Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

ИГРА “МОНОПОЛИЯ”

БГУИР КП 1-40 02 01 218 ПЗ

                                                                 Студент: гр. 250502 Котик А. Р.

Руководитель: Богдан Е. В.

МИНСК 2023

Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой

*––––––––––––––––––––––––*

(подпись)

–––––––––––––––––––2023 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Котику Алексею Романовичу*

Тема проекта *Игра “Монополия”*

2. Срок сдачи студентом законченного проекта *15 декабря 2023 г.*

3. Исходные данные к проекту *Контейнер двунаправленное кольцо для реализации списка игроков и ячеек*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

1. Лист задания.

2. Введение.

3. Обзор литературы.

4. Функциональное проектирование.

4.1. Структура входных и выходных данных.

4.2. Разработка диаграммы классов.

4.3. Описание классов.

5. Разработка программных модулей.

5.1. Разработка схем алгоритмов (два наиболее важных метода).

5.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов).

6. Результаты работы.

7. Заключение

8. Литература

9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. Диаграмма классов. ––––––––––––––––––––––––––––––*

*2. Схема алгоритма метода 1*

*3. Схема алгоритма метода 2––––––––––––––––––––––––––––––   –*

6. Консультант по проекту (с обозначением разделов проекта) А. М. Ковальчук

7. Дата выдачи задания –––––*15.09.2023г.––––––––––––––––––––––   –*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15 %;––––––––––––––––––––––––––––*

*разделы 2, 3 – 10 %;–––––––––––––––––––––––––––––––––––––*

*разделы 4 к –20 %;–––––––––––––––––––––––––––––––––––––––*

*разделы 5 к – 35 %;––––––––––––––––––––––––––––––––––––––*

*раздел 6,7,8 – 5 %;–––––––––––––––––––––––––––––––––––––––*

*раздел 9 к – 5%;–––––––––––––––––––––––––––––––––––––––––*

*оформление пояснительной записки и графического материала к 15.12.22 – 10 %*

*Защита курсового проекта с 21.12 по 28.12.23г.–––––––––––––––––––––––––*

РУКОВОДИТЕЛЬ Богдан Е. В.

(подпись)

Задание принял к исполнению *–––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_––*

(дата и подпись студента)

**СОДЕРЖАНИЕ**

ВВЕДЕНИE……………………………………………………………………….5

1.ОБЗОР ЛИТЕРАТУРЫ………………………………………………………...6

1.1 Анализ существующих аналогов…………………………………………..6

1.2 Обзор методов и алгоритмов решения поставленной задачи……………6

2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ……………………………….8

2.1 Структура входных и выходных данных………………………………….8

2.2 Разработка диаграммы классов………...…………………………………..9

2.3 Описание классов……………………………………………………………9

2.3.1 Класс контейнера………………………………………………………..9

2.3.2 Классы сущностей ………………………………...…………………...10

2.3.3 Классы ячеек…………………….……………………………………...12

2.3.4 Класс данных…………………………………………………………...16

2.3.5 Классы виджетов……………………………………………………….18

3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ……………………………...25

3.1 Разработка схем алгоритмов……………………………………….……...25

3.2 Разработка алгоритмов…………………………………………...………..25

3.2.1 Разработка алгоритма метода roll()…………………………………...25

3.2.2 Разработка алгоритма метода getActualRent() ……………………….26

4 РЕЗУЛЬТАТЫ РАБОТЫ……………………………………………………...29

ЗАКЛЮЧЕНИЕ……………………………………………………………….....31

СПИСОК ЛИТЕРАТУРЫ………………………………………………………32

ПРИЛОЖЕНИЕ А……………………………………………………………….33

ПРИЛОЖЕНИЕ Б………………………………………………………………..34

ПРИЛОЖЕНИЕ В……………………………………………………………….35

ПРИЛОЖЕНИЕ Г……………………………………………………………….36

**ВВЕДЕНИЕ**

Язык C++ является довольно мощным инструментом для умелого разработчика. Сам язык имеет 4 парадигмы, за счёт этого можно по-разному выстраивать структуру программы. Он сочетает свойства как высокоуровневых, так и низкоуровневых языков программирования.

Если сравнивать данный язык с C, то можно выделить разные парадигмы: C++ в первую очередь был создан для обретения других парадигм, речь идёт про ООП и обобщённое программирование.

Данный язык широко используется для разработки ПО, написания игр, создание операционных систем, написания драйверов и решения множества других задач.

Язык имеет богатую стандартную библиотеку, включающую в себя множество алгоритмов, контейнеров, ввод-вывод, строковые, языковую поддержку, регулярные выражения, поддержку многопоточности и многое другое.

Для реализации удобного графического интерфейса в данной курсовой работе был использован фреймворк под названием Qt.

"Монополия" – это настольная игра для двух и более игроков, где вы можете, используя свой стартовый капитал, покупать и продавать свою собственность для достижения наибольшей выгоды. Целью игры является банкротство соперников.

Таким образом для разработки игры “Монополия” язык C++ и фреймворк Qt подходят идеально.

**1 ОБЗОР ЛИТЕРАТУРЫ**

* 1. **Анализ существующих аналогов**

“Монополия” является одной из самых популярных настольных игр в мире.

У нее существует множество вариаций, например браузерная игра “Monopoly One”. В этой версии можно поиграть со своими друзьями или даже с незнакомцами.

Моя реализация “Монополии” имеет ряд недостатков по сравнению с вышеупомянутой Monopoly One”. Один из самых примечательных изъянов – отсутствие онлайн-режима. Однако в моей версии можно поиграть с ботами или со своими друзьями с одного устройства. Также в моей игре опущены некоторые правила и возможности. Такое решение я принял на основе своего опыта игры в “Монополию”. Опущенные мною правила и возможности никто не соблюдает и не использует, так как процесс игры может оказаться долгим и часто игроки ссорятся между собой.

Поэтому в данном смысле опущение некоторых особенностей является преимуществом над аналогами.

* 1. **Обзор методов и алгоритмов решения поставленной задачи**

Для создания “Монополии” был использован фреймворк Qt. Он содержит стандартные библиотеки С++, а также набор инструментов для создания удобного графического интерфейса.

Игрок может совершать действия, нажимая на различные кнопки. Такой метод основан на механизме сигналов и слотов. В моем проекте реализованы три окна: стартовое окно, где можно ввести стартовые параметры (StartWindow); главное окно (MainWindow), где происходит сам процесс игры; информационное окно (CellInfo), содержащее информацию о данной ячейке.

В моей версии игры игрок должен уметь бросить кости, перебросить кости, купить, заложить, продать собственность. В зависимости от позиции на игровом поле автоматически должны выполняться некоторые действия, например игрок попал в тюрьму, наступил на клетку “Шанс”, заплатил или получил деньги за аренду собственности и т.д. Также игрок должен видеть какие последние действия произошли в игре.

В программе реализовано наследование: живой игрок (Player) и бот (Bot) наследуются от абстрактного класса сущности (Entity); ячейки недвижимости (House), социальных предприятий (Estate) и событийных ячеек (Event) наследуются от абстрактного класса Cell. Наличие абстрактных классов позволяет хранит объекты производных классов в одном контейнере.

В программе реализовано шаблонное двунаправленное кольцо (Ring<T>), с помощью которого можно хранить информацию об игроках и игровом поле. В кольце отсутствует поле указателя на текущий элемент, так положение каждого игрока на кольцевом игровом поле разное. Вместо этого каждый игрок содержит указатель на текущую позицию, а также в классе данных Data есть отдельное поле для хранения указателя на текущего игрока.

Класс Data содержит данные об игре и оперирует ими. Класс MainWindow – интерфейс между игроком и программой.

В игре должны быть реализованы методы броска костей, переброса костей с возвращением поля и игроков в состояние до броска костей. К этому методу должен быть написан алгоритм копирования данных (игрового поля и игроков). Также должны присутствовать методы обработки ячейки, так как поведение программы зависит от того, какой тип данной ячейки, кто на ней стоит, кто ей владеет и т.д. Игрок должен знать о происходящих действиях на экране, видеть нужную информацию об игроках.

Должно быть организовано поведение ботов. В данной курсовой работе бот будет выполнять определенные действия, если он может, случайным образом.

Процесс игры продолжится пока все игроки, кроме одного, не сдадутся. Процесс сдачи каким-либо игроком удаляет его с игры и продает все его участки банку так, что остальные игроки теперь смогут купить его участки.

**2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

**2.1 Структура входных и выходных данных**

В начале игры пользователю представится стартовое окно (класс StartWindow), где он должен ввести количество игроков, их имена, а также, по желанию, количество ботов.

Остальные характеристики игроков и ботов определяются программой. Эти параметры равнозначны. Пример ввода пользователем стартовых параметров приведен на рисунке 2.1.

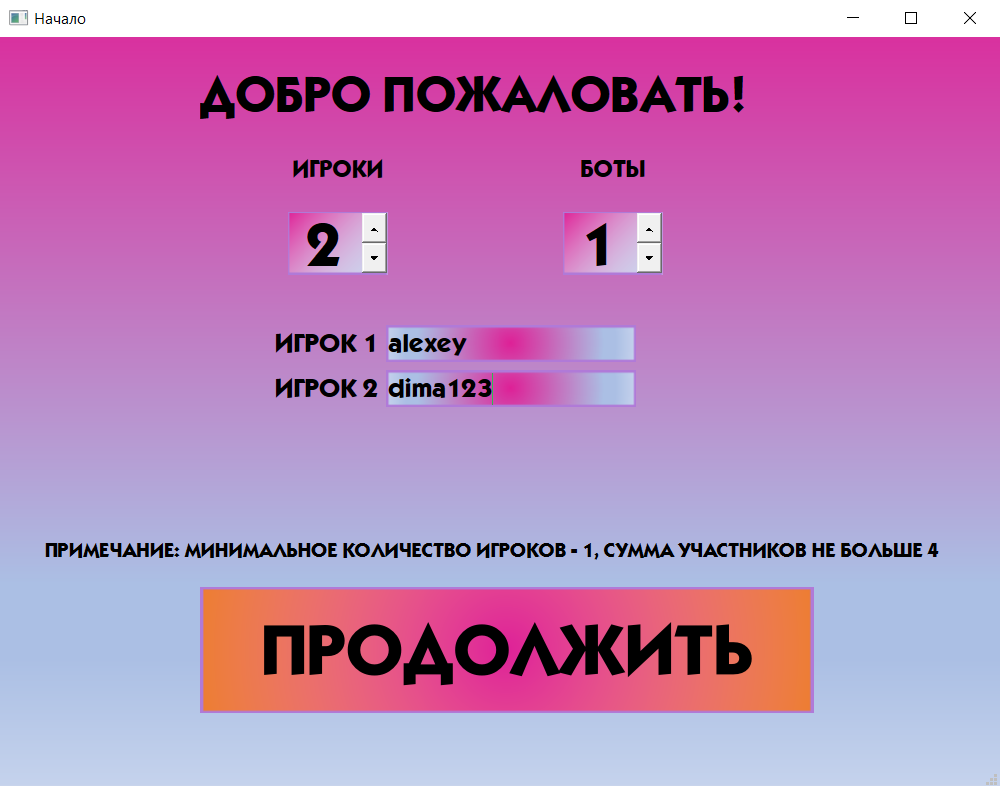


Рисунок 2.1 – Подготовка к игре

Соответствующие рисунку 2.1 полные стартовые параметры приведены в таблице 2.1.

Таблица 2.1 – Стартовые параметры

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Имя | ID | Текущий капитал (доллары) | Пропуск хода  (0 – нет, 1 - да) | Графическая  модель | Позиция на карте |
| alexey | 1 | 1000 | 0 | icon1.png | Старт |
| dima123 | 2 | 1000 | 0 | icon2.png | Старт |
| БОТ1 | 3 | 1000 | 0 | icon3.png | Старт |

**2.2 Разработка диаграммы классов**

Диаграмма классов данной курсовой работы приведена в приложении А.

**2.3 Описание классов**

**2.3.1 Класс контейнера**

В проекте используется шаблонный список двунаправленное кольцо Ring<T> для хранения игрового поля и игроков. Стоит отметить что Node<T> - узел, хранящий данные, а также указатель на предыдущий и следующий элемент.

Описание полей класса Ring<T>:

Node<T>\* first – указатель на первый элемент кольца

int size – размер кольца

Описание методов класса Ring<T>:

Ring() – конструктор по умолчанию

Ring(const Ring& ring) – конструктор копирования

~Ring() – деструктор

void push(T value) – добавление элемента в кольцо

void popValue(T value) – удалить элемент в соответствии с заданным аргументом

Node<T>\* getValue(T value) – возвращает элемент в соответствии с заданным аргументом

Node<T>\* getFirst() – возвращает первый элемент кольца

Node<T>\* getNext(Node<T>\* current) – возвращает элемент после current

Node<T>\* getNextN(Node<T>\* current, int N) – возвращает через N элемент после current

Node<T>\* getPrev(Node<T>\* current) – возвращает элемент перед current

Node<T>\* getPrevN(Node<T>\* current, int N) – возвращает через N элемент перед current

int&getSize() – возвращает размер кольца

**2.3.2 Классы сущностей**

Класс Entity – абстрактный класс для описания сущности (игрока или бота).

