void showGameResult(int playerID) = 0;

virtual void showEnemyCardCount(int count) = 0;

virtual void getCardInfo(QString cardInfo, int dest) = 0;

virtual void setCardSelectMode(bool enable, bool isOwned) = 0;

virtual void setContext(GameSession \*game, bool initial = true) = 0;

virtual void makeReady() = 0;

virtual void clearAllEntries() = 0;

};

#endif // GAMECONTROLLER\_H

**gameform.h**

#ifndef GAMEFORM\_H

#define GAMEFORM\_H

#include <QWidget>

#include <QMap>

#include <QMessageBox>

#include <QInputDialog>

#include <QListWidget>

#include "gamesession.h"

#include "command.h"

namespace Ui {

class GameForm;

}

class GameForm : public QWidget, public GameController

{

Q\_OBJECT

public:

explicit GameForm(QWidget \*parent = 0);

void setContext(GameSession \*game, bool initial = true);

void addCardEntry(listItem item, int dest);

void updateCardEntry(listItem item);

void removeCardEntry(listItem item);

void getCardInfo(QString cardInfo, int dest);

void setCardSelectMode(bool enable, bool isOwned);

void callForActionChoose(QList<effectInfoItem> effs);

void showGameResult(int playerID);

void showEnemyCardCount(int count);

void showMsg(QString msg);

void makeReady();

void clearAllEntries();

~GameForm();

public slots:

void initUI();

void readyToStart();

signals:

void useCard(QString id);

void activateCard(QString id);

void sendCardSelection(QVector<QString> ids, effectInfoItem eff);

void cardInfoRequest(int dest, QString id);

private slots:

void on\_btnUsedCardAct\_clicked();

void on\_btnPlayerCardUse\_clicked();

void on\_btnTargetCardSelect\_clicked();

void deselectAll();

void on\_listEnemyCards\_itemSelectionChanged();

void on\_listPlayerUsedCards\_itemSelectionChanged();

void on\_listPlayerCards\_itemSelectionChanged();

private:

Ui::GameForm \*ui;

GameSession \*gs;

bool selectOwnedCards;

QMap<QString, int> cardMapping;

QMap<QString, QListWidgetItem \*> cardItems;

QList<effectInfoItem> currEffs;

int currEffID;

};

#endif // GAMEFORM\_H

**gameform.cpp**

#include "gameform.h"

#include "ui\_gameform.h"

GameForm::GameForm(QWidget \*parent) :

QWidget(parent),

ui(new Ui::GameForm)

{

ui->setupUi(this);

selectOwnedCards = false;

}

void GameForm::setContext(GameSession \*game, bool initial)

{

gs = game;

if (initial)

{

int numCards = QInputDialog::getInt(this, "Новая игра", "По сколько карт?", 20, 1, 100);

gs->initialize(numCards);

initUI();

}

}

GameForm::~GameForm()

{

delete ui;

}

void GameForm::addCardEntry(listItem item, int dest)

{

QListWidgetItem \*it = new QListWidgetItem(QIcon(QPixmap::fromImage(item.drawable)), item.name);

cardMapping[item.uid] = dest;

cardItems[item.uid] = it;

switch (dest)

{

case 0:

{

ui->listPlayerCards->addItem(it);

}

case 1:

{

ui->listPlayerUsedCards->addItem(it);

}

case 2:

{

ui->listEnemyCards->addItem(it);

}

}

}

void GameForm::updateCardEntry(listItem item)

{

QListWidgetItem \*it = cardItems[item.uid];

it->setIcon(QIcon(QPixmap::fromImage(item.drawable)));

}

void GameForm::removeCardEntry(listItem item)

{

QListWidgetItem \*wi = cardItems[item.uid];

int place = cardMapping[item.uid];

switch (place)

{

case 0:

{

ui->listPlayerCards->removeItemWidget(wi);

break;

}

case 1:

{

ui->listPlayerUsedCards->removeItemWidget(wi);

break;

}

case 2:

{

ui->listEnemyCards->removeItemWidget(wi);

break;

}

}

cardMapping.remove(item.uid);

cardItems.remove(item.uid);

delete wi;

}

void GameForm::initUI()

{

QList<listItem> litems = gs->getDrawables();

for (int i = 0; i < litems.size(); i++)

{

listItem item = litems[i];

QListWidgetItem \*it = new QListWidgetItem(QIcon(QPixmap::fromImage(item.drawable)), item.name);

cardItems[item.uid] = it;

cardMapping[item.uid] = 0;

ui->listPlayerCards->addItem(it);

}

readyToStart();

}

void GameForm::readyToStart()

{

gs->nextTurn();

}

void GameForm::getCardInfo(QString cardInfo, int dest)

{

switch (dest)

{

case 0:

ui->lblPlayerCard->setText(cardInfo);

break;

case 1:

ui->lblPlayerUsedCard->setText(cardInfo);

break;

case 2:

ui->lblEnemyCard->setText(cardInfo);

break;

}

}

void GameForm::setCardSelectMode(bool enable, bool isOwned)

{

selectOwnedCards = isOwned;

ui->btnTargetCardSelect->setEnabled(enable);

}

void GameForm::callForActionChoose(QList<effectInfoItem> effs)

{

currEffs = effs;

int actionID = 0;

if (currEffs.size() > 1)

{

actionID = -1;

QStringList items;

for (int i = 0; i < currEffs.size(); i++)

items.push\_back(currEffs[i].info);

items.push\_back("(по умолчанию)");

QString selected = QInputDialog::getItem(this, "Выбор действия", "Выберите:", items);

for (int i = 0; i < currEffs.size(); i++)

if (currEffs[i].info == selected)

{

actionID = i;

break;

}

}

currEffID = actionID;

ui->btnTargetCardSelect->setEnabled(true);

}

void GameForm::showGameResult(int playerID)

{

// подразумевается, что:

// игрок 1 - компьютер

// игрок 0 - ТЫ

QMessageBox(QMessageBox::Information,

"Конец игры",

playerID == 0 ?

"Вы выстояли этот раунд, и вы победили!"

: "Вам больше нечем биться, а посему вы проиграли.",

QMessageBox::Ok,

this).exec();

this->close();

gs->installController(NULL);

}

void GameForm::showEnemyCardCount(int count)

{

ui->lcdEnemyNumCards->display(count);

}

void GameForm::showMsg(QString msg)

{

ui->listLog->addItem(msg);

}

void GameForm::makeReady()

{

ui->btnPlayerCardUse->setEnabled(true);

ui->btnUsedCardAct->setEnabled(true);

}

void GameForm::clearAllEntries()

{

currEffs.clear();

currEffID = -1;

QList<QString> keys = cardMapping.keys();

ui->listPlayerCards->clear();

ui->listPlayerUsedCards->clear();

ui->listEnemyCards->clear();

while (!keys.empty())

{

QString key = keys.takeFirst();

cardMapping.remove(key);

//delete cardItems[key];

cardItems.remove(key);

}

ui->listLog->clear();

}

void GameForm::on\_btnUsedCardAct\_clicked()

{

QListWidgetItem \*item = ui->listPlayerUsedCards->selectedItems()[0];

QString uid = cardItems.key(item);

deselectAll();

ui->btnUsedCardAct->setEnabled(false);

ui->btnPlayerCardUse->setEnabled(false);

PlayerCardActivateCommand \*pcac = new PlayerCardActivateCommand();

pcac->setContext(gs);

pcac->setPick(uid);

pcac->execute();

delete pcac;

}

void GameForm::on\_btnPlayerCardUse\_clicked()

{

QListWidgetItem \*item = ui->listPlayerCards->selectedItems()[0];

QString uid = cardItems.key(item);

deselectAll();

PlayerCardUseCommand \*pcuc = new PlayerCardUseCommand();

pcuc->setContext(gs);

pcuc->setPick(uid);

pcuc->execute();

delete pcuc;

}

void GameForm::on\_btnTargetCardSelect\_clicked()

{

QList<QListWidgetItem \*> sel;

if (currEffID != -1)

{

if (currEffs[currEffID].canAffectPlayer)

{

sel.append(ui->listPlayerUsedCards->selectedItems());

}

if (currEffs[currEffID].canAffectEnemy)

{

sel.append(ui->listEnemyCards->selectedItems());

}

}

else

{

sel.append(ui->listPlayerUsedCards->selectedItems());

sel.append(ui->listEnemyCards->selectedItems());

}

if (!sel.empty())

{

QVector<QString> ids;

for (int i = 0; i < sel.size(); i++)

{

ids.push\_back(cardItems.key(sel[i]));

}

gs->setCurrentAction(currEffID);

if (currEffID != -1)

gs->sendCardSelection(ids, currEffs[currEffID]);

else

gs->sendCardSelection(ids, effectInfoItem("COMBO", true, true, 0));

ui->btnTargetCardSelect->setEnabled(false);

ui->btnUsedCardAct->setEnabled(true);

ui->btnPlayerCardUse->setEnabled(true);

deselectAll();

}

}

void GameForm::deselectAll()

{

ui->listPlayerCards->clearSelection();

ui->listPlayerUsedCards->clearSelection();

ui->listEnemyCards->clearSelection();

}

void GameForm::on\_listEnemyCards\_itemSelectionChanged()

{

if (!ui->listEnemyCards->selectedItems().empty())

this->getCardInfo(gs->getCardFullInfo(cardItems.key(ui->listEnemyCards->selectedItems()[0])), 2);

else

ui->lblEnemyCard->setText("(нет)");

}

void GameForm::on\_listPlayerUsedCards\_itemSelectionChanged()

{

if (!ui->listPlayerUsedCards->selectedItems().empty())

this->getCardInfo(gs->getCardFullInfo(cardItems.key(ui->listPlayerUsedCards->selectedItems()[0])), 1);

else

ui->lblPlayerUsedCard->setText("(нет)");

}

void GameForm::on\_listPlayerCards\_itemSelectionChanged()

{

if (!ui->listPlayerCards->selectedItems().empty())

this->getCardInfo(gs->getCardFullInfo(cardItems.key(ui->listPlayerCards->selectedItems()[0])), 0);

else

ui->lblPlayerCard->setText("(нет)");

}

**consolegameform.h**

#ifndef CONSOLEGAMEFORM\_H

#define CONSOLEGAMEFORM\_H

#include <QWidget>

#include "gamecontroller.h"

#include "gamesession.h"

#include "defs.h"

#include "command.h"

namespace Ui {

class ConsoleGameForm;

}

enum commandID {

CMD\_CARDNUM = 0,

CMD\_SELECT\_CMD,

CMD\_SHOW\_PLAYERCARDS,

CMD\_SHOW\_PLAYERUSEDCARDS,

CMD\_SHOW\_ENEMYCARDS,

CMD\_SELECT\_ACTION,

CMD\_SELECT\_CARDTOACTIVATE,

CMD\_SELECT\_CARDTOUSE,

CMD\_SELECT\_TARGET,

CMD\_ENDGAME,

CMD\_IDLE

};

class ConsoleGameForm : public QWidget, public GameController

{

Q\_OBJECT

public:

explicit ConsoleGameForm(QWidget \*parent = 0);

void setContext(GameSession \*game, bool initial = true);

void callForActionChoose(QList<effectInfoItem> actions);

void showGameResult(int playerID);

void showEnemyCardCount(int count);

void getCardInfo(QString cardInfo, int dest);

void setCardSelectMode(bool enable, bool isOwned);

void showMsg(QString msg);

void addCardEntry(listItem item, int dest);

void updateCardEntry(listItem item);

void removeCardEntry(listItem item);

void makeReady();

void clearAllEntries();

void submitCommand(QString cmdText);

void execCommand(int cmdNum);

void showTargets();

~ConsoleGameForm();

private:

Ui::ConsoleGameForm \*ui;

GameSession \*gs;

bool selectOwnedCards;

QMap<QString, int> cardMapping;

QMap<QString, listItem> cardItems;

QList<effectInfoItem> currEffs;

QList<QString> currSelectionSet;

QString currPick, targetPick;

int currEffID;

commandID cmdID;

void initUI();

public slots:

private slots:

void on\_btnSubmit\_clicked();

};

#endif // CONSOLEGAMEFORM\_H

**consolegameform.cpp**

#include "consolegameform.h"

#include "ui\_consolegameform.h"

ConsoleGameForm::ConsoleGameForm(QWidget \*parent) :

QWidget(parent),

ui(new Ui::ConsoleGameForm)

{

ui->setupUi(this);

selectOwnedCards = false;

currEffID = -1;

cmdID = CMD\_CARDNUM;

}

void ConsoleGameForm::setContext(GameSession \*game, bool initial)

{

gs = game;

if (initial)

showMsg("Сколько карт?");

}

void ConsoleGameForm::callForActionChoose(QList<effectInfoItem> actions)

{

currEffs = actions;

if (currEffs.size() > 1)

{

showMsg("Выберите действие:");

for (int i = 0; i < currEffs.size(); i++)

showMsg(QString::number(i + 1) + ") " + currEffs[i].info);

showMsg(QString::number(currEffs.size() + 1) + ") " + "(по умолчанию)");

cmdID = CMD\_SELECT\_ACTION;

}

else

{

currEffID = 0;

cmdID = CMD\_SELECT\_TARGET;

showTargets();

}

}

void ConsoleGameForm::showGameResult(int playerID)

{

showMsg("Игра подошла к концу...");

showMsg(playerID == 0 ?

"Вы выстояли этот раунд, и вы победили!"