Описание полей:

int id – номер сущности

int money – текущий капитал

Node<Cell\*>\* pos – позиция на игровом поле

int skip – идентификатор пропуска хода

QString icon – хранит графическую модель сущности в виде строки

QString name – имя

Описание методов:

Entity(int id,int money,Node<Cell\*>\* pos,int skip, QString icon,QString name) - конструктор

Entity(const Entity&entity) – конструктор копирования

Entity& operator=(const Entity&source) – оператор присваивания

~Entity() - деструктор

void setID(int id) – установить ID

void setMoney(int money) – установить текущий капитал

void setPos(Node<Cell\*>\*pos) – установить текущую позицию

void setSkip(int skip) – установить идентификатор пропуска

void setIcon(QString icon) – установить графическую модель

void setName(QString name) – установить имя

QString& getIcon() – возвращает графическую модель

QString& getName() – возвращает имя

int& getSkip() – возвращает идентификатор пропуска

int& getID() – возвращает ID

int& getMoney() – возвращает текущий капитал

Node<Cell\*>\* getPos() – возвращает текущую позицию на карте

bool isEnough(int cost) – проверяет, может ли сущность заплатить заданную цену

bool mortgage(Cell\*cell) – кладет в залог или выкупает собственность и возвращает статус операции

bool buyHouse(Cell\*cell) – приобретает домик на заданной собственности и возвращает 1, если на участке стоит максимальное количество домиков

bool sellHouse(Cell\*cell) – продает домик на заданной собственности и возвращает 1, если до продажи стояло максимальное количество домиков

void sell(Cell\*cell) – продает заданную собственность

virtual void buy()=0 – виртуальный метод покупки собственности

void pay(int rent) – платит установленную цену

bool isPlayer() – возвращает 1, если данная сущность – реальный игрок

Класс Bot описывает бота (нереального игрока). Наследуется от класса Entity. Новых полей не имеет.

Описание методов:

Bot(int id=0,int money=0,Node<Cell\*>\*pos=nullptr,int skip=0, QString icon="",QString name="") – конструктор

Bot(const Bot&bot) – конструктор копирования

~Bot() – деструктор

void buy() override – перегрузка виртуального метода: бот купит собственность, на которой он стоит с некоторым шансом

Класс Player описывает реального игрока. Наследуется от класса Entity.

Новый полей не имеет.

Описание методов:

Player(int id=0,int money=0,Node<Cell\*>\*pos=nullptr,int skip=0,QString icon="",QString name="") – конструктор

Player(const Player& player) – конструктор копирования

~Player() – деструктор

void buy() override – перегрузка виртуального метода: покупка собственности, на которой стоит игрок.

**2.3.3 Классы ячеек**

Класс Cell – абстрактный класс для описания ячейки игрового поля.

Игровое поле составляет из себя кольцо. На экране это кольцо будет отображаться в виде контура прямоугольника. Каждая ячейка имеет позицию в этом контуре.

Описание полей:

QString name – имя ячейки

int posX – позиция ячейки по X

int posY – позиция ячейки по Y

Описание методов:

Cell(QString name="",int posX=0,int posY=0) – конструктор

Cell(const Cell& cell) – конструктор копирования

~Cell() – деструктор

void setPosX(int posX) – задать позицию ячейки по X

void setPosY(int posY) – задать позицию ячейки по Y

void setName(QString name) – установить имя

bool isEstate() – возвращает 1, если данная ячейка – ячейка социального предприятия

bool isHouse() – возвращает 1, если данная ячейка – ячейка недвижимости

bool isEvent() – возвращает 1, если данная ячейка – событийная

int&getPosX() – возвращает позицию ячейки по X

int&getPosY() – возвращает позицию ячейки по Y

QString&getName() – возвращает имя ячейки

Класс Estate описывает социальные предприятия, такие как железная дорога, водопровод и электростанция. Также является родителем класса House, описывающего недвижимость. Класс Estate наследуется от Cell.

Описание полей:

int ownerID – ID владельца собственности

int cost – стоимость покупки собственности

std::vector<int> rent – вектор хранит аренды участков в различных случаях

bool mortgage – 1, если собственность заложена

int type – тип собственности: 0, если недвижимость, 1, если железная дорога, 2, если электростанция или водопровод

Описание методов:

Estate(int ownerID=0,int cost=0,std::vector<int> rent={},bool mortgage=0,int type=0, QString name="",int posX=0,int posY=0) – конструктор

Estate(const Estate& estate) – конструктор копирования

Estate& operator=(const Estate&estate) – оператор присваивания

void setOwner(int ownerID) – установить ID владельца

void setCost(int cost) – установить стоимость собственности

void setRent(std::vector<int> rent) – установить стоимость аренды

void setType(int type) – установить тип собственности

void setMortgage(bool value) – заложить собственность

bool isMortgage() – возвращает 1, если собственность заложена

int& getType() – возвращает тип собственности

int& getOwnerID() – возвращает ID владельца собственности

int& getCost() – возвращает стоимость участка

std::vector<int>& getRent() – возвращает стоимость аренды участка

~Estate() – деструктор

Класс House описывает недвижимость. Каждая недвижимость принадлежит какой-либо группе. Игрок может строить дома на участке, если он владеет всеми участками в группе. Класс House наследуется от класса Estate.

Описание полей:

int houses – количество домиков на участке

int group – группа, которой принадлежит участок

Описание методов:

House(int ownerID=0,int cost=0,std::vector<int> rent={},bool mortgage=0,int houses=0,int group=0,QString name="",int posX=0,int posY=0) -конструктор

House(const House& house) – конструктор копирования

House& operator=(const House&house) – оператор присваивания

void setHouses(int houses) – установить количество домиков на участке

void setGroup(int group) – установить группу, которой принадлежит участок

int& getHouses() – возвращает количество домиков на участке

int& getGroup() – возвращает группу, которой принадлежит участок

int getHouseCost() – возвращает стоимость постройки одного домика на участке

~House() - деструктор

bool isHousesLimit() – возвращает 1, если участок полностью застроен домами (5 домов)

Класс Event описывает такие событийные ячейки как “Шанс”, “Старт”, “Налог”, “Тюрьма” и “Отдых”. Если игрок наступает на ячейку “Шанс”, то он получает или тратит случайное количество денег. Если игрок проходит “Старт”, он получает 200$. Если игрок попадает на ячейку “Налог”, он платит некоторую сумму денег. Если игрок попадает в “Тюрьму”, он пропускает 1 ход. На ячейке “Отдых” игрок может ничего не делать. Класс Event наследуется от класса Cell.

Описание полей:

int gift – сумма денег, которую получит игрок, если наступит на данную ячейку (сумма может быть отрицательной)

int skip – 1, если ячейка является “Тюрьмой”

int chance – 1, если ячейка является “Шансом”

Описание методов:

Event(int gift=0,int skip=0,int chance=0, QString name="",int posX=0,int posY=0) – конструктор

Event(const Event& event) – конструктор копирования

Event& operator=(const Event&event) – оператор присваивания

void setGift(int gift) – установить сумму денег, которую может получить игрок

void setSkip(int skip) – установить 1, если клетка является “Тюрьмой”

void setChance(int chance) – установить 1, если клетка является “Шансом”

int& getGift() – возвращает сумму денег, которую может получить игрок

int& getSkip() – возвращает 1, если клетка является “Тюрьмой”

int& getChance() – возвращает 1, если клетка является “Шансом”

~Event() – деструктор

**2.3.4 Класс данных**

Класс данных Data содержит в себе игровое поле, всех игроков, а также дополнительные данные для организации игрового процесса. Также он оперирует насколько это возможно этими данными, осуществляет основные взаимодействия между полем и игроками.

Описание полей:

Ring<Cell\*> field – игровое поле

Ring<Entity\*> players – игроки

Node<Entity\*>\*current – текущий игрок

std::vector<std::vector<QString>> images – графические модели игроков

int seed – число, влияющее на все случайные события в игре. Устанавливается в соответствии с текущим временем.

Ring<Cell\*> field\_buffer – буфер хранения игрового поля

Ring<Entity\*> players\_buffer – буфер хранения игроков

Описание методов:

Data() – конструктор по умолчанию

Data(int players,int bots,std::vector<QString> names) – конструктор

Data(const Data& data) – конструктор копирования

~Data() – деструктор

void setImages() – устанавливает графические модели игроков

void setSeed(int seed) – устанавливает seed

void saveBuffer() – копирует игровое поле и игроков в буфер

void loadBuffer() – выгружает игровое поле и игроков из буфера

void copyCells(Cell\* dest,Cell\*source) – данные из одной клетки в другую

int& getSeed() – возвращает значение seed

std::vector<std::vector<QString>>& getImages() – возвращает графические модели игроков

Ring<Entity\*>& getPlayers() – возвращает игроков

Ring<Cell\*>& getField() – возвращает игровое поле

void setNextCurrent() – устанавливает следующего текущего игрока

bool isGrouped(int id,Cell\*cell,int actionStatus) – возвращает 1, может ли игрок с заданным ID выполнить некоторое действие, предписанное параметром actionStatus, на заданной ячейке

int getActualRent() – возвращает стоимость аренды, которую должен заплатить текущий игрок

void copyField(Ring<Cell\*>& destination, Ring<Cell\*>& source) – копирует игровое поле из источника в приемник

void copyPlayers(Ring<Entity\*>& destination, Ring<Entity\*>& source) – копирует игроков из источника в приемник

bool isFree(Entity\*player) – возвращает 0, если ячейка, на которой стоит заданный игрок, принадлежит другому игроку

void setCurrent(Node<Entity\*>\*current) – установить текущего игрока

Node<Entity\*>\* getCurrent() – возвращает текущего игрока

Node<Entity\*>\* getByID(int id) – возвращает игрока в соответствии с заданным ID

void updateAfterForfeit(int id) – обновить данные после сдачи игры каким-либо игроком

Node<Cell\*>\* getCellByPos(int posX,int posY) – возвращает ячейку в соответствии с заданной позицией на игровом поле

**2.3.5 Классы виджетов**

В классах виджетов описывается графическое представление программы, интерфейс, а также некоторые методы игрового процесса, которые должны отображаться на экране.

Все использованные виджеты наследуются от встроенного в Qt класса QMainWindow. В данных виджетах вместе с модификаторами public, private, protected используются методы, имеющие модификатор signals (сигналы) и модификатор public/private slots (слоты). Сигнал вырабатывается когда происходит определенное событие. Слот – это функция, которая вызывается в ответ на определенный сигнал.

Класс StartWindow предназначен для того, чтобы пользователь мог ввести необходимую информацию для начала игры (как описано в подразделе 2.1). Когда игрок начинает игру, испускается сигнал, содержащий введенную информацию об игроках. MainWindow принимает эту информацию, формируя объект данных Data на основе полученных данных.

Описание полей:

Ui::StartWindow \*ui – пользовательский интерфейс, включает в себя все объекты (кнопки, поля ввода и т.д.), которые видит пользователь на экране

MainWindow\* mainWindow – указатель на главное окно

int players – количество игроков

int bots – количество ботов

Описание методов:

explicit StartWindow(QWidget \*parent = nullptr) – конструктор

~StartWindow() – деструктор

void startSettings() – устанавливает все графические настройки

bool eventFilter(QObject\* object,QEvent\*event) – обработчик действий пользователя

int& getPlayers() – возвращает количество игроков

int& getBots() – возвращает количество ботов

QFormLayout\* getFormLayout() – возвращает сетку на экране, где содержатся введенные имена игроков

Описание сигналов и слотов:

void sendData(int players,int bots,std::vector<QString> names) – сигнал, содержащий в себе всю нужную информацию для старта игры

void showLineEdit() – показывает столько граф для ввода имен, сколько игрок установил реальных игроков

void botSpinBox() – устанавливает поведение ползунка выбора количества ботов

void emitData() – передает данные объекту MainWindow посредством сигнала sendData.

Класс CellInfo служит для описания ячейки, на которую нажмет игрок в главном окне MainWindow. Когда игрок нажимает на некоторую ячейку, испускается сигнал, содержащий указатель на данную ячейку и CellInfo принимает эту ячейку и обрабатывает ее для вывода нужной информации на экран. Также в этом окне можно выполнять некоторые действия с ячейкой. При нажатии на соответствующие кнопки можно купить данную собственность, заложить, продать и т.д. Однако кнопки активируются по результатам обработки ячейки. Пример вывода информации об ячейке представлен на рисунке 2.2.



Рисунок 2.2 – Информация об участке “Ул. Полянка”

Описание полей:

MainWindow\* mainWindow – указатель на главное окно

Ui::CellInfo \*ui – пользовательский интерфейс, включает в себя все объекты (кнопки, поля ввода и т.д.), которые видит пользователь на экране

Cell\* cell – ячейка, информацию которой нужно вывести на экран

Описание методов:

explicit CellInfo(QWidget \*parent = nullptr,MainWindow\* mainWindow=nullptr) – конструктор

~CellInfo() – деструктор

void startSettings() – устанавливает стартовые графические настройки

bool eventFilter(QObject\* object,QEvent\*event) – обрабатывает действия

пользователя

void setButtons() – устанавливает графические настройки кнопок

void setLabels() – устанавливает графические настройки текстовых меток

void setEvent() – устанавливает графические настройки для ячейки типа Event

void setEstate() – устанавливает графические настройки для ячейки типа Estate

void setHouse() – устанавливает графические настройки для ячейки типа House

Описание сигналов и слотов:

void open(Node<Cell\*>\*cell) – применяет все настройки для показа нужной информации

void buyPressed() – слот, обрабатывающий покупку собственности

void housePressed() – слот, обрабатывающий покупку дома собственности

void mortgagePressed() – слот, обрабатывающий залог собственности

void sellHousePressed() – слот, обрабатывающий продажу дома собственности

void sellPropertyPressed() – слот, обрабатывающий продажу собственности

Класс MainWindow – описывает интерфейс главного окна, в котором происходит основной процесс игры. Также описывает некоторые методы игрового процесса. В главном окне пользователь видит игровое поле, информацию об игроках. Он может взаимодействовать с полем. Главное окно сообщает обо всех действиях, которые происходят в игре (алгоритм журнала событий). Главное окно MainWindow показано на рисунке 2.3.