: "Вам больше нечем биться, а посему вы проиграли.");

cmdID = CMD\_ENDGAME;

gs->installController(NULL);

}

void ConsoleGameForm::showEnemyCardCount(int count)

{

ui->textOutputBox->append("(у противника осталось " + QString::number(count) + " карт)");

}

void ConsoleGameForm::getCardInfo(QString cardInfo, int dest)

{

ui->textOutputBox->append(cardInfo);

}

void ConsoleGameForm::setCardSelectMode(bool enable, bool isOwned)

{

return;

}

void ConsoleGameForm::submitCommand(QString cmdText)

{

if (cmdID == CMD\_CARDNUM)

{

bool ok = false;

int cardNum = cmdText.toInt(&ok);

if (!ok || (cardNum < 1) || (cardNum > CARD\_POOL\_SIZE / 2))

{

showMsg("Неверный ввод, попробуйте еще раз!");

return;

}

gs->initialize(cardNum);

initUI();

gs->nextTurn();

}

else if (cmdID == CMD\_SELECT\_CMD)

{

bool ok = false;

int cmdNum = cmdText.toInt(&ok);

if (!ok || cmdNum > 5 || cmdNum < 1)

{

showMsg("Неверный ввод, попробуйте еще раз!");

return;

}

execCommand(cmdNum);

}

else if (cmdID == CMD\_SELECT\_CARDTOUSE)

{

bool ok = false;

int cardNum = cmdText.toInt(&ok);

if (!ok || (cardNum < 1) || (cardNum > currSelectionSet.size()))

{

showMsg("Неверный ввод, попробуйте еще раз!");

return;

}

currPick = currSelectionSet[cardNum - 1];

cmdID = CMD\_IDLE;

PlayerCardUseCommand \*pcuc = new PlayerCardUseCommand();

pcuc->setContext(gs);

pcuc->setPick(currPick);

pcuc->execute();

delete pcuc;

currPick = "";

}

else if (cmdID == CMD\_SELECT\_CARDTOACTIVATE)

{

bool ok = false;

int cardNum = cmdText.toInt(&ok);

if (!ok || (cardNum < 1) || (cardNum > currSelectionSet.size()))

{

showMsg("Неверный ввод, попробуйте еще раз!");

return;

}

currPick = currSelectionSet[cardNum - 1];

PlayerCardActivateCommand \*pcac = new PlayerCardActivateCommand();

pcac->setContext(gs);

pcac->setPick(currPick);

pcac->execute();

delete pcac;

}

else if (cmdID == CMD\_SELECT\_ACTION)

{

bool ok = false;

int effNum = cmdText.toInt(&ok);

if (!ok || (effNum < 1) || (effNum > (currEffs.size() + 1)))

{

showMsg("Неверный ввод, попробуйте еще раз!");

return;

}

currEffID = (effNum == (currEffs.size() + 1)) ? -1 : (effNum - 1);

cmdID = CMD\_SELECT\_TARGET;

showTargets();

}

else if (cmdID == CMD\_SELECT\_TARGET)

{

bool ok = false;

int cardNum = cmdText.toInt(&ok);

if (!ok || (cardNum < 1) || (cardNum > currSelectionSet.size()))

{

showMsg("Неверный ввод, попробуйте еще раз!");

return;

}

targetPick = currSelectionSet[cardNum - 1];

cmdID = CMD\_IDLE;

gs->setCurrentAction(currEffID);

gs->sendCardSelection({targetPick},

((currEffID == -1) ? effectInfoItem("COMBO", true, true, 0) : currEffs[currEffID]));

currPick = "";

targetPick = "";

}

else if (cmdID == CMD\_ENDGAME)

{

this->close();

}

}

void ConsoleGameForm::execCommand(int cmdNum)

{

int sizeInt = 0;

if (cmdNum == 1)

{

QList<QString> ids, keys = cardMapping.keys();

sizeInt = keys.size();

for (int i = 0; i < sizeInt; i++)

if (cardMapping[keys[i]] == 0)

ids.push\_back(keys[i]);

sizeInt = ids.size();

if (!ids.empty())

for (int i = 0; i < sizeInt; i++)

{

ui->textOutputBox->append(QString::number(i + 1) + " " + cardItems[ids[i]].pseudoDrawable + " " + gs->getCardFullInfo(ids[i]));

}

else

ui->textOutputBox->append("(нет)");

makeReady();

}

else if (cmdNum == 2)

{

QList<QString> ids, keys = cardMapping.keys();

sizeInt = keys.size();

for (int i = 0; i < sizeInt; i++)

if (cardMapping[keys[i]] == 1)

ids.push\_back(keys[i]);

sizeInt = ids.size();

if (!ids.empty())

for (int i = 0; i < sizeInt; i++)

{

ui->textOutputBox->append(QString::number(i + 1) + " " + cardItems[ids[i]].pseudoDrawable + " " + gs->getCardFullInfo(ids[i]));

}

else

ui->textOutputBox->append("(нет)");

makeReady();

}

else if (cmdNum == 3)

{

QList<QString> ids, keys = cardMapping.keys();

sizeInt = keys.size();

for (int i = 0; i < sizeInt; i++)

if (cardMapping[keys[i]] == 2)

ids.push\_back(keys[i]);

sizeInt = ids.size();

if (!ids.empty())

for (int i = 0; i < sizeInt; i++)

{

ui->textOutputBox->append(QString::number(i + 1) + " " + cardItems[ids[i]].pseudoDrawable + " " + gs->getCardFullInfo(ids[i]));

}

else

ui->textOutputBox->append("(нет)");

makeReady();

}

else if (cmdNum == 4)

{

QList<QString> ids, keys = cardMapping.keys();

sizeInt = keys.size();

for (int i = 0; i < sizeInt; i++)

if (cardMapping[keys[i]] == 0)

ids.push\_back(keys[i]);

sizeInt = ids.size();

if (!ids.empty())

for (int i = 0; i < sizeInt; i++)

{

ui->textOutputBox->append(QString::number(i + 1) + " " + cardItems[ids[i]].pseudoDrawable + " " + gs->getCardFullInfo(ids[i]));

}

else

{

ui->textOutputBox->append("Нечего выводить!");

makeReady();

return;

}

currSelectionSet = ids;

ui->textOutputBox->append("Выбор?");

cmdID = CMD\_SELECT\_CARDTOUSE;

}

else if (cmdNum == 5)

{

QList<QString> ids, keys = cardMapping.keys();

sizeInt = keys.size();

for (int i = 0; i < sizeInt; i++)

if (cardMapping[keys[i]] == 1)

ids.push\_back(keys[i]);

sizeInt = ids.size();

if (!ids.empty())

for (int i = 0; i < sizeInt; i++)

{

ui->textOutputBox->append(QString::number(i + 1) + " " + cardItems[ids[i]].pseudoDrawable + " " + gs->getCardFullInfo(ids[i]));

}

else

{

ui->textOutputBox->append("Нечего выбирать!");

makeReady();

return;

}

currSelectionSet = ids;

ui->textOutputBox->append("Выбор?");

cmdID = CMD\_SELECT\_CARDTOACTIVATE;

}

}

void ConsoleGameForm::showTargets()

{

QList<QString> sel;

int sizeInt = 0;

showMsg("Выберите цель:");

if (currEffID != -1)

{

sizeInt = cardMapping.size();

QList<QString> keys = cardMapping.keys();

for (int i = 0; i < sizeInt; i++)

{

if (currEffs[currEffID].canAffectPlayer && cardMapping[keys[i]] == 1)

{

ui->textOutputBox->append(QString::number(sel.size() + 1) + " (свое) " + cardItems[keys[i]].pseudoDrawable + " " + gs->getCardFullInfo(keys[i]));

sel.append(keys[i]);

}

if (currEffs[currEffID].canAffectEnemy && cardMapping[keys[i]] == 2)

{

ui->textOutputBox->append(QString::number(sel.size() + 1) + " " + cardItems[keys[i]].pseudoDrawable + " " + gs->getCardFullInfo(keys[i]));

sel.append(keys[i]);

}

}

}

else

{

sizeInt = cardMapping.size();

QList<QString> keys = cardMapping.keys();

for (int i = 0; i < sizeInt; i++)

{

if (cardMapping[keys[i]] == 1)

{

ui->textOutputBox->append(QString::number(sel.size() + 1) + " (свое) " + cardItems[keys[i]].pseudoDrawable + " " + gs->getCardFullInfo(keys[i]));

sel.append(keys[i]);

}

else if (cardMapping[keys[i]] == 2)

{

ui->textOutputBox->append(QString::number(sel.size() + 1) + " " + cardItems[keys[i]].pseudoDrawable + " " + gs->getCardFullInfo(keys[i]));

sel.append(keys[i]);

}

}

}

currSelectionSet = sel;

}

ConsoleGameForm::~ConsoleGameForm()

{

delete ui;

}

void ConsoleGameForm::initUI()

{

QList<listItem> litems = gs->getDrawables();

for (int i = 0; i < litems.size(); i++)

{

listItem item = litems[i];

cardItems[item.uid] = item;

cardMapping[item.uid] = 0;

}

}

void ConsoleGameForm::showMsg(QString msg)

{

ui->textOutputBox->append(msg);

}

void ConsoleGameForm::addCardEntry(listItem item, int dest)

{

cardMapping[item.uid] = dest;

cardItems[item.uid] = item;

}

void ConsoleGameForm::updateCardEntry(listItem item)

{

cardItems[item.uid] = item;

}

void ConsoleGameForm::removeCardEntry(listItem item)

{

cardMapping.remove(item.uid);

cardItems.remove(item.uid);

}

void ConsoleGameForm::makeReady()

{

cmdID = CMD\_SELECT\_CMD;

showMsg(QString("Что будем делать?\n")

+ QString("1) Посмотреть свои карты\n")

+ QString("2) Посмотреть свои карты на поле\n")

+ QString("3) Посмотреть карты противника на поле\n")

+ QString("4) Вывести карту на поле\n")

+ QString("5) Задействовать карту\n"));

}

void ConsoleGameForm::clearAllEntries()

{

currEffs.clear();

currSelectionSet.clear();

currEffID = -1;

currPick = "";

QList<QString> keys = cardMapping.keys();

while (!keys.empty())

{

QString key = keys.takeFirst();

cardMapping.remove(key);

cardItems.remove(key);

}

ui->textOutputBox->clear();

}

void ConsoleGameForm::on\_btnSubmit\_clicked()

{

QString text = ui->textInputBox->text();

ui->textInputBox->clear();

submitCommand(text);

}

**gamesession.h**

// класс сеанса игры

class GameSession : public QObject

{

Q\_OBJECT

private:

static GameSession \*instance;

// интерфейс для взаимодействия

GameController \*controller;

...

public:

static GameSession \* getInstance(bool reset = false);

void reset();

void installController(GameController \*gc);

...

public slots:

...

// повторители для работы со списками

// и отображением данных

// используют контроллер

void listItemRemoveRepeater(listItem item);

void listItemAddRepeater(listItem item, int dest);

void listItemUpdateRepeater(listItem item);

void selectionModeRepeater(bool enable, bool owned);

void cardCountRepeater(int count);

void effectsInfoRepeater(QList<effectInfoItem> effs);

...

void msgRepeater(QString msg);

...

};

**gamesession.cpp**

...

}

void GameSession::installController(GameController \*gc)

{

controller = gc;

}

void GameSession::msgRepeater(QString msg)

{

controller->showMsg(msg);

}

void GameSession::listItemRemoveRepeater(listItem item)

{

controller->removeCardEntry(item);

}

void GameSession::listItemAddRepeater(listItem item, int dest)

{

controller->addCardEntry(item, dest);

}

void GameSession::listItemUpdateRepeater(listItem item)

{

controller->updateCardEntry(item);

}

void GameSession::selectionModeRepeater(bool enable, bool owned)

{

controller->setCardSelectMode(enable, owned);

}

void GameSession::cardCountRepeater(int count)

{

controller->showEnemyCardCount(count);

}

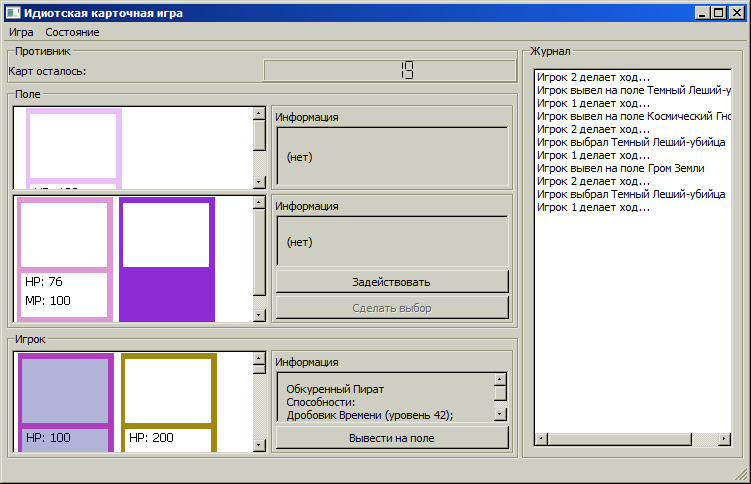
void GameSession::effectsInfoRepeater(QList<effectInfoItem> effs)

{

controller->callForActionChoose(effs);

}

**Пример:**

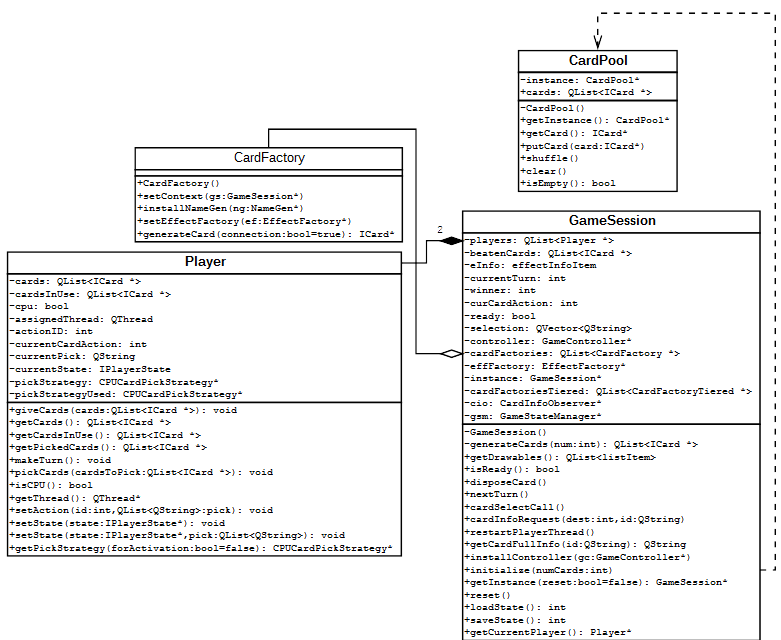


## Фасад

**Назначение:**

Позволяет управлять объектами модели, обращаясь к одному классу.

**Диаграмма классов:**

**Исходный код:**

**gamesession.h**

// класс сеанса игры

class GameSession : public QObject

{

Q\_OBJECT

private:

static GameSession \*instance;

// интерфейс для взаимодействия

GameController \*controller;

// игроки

QList<Player \*> players;

// битые карты

QList<ICard \*> beatenCards;

// выборка карт по UID

QVector<QString> selection;

// инфо по тому или иному действию

effectInfoItem eInfo;

// кто сейчас ходит

int currentTurn;

// победитель

int winner;

// текущее действие карты

int curCardAction;

// готовность

bool ready;

InfoTransmitter \*infoTrans;

// фабрики карт

QList<CardFactory \*> cardFactories;

// с уровнем

QList<CardFactoryTiered \*> cardFactoriesTiered;

// фабрика эффектов

EffectFactory \*effFactory;

// наблюдатель информации о состоянии карт

CardInfoObserver \*cio;

// менеджер состояний для загрузки и сохранения

GameStateManager \*gsm;

// посредник между игроками

PlayerMediator \*pMed;

// приватный конструктор

GameSession(QObject \*parent = 0);

// генерация карт

QList<ICard \*> generateCards(int num, int ctype = -1, bool connections = true);

public:

static GameSession \* getInstance(bool reset = false);

void reset();

void installController(GameController \*gc);

void initialize(int numCards);

...