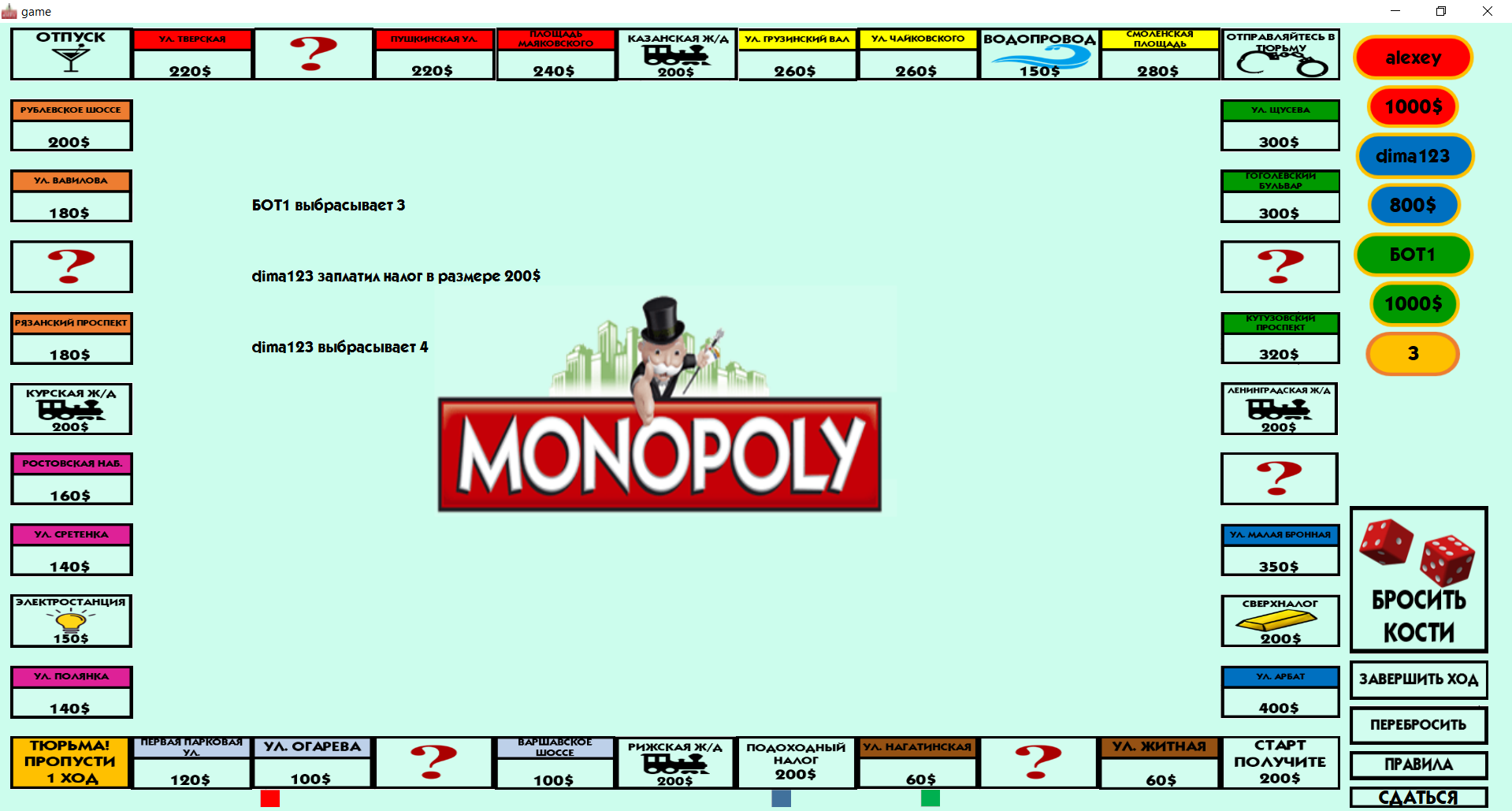


Рисунок 2.3 – Главное окно

Описание полей:

Ui::MainWindow \*ui – пользовательский интерфейс, включает в себя все объекты (кнопки, поля ввода и т.д.), которые видит пользователь на экране

CellInfo\* cellInfo – указатель на информационное окно

Data\* data – данные об игре

Описание методов:

MainWindow(QWidget \*parent = nullptr) – конструктор

~MainWindow() – деструктор

Data\* getData() – возвращает данные об игре

void connections() – метод, привязывающий сигналы к слотам

void startSettings() – устанавливает стартовые настройки графического интерфейса

void setIcon(int cellPosX,int cellPosY,QString icon,int playerID) – устанавливает заданную иконку игрока на заданную позицию

bool eventFilter(QObject\* object,QEvent\*event) – обрабатывает действия игрока

void roll() – бросок костей

void botCycle() – цикл ходов ботов

void log(QString info) – отображение на экране последних произошедших событий в игре

void updateMoney() – обновляет информацию о текущем капитале всех игроков

void botAction() – действие бота

void updateHouseLabel(Cell\*cell=nullptr,int id=0) – обновляет на каждой ячейке недвижимости количество домиков графически

void startCheck() – платит игроку 200$, если он пересек “Старт”

void prisonCheck() – садит игрока в “Тюрьму”, если он попал на определенные ячейки

void rentCheck() – выполняет некоторые действия, если аренда, которую должен заплатить игрок не равна нулю

void eventCheck() – выполняет некоторые действия, если игрок стоит на какой-либо событийной ячейке

void callChance() – случайным образом выполняется действие ячейки “Шанс”

QAbstractButton\* getButtonAt(int x,int y) – возвращает кнопку, которая привязана к определенной ячейке по ее позиции

Описание сигналов и слотов:

void openCellInfo(Node<Cell\*>\*cell) – сигнал, содержащий ячейку, информацию которой нужно вывести на экран

void receiveData(int players,int bots,std::vector<QString> names) – принимает данные от StartWindow

void press(Node<Cell\*>\*cell) – испускает сигнал openCellInfo

void rollPressed() – выполняется при нажатии кнопки “Бросить кости”

void rerollPressed() – выполняется при нажатии кнопки “Перебросить кости”

void finishPressed() – выполняется при нажатии кнопки “Завершить ход”

void forfeitPressed() – выполняется при нажатии кнопки “Сдаться”

**3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

**3.1 Разработка схем алгоритмов**

Метод rentCheck() класса MainWindow осуществляет плату аренды и делает соответствующие преобразования на экране. Схема алгоритма rentCheck() приведена в приложении Б.

Метод forfeitPressed() класса MainWindow удаляет игрока с игры, когда он сдается, и делает соответствующие преобразования на экране. Схема алгоритма forfeitPressed() приведена в приложении В.

**3.2 Разработка алгоритмов**

**3.2.1 Разработка алгоритма метода roll()**

Метод roll() класса MainWindow инициализирует бросок костей для текущего игрока.

Шаг 1. Начало.

Шаг 2. Задаем начальное значение для генератора псевдослучайных чисел с помощью переменной seed.

Шаг 3. Создаем переменную roll, присваивая ей случайное значение от 2 до 12. Это будет значение, выпавшее на кубиках.

Шаг 4. Известить пользователя соответствующими сообщениями о том, что были брошены кости.

Шаг 5. Запустить цикл for по переменной i=0.

Шаг 6. Запустить таймер на 200мс.

Шаг 7. Приостановить программу.

Шаг 8. Продолжить выполнение программы по истечении таймера.

Шаг 9. Удалить графическую модель игрока с ячейки, на которой он стоит.

Шаг 10. Передвинуть игрока на одну ячейку вперед.

Шаг 11. Установить графическую модель игрока на его позицию.

Шаг 12. Выполнение метода StartCheck() – платит игроку 200$, если он сейчас находится на “Старте”.

Шаг 13. Увеличить i на единицу.

Шаг 14. Если i<roll перейти к Шагу 5.

Шаг 15. Выполнить метод data->saveBuffer() – копирует данные об игровом поле и игроках в буфер. Это пригодится, если игрок захочет перебросить кости.

Шаг 16. Выполнение метода rentCheck() - игрок платит аренду и на экране выполняются соответствующие преобразования, если стоимость аренды не равна 0.

Шаг 17. Конец.

**3.2.2 Разработка алгоритма метода getActualRent()**

Метод getActualRent класса Data возвращает стоимость аренды для текущего игрока на его позиции. Это делается по классическим правилам игры.

Если ячейка является железнодорожной станцией (класс Estate, type=1), то аренда равна 25$, если владелец ячейки владеет одной ж/д станцией; 50$, если двумя; 100$, если тремя; 200$, если четырьмя. Если ячейка является коммунальным предприятием (класс Estate, type=2). Если владелец ячейки владеет одним коммунальным предприятием, то стоимость аренды равна четырехкратной сумме очков на кубиках, выпавшей текущему игроку; если двумя – десятикратной.

Если ячейка является недвижимостью (класс House), то аренда вычисляется следующим образом. Если владелец ячейки купил всю цветовую группу, то стоимость аренды составляет: базовая стоимость аренды, помноженная на 2, если на участке не стоит домов; некоторая сумма, определяющаяся количеством домом, стоящим на данной ячейке. Для каждой ячейки базовая аренда и аренда с учетом домов различна. Они хранятся в векторе rent.

Вышесказанную информацию пользователь может увидеть, нажав на нужную ячейку на игровом поле (см. описание класса CellInfo).

Опишем данный алгоритм по шагам:

Шаг 1. Начало.

Шаг 2. Входные данные: current – указатель на текущего игрока, roll – количество очков на кубиках, ownerID – ID владельца ячейки, на которой стоит игрок, counter – считает нужные ячейки во владении игрока по определенным условиям, temp – временный указатель на ячейку, будем проходится по всему полю. Изначально temp – ячейка, стоящая после старта.

Шаг 3. Если позиция current не является ячейкой типа Estate, перейти к Шагу 26.

Шаг 4. Создать переменную estate типа Estate, которая является текущей позицией игрока.

Шаг 5. Переменной ownerID присвоить ID владельца ячейки estate.

Шаг 6. Если ID владельца estate равен 0 (его нет) или равен ID игрока current, вернуть 0.

Шаг 7. Если estate->type (тип предприятия) не равен 1 (не ж/д станция), перейти к Шагу 17

Шаг 8. Запустить цикл for по проходу по всему игровому полю, пока temp не равен ячейке “Старт”.

Шаг 9. Если temp не имеет тип Estate перейти к Шагу 12

Шаг 10. Переменной estate присвоить значение temp.

Шаг 11. Если ID владельца estate равен ownerID и estate является ж/д станцией, увеличить counter на единицу.

Шаг 12. Переменной temp присвоить ячейку, стоящую после temp.

Шаг 13. Если temp не равен ячейке “Старт” перейти к Шагу 9.

Шаг 14. Если counter равен 3, вернуть 100.

Шаг 15. Если counter равен 4, вернуть 200.

Шаг 16. Вернуть counter\*25.

Шаг 17. Запустить цикл for по проходу по всему игровому полю, пока temp не равен ячейке “Старт”.

Шаг 18. Если temp не имеет тип Estate перейти к Шагу 21

Шаг 19. Переменной estate присвоить значение temp.

Шаг 20. Если ID владельца estate равен ownerID и estate является коммунальным предприятием (электростанция или водопровод), увеличить counter на единицу.

Шаг 21. Переменной temp присвоить ячейку, стоящую после temp.

Шаг 22. Если temp не равен ячейке “Старт” перейти к Шагу 18.

Шаг 23. Если counter равен 1, вернуть roll\*4.

Шаг 24. Если counter равен 2, вернуть roll\*10.

Шаг 25. Вернуть 0.

Шаг 26. Если позиция current не является ячейкой типа House, перейти к Шагу 32.

Шаг 27. Создать переменную house типа House, которая является текущей позицией игрока.

Шаг 28. Если метод isGrouped(ID текущего игрока, позиция текущего игрока, 2) вернет 0 перейти к Шагу 31.

Шаг 29. Если количество домов на ячейке house равно 0, вернуть базовую стоимость аренды ячейки house, помноженную на 2.

Шаг 30. Вернуть стоимость аренды house, в соответствии с количеством домов, построенных на ячейке house.

Шаг 31. Если ID владельца ячейки house не равно 0 и не равно ID текущего игрока, вернуть стоимость базовой аренды house.

Шаг 32. Вернуть 0.

**4 РЕЗУЛЬТАТЫ РАБОТЫ**

Перед началом игры пользователь должен выбрать количество игроков и ботов, ввести имена реальных игроков. Минимальное количество участников – 2, но хотя бы один реальный игрок. Подготовка к началу игры приведена на рисунке 4.1.

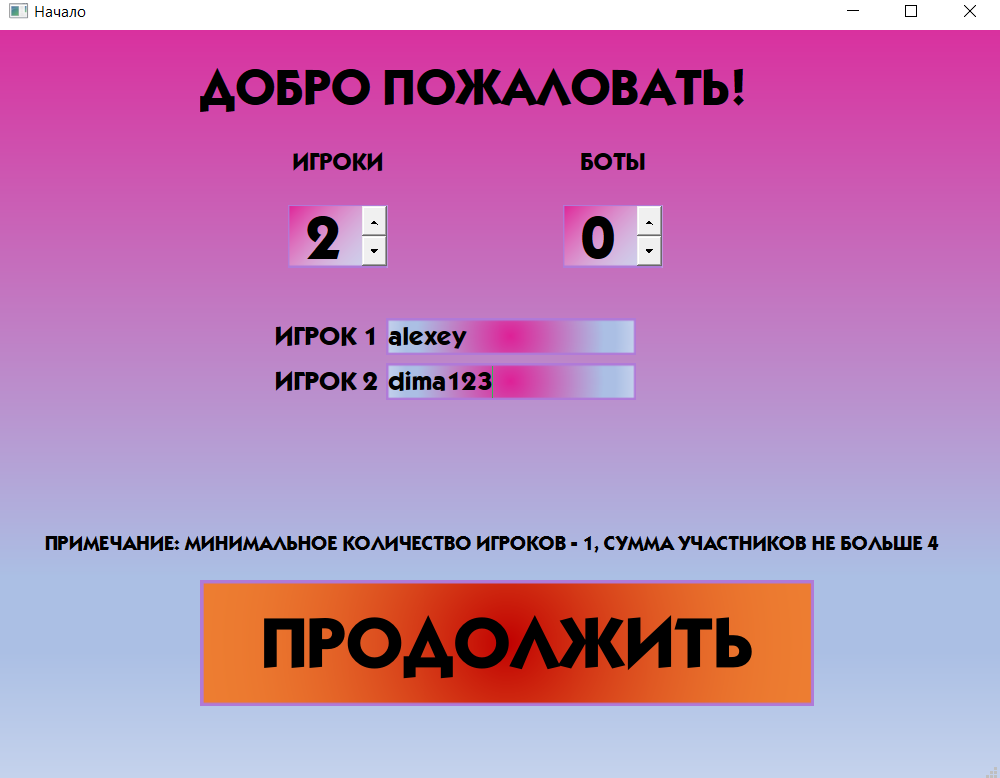


Рисунок 4.1 – Подготовка к игре

Когда игрок нажмет кнопку “Продолжить”, стартовое окно (рисунок 4.1) закроется и откроется главное окно игры, где пользователь видит игровое поле. Справа сверху игроки могут видеть информацию об игроках (имена, текущий капитал). Справа снизу находятся кнопки, которые выполняют действия, написанные на этих кнопках. Главное окно приведено на рисунке 4.2.