};

**gamesession.cpp**

GameSession::GameSession(QObject \*parent):QObject(parent)

{

int sizeInt = 0;

infoTrans = new InfoTransmitter(this);

cio = new CardInfoObserver(this);

gsm = new GameStateManager(this);

pMed = new PlayerMediator();

NameGen \*ng = new NameGen();

effFactory = new EffectFactory(this);

effFactory->installNamegen(ng);

effFactory->setContext(this);

// создаем фабрики карт

cardFactories.push\_back(new ActorCardFactory(this));

cardFactories.push\_back(new ActionCardFactory(this));

TeamCardFactory \*tcf = new TeamCardFactory(this);

tcf->setSubfactories(cardFactories[0], cardFactories[1]);

cardFactories.push\_back(tcf);

sizeInt = cardFactories.size();

for (int i = 0; i < sizeInt; i++)

{

cardFactories[i]->setContext(this);

cardFactories[i]->installNamegen(ng);

cardFactories[i]->setEffectFactory(effFactory);

}

CardFactoryTiered \*cft2 = new CardFactoryTier2(this);

cft2->setContext(this);

cft2->installNamegen(ng);

cft2->setEffectFactory(effFactory);

cardFactoriesTiered.push\_back(cft2);

CardFactoryTiered \*cft3 = new CardFactoryTier3(this);

cft3->setContext(this);

cft3->installNamegen(ng);

cft3->setEffectFactory(effFactory);

cardFactoriesTiered.push\_back(cft3);

// генерируем карты и забиваем в пул

CardPool \*cp = CardPool::getInstance();

QList<ICard \*> pregenCards = generateCards(CARD\_POOL\_SIZE);

while (!pregenCards.empty())

cp->putCard(pregenCards.takeFirst());

cp->shuffle();

connect(this, SIGNAL(sendMsg(QString)), infoTrans, SLOT(receiveMsg(QString)));

connect(infoTrans, SIGNAL(sendMsg(QString)), this, SLOT(msgRepeater(QString)));

ready = false;

currentTurn = -1;

winner = -1;

}

void GameSession::initialize(int numCards)

{

CardPool \*cp = CardPool::getInstance();

// настраиваем игроков

// пока что здесь их всего два

// так что пусть игроком 0 будет человек

players.push\_back(new Player(false));

players.push\_back(new Player(true));

foreach (Player \*p, players) {

connect(p, SIGNAL(turnEnd()), this, SLOT(nextTurn()));

connect(p, SIGNAL(removeID(listItem)), this, SLOT(listItemRemoveRepeater(listItem)));

connect(p, SIGNAL(addID(listItem,int)), this, SLOT(listItemAddRepeater(listItem,int)));

connect(p, SIGNAL(restart()), this, SLOT(restartPlayerThread()));

connect(p, SIGNAL(showPickedCardInfo(QList<effectInfoItem>)), this, SLOT(effectsInfoRepeater(QList<effectInfoItem>)));

connect(p, SIGNAL(sendActionSelect(int)), this, SLOT(setCurrentAction(int)));

connect(p, SIGNAL(showPickedCardInfoSingle(effectInfoItem)), this, SLOT(setCurEffInfo(effectInfoItem)));

connect(p, SIGNAL(sendMsg(QString)), infoTrans, SLOT(receiveMsg(QString)));

p->attachMediator(pMed);

}

connect(players[1], SIGNAL(showCardCount(int)), this, SLOT(cardCountRepeater(int)));

for (int k = 0; k < players.size(); k++)

{

// раздадим карты

QList<ICard \*> cards;

for (int i = 0; i < numCards; i++)

cards.push\_back(cp->getCard());

players[k]->giveCards(cards);

for (int i = 0; i < cards.size(); i++)

{

cards[i]->attachObserver(cio);

connect(cards[i], SIGNAL(disposeMe()), players[k], SLOT(disposeCard()));

connect(cards[i], SIGNAL(callActionSelect()), players[k], SLOT(callActionSelect()));

connect(players[k], SIGNAL(sendActionSelect(int)), cards[i], SLOT(makeActionSelection(int)));

}

}

ready = true;

}

QList<ICard \*> GameSession::generateCards(int num, int ctype, bool connections)

{

QList<ICard \*> cards;

for (int i = 0; i < num; i++)

{

int cardType = 0;

if (ctype == -1)

cardType = Random::randIntWeighted({25, 15, 5});

else

cardType = ctype;

switch (Random::randIntWeighted({25, 5}))

{

case 0:

cards.push\_back(cardFactories[cardType]->generateCard(connections));

break;

case 1:

CardFactoryTiered \*cft = cardFactoriesTiered[Random::randInt(0, cardFactoriesTiered.size() - 1)];

switch (cardType)

{

case 0:

cards.push\_back(cft->generateActorCard());

break;

case 1:

cards.push\_back(cft->generateActionCard());

break;

case 2:

cards.push\_back(cft->generateTeamCard());

break;

}

break;

}

}

return cards;

}

void GameSession::reset()

{

int sizeInt = 0;

CardPool \*cp = CardPool::getInstance();

QList<ICard \*> usedCards;

sizeInt = players.size();

for (int i = 0; i < sizeInt; i++)

{

usedCards.append(players[i]->getCards());

usedCards.append(players[i]->getCardsInUse());

}

while (!players.empty())

{

Player \*plr = players.takeFirst();

plr->detachMediator(pMed);

delete plr;

}

while (!usedCards.empty())

cp->putCard(usedCards.takeFirst());

while (!beatenCards.empty())

cp->putCard(beatenCards.takeFirst());

cp->shuffle();

ready = false;

currentTurn = -1;

winner = -1;

}

...

## Информационный эксперт

**Назначение:**

Используется для сбора информации, её обработки и перенаправления.

В качестве информационного эксперта выступает главный класс GameSession, собирающий и обрабатывающий информацию об игроках, картах, действиях и т. д.

**Диаграмма классов:**

**Исходный код:**

**gamesession.h**

// класс сеанса игры

class GameSession : public QObject

{

Q\_OBJECT

private:

static GameSession \*instance;

// интерфейс для взаимодействия

GameController \*controller;

// игроки

QList<Player \*> players;

// битые карты

QList<ICard \*> beatenCards;

// выборка карт по UID

QVector<QString> selection;

// инфо по тому или иному действию

effectInfoItem eInfo;

// кто сейчас ходит

int currentTurn;

// победитель

int winner;

// текущее действие карты

int curCardAction;

// готовность

bool ready;

InfoTransmitter \*infoTrans;

// фабрики карт

QList<CardFactory \*> cardFactories;

// с уровнем

QList<CardFactoryTiered \*> cardFactoriesTiered;

// фабрика эффектов

EffectFactory \*effFactory;

// наблюдатель информации о состоянии карт

CardInfoObserver \*cio;

// менеджер состояний для загрузки и сохранения

GameStateManager \*gsm;

// посредник между игроками

PlayerMediator \*pMed;

// приватный конструктор

GameSession(QObject \*parent = 0);

// генерация карт

QList<ICard \*> generateCards(int num, int ctype = -1, bool connections = true);

public:

...

QList<listItem> getDrawables();

// готовность

bool isReady();

InfoTransmitter \* getInfoTrans();

QString getCardFullInfo(QString uid);

...

Player \* getCurrentPlayer();

Player \* getNextPlayer();

...

};

**player.h**

// класс игрока

class Player : public QObject

{

Q\_OBJECT

private:

// карты

QList<ICard \*> cards;

// карты на поле

QList<ICard \*> cardsInUse;

// показатель того, компьютер это, или человек

bool cpu;

// состояние

int actionID;

QList<QString> actionPick;

QVector<QString> selection;

// выбранные для действия карты

QList<ICard \*> pickedCards;

int currentCardAction;

QString currentPick;

// тред

QThread \*assignedThread;

// текущее состояние

IPlayerState \*currentState;

//

bool needToRestartThread;

effectInfoItem eInfo;

CPUCardPickStrategy \*pickStrategy, \*pickStrategyUsed;

PlayerMediator \*mediator;

public:

...

// получение карт

QList <ICard \*> getCards();

QList <ICard \*> getCardsInUse();

QList <ICard \*> getPickedCards();

...

bool isCPU();

...

QList<QString> getActionPick();

QString getCurrentPick();

QVector<QString> getSelection();

...

int getCardAction();

QString getCardFullInfo(QString uid);

...

CPUCardPickStrategy \* getStrategy(bool forActivation = false);

...

QList <ICard \*> lookForNextPlayerUsedCards();

void tellSelectedEffectInfo(effectInfoItem e);

effectInfoItem getSelectedEffectInfo();

...

};

## Приспособленец

**Назначение:**

В данном примере расширяется класс ExternalRuleGen, чтобы за каждым разом не генерировать новый класс, когда понадобится сгенерировать определённое имя.

**Диаграмма классов:**

**Исходный код:**

(см. Адаптер)

## Фабричный метод

**Назначение:**

Используется для генерации объектов различного типа (в данной предметной области – карты и их действия).

**Диаграмма классов:**



**Исходный код:**

**cardfactory.h**

// фабрика карт

class CardFactory : public QObject

{

Q\_OBJECT

public:

explicit CardFactory(QObject \*parent = 0);

virtual void setContext(GameSession \*gs) = 0;

virtual void installNamegen(NameGen \*ng) = 0;

virtual void setEffectFactory(EffectFactory \*ef) = 0;

virtual ICard \* generateCard(bool connect = true) = 0;

private:

signals:

public slots:

};

// фабрика карт-персонажей

class ActorCardFactory : public CardFactory

{

Q\_OBJECT

public:

ActorCardFactory(QObject \*parent = 0);

void setContext(GameSession \*gs);

void installNamegen(NameGen \*ng);

void setEffectFactory(EffectFactory \*ef);

ICard \* generateCard(bool connection = true);

private:

GameSession \*context;

NameGen \*nameGen;

EffectFactory \*effFactory;

};

// фабрика карт-действий

class ActionCardFactory : public CardFactory

{

Q\_OBJECT

public:

ActionCardFactory(QObject \*parent = 0);

void setContext(GameSession \*gs);

void installNamegen(NameGen \*ng);

void setEffectFactory(EffectFactory \*ef);

ICard \* generateCard(bool connection = true);

private:

GameSession \*context;

NameGen \*nameGen;

EffectFactory \*effFactory;

};

// фабрика карт-команд

class TeamCardFactory : public CardFactory

{

Q\_OBJECT

public:

TeamCardFactory(QObject \*parent = 0);

void setContext(GameSession \*gs);

void installNamegen(NameGen \*ng);

void setEffectFactory(EffectFactory \*ef);

void setSubfactories(CardFactory \*forActor, CardFactory \*forAction);

ICard \* generateCard(bool connection = true);

private:

GameSession \*context;

NameGen \*nameGen;

EffectFactory \*effFactory;

CardFactory \*actorFact, \*actionFact;

};

**cardfactory.cpp**

CardFactory::CardFactory(QObject \*parent) : QObject(parent)

{

}

ActorCardFactory::ActorCardFactory(QObject \*parent) : CardFactory(parent)

{

}

void ActorCardFactory::setContext(GameSession \*gs)

{

context = gs;

}

void ActorCardFactory::installNamegen(NameGen \*ng)

{

nameGen = ng;

}

void ActorCardFactory::setEffectFactory(EffectFactory \*ef)

{

effFactory = ef;

}

ICard \*ActorCardFactory::generateCard(bool connection)

{

ActorCard \*card = new ActorCard();

card->setName(nameGen->randomActorName());

QList<IEffect \*> effects;

card->setStat("health", 100, false);

card->setStat("energy", 100, false);

int numEffects = Random::randInt(1, 3);

for (int j = 0; j < numEffects; j++)

{

IEffect \*e = effFactory->generateEffect(connection);

effects.push\_back(e);

}

IEffectSelector \*selector;

switch (Random::randIntWeighted({5, 5}))

{

case 0:

selector = new EffectSelectorRandom();

break;

case 1:

selector = new EffectSelectorAll();

break;

}

card->setEffectSelector(selector);

card->setEffects(effects);

if (connection)

{

//connect(card, SIGNAL(updateDrawable(listItem)), context, SLOT(listItemUpdateRepeater(listItem)));

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

}

return card;

}

ActionCardFactory::ActionCardFactory(QObject \*parent) : CardFactory(parent)

{

}

void ActionCardFactory::setContext(GameSession \*gs)

{

context = gs;

}

void ActionCardFactory::installNamegen(NameGen \*ng)

{

nameGen = ng;

}

void ActionCardFactory::setEffectFactory(EffectFactory \*ef)

{

effFactory = ef;

}

ICard \*ActionCardFactory::generateCard(bool connection)

{

ActionCard \*card = new ActionCard();

card->setName(nameGen->randomActionName());

IEffect \*e = effFactory->generateEffect(connection);

card->setEffects({e});

if (connection)

{

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

}

return card;

}

TeamCardFactory::TeamCardFactory(QObject \*parent) : CardFactory(parent)

{

}

void TeamCardFactory::setContext(GameSession \*gs)

{

context = gs;

}

void TeamCardFactory::installNamegen(NameGen \*ng)

{

nameGen = ng;

}

void TeamCardFactory::setEffectFactory(EffectFactory \*ef)

{

effFactory = ef;

}

void TeamCardFactory::setSubfactories(CardFactory \*forActor, CardFactory \*forAction)

{

actorFact = forActor;

actionFact = forAction;

}

ICard \*TeamCardFactory::generateCard(bool connection)

{

int numMembers = Random::randInt(2, 5);

TeamCard \*card = new TeamCard();

card->setName(nameGen->randomTeamName());

QList<ICard \*> members;

for (int i = 0; i < numMembers; i++)

{

ICard \*member;

switch (Random::randIntWeighted({25, 15, 5}))

{

case 0:

member = actorFact->generateCard(false);

break;

case 1:

member = actionFact->generateCard(false);

break;

case 2:

member = this->generateCard(false);

break;

}

members.push\_back(member);

}

card->setMembers(members);

if (connection)

{

//connect(card, SIGNAL(updateDrawable(listItem)), context, SLOT(listItemUpdateRepeater(listItem)));

}

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

return card;

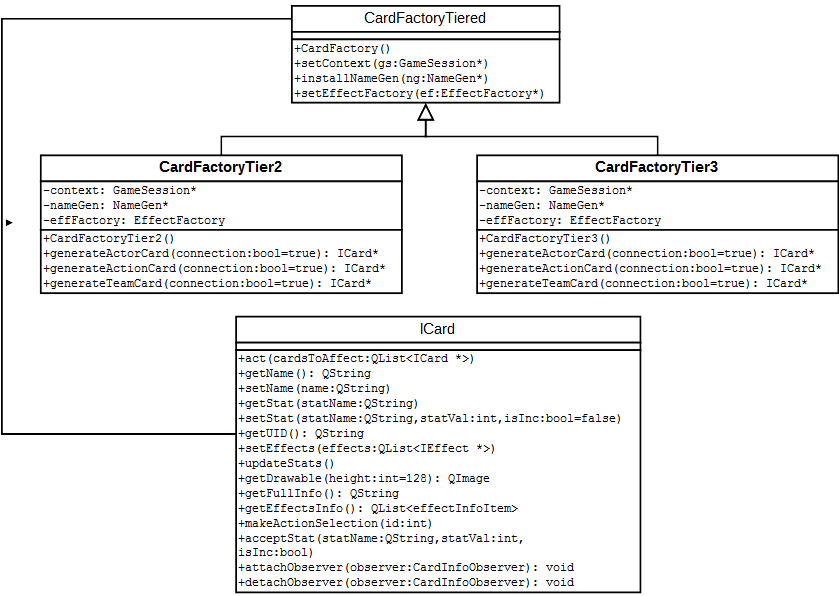
}

## Абстрактная фабрика

**Назначение:**

Используется для генерации объектов одного типа, но с разными характеристиками, как правило, идущих в одном наборе, взаимосвязывающем их.

**Диаграмма классов:**

**Исходный код:**

**cardfactory.h**

class CardFactoryTiered : public QObject

{

Q\_OBJECT

public:

CardFactoryTiered(QObject \*parent = 0);

virtual void setContext(GameSession \*gs) = 0;

virtual void installNamegen(NameGen \*ng) = 0;

virtual void setEffectFactory(EffectFactory \*ef) = 0;

virtual ICard \* generateActorCard(bool connection = true) = 0;

virtual ICard \* generateActionCard(bool connection = true) = 0;

virtual ICard \* generateTeamCard(bool connection = true) = 0;

};

class CardFactoryTier2 : public CardFactoryTiered

{

Q\_OBJECT

public:

CardFactoryTier2(QObject \*parent = 0);

void setContext(GameSession \*gs);

void installNamegen(NameGen \*ng);

void setEffectFactory(EffectFactory \*ef);

ICard \* generateActorCard(bool connection = true);

ICard \* generateActionCard(bool connection = true);

ICard \* generateTeamCard(bool connection = true);

private:

GameSession \*context;

NameGen \*nameGen;

EffectFactory \*effFactory;

};

class CardFactoryTier3 : public CardFactoryTiered

{

Q\_OBJECT

public:

CardFactoryTier3(QObject \*parent = 0);

void setContext(GameSession \*gs);

void installNamegen(NameGen \*ng);

void setEffectFactory(EffectFactory \*ef);

ICard \* generateActorCard(bool connection = true);

ICard \* generateActionCard(bool connection = true);

ICard \* generateTeamCard(bool connection = true);

private:

GameSession \*context;

NameGen \*nameGen;

EffectFactory \*effFactory;

};

**cardfactory.cpp**

CardFactoryTiered::CardFactoryTiered(QObject \*parent) : QObject(parent)

{

}

CardFactoryTier2::CardFactoryTier2(QObject \*parent) : CardFactoryTiered(parent)

{

}

void CardFactoryTier2::setEffectFactory(EffectFactory \*ef)

{

effFactory = ef;

}

ICard \*CardFactoryTier2::generateActorCard(bool connection)

{

ActorCard \*card = new ActorCard();

card->setName(nameGen->randomActorName());

QList<IEffect \*> effects;

card->setStat("health", 200, false);

card->setStat("energy", 200, false);

card->setStat("healthInit", 200, false);

card->setStat("energyInit", 200, false);

int numEffects = Random::randInt(1, 3);

for (int j = 0; j < numEffects; j++)

{

IEffect \*e = effFactory->generateEffect(connection);

e->setStat("power", e->getStat("power") \* 2);

e->setStat("amount", e->getStat("amount") \* 2);

effects.push\_back(e);

}

IEffectSelector \*selector;

switch (Random::randIntWeighted({5, 5}))

{

case 0:

selector = new EffectSelectorRandom();

break;

case 1:

selector = new EffectSelectorAll();

break;

}

card->setEffectSelector(selector);

card->setEffects(effects);

if (connection)

{

//connect(card, SIGNAL(updateDrawable(listItem)), context, SLOT(listItemUpdateRepeater(listItem)));

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

}

return card;

}

ICard \*CardFactoryTier2::generateActionCard(bool connection)

{

ActionCard \*card = new ActionCard();

card->setName(nameGen->randomActionName());

IEffect \*e = effFactory->generateEffect(connection);

card->setEffects({e});

e->setStat("power", e->getStat("power") \* 2);

e->setStat("amount", e->getStat("amount") \* 2);

if (connection)

{

//connect(card, SIGNAL(updateDrawable(listItem)), context, SLOT(listItemUpdateRepeater(listItem)));

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

}

return card;

}

ICard \*CardFactoryTier2::generateTeamCard(bool connection)

{

int numMembers = Random::randInt(5, 8);

TeamCard \*card = new TeamCard();

card->setName(nameGen->randomTeamName());

QList<ICard \*> members;

for (int i = 0; i < numMembers; i++)

{

ICard \*member;

switch (Random::randIntWeighted({25, 15, 5}))

{

case 0:

member = this->generateActorCard(false);

break;

case 1:

member = this->generateActionCard(false);

break;

case 2:

member = this->generateTeamCard(false);

break;

}

members.push\_back(member);

}

card->setMembers(members);

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

return card;

}

void CardFactoryTier2::installNamegen(NameGen \*ng)

{

nameGen = ng;

}

void CardFactoryTier2::setContext(GameSession \*gs)

{

context = gs;

}

CardFactoryTier3::CardFactoryTier3(QObject \*parent) : CardFactoryTiered(parent)

{

}

void CardFactoryTier3::setEffectFactory(EffectFactory \*ef)

{

effFactory = ef;

}

ICard \*CardFactoryTier3::generateActorCard(bool connection)

{

ActorCard \*card = new ActorCard();

card->setName(nameGen->randomActorName());

QList<IEffect \*> effects;

card->setStat("health", 300, false);

card->setStat("energy", 300, false);

card->setStat("healthInit", 300, false);

card->setStat("energyInit", 300, false);

int numEffects = Random::randInt(1, 3);

for (int j = 0; j < numEffects; j++)

{

IEffect \*e = effFactory->generateEffect(connection);

e->setStat("power", e->getStat("power") \* 3);

e->setStat("amount", e->getStat("amount") \* 3);

effects.push\_back(e);

}

IEffectSelector \*selector;

switch (Random::randIntWeighted({5, 5}))

{

case 0:

selector = new EffectSelectorRandom();

break;

case 1:

selector = new EffectSelectorAll();

break;

}

card->setEffectSelector(selector);

card->setEffects(effects);

if (connection)

{

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

}

return card;

}

ICard \*CardFactoryTier3::generateActionCard(bool connection)

{

ActionCard \*card = new ActionCard();

card->setName(nameGen->randomActionName());

IEffect \*e = effFactory->generateEffect(connection);

card->setEffects({e});

e->setStat("power", e->getStat("power") \* 3);

e->setStat("amount", e->getStat("amount") \* 3);

if (connection)

{

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

}

return card;

}

ICard \*CardFactoryTier3::generateTeamCard(bool connection)

{

int numMembers = Random::randInt(5, 8);

TeamCard \*card = new TeamCard();

card->setName(nameGen->randomTeamName());

QList<ICard \*> members;

for (int i = 0; i < numMembers; i++)

{

ICard \*member;

switch (Random::randIntWeighted({25, 15, 5}))

{

case 0:

member = this->generateActorCard(false);

break;

case 1:

member = this->generateActionCard(false);

break;

case 2:

member = this->generateTeamCard(false);

break;

}

members.push\_back(member);

}

card->setMembers(members);

connect(card, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

return card;

}

void CardFactoryTier3::installNamegen(NameGen \*ng)

{

nameGen = ng;

}

void CardFactoryTier3::setContext(GameSession \*gs)

{

context = gs;

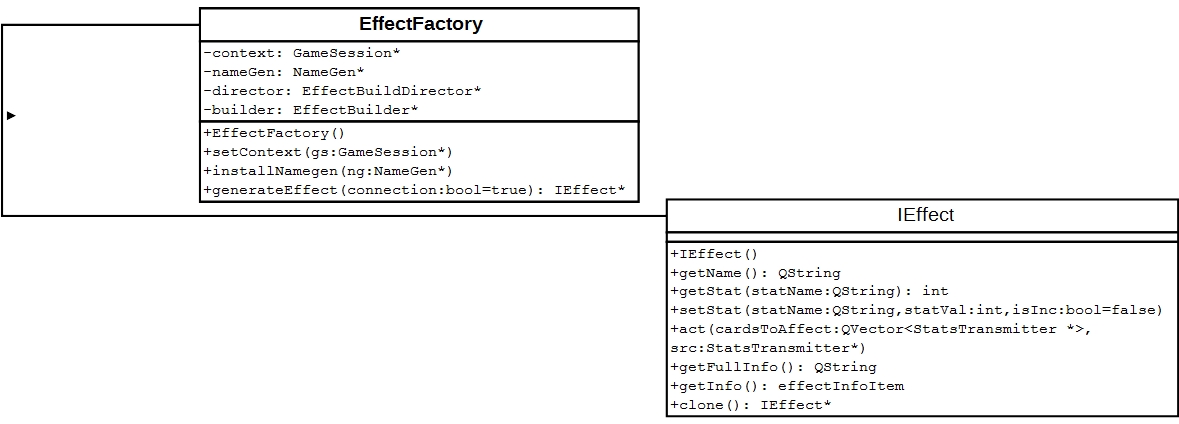
}

## Фабрика

**Назначение:**

Используется для генерации объектов одного конкретного типа с заданными характеристиками

**Диаграмма классов:**

****

**Исходный код:**

**effectfactory.h**

class EffectFactory : public QObject

{

Q\_OBJECT

public:

explicit EffectFactory(QObject \*parent = 0);

void setContext(GameSession \*gs);

void installNamegen(NameGen \*ng);

IEffect \* generateEffect(bool connection = true);

private:

GameSession \*context;

NameGen \*nameGen;

EffectBuilder \*builder;

EffectBuildDirector \*director;

};

**effectfactory.cpp**

EffectFactory::EffectFactory(QObject \*parent) : QObject(parent)

{

builder = new EffectBuilder(this);

director = new EffectBuildDirector(builder, this);

}

void EffectFactory::setContext(GameSession \*gs)

{

context = gs;

}

void EffectFactory::installNamegen(NameGen \*ng)

{

nameGen = ng;

}

IEffect \*EffectFactory::generateEffect(bool connection)

{

IEffect \*e;

director->construct();

e = builder->getEffect();

if (connection)

connect(e, SIGNAL(callCardSelect()), context, SLOT(cardSelectCall()));

connect(e, SIGNAL(sendMsg(QString)), context->getInfoTrans(), SLOT(receiveMsg(QString)));

return e;

}

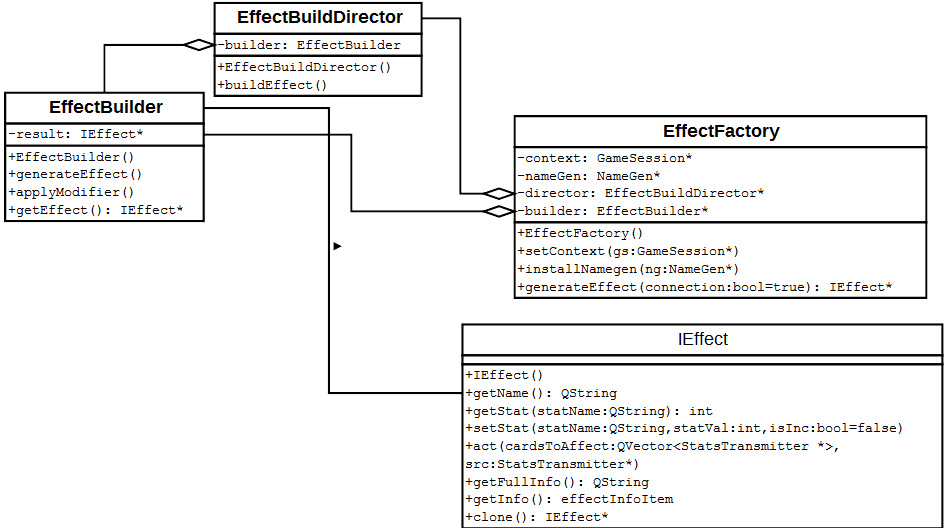
## Строитель

**Назначение:**

Позволяет генерировать объекты определённого типа покомпонентно, управляя процессом сборки извне.

В данном примере Строитель скомбинирован с паттерном Фабрика, будучи встроенным в него.

**Диаграмма классов:**

**Исходный код:**

**effectfactory.h**

class EffectBuilder : public QObject

{

Q\_OBJECT

public:

EffectBuilder(QObject \*parent = 0);

void generateEffect(int eType);

void applyModifier();

IEffect \*getEffect();

private:

IEffect \*result;

};

class EffectBuildDirector : public QObject

{

Q\_OBJECT

public:

EffectBuildDirector(QObject \*parent = 0);

EffectBuildDirector(EffectBuilder \*bldr, QObject \*parent = 0);

void construct();

private:

EffectBuilder \*builder;

};

class EffectFactory : public QObject

{

Q\_OBJECT

public:

explicit EffectFactory(QObject \*parent = 0);

void setContext(GameSession \*gs);

void installNamegen(NameGen \*ng);

IEffect \* generateEffect(bool connection = true);

private:

...

EffectBuilder \*builder;

EffectBuildDirector \*director;

...

};

**effectfactory.cpp**

EffectBuilder::EffectBuilder(QObject \*parent)

{

}

void EffectBuilder::generateEffect(int eType)

{

NameGen \*nameGen = new NameGen();

switch (eType)

{

case 0:

{

result = new AttackEffect();

result->setName(nameGen->randomEffectName());

result->setStat("cost", Random::randInt(1, 5));

result->setStat("power", Random::randInt(10, 40));

break;

}

case 1:

{

result = new HelpEffect();

result->setName(nameGen->randomEffectName());

result->setStat("amount", Random::randInt(10, 50));

result->setStat("limit", 1);

result->setStat("helptype", Random::randIntWeighted({10, 5}));

break;

}

}

}

void EffectBuilder::applyModifier()

{

EffectDestabilizer \*destab = new EffectDestabilizer(result);

destab->setName("разброс");

destab->setStat("distAmount", Random::randInt(5, 10));

result = destab;

}

IEffect \*EffectBuilder::getEffect()

{

return result;

}

EffectBuildDirector::EffectBuildDirector(QObject \*parent) : QObject(parent)

{

}

EffectBuildDirector::EffectBuildDirector(EffectBuilder \*bldr, QObject \*parent) : EffectBuildDirector(parent)

{

builder = bldr;

}

void EffectBuildDirector::construct()

{

builder->generateEffect(Random::randIntWeighted({75, 25}));

if (Random::probability(0.5))

builder->applyModifier();

}

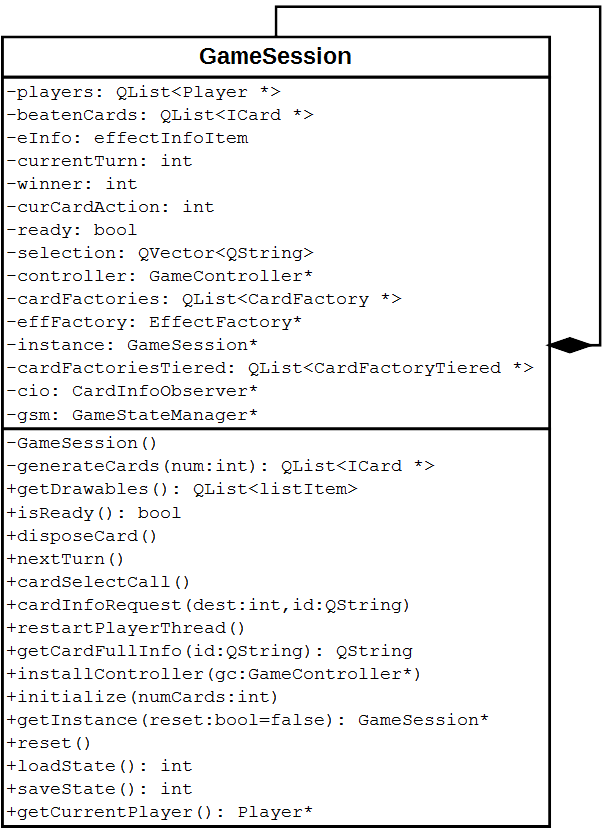
## Синглтон

**Назначение:**

Этот паттерн следует применять при проектировании классов, объект которого должен быть в единственном экземпляре за всё время работы.