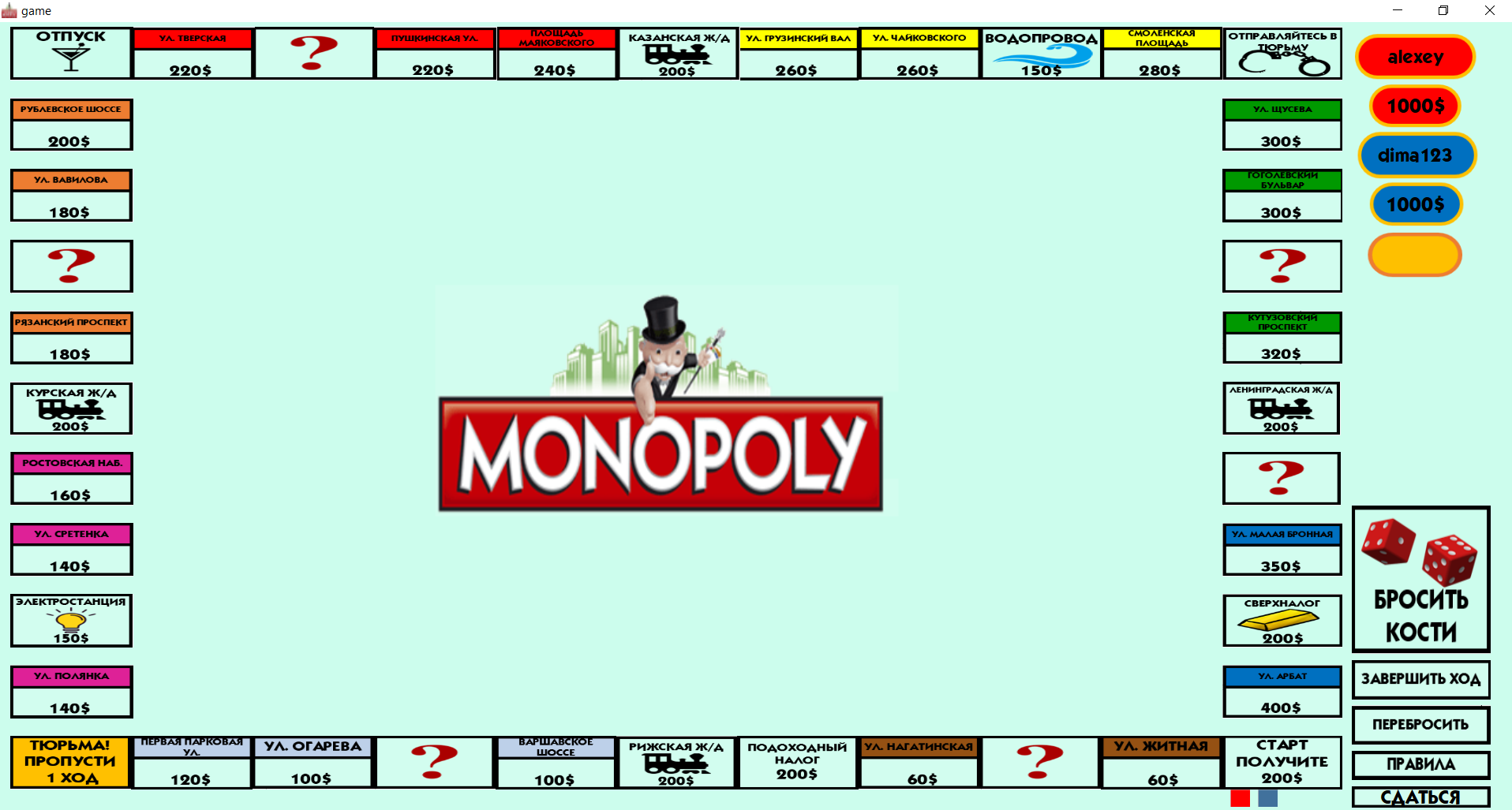


Рисунок 4.2 – Начало игры

Игрок может смотреть информацию об ячейке, просто нажав на нее. Тогда откроется информационное окно (объект CellInfo). В данном окне игрок также может купить или продать собственность, если ячейка таковою является.

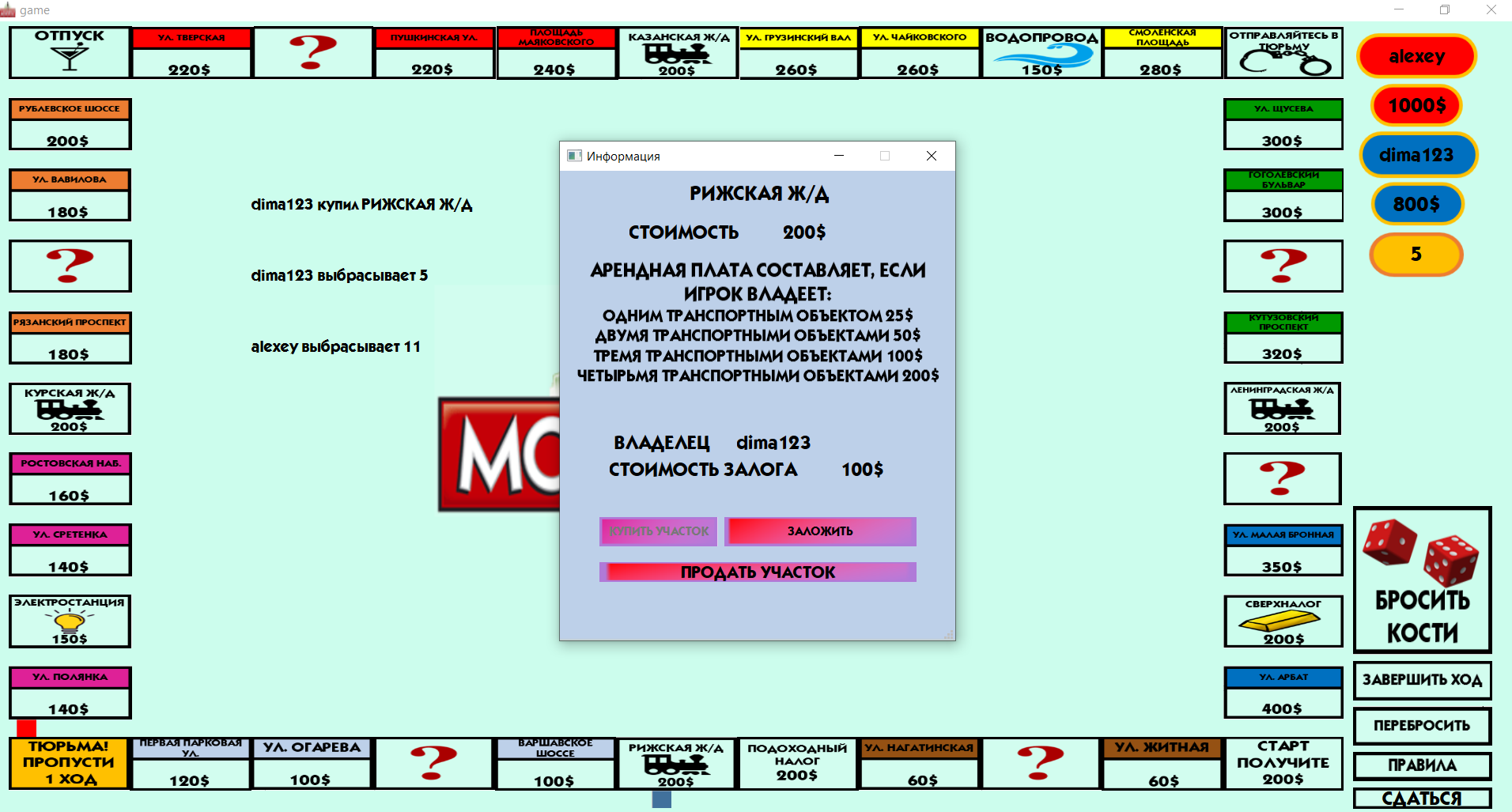


Рисунок 4.3 – Игрок dima123 только что купил Рижскую железную дорогу

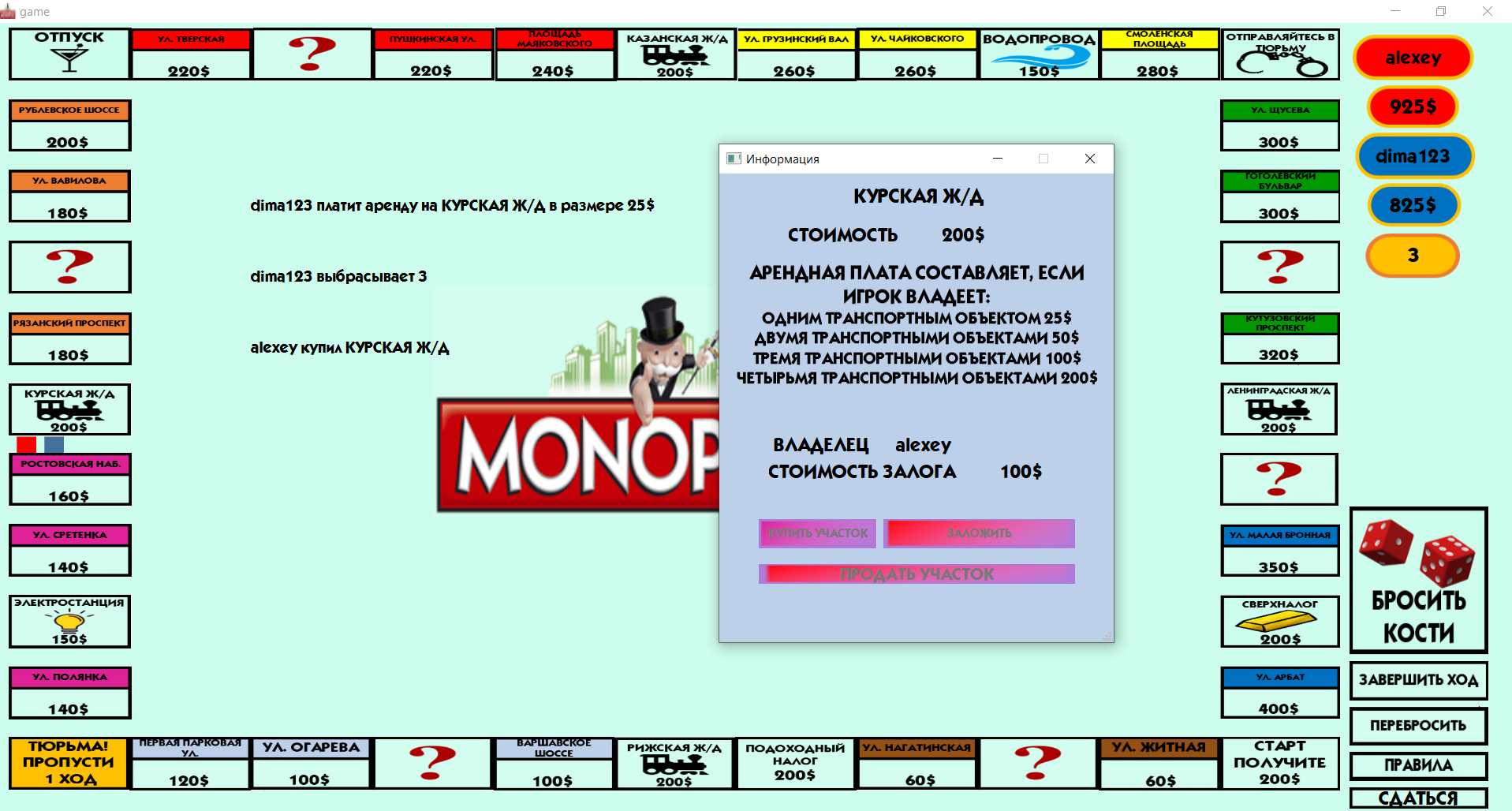


Рисунок 4.4 – Игрок dima123 заплатил аренду в размере 25$ игроку alexey,

он попал на его участок. Об этом сообщает игра.

**ЗАКЛЮЧЕНИЕ**

В процессе выполнения курсовой работы были закреплены знания основ программирования и алгоритмизации. Были приобретены знания в объектно-ориентированном программировании. Также я познакомился с созданием полноценных проектов на фреймворке Qt и убедился, что он идеально подходит для создания несложных проектов с удобным интерфейсом.

В процессе разработки игры “Монополия” были реализованы алгоритмы отмены последнего действия, журнала событий, броска костей и т. д. Как упоминалось раннее, моя игра является неполной версией оригинальной игры “Монополия”. Однако она открыта для модернизации. Например, можно создать новый виджет, где игроки смогут обмениваться недвижимостью.

Игра оказалась простой в изучении, имеет удобный интерфейс. В нее можно играть одному с ботами или с друзьями с одного компьютера.

На основе полученных знаний при создании игры можно несложно будет создать приложения, где от пользователя будет требоваться ввод каких-либо данных и нажимание кнопок на экране. Такими примерами могут быть приложение для ведения заметок, ежедневник и т.д.

Среда разработки – QtCreator.

Код программы приведен в приложении Г.

**СПИСОК ЛИТЕРАТУРЫ**

1. Язык программирования С++. Лекции и упражнения. – Стивен Прата.

2. Конструирование программ и языки программирования: метод. указания по курсовому проектированию – А. В. Бушкевич, А. М. Ковальчук, И. В. Лукьянова. – Минск : БГУИР, 2009.

3. Qt 5.10. Профессиональное программирование на С++ – М. Шлее.

# ПРИЛОЖЕНИЕ А

*(обязательное)*

Диаграмма классов

# ПРИЛОЖЕНИЕ Б

*(обязательное)*

Схема метода rentCheck()

# ПРИЛОЖЕНИЕ В

*(обязательное)*

Схема метода forfeitPressed()

# ПРИЛОЖЕНИЕ Г

*(обязательное)*

Код программы

#include "entity.h"

class Data;

class Bot:public Entity

{

public:

Bot(int id=0,int money=0,Node<Cell\*>\*pos=nullptr,int skip=0, QString icon="",QString name="");

Bot(const Bot&bot);

~Bot();

void buy() override;

};

#include <QString>

class Estate;

class House;

class Event;

class Cell

{

protected:

QString name;

int posX;

int posY;

public:

Cell(QString name="",int posX=0,int posY=0);

Cell(const Cell& cell);

~Cell();

void setPosX(int posX);

void setPosY(int posY);

void setName(QString name);

bool isEstate();

bool isHouse();

bool isEvent();

int&getPosX();

int&getPosY();

QString&getName();

virtual void show()=0;

};

template<typename T> struct Node;

class Cell;

class MainWindow;

#include <QMainWindow>

namespace Ui {

class CellInfo;

}

class CellInfo : public QMainWindow

{

Q\_OBJECT

private:

MainWindow\* mainWindow;

Ui::CellInfo \*ui;

Cell\* cell;

public:

explicit CellInfo(QWidget \*parent = nullptr,MainWindow\* mainWindow=nullptr);

~CellInfo();

void startSettings();

bool eventFilter(QObject\* object,QEvent\*event);

void setButtons();

void setLabels();

void setEvent();

void setEstate();

void setHouse();

signals:

public slots:

void open(Node<Cell\*>\*cell);

private slots:

void buyPressed();

void housePressed();

void mortgagePressed();

void sellHousePressed();

void sellPropertyPressed();

};

#include "ring.h"

#include <iostream>

#include <vector>

#include <QDebug>

#include <QTime>

class Data

{

private:

Ring<Cell\*> field;

Ring<Entity\*> players;

Node<Entity\*>\*current;

std::vector<std::vector<QString>> images;

std::vector<Point> points;

int seed;

Ring<Cell\*> field\_buffer;

Ring<Entity\*> players\_buffer;

public:

Data();

Data(int players,int bots,std::vector<QString> names);

Data(const Data& data);

~Data();

void setImages();

void setPoints();

void setSeed(int seed);

void saveBuffer();

void loadBuffer();

void copyCells(Cell\* dest,Cell\*source);

int& getSeed();

std::vector<std::vector<QString>>& getImages();

Ring<Entity\*>& getPlayers();

Ring<Cell\*>& getField();

void setNextCurrent();

bool isGrouped(int id,Cell\*cell,int actionStatus);

int getActualRent();

void copyField(Ring<Cell\*>& destination, Ring<Cell\*>& source);

void copyPlayers(Ring<Entity\*>& destination, Ring<Entity\*>& source);

bool isFree(Entity\*player);

void setCurrent(Node<Entity\*>\*current);

Node<Entity\*>\* getCurrent();

Node<Entity\*>\* getByID(int id);

void updateAfterForfeit(int id);

Node<Cell\*>\* getCellByPos(int posX,int posY);

};

#include <iostream>

#include "ring.h"

#include <QString>

#include "house.h"

#include "event.h"

class Entity

{

protected:

int id;

int money; //current money

Node<Cell\*>\* pos; //position on the map

int skip;

QString icon;

QString name;

public:

Entity(int id,int money,Node<Cell\*>\* pos,int skip, QString icon,QString name);

Entity(const Entity&entity);

Entity& operator=(const Entity&source);

~Entity();

void setID(int id);

void setMoney(int money);

void setPos(Node<Cell\*>\*pos);

void setSkip(int skip);

void setIcon(QString icon);

void setName(QString name);

QString& getIcon();

QString& getName();

int& getSkip();

int& getID();

int& getMoney();

Node<Cell\*>\* getPos();

bool isEnough(int cost);

bool mortgage(Cell\*cell);

bool buyHouse(Cell\*cell);

bool sellHouse(Cell\*cell);

void sell(Cell\*cell);

virtual void buy()=0;

void pay(int rent);

bool isPlayer();

};

#include "cell.h"

#include <iostream>

class Estate:public Cell

{

protected:

int ownerID;

int cost; //base cost

std::vector<int> rent;

bool mortgage;

int type;//1-trainSt 2-publicService

public:

Estate(int ownerID=0,int cost=0,std::vector<int> rent={},bool mortgage=0,int type=0, QString name="",int posX=0,int posY=0);

Estate(const Estate& estate);

Estate& operator=(const Estate&estate);

~Estate();

void setOwner(int ownerID);

void setCost(int cost);

void setRent(std::vector<int> rent);

void setType(int type);

void setMortgage(bool value);

bool isMortgage();

int& getType();

int& getOwnerID();

int& getCost();

std::vector<int>& getRent();

virtual void show() override;

};

#include "cell.h"

class Event:public Cell

{

protected:

int gift;

int skip;

int chance;

public:

Event(int gift=0,int skip=0,int chance=0, QString name="",int posX=0,int posY=0);

Event(const Event& event);

Event& operator=(const Event&event);

void setGift(int gift);

void setSkip(int skip);

void setChance(int chance);

int& getGift();

int& getSkip();

int& getChance();

virtual void show() override;

~Event();

};

#include "estate.h"

class House:public Estate

{

private:

int houses;

int group;

public:

House(int ownerID=0,int cost=0,std::vector<int> rent={},bool mortgage=0,int houses=0,int group=0, QString name="",int posX=0,int posY=0);

House(const House& house);

House& operator=(const House&house);

void setHouses(int houses);

void setGroup(int group);

int& getHouses();

int& getGroup();

int getHouseCost();

~House();

bool isHousesLimit();

virtual void show() override;

};

#include <QMainWindow>

#include <QTabWidget>

#include <QTabBar>

#include "data.h"

#include "cellinfo.h"

#include <QLineEdit>

#include <QDateTime>

#include <QtGlobal>

#include <QRandomGenerator>

#include <QTimer>

#include <windows.h>

#include <QThread>

#include <QAbstractButton>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

class CellInfo;

class MainWindow : public QMainWindow

{

Q\_OBJECT

private:

Ui::MainWindow \*ui;

CellInfo\* cellInfo;

Data\* data;

public:

MainWindow(QWidget \*parent = nullptr);

~MainWindow();

Data\* getData();

void connections();

void startSettings();

void setIcon(int cellPosX,int cellPosY,QString icon,int playerID);

bool eventFilter(QObject\* object,QEvent\*event);

void roll();

void botCycle();

void log(QString info);

void updateMoney();

void botAction();

void updateHouseLabel(Cell\*cell=nullptr,int id=0);

void startCheck();

void prisonCheck();

void rentCheck();

void eventCheck();

void callChance();

QAbstractButton\* getButtonAt(int x,int y);

signals:

void openCellInfo(Node<Cell\*>\*cell);

public slots:

void receiveData(int players,int bots,std::vector<QString> names);

private slots:

void press(Node<Cell\*>\*cell);

void rollPressed();

void rerollPressed();

void finishPressed();

void forfeitPressed();

};

#include "entity.h"

class Player:public Entity

{

public:

Player(int id=0,int money=0,Node<Cell\*>\*pos=nullptr,int skip=0,QString icon="",QString name="");

Player(const Player& player);

~Player();

void buy() override;

};

template <typename T>

struct Node{

T value;

Node\* next;

Node\* prev;

};

template <typename T>

class Ring

{

private:

Node<T>\* first;

int size;

public:

Ring() {

first = nullptr;

size=0;

}

Ring(const Ring& ring)

{

first = new Node<T>;

this->first = ring.first;

this->size=ring.size;

}

void popValue(T value) {

Node<T>\* target = getValue(value);

if (first == nullptr) {

return;

}

if (target->next == target) {

delete target;

this->first = nullptr;

size--;

}

else {

target->prev->next = target->next;

target->next->prev = target->prev;

if (target == first) first = target->next;

delete target;

size--;

}

}

~Ring() {

while (first) {

popValue(first->value);

}

}

void push(T value) {

Node<T>\* temp = new Node<T>;

temp->value = value;

size++;

if (!first) {

temp->next = temp->prev = temp;

first = temp;

}

else {

temp->next = first;

temp->prev = first->prev;

first->prev->next = temp;

first->prev = temp;

}

}

Node<T>\* getValue(T value) {

Node<T>\* temp = first->next;

for (; temp!=first; temp=temp->next) {

if (temp->value == value) return temp;

}

}

Node<T>\* getFirst()

{

return first;

}

Node<T>\* getNext(Node<T>\* current) {

return current->next;

}

Node<T>\* getNextN(Node<T>\* current, int N)

{

for (int i = 0; i < N; i++) {

current = getNext(current);

}

return current;

}

Node<T>\* getPrev(Node<T>\* current) {

return current->prev;

}

Node<T>\* getPrevN(Node<T>\* current, int N) {

for (int i = 0; i < N; i++) {

current = getPrev(current);

}

return current;

}

int&getSize(){

return size;

}

void deleteRing(){

while(first){

popValue(first->value);

}

}

};

#include <QMainWindow>

#include <QFormLayout>

#include "mainwindow.h"

namespace Ui {

class StartWindow;

}

class StartWindow : public QMainWindow

{

Q\_OBJECT

public:

explicit StartWindow(QWidget \*parent = nullptr);

~StartWindow();

void startSettings();

bool eventFilter(QObject\* object,QEvent\*event);

int& getPlayers();

int& getBots();

QFormLayout\* getFormLayout();

private:

Ui::StartWindow \*ui;

MainWindow\* mainWindow;

int players;

int bots;

private slots:

void showLineEdit();

void botSpinBox();

void emitData();

signals:

void sendData(int players,int bots,std::vector<QString> names);

};

#include "cell.h"

#include "house.h"

#include "event.h"

#include <iostream>

Cell::Cell(QString name,int posX,int posY)

{

this->name=name;

this->posX=posX;

this->posY=posY;

}

Cell::Cell(const Cell &cell)

{

this->name=cell.name;

this->posX=cell.posX;

this->posY=cell.posY;

}

Cell::~Cell()

{

}

void Cell::setPosX(int posX)

{

this->posX=posX;

}

void Cell::setPosY(int posY)

{

this->posY=posY;

}

void Cell::setName(QString name){

this->name=name;

}

int& Cell::getPosX()

{

return posX;

}

int& Cell::getPosY()

{

return posY;

}

QString& Cell::getName(){

return name;

}

#include "mainwindow.h"

#include "cellinfo.h"

#include "ui\_cellinfo.h"

CellInfo::CellInfo(QWidget \*parent,MainWindow\* mainWindow) :

QMainWindow(parent),

ui(new Ui::CellInfo)

{

ui->setupUi(this);

this->mainWindow=mainWindow;

cell=nullptr;

}

CellInfo::~CellInfo()

{

delete ui;

}

void CellInfo::startSettings()

{

setStyleSheet("QMainWindow{border-image:url(:images/cellinfobackground.png); stretch}");

ui->buyButton->setStyleSheet("border-image: url(:/images/buy.png) stretch; ");

ui->houseButton->setStyleSheet("border-image: url(:/images/buy.png) stretch; ");

ui->mortgageButton->setStyleSheet("border-image: url(:/images/sell.png) stretch; ");

ui->sellButton->setStyleSheet("border-image: url(:/images/sell.png) stretch; ");

ui->sellhouseButton->setStyleSheet("border-image: url(:/images/sell.png) stretch; ");

ui->buyButton->setText("КУПИТЬ УЧАСТОК");

ui->houseButton->setText("КУПИТЬ ДОМ");

ui->mortgageButton->setText("ЗАЛОЖИТЬ");

ui->sellButton->setText("ПРОДАТЬ УЧАСТОК");

ui->sellhouseButton->setText("ПРОДАТЬ ДОМ");

ui->buyButton->installEventFilter(this);

ui->houseButton->installEventFilter(this);

ui->mortgageButton->installEventFilter(this);

ui->sellButton->installEventFilter(this);

ui->sellhouseButton->installEventFilter(this);

connect(ui->buyButton,&QAbstractButton::clicked,this,&CellInfo::buyPressed);

connect(ui->mortgageButton,&QAbstractButton::clicked,this,&CellInfo::mortgagePressed);

connect(ui->houseButton,&QAbstractButton::clicked,this,&CellInfo::housePressed);

connect(ui->sellhouseButton,&QAbstractButton::clicked,this,&CellInfo::sellHousePressed);

connect(ui->sellButton,&QAbstractButton::clicked,this,&CellInfo::sellPropertyPressed);

}

bool CellInfo::eventFilter(QObject \*object, QEvent \*event)