В данном случае такой паттерн реализован в классе GameSession.

**Диаграмма классов:**

**Исходный код:**

**gamesession.h**

// класс сеанса игры

class GameSession : public QObject

{

Q\_OBJECT

private:

static GameSession \*instance;

// интерфейс для взаимодействия

...

// приватный конструктор

GameSession(QObject \*parent = 0);

...

public:

static GameSession \* getInstance(bool reset = false);

void reset();

...

};

**gamesession.cpp**

GameSession \* GameSession::instance = NULL;

GameSession \*GameSession::getInstance(bool reset)

{

if (instance == NULL)

instance = new GameSession();

if (reset)

instance->reset();

return instance;

}

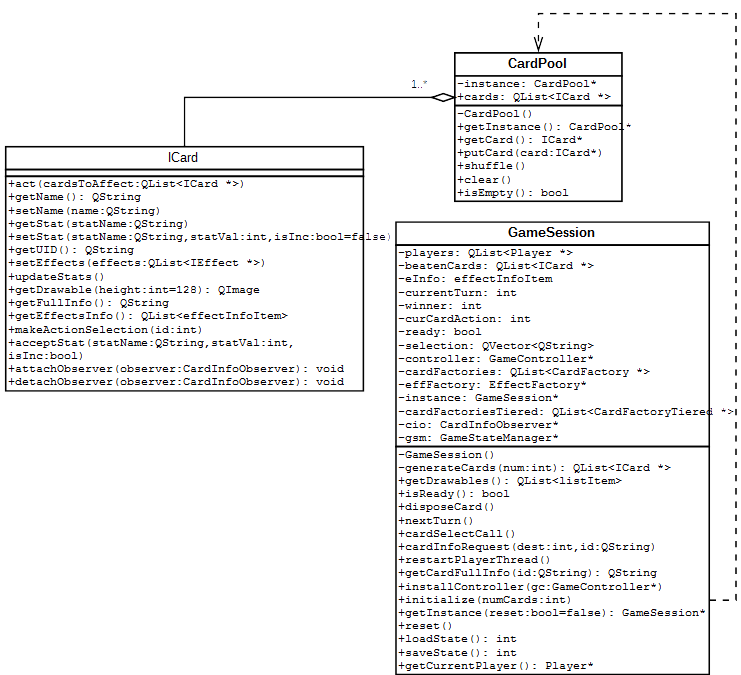
## Пул

**Назначение:**

Требуется для хранения многоразово используемых объектов ограниченного количества.

К примеру, в данной предметной области карты не генерируются каждый раунд, а выдаются из пула в начале раунда и кладутся туда обратно по окончании, предварительно сбросив состояние.

**Диаграмма классов:**

**Исходный код:**

**cardpool.h**

class CardPool : public QObject

{

Q\_OBJECT

public:

static CardPool \* getInstance();

// добавление в пул и извлечение из него

void putCard(ICard \*card);

ICard \* getCard();

// перетасовка "колоды"

void shuffle();

void clear();

bool isEmpty();

private:

static CardPool \*instance;

QList<ICard \*> cards;

explicit CardPool(QObject \*parent = 0);

signals:

public slots:

};

**cardpool.cpp**

CardPool\* CardPool::instance = NULL;

CardPool \*CardPool::getInstance()

{

if (instance == NULL)

instance = new CardPool();

return instance;

}

void CardPool::putCard(ICard \*card)

{

ICard \*cardToPut = card;

card->reset();

cards.push\_back(cardToPut);

}

ICard \*CardPool::getCard()

{

if (cards.empty())

return NULL;

ICard \*cardToGet = cards.back();

cards.pop\_back();

return cardToGet;

}

void CardPool::shuffle()

{

QList<ICard \*> srcCards = cards;

cards.clear();

while (!srcCards.empty())

cards.push\_back(srcCards.takeAt(Random::randInt(0, srcCards.size() - 1)));

}

void CardPool::clear()

{

while (!cards.empty())

delete cards.takeFirst();

}

bool CardPool::isEmpty()

{

return (cards.size() == 0);

}

CardPool::CardPool(QObject \*parent) : QObject(parent)

{

}

**gamesession.h**

// класс сеанса игры

class GameSession : public QObject

{

Q\_OBJECT

private:

...

public:

...

void reset();

void initialize(int numCards);

...

};

**gamesesison.cpp**

void GameSession::initialize(int numCards)

{

CardPool \*cp = CardPool::getInstance();

// настраиваем игроков

// пока что здесь их всего два

// так что пусть игроком 0 будет человек

players.push\_back(new Player(false));

players.push\_back(new Player(true));

foreach (Player \*p, players) {

connect(p, SIGNAL(turnEnd()), this, SLOT(nextTurn()));

connect(p, SIGNAL(removeID(listItem)), this, SLOT(listItemRemoveRepeater(listItem)));

connect(p, SIGNAL(addID(listItem,int)), this, SLOT(listItemAddRepeater(listItem,int)));

connect(p, SIGNAL(restart()), this, SLOT(restartPlayerThread()));

connect(p, SIGNAL(showPickedCardInfo(QList<effectInfoItem>)), this, SLOT(effectsInfoRepeater(QList<effectInfoItem>)));

connect(p, SIGNAL(sendActionSelect(int)), this, SLOT(setCurrentAction(int)));

connect(p, SIGNAL(showPickedCardInfoSingle(effectInfoItem)), this, SLOT(setCurEffInfo(effectInfoItem)));

connect(p, SIGNAL(sendMsg(QString)), infoTrans, SLOT(receiveMsg(QString)));

p->attachMediator(pMed);

}

connect(players[1], SIGNAL(showCardCount(int)), this, SLOT(cardCountRepeater(int)));

for (int k = 0; k < players.size(); k++)

{

// раздадим карты

QList<ICard \*> cards;

for (int i = 0; i < numCards; i++)

cards.push\_back(cp->getCard());

players[k]->giveCards(cards);

for (int i = 0; i < cards.size(); i++)

{

cards[i]->attachObserver(cio);

connect(cards[i], SIGNAL(disposeMe()), players[k], SLOT(disposeCard()));

connect(cards[i], SIGNAL(callActionSelect()), players[k], SLOT(callActionSelect()));

connect(players[k], SIGNAL(sendActionSelect(int)), cards[i], SLOT(makeActionSelection(int)));

}

}

ready = true;

}

void GameSession::reset()

{

int sizeInt = 0;

CardPool \*cp = CardPool::getInstance();

QList<ICard \*> usedCards;

sizeInt = players.size();

for (int i = 0; i < sizeInt; i++)

{

usedCards.append(players[i]->getCards());

usedCards.append(players[i]->getCardsInUse());

}

while (!players.empty())

{

Player \*plr = players.takeFirst();

plr->detachMediator(pMed);

delete plr;

}

while (!usedCards.empty())

cp->putCard(usedCards.takeFirst());

while (!beatenCards.empty())

cp->putCard(beatenCards.takeFirst());

cp->shuffle();

ready = false;

currentTurn = -1;

winner = -1;

}

**card.h**

class ICard : public QObject

{

Q\_OBJECT

public:

...

virtual void reset() = 0;

...

};

## Наблюдатель

**Назначение:**

Позволяет контролировать состояние объекта, к которому прикреплён объект наблюдателя и управлять им извне в определённый момент.

К примеру, мы можем получать информацию от объекта о его изменённом состоянии или удалять его по его же запросу.

**Диаграмма классов:**

****

**Исходный код:**

**card.h**

class ICard : public QObject

{

Q\_OBJECT

public:

...

// обновление статистики

virtual void updateStats() = 0;

...

virtual void attachObserver(CardInfoObserver \*observer) = 0;

virtual void detachObserver(CardInfoObserver \*observer) = 0;

...

};

class CardInfoObserver : public QObject

{

Q\_OBJECT

private:

public:

CardInfoObserver(QObject \*parent = 0);

void notify(listItem litem);

void dispose(ICard \*me);

};

// карта-персонаж

// пока позволяют "здоровье" и "энергия", может действовать сколько угодно раз

// может иметь несколько действий

class ActorCard : public ICard

{

Q\_OBJECT

private:

...

CardInfoObserver \*cio;

public:

...

void updateStats();

...

void attachObserver(CardInfoObserver \*observer);

void detachObserver(CardInfoObserver \*observer);

...

};

// карта-действие

// имеет только одно действие

// действует только один раз, после чего уходит с поля

// зато неуязвима

class ActionCard : public ICard

{

Q\_OBJECT

private:

...

CardInfoObserver \*cio;

public:

...

void updateStats();

...

void attachObserver(CardInfoObserver \*observer);

void detachObserver(CardInfoObserver \*observer);

...

};

// карта-команда

// объединяет в себе несколько карт

class TeamCard : public ICard

{

Q\_OBJECT

private:

...

CardInfoObserver \*cio;

public:

...

void updateStats();

...

void attachObserver(CardInfoObserver \*observer);

void detachObserver(CardInfoObserver \*observer);

...

};

**card.cpp**

void ActorCard::updateStats()

{

if (health <= 0)

{

disposeMe();

if (cio)

cio->dispose(this);

}

else

{

if (cio)

cio->notify(listItem(this->getDrawable(), this->getPseudoDrawable(), this->name, this->uid));

}

}

…

void ActorCard::attachObserver(CardInfoObserver \*observer)

{

cio = observer;

}

void ActorCard::detachObserver(CardInfoObserver \*observer)

{

if (cio == observer)

cio = NULL;

}

…

void ActionCard::attachObserver(CardInfoObserver \*observer)

{

cio = observer;

}

void ActionCard::detachObserver(CardInfoObserver \*observer)

{

if (cio == observer)

cio = NULL;

}

…

void TeamCard::updateStats()

{

if (this->getStat("health") <= 0)

{

disposeMe();

if (cio)

cio->dispose(this);

}

else

{

if (cio)

cio->notify(listItem(this->getDrawable(), this->getPseudoDrawable(), this->name, this->uid));

}

}

…

void TeamCard::attachObserver(CardInfoObserver \*observer)

{

cio = observer;

}

void TeamCard::detachObserver(CardInfoObserver \*observer)

{

if (cio == observer)

cio = NULL;

}

…

CardInfoObserver::CardInfoObserver(QObject \*parent) : QObject(parent)

{

}

void CardInfoObserver::notify(listItem litem)

{

GameSession::getInstance()->listItemUpdateRepeater(litem);

}

void CardInfoObserver::dispose(ICard \*me)

{

GameSession::getInstance()->disposeCard(me);

}

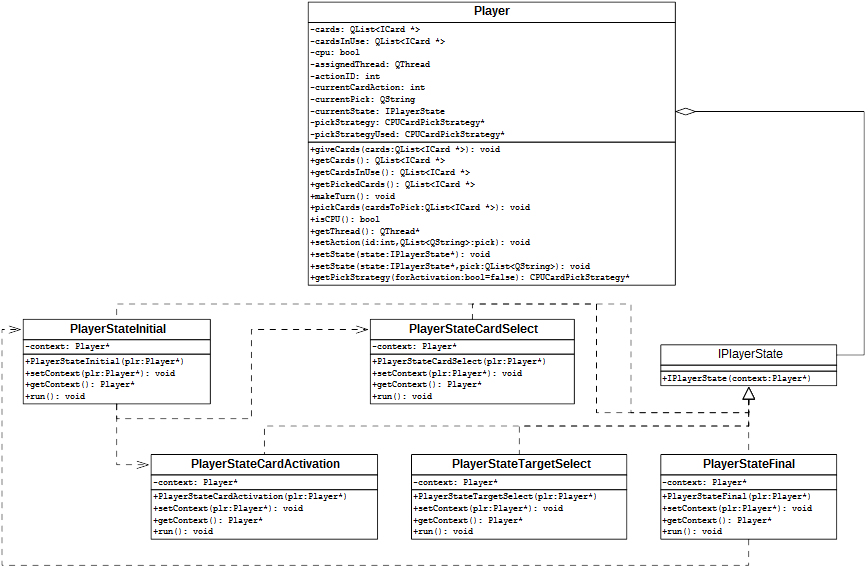
## Состояние

**Назначение:**

Позволяет при наличии у объекта поведения с явно выраженными дискретными состояниями обойтись без громоздких и неудобных конструкций, основанных на if/switch.

Например, у объектов класса Player несколько состояний: начальное, выбор карты для вывода, выбор карты для действия, выбор цели, конечное.

**Диаграмма классов:**

**Исходный код:**

**player.h**

// интерфейс классов состояний

class IPlayerState : public QObject

{

Q\_OBJECT

public:

IPlayerState(QObject \*parent = 0);

IPlayerState(Player \*plr, QObject \*parent = 0);

virtual void setContext(Player \*plr) = 0;

virtual Player \* getContext() = 0;

virtual void run() = 0;

};

// начальное состояние

class PlayerStateInitial : public IPlayerState

{

Q\_OBJECT

private:

Player \*context;

public:

PlayerStateInitial(QObject \*parent = 0);

PlayerStateInitial(Player \*plr, QObject \*parent = 0);

void setContext(Player \*plr);

Player \* getContext();

void run();

};

// состояние выбора карты

class PlayerStateCardSelect : public IPlayerState

{

Q\_OBJECT

private:

Player \*context;

public:

PlayerStateCardSelect(QObject \*parent = 0);

PlayerStateCardSelect(Player \*plr, QObject \*parent = 0);

void setContext(Player \*plr);

Player \* getContext();

void run();

};

// состояние задействования карты

class PlayerStateCardActivation : public IPlayerState

{

Q\_OBJECT

private:

Player \*context;

public:

PlayerStateCardActivation(QObject \*parent = 0);

PlayerStateCardActivation(Player \*plr, QObject \*parent = 0);

void setContext(Player \*plr);

Player \* getContext();

void run();

};

// состояние выбора цели

class PlayerStateTargetSelect : public IPlayerState

{

Q\_OBJECT

private:

Player \*context;

public:

PlayerStateTargetSelect(QObject \*parent = 0);

PlayerStateTargetSelect(Player \*plr, QObject \*parent = 0);

void setContext(Player \*plr);

Player \* getContext();

void run();

};

// конечное состояние

class PlayerStateFinal : public IPlayerState

{

Q\_OBJECT

private:

Player \*context;

public:

PlayerStateFinal(QObject \*parent = 0);

PlayerStateFinal(Player \*plr, QObject \*parent = 0);

void setContext(Player \*plr);

Player \* getContext();

void run();

};

// класс игрока

class Player : public QObject

{

Q\_OBJECT

private:

...

// текущее состояние

IPlayerState \*currentState;

//

bool needToRestartThread;

...

public:

// конструкторы

Player(QObject \*parent = 0);

Player(bool isCPU, QObject \*parent = 0);

...

void setState(IPlayerState \*state);

void setState(IPlayerState \*state, QList<QString> pick);

...

// ход

void makeTurn();

...