{

QAbstractButton\*button=qobject\_cast<QAbstractButton\*>(object);

if((button==ui->buyButton||button==ui->houseButton)&&event->type()==QEvent::Enter&&button->isEnabled()) button->setStyleSheet("border-image: url(:/images/buyenter.png) stretch; ");

else if((button==ui->buyButton||button==ui->houseButton)&&event->type()==QEvent::Leave&&button->isEnabled()) button->setStyleSheet("border-image: url(:/images/buy.png) stretch; ");

else if((button==ui->mortgageButton||button==ui->sellButton||button==ui->sellhouseButton)&&event->type()==QEvent::Enter&&button->isEnabled()) button->setStyleSheet("border-image: url(:/images/sellenter.png) stretch; ");

else if((button==ui->mortgageButton||button==ui->sellButton||button==ui->sellhouseButton)&&event->type()==QEvent::Leave&&button->isEnabled()) button->setStyleSheet("border-image: url(:/images/sell.png) stretch; ");

return false;

}

void CellInfo::setButtons()

{

ui->mortgageButton->setDisabled(1);

ui->buyButton->setDisabled(1);

ui->houseButton->setDisabled(1);

ui->sellButton->setDisabled(1);

ui->sellhouseButton->setDisabled(1);

setEvent();

setEstate();

setHouse();

}

void CellInfo::setLabels()

{

QString buffer=QString::number(123);

ui->name->setText(cell->getName());

Node<Entity\*>\*temp;

if(cell->isHouse()){

ui->rentlabel->show();

ui->rent->show();

ui->costlabel->show();

ui->cost->show();

ui->name->show();

ui->house1label->show();

ui->house2label->show();

ui->house3label->show();

ui->house4label->show();

ui->hotellabel->show();

ui->house1->show();

ui->house2->show();

ui->house3->show();

ui->house4->show();

ui->hotel->show();

ui->housecostlabel->show();

ui->housecost->show();

ui->mortgage->show();

ui->mortgagelabel->show();

ui->buyButton->show();

ui->houseButton->show();

ui->mortgageButton->show();

ui->ownerlabel->show();

ui->owner->show();

House\*house=dynamic\_cast<House\*>(cell);

buffer=QString::number(house->getCost());

ui->cost->setText(buffer+'$');

buffer=QString::number(house->getCost()/2);

ui->mortgage->setText(buffer+'$');

buffer=QString::number(house->getHouseCost());

ui->housecost->setText(buffer+'$');

ui->rentlabel->setText("АРЕНДА");

ui->house1label->setText("С ОДНИМ ДОМОМ");

ui->house2label->setText("С ДВУМЯ ДОМАМИ");

ui->house3label->setText("С ТРЕМЯ ДОМАМИ");

ui->house4label->setText("С ЧЕТЫРЬМЯ ДОМАМИ");

ui->hotellabel->setText("С ОДНИМ ОТЕЛЕМ");

setStyleSheet("QMainWindow{border-image:url(:images/cellinfobackground.png); stretch}");

buffer=QString::number(house->getRent()[0]);

ui->rent->setText(buffer+'$');

buffer=QString::number(house->getRent()[1]);

ui->house1->setText(buffer+'$');

buffer=QString::number(house->getRent()[2]);

ui->house2->setText(buffer+'$');

buffer=QString::number(house->getRent()[3]);

ui->house3->setText(buffer+'$');

buffer=QString::number(house->getRent()[4]);

ui->house4->setText(buffer+'$');

buffer=QString::number(house->getRent()[5]);

ui->hotel->setText(buffer+'$');

temp=mainWindow->getData()->getByID(house->getOwnerID());

if(temp&&house->isMortgage()){

ui->mortgageButton->setText("ВЫКУПИТЬ");

ui->owner->setText(temp->value->getName()+" (УЧАСТОК ЗАЛОЖЕН)");

}

else if(temp){

ui->mortgageButton->setText("ЗАЛОЖИТЬ");

ui->owner->setText(temp->value->getName());

}

else ui->owner->setText("-");

}

else{

ui->rentlabel->hide();

ui->rent->hide();

ui->house1label->hide();

ui->house2label->hide();

ui->house3label->hide();

ui->house4label->hide();

ui->hotellabel->hide();

ui->house1->hide();

ui->house2->hide();

ui->house3->hide();

ui->house4->hide();

ui->hotel->hide();

ui->housecostlabel->hide();

ui->housecost->hide();

if(cell->isEstate()){

Estate\*estate=dynamic\_cast<Estate\*>(cell);

buffer=QString::number(estate->getCost());

ui->mortgage->show();

ui->mortgagelabel->show();

ui->costlabel->show();

ui->cost->show();

ui->name->show();

ui->owner->show();

ui->ownerlabel->show();

ui->cost->setText(buffer+'$');

buffer=QString::number(estate->getCost()/2);

ui->mortgage->setText(buffer+'$');

temp=mainWindow->getData()->getByID(estate->getOwnerID());

if(temp&&estate->isMortgage()){

ui->mortgageButton->setText("ВЫКУПИТЬ");

ui->owner->setText(temp->value->getName()+" (УЧАСТОК ЗАЛОЖЕН)");

}

else if(temp){

ui->mortgageButton->setText("ЗАЛОЖИТЬ");

ui->owner->setText(temp->value->getName());

}

else ui->owner->setText("-");

if(estate->getType()==1){

setStyleSheet("QMainWindow{border-image:url(:images/trainbackground.png); stretch}");

}

else{

setStyleSheet("QMainWindow{border-image:url(:images/supplybackground.png); stretch}");

}

}

else{

Event\*event=dynamic\_cast<Event\*>(cell);

ui->mortgage->hide();

ui->mortgagelabel->hide();

ui->costlabel->hide();

ui->cost->hide();

ui->name->hide();

ui->owner->hide();

ui->ownerlabel->hide();

if(event->getSkip()==1) setStyleSheet("QMainWindow{border-image:url(:images/prisonbackground.png); stretch}");

else if(event->getChance()==1) setStyleSheet("QMainWindow{border-image:url(:images/chancebackground.png); stretch}");

else if(event->getGift()>0) setStyleSheet("QMainWindow{border-image:url(:images/startbackground.png); stretch}");

else if(event->getGift()<0) setStyleSheet("QMainWindow{border-image:url(:images/taxbackground.png); stretch}");

else setStyleSheet("QMainWindow{border-image:url(:images/relaxbackground.png); stretch}");

}

}

}

void CellInfo::setEvent()

{

if(cell->isEvent()){

ui->mortgageButton->hide();

ui->buyButton->hide();

ui->houseButton->hide();

ui->sellButton->hide();

ui->sellhouseButton->hide();

}

}

void CellInfo::setEstate()

{

if(!cell->isEstate()) return;

ui->mortgageButton->show();

ui->buyButton->show();

ui->sellButton->show();

ui->houseButton->hide();

ui->sellhouseButton->hide();

Estate\* estate=dynamic\_cast<Estate\*>(cell);

if(mainWindow->getData()->getCurrent()->value->isPlayer()){

if(estate->getOwnerID()==0&&mainWindow->getData()->getCurrent()->value->getPos()->value==cell

&&mainWindow->getData()->getCurrent()->value->isEnough(estate->getCost())) ui->buyButton->setDisabled(0);

if(estate->getOwnerID()==mainWindow->getData()->getCurrent()->value->getID()){

if((!estate->isMortgage())

||(estate->isMortgage()&&mainWindow->getData()->getCurrent()->value->isEnough(estate->getCost()\*0.55))) ui->mortgageButton->setDisabled(0);

if(!estate->isMortgage()) ui->sellButton->setDisabled(0);

}

}

}

void CellInfo::setHouse()

{

if(!cell->isHouse()) return;

ui->mortgageButton->show();

ui->buyButton->show();

ui->houseButton->show();

ui->sellButton->show();

ui->sellhouseButton->show();

House\* house=dynamic\_cast<House\*>(cell);

if(mainWindow->getData()->getCurrent()->value->isPlayer()){

if(house->getOwnerID()==0&&mainWindow->getData()->getCurrent()->value->getPos()->value==cell

&&mainWindow->getData()->getCurrent()->value->isEnough(house->getCost())) ui->buyButton->setDisabled(0);

if(house->getOwnerID()==mainWindow->getData()->getCurrent()->value->getID()){

if((!house->isMortgage())

||(house->isMortgage()&&mainWindow->getData()->getCurrent()->value->isEnough(house->getCost()\*0.55))) ui->mortgageButton->setDisabled(0);

if(mainWindow->getData()->isGrouped(mainWindow->getData()->getCurrent()->value->getID(),cell,1)&&!house->isHousesLimit()

&&mainWindow->getData()->getCurrent()->value->isEnough(house->getHouseCost())) ui->houseButton->setDisabled(0);

if(mainWindow->getData()->isGrouped(mainWindow->getData()->getCurrent()->value->getID(),cell,0)){

if(!house->isMortgage()){

if(house->getHouses()!=0){

ui->sellButton->setDisabled(1);

ui->sellhouseButton->setDisabled(0);

}

else{

ui->sellButton->setDisabled(0);

ui->sellhouseButton->setDisabled(1);

}

}

}

}

}

}

void CellInfo::open(Node<Cell \*> \*cell)

{

this->cell=cell->value;

setButtons();

setLabels();

}

void CellInfo::buyPressed()

{

hide();

mainWindow->getData()->getCurrent()->value->buy();

mainWindow->log(mainWindow->getData()->getCurrent()->value->getName()+" купил "+mainWindow->getData()->getCurrent()->value->getPos()->value->getName());

mainWindow->updateMoney();

}

void CellInfo::housePressed()

{

QString status=" купил дом на ";

hide();

if(mainWindow->getData()->getCurrent()->value->buyHouse(cell)) status=" купил отель на ";

mainWindow->log(mainWindow->getData()->getCurrent()->value->getName()+status+mainWindow->getData()->getCurrent()->value->getPos()->value->getName());

mainWindow->updateMoney();

mainWindow->updateHouseLabel(cell,0);

}

void CellInfo::mortgagePressed()

{

QString status=" выкупил ";

hide();

if(mainWindow->getData()->getCurrent()->value->mortgage(cell)) status=" заложил ";

mainWindow->log(mainWindow->getData()->getCurrent()->value->getName()+status+mainWindow->getData()->getCurrent()->value->getPos()->value->getName());

mainWindow->updateMoney();

}

void CellInfo::sellHousePressed()

{

QString status=" продал дом на ";

hide();

if(mainWindow->getData()->getCurrent()->value->sellHouse(cell)) status=" продал отель на ";

mainWindow->log(mainWindow->getData()->getCurrent()->value->getName()+status+mainWindow->getData()->getCurrent()->value->getPos()->value->getName());

mainWindow->updateMoney();

mainWindow->updateHouseLabel(cell,0);

}

void CellInfo::sellPropertyPressed()

{

hide();

mainWindow->getData()->getCurrent()->value->sell(cell);

mainWindow->log(mainWindow->getData()->getCurrent()->value->getName()+" продал "+mainWindow->getData()->getCurrent()->value->getPos()->value->getName());

mainWindow->updateMoney();

}

#include "data.h"

Data::Data(int players,int bots,std::vector<QString> names)

{

seed=QTime::currentTime().msec();

setImages();

std::vector<QString> icons={"border-image: url(:/images/player1icon.png) stretch; ","border-image: url(:/images/player2icon.png) stretch; ",

"border-image: url(:/images/player3icon.png) stretch; ","border-image: url(:/images/player4icon.png) stretch; "

};

int id = 1;

Player player;

Bot bot;

Estate estate;

Event event;

House house;

QString botname="БОТ";

event={200,0,0,"",10,10}; //start

field.push(new Event (event));

field\_buffer.push(new Event(event));

Node<Cell\*>\* first=field.getFirst();

for(int i=0;i<players;i++){

player={id,1000,first,0,icons[i],names[i]};

this->players.push(new Player (player));

players\_buffer.push(new Player(player));

id++;

}

for (int i = 0; i <bots; i++) {

bot = {id,1000,first,0,icons[players+i],botname+QString::number(i+1) };

this->players.push(new Bot (bot));

players\_buffer.push(new Bot(bot));

id++;

}

current=this->players.getFirst();

house={0,60,{2,10,30,90,160,250},0,0,1,"ЖИТНАЯ УЛ.",10,9}; //житная

field.push(new House (house));

field\_buffer.push(new House(house));

event={0,0,1,"ШАНС",10,8}; //шанс

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,60,{4,20,60,180,320,450},0,0,1,"НАГАТИНСКАЯ УЛ.",10,7}; //нагатинская

field.push(new House (house));

field\_buffer.push(new House(house));

event={-200,0,0,"ПОДОХОДНЫЙ НАЛОГ",10,6}; //подоходный налог

field.push(new Event (event));

field\_buffer.push(new Event(event));

estate={0,200,{25,50,100,200},0,1,"РИЖСКАЯ Ж/Д",10,5}; //жд1

field.push(new Estate (estate));

field\_buffer.push(new Estate (estate));

house={0,100,{6,30,90,270,400,550},0,0,2,"ВАРШАВСКОЕ ШОССЕ",10,4}; //варшавское шоссе

field.push(new House (house));

field\_buffer.push(new House(house));

event={0,0,1,"ШАНС",10,3}; //шанс

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,100,{6,30,90,270,400,550},0,0,2,"УЛ. ОГАРЕВА",10,2}; //огарева

field.push(new House (house));

field\_buffer.push(new House(house));

house={0,120,{8,40,100,300,450,600},0,0,2,"ПЕРВАЯ ПАРКОВАЯ УЛ.",10,1}; //первая парковая

field.push(new House (house));

field\_buffer.push(new House(house));

event={0,1,0,"ТЮРЬМА",10,0}; //тюрьма

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,140,{10,50,150,450,625,750},0,0,3,"УЛ. ПОЛЯНКА",9,0}; //полянка

field.push(new House (house));

field\_buffer.push(new House(house));

estate={0,150,{4,10},0,2,"ЭЛЕКТРОСТАНЦИЯ",8,0}; //электроэнергия

field.push(new Estate (estate));

field\_buffer.push(new Estate (estate));

house={0,140,{10,50,150,450,625,750},0,0,3,"УЛ. СРЕТЕНКА",7,0}; //сретенка

field.push(new House (house));

field\_buffer.push(new House(house));

house={0,160,{12,60,180,500,700,900},0,0,3,"РОСТОВСКАЯ НАБЕРЕЖНАЯ",6,0}; //ростовкая набережная

field.push(new House (house));

field\_buffer.push(new House(house));

estate={0,200,{25,50,100,200},0,1,"КУРСКАЯ Ж/Д",5,0}; //жд2

field.push(new Estate (estate));

field\_buffer.push(new Estate (estate));

house={0,180,{14,70,200,550,750,950},0,0,4,"РЯЗАНСКИЙ ПРОСПЕКТ",4,0}; //рязанский проспект

field.push(new House (house));

field\_buffer.push(new House(house));

event={0,0,1,"ШАНС",3,0}; //шанс

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,180,{14,70,200,550,750,950},0,0,4,"УЛ. ВАВИЛОВА",2,0}; //вавилова

field.push(new House (house));

field\_buffer.push(new House(house));

house={0,200,{16,80,220,600,800,1000},0,0,4,"РУБЛЕВСКОЕ ШОССЕ",1,0}; //рублевское шоссе

field.push(new House (house));

field\_buffer.push(new House(house));

event={0,0,0,"ОТДЫХ",0,0}; //бесплатная парковка

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,220,{18,90,250,700,875,1050},0,0,5,"УЛ. ТВЕРСКАЯ",0,1}; //тверская

field.push(new House (house));

field\_buffer.push(new House(house));

event={0,0,1,"ШАНС",0,2}; //шанс

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,220,{18,90,250,700,875,1050},0,0,5,"ПУШКИНСКАЯ УЛ.",0,3}; //пушкинская

field.push(new House (house));

field\_buffer.push(new House(house));

house={0,240,{20,100,300,750,925,1100},0,0,5,"ПЛОЩАДЬ МАЯКОВСКОГО",0,4}; //площадь маяковского

field.push(new House (house));

field\_buffer.push(new House(house));

estate={0,200,{25,50,100,200},0,1,"КАЗАНСКАЯ Ж/Д",0,5}; //жд3

field.push(new Estate (estate));

field\_buffer.push(new Estate (estate));

house={0,260,{22,110,330,800,975,1150},0,0,6,"УЛ. ГРУЗИНСКИЙ ВАЛ",0,6}; //грузинский вал

field.push(new House (house));

field\_buffer.push(new House(house));

house={0,260,{22,110,330,800,975,1150},0,0,6,"УЛ. ЧАЙКОВСКОГО",0,7}; //чайковского

field.push(new House (house));

field\_buffer.push(new House(house));

estate={0,150,{4,10},0,2,"ВОДОПРОВОД",0,8}; //водопровод

field.push(new Estate (estate));

field\_buffer.push(new Estate (estate));

house={0,280,{24,120,360,850,1025,1200},0,0,6,"СМОЛЕНСКАЯ ПЛОЩАДЬ",0,9}; //смоленская площадь

field.push(new House (house));

field\_buffer.push(new House(house));

event={0,1,0,"ТЮРЬМА",0,10}; //тюрьма

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,300,{26,130,390,900,1100,1275},0,0,7,"УЛ. ЩУСЕВА",1,10}; //щусева

field.push(new House (house));

field\_buffer.push(new House(house));

house={0,300,{26,130,390,900,1100,1275},0,0,7,"ГОГОЛЕВСКИЙ БУЛЬВАР",2,10}; //гоголевский бульвар

field.push(new House (house));

field\_buffer.push(new House(house));

event={0,0,1,"ШАНС",3,10}; //шанс

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,320,{28,150,450,1000,1200,1400},0,0,7,"КУТУЗОВСКИЙ ПРОСПЕКТ",4,10}; //кутузовский проспект

field.push(new House (house));

field\_buffer.push(new House(house));

estate={0,200,{25,50,100,200},0,1,"ЛЕНИНГРАДСКАЯ Ж/Д",5,10}; //жд4

field.push(new Estate (estate));

field\_buffer.push(new Estate (estate));

event={0,0,1,"ШАНС",6,10}; //шанс

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,350,{35,175,500,1100,1300,1500},0,0,8,"УЛ. МАЛАЯ БРОННАЯ",7,10}; //малая бронная

field.push(new House (house));

field\_buffer.push(new House(house));

event={-100,0,0,"СВЕРХНАЛОГ",8,10}; //сверхналог

field.push(new Event (event));

field\_buffer.push(new Event(event));

house={0,400,{50,200,600,1400,1700,2000},0,0,8,"УЛ. АРБАТ",9,10}; //арбат

field.push(new House (house));

field\_buffer.push(new House(house));

}

Data::Data(const Data &data)

{

this->players=data.players;

this->field=data.field;

this->current=data.current;

}

Data::~Data()

{

}

void Data::setNextCurrent()

{

current=current->next;

for(;current->value->getSkip()==1;current=current->next){

if(current->value->getSkip()==1) current->value->setSkip(0);

}

}

void Data::setImages()

{

images={

{"border-image: url(:/images/start.png) stretch; ","border-image: url(:/images/startenter.png) stretch; "},

{"border-image: url(:/images/brown1.png) stretch; ","border-image: url(:/images/brown1enter.png) stretch; "},

{"border-image: url(:/images/chance.png) stretch; ","border-image: url(:/images/chanceenter.png) stretch; "},

{"border-image: url(:/images/brown2.png) stretch; ","border-image: url(:/images/brown2enter.png) stretch; "},

{"border-image: url(:/images/tax1.png) stretch; ","border-image: url(:/images/tax1enter.png) stretch; "},

{"border-image: url(:/images/train1.png) stretch; ","border-image: url(:/images/train1enter.png) stretch; "},

{"border-image: url(:/images/blue1.png) stretch; ","border-image: url(:/images/blue1enter.png) stretch; "},

{"border-image: url(:/images/chance.png) stretch; ","border-image: url(:/images/chanceenter.png) stretch; "},

{"border-image: url(:/images/blue2.png) stretch; ","border-image: url(:/images/blue2enter.png) stretch; "},

{"border-image: url(:/images/blue3.png) stretch; ","border-image: url(:/images/blue3enter.png) stretch; "},

{"border-image: url(:/images/prison.png) stretch; ","border-image: url(:/images/prisonenter.png) stretch; "},

{"border-image: url(:/images/pink1.png) stretch; ","border-image: url(:/images/pink1enter.png) stretch; "},

{"border-image: url(:/images/electricity.png) stretch; ","border-image: url(:/images/electricityenter.png) stretch; "},

{"border-image: url(:/images/pink2.png) stretch; ","border-image: url(:/images/pink2enter.png) stretch; "},

{"border-image: url(:/images/pink3.png) stretch; ","border-image: url(:/images/pink3enter.png) stretch; "},

{"border-image: url(:/images/train2.png) stretch; ","border-image: url(:/images/train2enter.png) stretch; "},

{"border-image: url(:/images/orange1.png) stretch; ","border-image: url(:/images/orange1enter.png) stretch; "},

{"border-image: url(:/images/chance.png) stretch; ","border-image: url(:/images/chanceenter.png) stretch; "},

{"border-image: url(:/images/orange2.png) stretch; ","border-image: url(:/images/orange2enter.png) stretch; "},

{"border-image: url(:/images/orange3.png) stretch; ","border-image: url(:/images/orange3enter.png) stretch; "},

{"border-image: url(:/images/parking.png) stretch; ","border-image: url(:/images/parkingenter.png) stretch; "},

{"border-image: url(:/images/red1.png) stretch; ","border-image: url(:/images/red1enter.png) stretch; "},

{"border-image: url(:/images/chance.png) stretch; ","border-image: url(:/images/chanceenter.png) stretch; "},

{"border-image: url(:/images/red2.png) stretch; ","border-image: url(:/images/red2enter.png) stretch; "},

{"border-image: url(:/images/red3.png) stretch; ","border-image: url(:/images/red3enter.png) stretch; "},

{"border-image: url(:/images/train3.png) stretch; ","border-image: url(:/images/train3enter.png) stretch; "},

{"border-image: url(:/images/yellow1.png) stretch; ","border-image: url(:/images/yellow1enter.png) stretch; "},

{"border-image: url(:/images/yellow2.png) stretch; ","border-image: url(:/images/yellow2enter.png) stretch; "},

{"border-image: url(:/images/watersupply.png) stretch; ","border-image: url(:/images/watersupplyenter.png) stretch; "},

{"border-image: url(:/images/yellow3.png) stretch; ","border-image: url(:/images/yellow3enter.png) stretch; "},

{"border-image: url(:/images/gotoprison.png) stretch; ","border-image: url(:/images/gotoprisonenter.png) stretch; "},

{"border-image: url(:/images/green1.png) stretch; ","border-image: url(:/images/green1enter.png) stretch; "},

{"border-image: url(:/images/green2.png) stretch; ","border-image: url(:/images/green2enter.png) stretch; "},

{"border-image: url(:/images/chance.png) stretch; ","border-image: url(:/images/chanceenter.png) stretch; "},

{"border-image: url(:/images/green3.png) stretch; ","border-image: url(:/images/green3enter.png) stretch; "},

{"border-image: url(:/images/train4.png) stretch; ","border-image: url(:/images/train4enter.png) stretch; "},

{"border-image: url(:/images/chance.png) stretch; ","border-image: url(:/images/chanceenter.png) stretch; "},

{"border-image: url(:/images/darkblue1.png) stretch; ","border-image: url(:/images/darkblue1enter.png) stretch; "},

{"border-image: url(:/images/tax2.png) stretch; ","border-image: url(:/images/tax2enter.png) stretch; "},

{"border-image: url(:/images/darkblue2.png) stretch; ","border-image: url(:/images/darkblue2enter.png) stretch; "},

{"border-image: url(:/images/roll.png) stretch; ","border-image: url(:/images/rollenter.png) stretch; "},

{"border-image: url(:/images/finish.png) stretch; ","border-image: url(:/images/finishenter.png) stretch; "},

{"border-image: url(:/images/reroll.png) stretch; ","border-image: url(:/images/rerollenter.png) stretch; "},

{"border-image: url(:/images/about.png) stretch; ","border-image: url(:/images/aboutenter.png) stretch; "},

{"border-image: url(:/images/forfeit.png) stretch; ","border-image: url(:/images/forfeitenter.png) stretch; "}

};

}

void Data::setPoints()

{

}

void Data::setSeed(int seed)

{

this->seed=seed;

}

void Data::saveBuffer()

{

copyField(field\_buffer,field);

copyPlayers(players\_buffer,players);

}

void Data::loadBuffer()

{

copyField(field,field\_buffer);

copyPlayers(players,players\_buffer);

}

void Data::copyCells(Cell \*dest, Cell \*source)

{

if(dest->isEstate()){

Estate& dest\_={\*dynamic\_cast<Estate\*>(dest)};

const Estate& source\_={\*dynamic\_cast<Estate\*>(source)};

dest\_=source\_;

}

if(dest->isHouse()){

House& dest\_={\*dynamic\_cast<House\*>(dest)};

const House& source\_={\*dynamic\_cast<House\*>(source)};

dest\_=source\_;

}

if(dest->isEvent()){

Event& dest\_={\*dynamic\_cast<Event\*>(dest)};

const Event& source\_={\*dynamic\_cast<Event\*>(source)};

dest\_=source\_;

}

}

void Data::copyField(Ring<Cell \*>& destination, Ring<Cell\*>& source)

{

Node<Cell\*>\*source\_cell=source.getFirst();

Node<Cell\*>\*dest\_cell=destination.getFirst();

for(int i=0;i<source.getSize();i++,source\_cell=source\_cell->next,dest\_cell=dest\_cell->next){

copyCells(dest\_cell->value,source\_cell->value);

}

}

void Data::copyPlayers(Ring<Entity \*>& destination, Ring<Entity \*>& source)

{

Node<Entity\*>\* source\_player=source.getFirst();

Node<Entity\*>\* dest\_player=destination.getFirst();

for(int i=0;i<source.getSize();i++,source\_player=source\_player->next,dest\_player=dest\_player->next){

\*dest\_player->value=\*source\_player->value;

copyCells(dest\_player->value->getPos()->value,source\_player->value->getPos()->value);

}

}

int &Data::getSeed()

{

return seed;

}

std::vector<std::vector<QString> > &Data::getImages()

{

return images;

}

void Data::setCurrent(Node<Entity \*> \*current)

{

this->current=current;

}

Node<Entity \*> \*Data::getCurrent()

{

return current;

}

Node<Entity \*>\* Data::getByID(int id)

{

Node<Entity\*>\* temp=players.getFirst();

do{

if(temp->value->getID()==id) return temp;

temp=temp->next;

}while(temp!=players.getFirst());

return nullptr;

}

void Data::updateAfterForfeit(int id)

{

Node<Cell\*>\*temp=field.getFirst()->next;

Estate\* estate;

House\* house;

for(;temp!=field.getFirst();temp=temp->next){

if(temp->value->isEstate()||temp->value->isHouse()){

estate=dynamic\_cast<Estate\*>(temp->value);

if(estate->getOwnerID()==id){

estate->setOwner(0);

estate->setMortgage(0);

}

}

if(temp->value->isHouse()){

house=dynamic\_cast<House\*>(temp->value);

if(house->getOwnerID()==id){

house->setHouses(0);

}

}

}

}

Ring<Entity \*> &Data::getPlayers()

{

return players;

}

Ring<Cell \*> &Data::getField()

{

return field;

}

bool Data::isGrouped(int id,Cell\*cell,int actionStatus)

{

House\*house=dynamic\_cast<House\*>(cell);