};

**player.cpp**

void Player::makeTurn()

{

// ID действий:

// 0: начальное

// 1: вывод карты на поле

// 2: задействование выведенной карты

// 3: выбор целевой карты

// 10: окончание хода

currentState->run();

if (needToRestartThread)

{

needToRestartThread = false;

emit restart();

}

}

void Player::setState(IPlayerState \*state)

{

currentState = state;

}

void Player::setState(IPlayerState \*state, QList<QString> pick)

{

actionPick = pick;

currentState = state;

}

IPlayerState::IPlayerState(QObject \*parent) : QObject(parent)

{

}

IPlayerState::IPlayerState(Player \*plr, QObject \*parent) : IPlayerState(parent)

{

}

PlayerStateInitial::PlayerStateInitial(QObject \*parent) : IPlayerState(parent)

{

}

PlayerStateInitial::PlayerStateInitial(Player \*plr, QObject \*parent) : PlayerStateInitial(parent)

{

context = plr;

}

void PlayerStateInitial::setContext(Player \*plr)

{

context = plr;

}

Player \*PlayerStateInitial::getContext()

{

return context;

}

void PlayerStateInitial::run()

{

bool isCPU = context->isCPU();

if (isCPU)

{

int actionID = Random::randIntWeighted({context->getCards().empty() ? 0 : 5, context->getCardsInUse().empty() ? 0 : 5});

if (actionID == 0)

context->setState(new PlayerStateCardSelect(dynamic\_cast<Player\*>(context)));

else if (actionID == 1)

context->setState(new PlayerStateCardActivation(dynamic\_cast<Player\*>(context)));

context->setRestartTask(true);

}

else

{

return;

}

}

PlayerStateCardSelect::PlayerStateCardSelect(QObject \*parent) : IPlayerState(parent)

{

}

PlayerStateCardSelect::PlayerStateCardSelect(Player \*plr, QObject \*parent) : PlayerStateCardSelect(parent)

{

context = plr;

}

void PlayerStateCardSelect::setContext(Player \*plr)

{

context = plr;

}

Player \*PlayerStateCardSelect::getContext()

{

return context;

}

void PlayerStateCardSelect::run()

{

GameSession \*gs = GameSession::getInstance();

bool isCPU = context->isCPU();

if (isCPU)

{

QList<ICard \*> cards = context->getCards(),

cardsInUse = context->getCardsInUse();

int randNum = context->getStrategy()->pickCard(cards);

cardsInUse.push\_back(cards.takeAt(randNum));

context->setCards(cards);

context->setCardsInUse(cardsInUse);

context->addID(listItem(cardsInUse.back()->getDrawable(),

cardsInUse.back()->getPseudoDrawable(),

cardsInUse.back()->getName(),

cardsInUse.back()->getUID()), 2);

context->showCardCount(cards.size());

gs->msgRepeater("Игрок вывел на поле " + cardsInUse.back()->getName());

context->setState(new PlayerStateInitial(dynamic\_cast<Player\*>(context)), QList<QString>());

context->getThread()->msleep(250);

context->turnEnd();

}

else

{

int pick = 0;

QList<ICard \*> cards = context->getCards(),

cardsInUse = context->getCardsInUse();

QList<QString> actionPick = context->getActionPick();

for (int i = 0; i < cards.size(); i++)

if (cards[i]->getUID() == actionPick[0])

{

pick = i;

break;

}

context->removeID(listItem(cards[pick]->getDrawable(), cards[pick]->getPseudoDrawable(), cards[pick]->getName(), cards[pick]->getUID()));

cardsInUse.push\_back(cards.takeAt(pick));

context->setCards(cards);

context->setCardsInUse(cardsInUse);

context->addID(listItem(cardsInUse.back()->getDrawable(), cardsInUse.back()->getPseudoDrawable(), cardsInUse.back()->getName(), cardsInUse.back()->getUID()), 1);

gs->msgRepeater("Игрок вывел на поле " + cardsInUse.back()->getName());

// вывели карту, можно переводить ход

context->setState(new PlayerStateInitial(dynamic\_cast<Player\*>(context)), QList<QString>());

context->getThread()->msleep(250);

context->turnEnd();

}

}

PlayerStateCardActivation::PlayerStateCardActivation(QObject \*parent) : IPlayerState(parent)

{

}

PlayerStateCardActivation::PlayerStateCardActivation(Player \*plr, QObject \*parent) : PlayerStateCardActivation(parent)

{

context = plr;

}

void PlayerStateCardActivation::setContext(Player \*plr)

{

context = plr;

}

Player \*PlayerStateCardActivation::getContext()

{

return context;

}

void PlayerStateCardActivation::run()

{

GameSession \*gs = GameSession::getInstance();

bool isCPU = context->isCPU();

if (isCPU)

{

QList<ICard \*> cardsInUse = context->getCardsInUse();

int randNum = (cardsInUse.size() != 1) ? context->getStrategy(true)->pickCard(cardsInUse) : 0;

//actionPick = randNum;

context->setState(new PlayerStateTargetSelect(dynamic\_cast<Player\*>(context)));

context->setCurrentPick(cardsInUse[randNum]->getUID());

gs->msgRepeater("Игрок выбрал " + cardsInUse[randNum]->getName());

QList<effectInfoItem> effs = cardsInUse[randNum]->getEffectsInfo();

int effID = Random::randInt(0, effs.size() - 1);

context->chooseCardAction(effID);

emit context->showPickedCardInfoSingle(effs[effID]);

QList<ICard \*> pick;

if (effs[effID].canAffectEnemy)

{

QList<ICard \*> playerCards = context->lookForNextPlayerUsedCards();

if (!playerCards.empty())

{

int cardId = Random::randInt(0, playerCards.size() - 1);

pick.push\_back(playerCards[cardId]);

}

}

if (effs[effID].canAffectPlayer)

{

int cardId = Random::randInt(0, cardsInUse.size() - 1);

pick.push\_back(cardsInUse[cardId]);

}

context->pickCards(pick);

context->restart();

}

else

{

QList<ICard \*> cardsInUse = context->getCardsInUse();

QList<QString> actionPick = context->getActionPick();

int pick = 0;

for (int i = 0; i < cardsInUse.size(); i++)

if (cardsInUse[i]->getUID() == actionPick[0])

{

pick = i;

break;

}

emit context->showPickedCardInfo(cardsInUse[pick]->getEffectsInfo());

gs->msgRepeater("Игрок выбрал " + cardsInUse[pick]->getName());

context->setState(new PlayerStateTargetSelect(dynamic\_cast<Player\*>(context)));

}

}

PlayerStateTargetSelect::PlayerStateTargetSelect(QObject \*parent) : IPlayerState(parent)

{

}

PlayerStateTargetSelect::PlayerStateTargetSelect(Player \*plr, QObject \*parent) : PlayerStateTargetSelect(parent)

{

context = plr;

}

void PlayerStateTargetSelect::setContext(Player \*plr)

{

context = plr;

}

Player \*PlayerStateTargetSelect::getContext()

{

return context;

}

void PlayerStateTargetSelect::run()

{

GameSession \*gs = GameSession::getInstance();

bool isCPU = context->isCPU();

if (isCPU)

{

QList<ICard \*> cardsInUse = context->getCardsInUse();

QList<ICard \*> pickedCards = context->getPickedCards();

QString currentPick = context->getCurrentPick();

int pick = 0;

if (pickedCards.empty())

{

gs->msgRepeater("Игрок ничего не выбрал и пропускает ход.");

context->callTurnEnd();

return;

}

for (int i = 0; i < cardsInUse.size(); i++)

if (cardsInUse[i]->getUID() == currentPick)

{

pick = i;

break;

}

cardsInUse[pick]->makeActionSelection(context->getCardAction());

cardsInUse[pick]->act(context->getPickedCards());

context->callTurnEnd();

}

else

{

QList<ICard \*> cardsInUse = context->getCardsInUse();

QList<QString> actionPick = context->getActionPick();

int pick = 0;

for (int i = 0; i < cardsInUse.size(); i++)

if (cardsInUse[i]->getUID() == actionPick[0])

{

pick = i;

break;

}

cardsInUse[pick]->makeActionSelection(context->getCardAction());

QList<ICard \*> tCards, tPick;

QVector<QString> selection = context->getSelection();

effectInfoItem eInfo = context->getSelectedEffectInfo();

if (eInfo.canAffectEnemy)

tCards.append(context->lookForNextPlayerUsedCards());

if (eInfo.canAffectPlayer)

tCards.append(cardsInUse);

for (int i = 0; i < selection.size(); i++)

{

for (int j = 0; j < tCards.size(); j++)

{

if (selection[i] == tCards[j]->getUID())

{

tPick.push\_back(tCards[j]);

}

}

}

context->pickCards(tPick);

cardsInUse[pick]->act(context->getPickedCards());

context->callTurnEnd();

}

}

PlayerStateFinal::PlayerStateFinal(QObject \*parent) : IPlayerState(parent)

{

}

PlayerStateFinal::PlayerStateFinal(Player \*plr, QObject \*parent) : PlayerStateFinal(parent)

{

context = plr;

}

void PlayerStateFinal::setContext(Player \*plr)

{

context = plr;

}

Player \*PlayerStateFinal::getContext()

{

return context;

}

void PlayerStateFinal::run()

{

bool isCPU = context->isCPU();

if (isCPU)

{

context->setState(new PlayerStateInitial(dynamic\_cast<Player \*>(context)), {});

context->setCurrentPick("");

context->chooseCardAction(-1);

context->getThread()->msleep(250);

context->turnEnd();

}

else

{

context->setState(new PlayerStateInitial(dynamic\_cast<Player \*>(context)), {});

context->setCurrentPick("");

context->chooseCardAction(-1);

context->getThread()->msleep(250);

context->turnEnd();

}

}

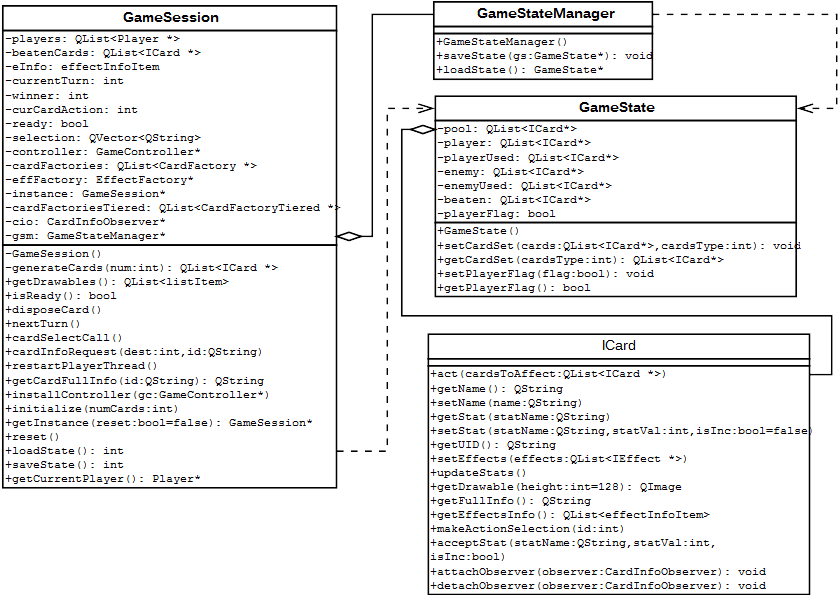
## Хранитель

**Назначение:**

Позволяет сохранять состояние объекта или группы объектов и, при необходимости, откатываться к нему.

В данном примере сохраняется в файл полное состояние игры.

**Диаграмма классов:**

**Исходный код:**

**gamestate.h**

enum GameStateCardsSet

{

GS\_POOL = 0,

GS\_PLAYER,

GS\_PLAYERUSED,

GS\_ENEMY,

GS\_ENEMYUSED,

GS\_BEATEN

};

class GameState

{

private:

QList <ICard \*> pool,

player,

playerUsed,

enemy,

enemyUsed,

beaten;

bool playersFlag;

public:

GameState();

void setCardSet(QList<ICard \*> cards, int cardsType);

QList<ICard \*> getCardSet(int cardsType);

void setPlayersFlag(bool flag);

bool getPlayersFlag();

};

**gamestate.cpp**

GameState::GameState()

{

}

void GameState::setCardSet(QList<ICard \*> cards, int cardsType)

{

switch (cardsType)

{

case GS\_POOL:

pool = cards;

break;

case GS\_PLAYER:

player = cards;

break;

case GS\_PLAYERUSED:

playerUsed = cards;

break;

case GS\_ENEMY:

enemy = cards;

break;

case GS\_ENEMYUSED:

enemyUsed = cards;

break;

case GS\_BEATEN:

beaten = cards;

break;

default:

break;

}

}

QList<ICard \*> GameState::getCardSet(int cardsType)

{

switch (cardsType)

{

case GS\_POOL:

return pool;

break;

case GS\_PLAYER:

return player;

break;

case GS\_PLAYERUSED:

return playerUsed;

break;

case GS\_ENEMY:

return enemy;

break;

case GS\_ENEMYUSED:

return enemyUsed;

break;

case GS\_BEATEN:

return beaten;

break;

default:

return QList<ICard \*>();

break;

}

}

void GameState::setPlayersFlag(bool flag)

{

playersFlag = flag;

}

bool GameState::getPlayersFlag()

{

return playersFlag;

}

**gamestatemanager.h**

class GameStateManager : public QObject

{

Q\_OBJECT

public:

explicit GameStateManager(QObject \*parent = 0);

void saveState(GameState \*gs, int \*ok = NULL);

GameState \* loadState(int \*ok = NULL);

signals:

public slots:

};

**gamestatemanager.cpp**

GameStateManager::GameStateManager(QObject \*parent) : QObject(parent)

{

}

void GameStateManager::saveState(GameState \*gs, int \*ok)

{

int result = 0;

int sizeInt = 0;

try

{

CardStreamWriter \*csw = new CardStreamWriter();

QFile outf("savegame.bin");

outf.open(QFile::WriteOnly | QFile::Truncate);

QDataStream outs(&outf);

outs.setVersion(QDataStream::Qt\_4\_0);

csw->setupStream(&outs);

QList<ICard \*> cards;

cards = gs->getCardSet(GameStateCardsSet::GS\_POOL);

sizeInt = cards.size();

outs << sizeInt;

while (!cards.empty())

csw->writeCard(cards.takeFirst());

bool players = gs->getPlayersFlag();

outs << players;

if (players)

{

cards = gs->getCardSet(GameStateCardsSet::GS\_PLAYER);

sizeInt = cards.size();

outs << sizeInt;

while (!cards.empty())

csw->writeCard(cards.takeFirst());

cards = gs->getCardSet(GameStateCardsSet::GS\_PLAYERUSED);

sizeInt = cards.size();

outs << sizeInt;

while (!cards.empty())

csw->writeCard(cards.takeFirst());

cards = gs->getCardSet(GameStateCardsSet::GS\_ENEMY);

sizeInt = cards.size();

outs << sizeInt;

while (!cards.empty())

csw->writeCard(cards.takeFirst());

cards = gs->getCardSet(GameStateCardsSet::GS\_ENEMYUSED);

sizeInt = cards.size();

outs << sizeInt;

while (!cards.empty())

csw->writeCard(cards.takeFirst());

cards = gs->getCardSet(GameStateCardsSet::GS\_BEATEN);

sizeInt = cards.size();

outs << sizeInt;

while (!cards.empty())

csw->writeCard(cards.takeFirst());

}

}

catch (...)

{

result = 1;

}

if (ok != NULL)

\*ok = result;

}

GameState \*GameStateManager::loadState(int \*ok)

{

int result = 0;

int sizeInt = 0;

GameState \*gs = new GameState();

try

{

CardStreamReader \*csr = new CardStreamReader();

QFile inf("savegame.bin");

inf.open(QFile::ReadOnly);

QList<ICard \*> cards;

QDataStream ins(&inf);

ins.setVersion(QDataStream::Qt\_4\_0);

csr->setupStream(&ins);

bool players = false;

int cardType;

ICard \*card = NULL;

ins >> sizeInt;

for (int i = 0; i < sizeInt; i++)

{

ins >> cardType;

card = matchCardPointer(cardType);

csr->readCard(card);

cards.push\_back(card);

}

gs->setCardSet(cards, GameStateCardsSet::GS\_POOL);

ins >> players;

gs->setPlayersFlag(players);

if (players)

{

ins >> sizeInt;

cards.clear();

for (int i = 0; i < sizeInt; i++)

{

ins >> cardType;

card = matchCardPointer(cardType);

csr->readCard(card);

cards.push\_back(card);

}

gs->setCardSet(cards, GameStateCardsSet::GS\_PLAYER);

ins >> sizeInt;

cards.clear();

for (int i = 0; i < sizeInt; i++)

{

ins >> cardType;

card = matchCardPointer(cardType);

csr->readCard(card);

cards.push\_back(card);

}

gs->setCardSet(cards, GameStateCardsSet::GS\_PLAYERUSED);

ins >> sizeInt;

cards.clear();

for (int i = 0; i < sizeInt; i++)

{

ins >> cardType;

card = matchCardPointer(cardType);

csr->readCard(card);

cards.push\_back(card);

}

gs->setCardSet(cards, GameStateCardsSet::GS\_ENEMY);

ins >> sizeInt;

cards.clear();

for (int i = 0; i < sizeInt; i++)

{

ins >> cardType;

card = matchCardPointer(cardType);

csr->readCard(card);

cards.push\_back(card);

}

gs->setCardSet(cards, GameStateCardsSet::GS\_ENEMYUSED);

ins >> sizeInt;

cards.clear();

for (int i = 0; i < sizeInt; i++)

{

ins >> cardType;

card = matchCardPointer(cardType);

csr->readCard(card);

cards.push\_back(card);

}

gs->setCardSet(cards, GameStateCardsSet::GS\_BEATEN);

}

}

catch (...)

{

result = 1;

}

if (ok != NULL)

\*ok = result;

return gs;

}

**gamesession.h**

// класс сеанса игры

class GameSession : public QObject

{

Q\_OBJECT

private:

...

GameStateManager \*gsm;

...

public:

...

int saveState();

int loadState();

...

};

**gamesession.cpp**

int GameSession::saveState()

{

CardPool \*cp = CardPool::getInstance();

GameState \*gs = new GameState();

gs->setCardSet(beatenCards, GameStateCardsSet::GS\_BEATEN);

if (players.size() >= 2)

{

gs->setPlayersFlag(true);

gs->setCardSet(players[0]->getCards(), GameStateCardsSet::GS\_PLAYER);

gs->setCardSet(players[0]->getCardsInUse(), GameStateCardsSet::GS\_PLAYERUSED);

gs->setCardSet(players[1]->getCards(), GameStateCardsSet::GS\_ENEMY);

gs->setCardSet(players[1]->getCardsInUse(), GameStateCardsSet::GS\_ENEMYUSED);

}

else

gs->setPlayersFlag(false);

QList<ICard \*> poolCards;

while (!cp->isEmpty())

poolCards.push\_back(cp->getCard());

gs->setCardSet(poolCards, GameStateCardsSet::GS\_POOL);

while (!poolCards.empty())

cp->putCard(poolCards.takeFirst());

int result = 0;

gsm->saveState(gs, &result);

delete gs;

return result;

}

int GameSession::loadState()

{

int result = 0;

GameState \*gs;

gs = gsm->loadState(&result);

if (result != 0)

return result;

reset();

CardPool \*cp = CardPool::getInstance();

beatenCards = gs->getCardSet(GameStateCardsSet::GS\_BEATEN);

QList<ICard \*> cards;

cards = gs->getCardSet(GameStateCardsSet::GS\_POOL);

cp->clear();

while (!cards.empty())

cp->putCard(cards.takeFirst());

cp->shuffle();

if (gs->getPlayersFlag())

{

initialize(0);

players[0]->setCards(gs->getCardSet(GameStateCardsSet::GS\_PLAYER));

players[0]->setCardsInUse(gs->getCardSet(GameStateCardsSet::GS\_PLAYERUSED));

players[1]->setCards(gs->getCardSet(GameStateCardsSet::GS\_ENEMY));

players[1]->setCardsInUse(gs->getCardSet(GameStateCardsSet::GS\_ENEMYUSED));

beatenCards = gs->getCardSet(GameStateCardsSet::GS\_BEATEN);

}

if (controller)

{

controller->clearAllEntries();

if (gs->getPlayersFlag())

{

cards = gs->getCardSet(GameStateCardsSet::GS\_PLAYER);

while (!cards.empty())

controller->addCardEntry(getCardListItem(cards.takeFirst()), 0);

cards = gs->getCardSet(GameStateCardsSet::GS\_PLAYERUSED);

while (!cards.empty())

controller->addCardEntry(getCardListItem(cards.takeFirst()), 1);

cards = gs->getCardSet(GameStateCardsSet::GS\_ENEMY);

controller->showEnemyCardCount(cards.size());

while (!cards.empty())

controller->addCardEntry(getCardListItem(cards.takeFirst()), 2);

cards = gs->getCardSet(GameStateCardsSet::GS\_ENEMYUSED);

controller->makeReady();

}

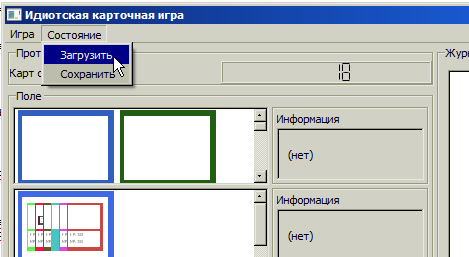
}

delete gs;

return result;

}

**Пример:**



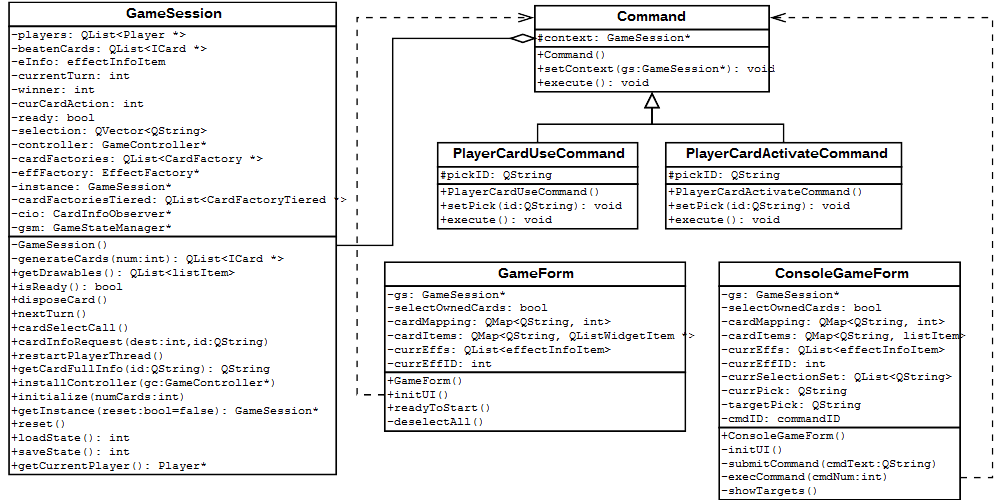
## Команда

**Назначение:**

Команда годится для отправки запросов от «клиента» независимо от того, кто будет это исполнять.

К примеру, здесь есть команды для выбора карты, задействования карты и т. д.

**Диаграмма классов:**

**Исходный код:**

**command.h**

class Command

{

protected:

GameSession \*context;

public:

Command();

void setContext(GameSession \*gs);

virtual void execute() = 0;

};

class PlayerCardUseCommand : public Command

{

protected:

QString pickID;

public:

PlayerCardUseCommand();

void execute();

void setPick(QString id);

};

class PlayerCardActivateCommand : public Command

{

protected:

QString pickID;

public:

PlayerCardActivateCommand();

void execute();

void setPick(QString id);

};

**command.cpp**

Command::Command()

{

context = nullptr;

}

void Command::setContext(GameSession \*gs)

{

context = gs;

}

PlayerCardUseCommand::PlayerCardUseCommand() : Command()

{

pickID = "";

}

void PlayerCardUseCommand::execute()

{

Player \*plr = context->getCurrentPlayer();

if (plr != nullptr)

{

plr->setState(new PlayerStateCardSelect(dynamic\_cast<Player\*>(plr)), {pickID});

plr->getThread()->terminate();

plr->getThread()->start();

}

}

void PlayerCardUseCommand::setPick(QString id)

{

pickID = id;

}

PlayerCardActivateCommand::PlayerCardActivateCommand() : Command()

{

pickID = "";

}

void PlayerCardActivateCommand::execute()

{

Player \*plr = context->getCurrentPlayer();

if (plr != nullptr)

{

plr->setState(new PlayerStateCardActivation(dynamic\_cast<Player\*>(plr)), {pickID});

plr->getThread()->terminate();

plr->getThread()->start();

}

}

void PlayerCardActivateCommand::setPick(QString id)

{

pickID = id;

}

**Пример:**

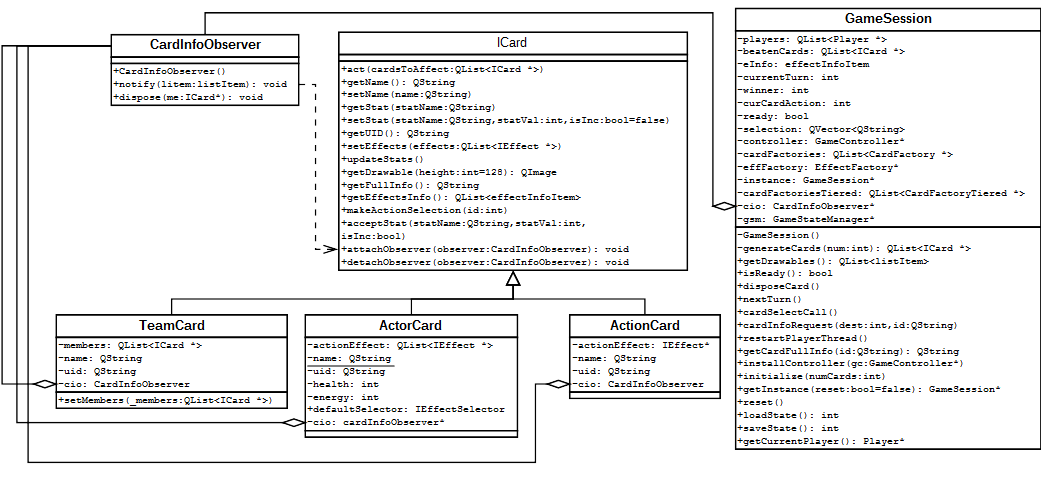
## Перенаправление

**Назначение:**

Позволяет разорвать явную связь между объектами, перераспределив обязанности между ними, делегировав их дополнительному звену между ними.

В данном примере это представлено паттерном Наблюдатель, разрывающим связь между картами и главным объектом GameSession.

**Диаграмма классов:**

**Исходный код:**

(см. Наблюдатель)

## Посетитель

**Назначение:**

Используется для ввода дополнительных операций над объектом без изменения структуры объекта.

Например, теперь карту можно записать в файл или считать из него.

**Диаграмма классов:**

**Исходный код:**

**card.h**

// интерфейс карты

class ICard : public QObject

{

Q\_OBJECT

public:

...

// работа с "посетителями"

virtual void acceptVisitor(CardVisitor \*visitor) = 0;

};

// карта-персонаж

// пока позволяют "здоровье" и "энергия", может действовать сколько угодно раз

// может иметь несколько действий

class ActorCard : public ICard

{

Q\_OBJECT

private:

...

public:

...

void acceptVisitor(CardVisitor \*visitor);

};

class CardVisitor

{

public:

CardVisitor();

virtual void visitActorCard(ActorCard \*card) = 0;

virtual void visitActionCard(ActionCard \*card) = 0;

virtual void visitTeamCard(TeamCard \*card) = 0;

};

ICard \* matchCardPointer(char cardType);

class CardStreamWriter : public CardVisitor

{

private:

QDataStream \*stream;

public:

CardStreamWriter();

void setupStream(QDataStream \*stream);

void writeCard(ICard \*card);

void visitActorCard(ActorCard \*card);

void visitActionCard(ActionCard \*card);

void visitTeamCard(TeamCard \*card);

};

class CardStreamReader : public CardVisitor

{

private:

QDataStream \*stream;

public:

CardStreamReader();

void setupStream(QDataStream \*stream);

void readCard(ICard \*card);

void visitActorCard(ActorCard \*card);

void visitActionCard(ActionCard \*card);

void visitTeamCard(TeamCard \*card);

};

**card.cpp**

...

void ActorCard::acceptVisitor(CardVisitor \*visitor)

{

visitor->visitActorCard(this);

}

…

CardStreamWriter::CardStreamWriter() : CardVisitor()

{

}

void CardStreamWriter::setupStream(QDataStream \*stream)

{

this->stream = stream;

}

void CardStreamWriter::writeCard(ICard \*card)

{

card->acceptVisitor(this);

}

void CardStreamWriter::visitActorCard(ActorCard \*card)

{

int sizeInt = 0;

\*stream << 'A';

\*stream << card->getUID();

\*stream << card->getName();

\*stream << card->getBG();

\*stream << card->getStat("healthInit");

\*stream << card->getStat("energyInit");

\*stream << card->getStat("health");

\*stream << card->getStat("energy");

QList<IEffect \*> effs = card->getEffects();

sizeInt = effs.size();

\*stream << sizeInt;

for (int i = 0; i < sizeInt; i++)

{

effs[i]->saveToStream(stream);

}

\*stream << card->getEffectSelector()->getType();

}

void CardStreamWriter::visitActionCard(ActionCard \*card)

{

\*stream << 'E';

\*stream << card->getUID();

\*stream << card->getName();

\*stream << card->getBG();

card->getEffects().at(0)->saveToStream(stream);

}

void CardStreamWriter::visitTeamCard(TeamCard \*card)

{

int sizeInt = 0;

\*stream << 'T';

\*stream << card->getUID();

\*stream << card->getName();

\*stream << card->getBG();

QList<ICard \*> members = card->getMembers(), membersBeaten = card->getMembers(true);

sizeInt = members.size();

\*stream << sizeInt;

while (!members.empty())

writeCard(members.takeFirst());

sizeInt = membersBeaten.size();

\*stream << sizeInt;

while (!membersBeaten.empty())

writeCard(membersBeaten.takeFirst());

}

ICard \*matchCardPointer(char cardType)

{

ICard \*card;

switch (cardType)

{

case 'A':

card = new ActorCard();

break;

case 'E':

card = new ActionCard();

break;

case 'T':

card = new TeamCard();

break;

default:

card = nullptr;

break;

}

return card;

}

CardVisitor::CardVisitor()

{

}

CardStreamReader::CardStreamReader() : CardVisitor()

{

}

void CardStreamReader::setupStream(QDataStream \*stream)

{

this->stream = stream;

}

void CardStreamReader::readCard(ICard \*card)

{

card->acceptVisitor(this);

}

void CardStreamReader::visitActorCard(ActorCard \*card)

{

int bufInt;

QString bufString;

QColor bufCol;

\*stream >> bufString;

card->setUID(bufString);

\*stream >> bufString;

card->setName(bufString);

\*stream >> bufCol;

card->setBG(bufCol);

\*stream >> bufInt;

card->setStat("healthInit", bufInt);

\*stream >> bufInt;

card->setStat("energyInit", bufInt);

\*stream >> bufInt;

card->setStat("health", bufInt);

\*stream >> bufInt;

card->setStat("energy", bufInt);

int effCount = 0;

\*stream >> effCount;

QList<IEffect \*> effs;

for (int i = 0; i < effCount; i++)

{

int effType;

\*stream >> effType;

IEffect \*eff;

switch (effType)

{

case 'A':

eff = new AttackEffect();

break;

case 'H':

eff = new HelpEffect();

break;

case 'D':

eff = new EffectDestabilizer();

break;

}

eff->loadFromStream(stream);

effs.push\_back(eff);

}

card->setEffects(effs);

IEffectSelector \*effSel = nullptr;

int selectorType = 0;

\*stream >> selectorType;

switch (selectorType)

{

case 0:

effSel = new EffectSelectorAll();

break;

case 1:

effSel = new EffectSelectorRandom();

break;

default:

break;

}

card->setEffectSelector(effSel);

}

void CardStreamReader::visitActionCard(ActionCard \*card)

{

QString bufString;

QColor bufCol;

\*stream >> bufString;

card->setUID(bufString);

\*stream >> bufString;

card->setName(bufString);

\*stream >> bufCol;

card->setBG(bufCol);

IEffect \*eff;

int effType;

\*stream >> effType;

switch (effType)

{

case 'A':

eff = new AttackEffect();

break;

case 'H':

eff = new HelpEffect();

break;

case 'D':

eff = new EffectDestabilizer();

break;

}

eff->loadFromStream(stream);

card->setEffects({eff});

}

void CardStreamReader::visitTeamCard(TeamCard \*card)

{

int sizeInt = 0;

QString bufString;

QColor bufCol;

\*stream >> bufString;

card->setUID(bufString);

\*stream >> bufString;

card->setName(bufString);

\*stream >> bufCol;

card->setBG(bufCol);

QList<ICard \*> storedMembers, storedMembersBeaten;

\*stream >> sizeInt;

for (int i = 0; i < sizeInt; i++)

{

int cardType;

\*stream >> cardType;

ICard \*member = matchCardPointer(cardType);

readCard(member);

storedMembers.push\_back(member);

}

\*stream >> sizeInt;

for (int i = 0; i < sizeInt; i++)

{

int cardType;

\*stream >> cardType;

ICard \*member = matchCardPointer(cardType);

readCard(member);

QObject::connect(member, SIGNAL(disposeMe()), card, SLOT(cardDispose()));

storedMembersBeaten.push\_back(member);

}

card->setMembers(storedMembers);

card->setMembers(storedMembersBeaten, true);

}

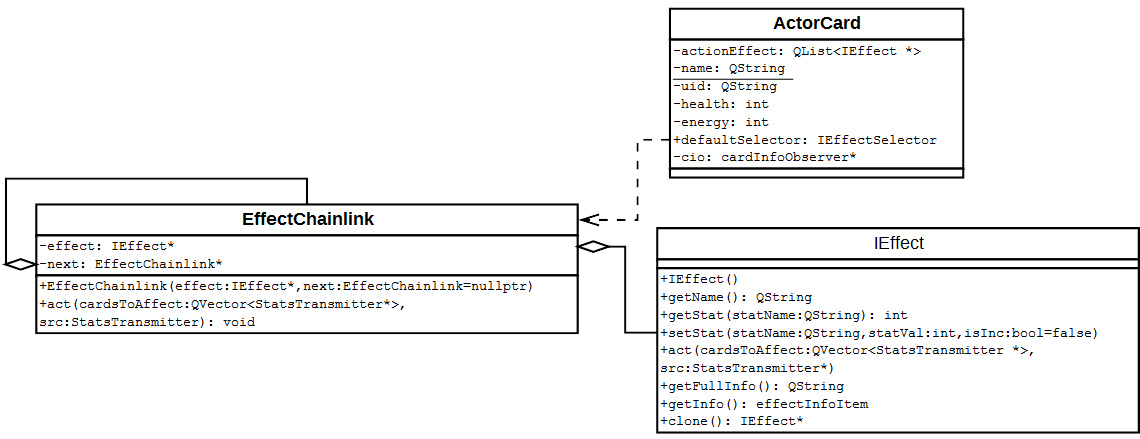
## Цепочка обязанности

**Назначение:**

Используется, когда запрос необходимо обработать с помощью нескольких объектов.

Например, когда карта-персонаж использует несколько действий одно за другим, используя класс-обёртку.

**Диаграмма классов:**

**Исходный код:**

**effect.h**

class EffectChainlink

{

private:

IEffect \*effect;

EffectChainlink \*next;

public:

EffectChainlink(IEffect \*effect, EffectChainlink \*next = nullptr);

void act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src);

~EffectChainlink();

};

**effect.cpp**

EffectChainlink::EffectChainlink(IEffect \*effect, EffectChainlink \*next)

{

this->effect = effect;

this->next = next;

}

void EffectChainlink::act(QVector<StatsTransmitter \*> cardsToAffect, StatsTransmitter \*src)

{

if (effect != nullptr)

effect->act(cardsToAffect, src);

if (next != nullptr)

next->act(cardsToAffect, src);

}

EffectChainlink::~EffectChainlink()

{

if (next != nullptr)

delete next;

}

**card.cpp**

…

void ActorCard::act(QList<ICard \*> cardsToAffect)

{

GameSession \*gs = GameSession::getInstance();

if (energy <= 0)

return;

QVector<StatsTransmitter \*> transmitters;

QMap< StatsTransmitter \*, ICard \* > transMapping;

StatsTransmitter \*srcTrans = new StatsTransmitter();

for (int i = 0; i < cardsToAffect.size(); i++)

{

StatsTransmitter \*st = new StatsTransmitter();

connect(st, SIGNAL(sendStat(QString,int,bool)), cardsToAffect[i], SLOT(acceptStat(QString,int,bool)));

transmitters.push\_back(st);

transMapping[st] = cardsToAffect[i];

}

connect(srcTrans, SIGNAL(sendStat(QString,int,bool)), this, SLOT(acceptStat(QString,int,bool)));

int effectsNum = actionEffect.size();

if (effectsNum == 1)

{

gs->msgRepeater(this->getName() + " использует " + actionEffect[0]->getName());

actionEffect[0]->act(transmitters, srcTrans);

}

else

{

if (selectedAction != -1)

{

actionEffect[selectedAction]->act(transmitters, srcTrans);

selectedAction = -1;

}

else

{

QList<IEffect \*> pick = defaultSelector->selectEffect(actionEffect);

for (int i = 0; i < pick.size(); i++)

{

gs->msgRepeater(this->getName() + " использует " + pick[i]->getName());

}

QVector<StatsTransmitter \*> selectedTrans;

for (int j = 0; j < transmitters.size(); j++)

{

if ((transMapping[transmitters[j]])->getStat("health") > 0)

{

gs->msgRepeater(this->getName() + " воздействует на " + transMapping[transmitters[j]]->getName());

selectedTrans.push\_back(transmitters[j]);

}

}

EffectChainlink \*efc = new EffectChainlink(pick.takeLast());

while (!pick.empty())

efc = new EffectChainlink(pick.takeLast(), efc);

efc->act(selectedTrans, srcTrans);

delete efc;

}

}

while (!transmitters.empty())

delete transmitters.takeAt(0);

delete srcTrans;

}

...

## Медиатор

**Назначение:**

Используется для взаимодействия между разнообразными объектами без использования явных связей между ними.

Например, через него игрок может посмотреть карты следующего игрока, чтобы решить, что атаковать.

**Диаграмма классов:**

**Исходный код:**

**player.h**

// класс игрока

class Player : public QObject

{

Q\_OBJECT

private:

...

public:

...

void attachMediator(PlayerMediator \*mediator);

void detachMediator(PlayerMediator \*mediator);

QList <ICard \*> lookForNextPlayerUsedCards();

void tellSelectedEffectInfo(effectInfoItem e);

...

};

class PlayerMediator : public QObject

{

Q\_OBJECT

private:

QList<Player \*> subscribers;

public:

PlayerMediator(QObject \*parent = 0);

QList<ICard \*> showNextPlayerCards();

};

**player.cpp**

void Player::attachMediator(PlayerMediator \*mediator)

{

if (this->mediator != mediator)

this->mediator = mediator;

}

void Player::detachMediator(PlayerMediator \*mediator)

{

if (this->mediator == mediator)

this->mediator = nullptr;

}

QList<ICard \*> Player::lookForNextPlayerUsedCards()

{

QList<ICard \*> result;

if (mediator != nullptr)

result = mediator->showNextPlayerCards();

return result;

}

…

PlayerMediator::PlayerMediator(QObject \*parent) : QObject(parent)

{

}

QList<ICard \*> PlayerMediator::showNextPlayerCards()

{

QList<ICard \*> result;

Player \*nextPlayer = GameSession::getInstance()->getNextPlayer();

if (nextPlayer != nullptr)

result = nextPlayer->getCardsInUse();

return result;

}

**Пример:**

void PlayerStateCardActivation::run()

{

GameSession \*gs = GameSession::getInstance();

bool isCPU = context->isCPU();

if (isCPU)

{

...

if (effs[effID].canAffectEnemy)

{

QList<ICard \*> playerCards = context->lookForNextPlayerUsedCards();

if (!playerCards.empty())

{

int cardId = Random::randInt(0, playerCards.size() - 1);

pick.push\_back(playerCards[cardId]);

}

}

if (effs[effID].canAffectPlayer)

{

int cardId = Random::randInt(0, cardsInUse.size() - 1);

pick.push\_back(cardsInUse[cardId]);

}

context->pickCards(pick);

context->restart();

}

else

{

...

}

}

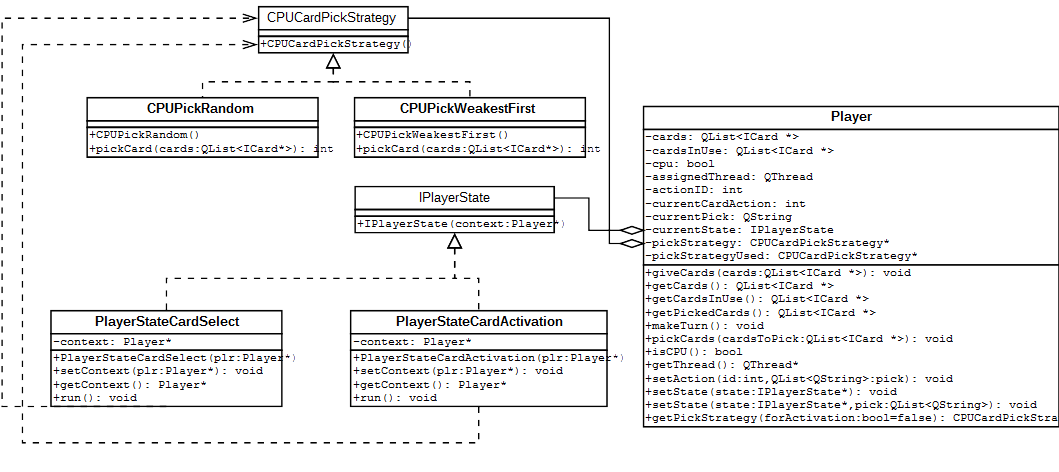
## Стратегия

**Назначение:**

Предназначается для создания возможности решения одной и той же проблемы разными способами.

К примеру, компьютер может выбирать карты для хода по-разному: выбрать случайную или выбрать самую слабую.

**Диаграмма классов:**

**Исходный код:**

**player.h**

class Player : public QObject

{

Q\_OBJECT

private:

..

CPUCardPickStrategy \*pickStrategy, \*pickStrategyUsed;

..

public:

..

CPUCardPickStrategy \* getStrategy(bool forActivation = false);

..

};

class CPUCardPickStrategy

{

public:

CPUCardPickStrategy();

virtual int pickCard(QList<ICard \*> cards) = 0;

};

class CPUPickRandom : public CPUCardPickStrategy

{

public:

CPUPickRandom();

int pickCard(QList<ICard \*> cards);

};

class CPUPickWeakestFirst : public CPUCardPickStrategy

{

public:

CPUPickWeakestFirst();

int pickCard(QList<ICard \*> cards);

};

**player.cpp**

CPUCardPickStrategy::CPUCardPickStrategy()

{

}

CPUPickRandom::CPUPickRandom() : CPUCardPickStrategy()

{

}

int CPUPickRandom::pickCard(QList<ICard \*> cards)

{

return Random::randInt(0, cards.size() - 1);

}

CPUPickWeakestFirst::CPUPickWeakestFirst()

{

}

int CPUPickWeakestFirst::pickCard(QList<ICard \*> cards)

{

QList<int> actors, actions;

int sizeInt = cards.size();

int result = 0;

for (int i = 0; i < sizeInt; i++)

{

// если это действие (а оно всега имеет 999 HP)

if (cards[i]->getStat("health") == 999)

actions.push\_back(i);

else

actors.push\_back(i);

}

bool selectActions = ((!actors.empty() && !actions.empty() && Random::probability(0.5))

|| (actors.empty() && !actions.empty()));

if (!selectActions)

{

// выбираем из актеров и команд

sizeInt = actors.size();

int curHealth = cards[actors[0]]->getStat("health");

result = actors[0];

for (int i = 1; i < sizeInt; i++)

{

int nextHealth = cards[actors[i]]->getStat("health");

if (nextHealth < curHealth)

{

result = actors[i];

curHealth = nextHealth;

}

}

}

else

{

// выбираем из действий

result = actions[Random::randInt(0, actions.size() - 1)];

}

return result;

}

**Пример:**

void PlayerStateCardSelect::run()

{

GameSession \*gs = GameSession::getInstance();

bool isCPU = context->isCPU();

if (isCPU)

{

QList<ICard \*> cards = context->getCards(),

cardsInUse = context->getCardsInUse();

int randNum = context->getStrategy()->pickCard(cards);

cardsInUse.push\_back(cards.takeAt(randNum));

context->setCards(cards);

context->setCardsInUse(cardsInUse);

context->addID(listItem(cardsInUse.back()->getDrawable(),

cardsInUse.back()->getPseudoDrawable(),

cardsInUse.back()->getName(),

cardsInUse.back()->getUID()), 2);

context->showCardCount(cards.size());

gs->msgRepeater("Игрок вывел на поле " + cardsInUse.back()->getName());

context->setState(new PlayerStateInitial(dynamic\_cast<Player\*>(context)), QList<QString>());

context->getThread()->msleep(250);

context->turnEnd();

}

else

{

...

}

}

# Приложение

Исходные коды программы находятся на прилагаемом диске в каталоге «Исходные коды».