Node<Cell\*>\*temp=field.getFirst()->next;

int counter=0;

int group=house->getGroup();

int houseQuantity=house->getHouses();

int ownerID=house->getOwnerID();

for(;temp!=field.getFirst();temp=temp->next){

if(temp->value->isHouse()){

house=dynamic\_cast<House\*>(temp->value);

if(house->getGroup()!=group) continue;

else{

if(actionStatus==1&&house->getOwnerID()==id&&houseQuantity<=house->getHouses()) counter++; //игрок покупает дом на данном участке

else if(actionStatus==0&&houseQuantity>=house->getHouses()) counter++; //игрок хочет продает дом на данном участке

else if(actionStatus==2&&ownerID!=0&&ownerID!=id&&ownerID==house->getOwnerID()) counter++; //2- игрок платит аренду

}

}

}

if((group==1||group==8)&&counter==2) return 1;

if(counter==3) return 1;

return 0;

}

int Data::getActualRent()

{

srand(seed);

int roll=2+rand()%11;

int counter=0;

Node<Cell\*>\* temp=field.getFirst()->next;

int ownerID=0;

if(current->value->getPos()->value->isEstate()){

Estate\*estate=dynamic\_cast<Estate\*>(current->value->getPos()->value);

ownerID=estate->getOwnerID();

if(estate->getOwnerID()==0||estate->getOwnerID()==current->value->getID()) return 0;

if(estate->getType()==1){ //trainSt

for(;temp!=field.getFirst();temp=temp->next){

if(temp->value->isEstate()){

estate=dynamic\_cast<Estate\*>(temp->value);

if((estate->getOwnerID()==ownerID)&&(estate->getType()==1)) counter++;

}

}

if(counter==3) return 100;

if(counter==4) return 200;

else return counter\*25;

}

else{ //service

for(;temp!=field.getFirst();temp=temp->next){

if(temp->value->isEstate()){

estate=dynamic\_cast<Estate\*>(temp->value);

if((estate->getOwnerID()==ownerID)&&(estate->getType()==2)) counter++;

}

}

if(counter==1) return roll\*4;

else if(counter==2) return roll\*10;

else return 0;

}

}

else if(current->value->getPos()->value->isHouse()){

House\* house=dynamic\_cast<House\*>(current->value->getPos()->value);

if(isGrouped(current->value->getID(),current->value->getPos()->value,2)){

if(house->getHouses()==0) return house->getRent()[0]\*2;

else return house->getRent()[house->getHouses()];

}

else{

if(house->getOwnerID()!=0&&house->getOwnerID()!=current->value->getID()) return house->getRent()[0];

else return 0;

}

}

return 0;

}

bool Data::isFree(Entity \*player)

{

Estate\*estate=dynamic\_cast<Estate\*>(player->getPos()->value);

if(estate->getOwnerID()!=player->getID()&&estate->getOwnerID()!=0) return 0;

return 1;

}

Node<Cell \*>\* Data::getCellByPos(int posX, int posY)

{

Node<Cell\*>\*temp=field.getFirst();

for(;;temp=temp->next){

if(temp->value->getPosX()==posX&&temp->value->getPosY()==posY) return temp;

}

}

#include "entity.h"

#include "player.h"

#include "bot.h"

Entity::Entity(int id,int money,Node<Cell\*>\*pos,int skip, QString icon,QString name)

{

this->id=id;

this->money=money;

this->pos=pos;

this->skip=skip;

this->icon=icon;

this->name=name;

}

Entity::Entity(const Entity &entity)

{

this->id=entity.id;

this->money=entity.money;

this->pos=entity.pos;

this->skip=entity.skip;

this->icon=entity.icon;

this->name=entity.name;

}

Entity::~Entity()

{

}

void Entity::setID(int id)

{

this->id=id;

}

void Entity::setMoney(int money)

{

this->money=money;

}

void Entity::setPos(Node<Cell\*>\*pos)

{

this->pos=pos;

}

void Entity::setSkip(int skip)

{

this->skip=skip;

}

void Entity::setIcon(QString icon)

{

this->icon=icon;

}

void Entity::setName(QString name)

{

this->name=name;

}

QString&Entity::getIcon()

{

return icon;

}

QString &Entity::getName()

{

return name;

}

int &Entity::getSkip()

{

return skip;

}

int& Entity::getID()

{

return id;

}

int& Entity::getMoney()

{

return money;

}

Node<Cell\*>\* Entity::getPos()

{

return pos;

}

Entity& Entity::operator=(const Entity& source)

{

this->id=source.id;

this->money=source.money;

this->pos=source.pos;

this->skip=source.skip;

this->icon=source.icon;

this->name=source.name;

return \*this;

}

bool Entity::isEnough(int cost)

{

if(money<cost) return 0;

return 1;

}

bool Entity::mortgage(Cell\*cell)

{

Estate\* estate=dynamic\_cast<Estate\*>(cell);

if(estate->isMortgage()){

estate->setMortgage(0);

this->money-=estate->getCost()\*0.55;

return 0;

}

else{

estate->setMortgage(1);

this->money+=estate->getCost()/2;

return 1;

}

}

bool Entity::buyHouse(Cell \*cell)

{

House\*house=dynamic\_cast<House\*>(cell);

house->setHouses(house->getHouses()+1);

this->money-=house->getHouseCost();

if(house->getHouses()==5) return 1;

else return 0;

}

bool Entity::sellHouse(Cell \*cell)

{

House\*house=dynamic\_cast<House\*>(cell);

house->setHouses(house->getHouses()-1);

this->money+=house->getHouseCost()/2;

if(house->getHouses()+1==5) return 1;

else return 0;

}

void Entity::sell(Cell \*cell)

{

Estate\* estate=dynamic\_cast<Estate\*>(cell);

estate->setOwner(0);

money+=estate->getCost()\*0.75;

}

void Entity::pay(int rent){

money-=rent;

}

#include "estate.h"

Estate::Estate(int ownerID, int cost, std::vector<int> rent,bool mortgage,int type, QString name,int posX,int posY):Cell(name,posX,posY)

{

this->ownerID=ownerID;

this->cost=cost;

this->rent=rent;

this->type=type;

this->mortgage=mortgage;

}

Estate::Estate(const Estate &estate):Cell(estate)

{

this->ownerID=estate.ownerID;

this->cost=estate.cost;

this->rent=estate.rent;

this->type=estate.type;

this->mortgage=estate.mortgage;

}

Estate& Estate::operator=(const Estate& estate)

{

Estate& estate\_=const\_cast<Estate&>(estate);

this->setName(estate\_.getName());

this->setPosX(estate\_.getPosX());

this->setPosY(estate\_.getPosY());

this->ownerID=estate.ownerID;

this->cost=estate.cost;

this->rent=estate.rent;

this->type=estate.type;

this->mortgage=estate.mortgage;

return \*this;

}

void Estate::setOwner(int ownerID){

this->ownerID=ownerID;

}

void Estate::setCost(int cost){

this->cost=cost;

}

void Estate::setRent(std::vector<int> rent){

this->rent=rent;

}

int& Estate::getOwnerID(){

return ownerID;

}

int& Estate::getCost(){

return cost;

}

std::vector<int> & Estate::getRent(){

return rent;

}

void Estate::setType(int type){

this->type=type;

}

void Estate::setMortgage(bool value)

{

mortgage=value;

}

bool Estate::isMortgage()

{

return mortgage;

}

int&Estate::getType(){

return type;

}

void Estate::show()

{

}

Estate::~Estate()

{

}

#include "event.h"

Event::Event(int gift,int skip,int chance, QString name,int posX,int posY):Cell(name,posX,posY){

this->gift=gift;

this->skip=skip;

this->chance=chance;

}

Event::Event(const Event& event):Cell(event){

this->gift=event.gift;

this->skip=event.skip;

this->chance=event.chance;

}

Event &Event::operator=(const Event &event)

{

Event& event\_=const\_cast<Event&>(event);

this->setName(event\_.getName());

this->setPosX(event\_.getPosX());

this->setPosY(event\_.getPosY());

this->gift=event.gift;

this->skip=event.skip;

this->chance=event.chance;

return \*this;

}

void Event::setGift(int gift){

this->gift=gift;

}

void Event::setSkip(int skip){

this->skip=skip;

}

void Event::setChance(int chance){

this->chance=chance;

}

int& Event::getGift(){

return gift;

}

int& Event::getSkip(){

return skip;

}

int& Event::getChance(){

return chance;

}

void Event::show()

{

}

bool Cell::isEvent()

{

Event\* event;

if(event=dynamic\_cast<Event\*>(this)) return 1;

return 0;

}

Event::~Event(){

}

#include "house.h"

House::House(int ownerID, int cost, std::vector<int> rent,bool mortgage, int houses, int group, QString name,int posX,int posY):Estate(ownerID,cost,rent,mortgage,0,name,posX,posY)

{

this->houses=houses;

this->group=group;

}

House::House(const House &house):Estate(house)

{

this->houses=house.houses;

this->group=house.group;

}

House &House::operator=(const House &house)

{

House& house\_=const\_cast<House&>(house);

this->setName(house\_.getName());

this->setPosX(house\_.getPosX());

this->setPosY(house\_.getPosY());

this->setOwner(house\_.getOwnerID());

this->setCost(house\_.getCost());

this->setRent(house\_.getRent());

this->setType(house\_.getType());

this->setMortgage(house\_.isMortgage());

this->houses=house.houses;

this->group=house.group;

return \*this;

}

House::~House()

{

}

bool House::isHousesLimit()

{

if(houses==5) return 1;

return 0;

}

void House::show()

{

}

void House::setHouses(int houses){

this->houses=houses;

}

void House::setGroup(int group){

this->group=group;

}

int& House::getHouses(){

return houses;

}

int& House::getGroup(){

return group;

}

int House::getHouseCost()

{

int tempGroup=group;

if(group%2==1) tempGroup++;

tempGroup/=2;

return tempGroup\*50;

}

bool Cell::isHouse()

{

House\*house;

if(house=dynamic\_cast<House\*>(this)) return 1;

return 0;

}

bool Cell::isEstate()

{

Estate\*estate;

if(!(estate=dynamic\_cast<House\*>(this))&&(estate=dynamic\_cast<Estate\*>(this))) return 1;

return 0;

}

#include "mainwindow.h"

#include "ui\_cellinfo.h"

#include "ui\_mainwindow.h"

#include <QWidget>

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

// this->iterator=0;

// this->points=setPoints();

ui->setupUi(this);

data=nullptr;

cellInfo=nullptr;

}

MainWindow::~MainWindow()

{

delete ui;

}

Data \*MainWindow::getData()

{

return data;

}

void MainWindow::receiveData(int players,int bots,std::vector<QString> names){

data=new Data(players,bots,names);

startSettings();

cellInfo=new CellInfo(nullptr,this);

cellInfo->startSettings();

connections();

}

void MainWindow::connections()

{

Node<Cell\*>\*temp=data->getField().getFirst();

QAbstractButton\*button;

QGridLayout\*grid=qobject\_cast<QGridLayout\*>(ui->gridLayout);

QVBoxLayout\*vlayout;

connect(this,&MainWindow::openCellInfo,cellInfo,&CellInfo::open);

for(int i=0;i<40;i++){

vlayout=qobject\_cast<QVBoxLayout\*>(grid->itemAtPosition(temp->value->getPosX(),temp->value->getPosY())->layout());

button=qobject\_cast<QAbstractButton\*>(vlayout->itemAt(0)->widget());

button->installEventFilter(this);

button->setStyleSheet(data->getImages()[i][0]);

connect(button,&QAbstractButton::clicked,this,[=](void){

emit openCellInfo(temp);

});

connect(button,&QAbstractButton::clicked,cellInfo,&QWidget::show);

temp=temp->next;

}

vlayout=qobject\_cast<QVBoxLayout\*>(ui->verticalLayout\_2->layout());

for(int i=0;i<5;i++){

button=qobject\_cast<QAbstractButton\*>(vlayout->itemAt(i)->widget());

button->installEventFilter(this);

}

connect(ui->rollButton,&QAbstractButton::clicked,this,&MainWindow::rollPressed);

connect(ui->finishButton,&QAbstractButton::clicked,this,&MainWindow::finishPressed);

connect(ui->rerollButton,&QAbstractButton::clicked,this,&MainWindow::rerollPressed);

connect(ui->forfeitButton,&QAbstractButton::clicked,this,&MainWindow::forfeitPressed);

}

bool MainWindow::eventFilter(QObject \*object, QEvent \*event)

{

Node<Cell\*>\*temp=data->getField().getFirst();

QGridLayout\*grid=qobject\_cast<QGridLayout\*>(ui->gridLayout);

QVBoxLayout\*vlayout;

QAbstractButton\* target=qobject\_cast<QAbstractButton\*>(object);

QAbstractButton\*button;

for(int i=0;i<40;i++){

vlayout=qobject\_cast<QVBoxLayout\*>(grid->itemAtPosition(temp->value->getPosX(),temp->value->getPosY())->layout());

button=qobject\_cast<QAbstractButton\*>(vlayout->itemAt(0)->widget());

if(target==button){

if(event->type()==QEvent::Enter){

button->setStyleSheet(data->getImages()[i][1]);

return true;

}

else if(event->type()==QEvent::Leave){

button->setStyleSheet(data->getImages()[i][0]);

return false;

}

}

temp=temp->next;

}

vlayout=qobject\_cast<QVBoxLayout\*>(ui->verticalLayout\_2->layout());

for(int i=0;i<5;i++){

button=qobject\_cast<QAbstractButton\*>(vlayout->itemAt(i)->widget());

if(target==button){

if(event->type()==QEvent::Enter&&button->isEnabled()){

button->setStyleSheet(data->getImages()[i+40][1]);

return true;

}

else if(event->type()==QEvent::Leave&&button->isEnabled()){

button->setStyleSheet(data->getImages()[i+40][0]);

return false;

}

else if(event->type()==QEvent::Leave&&!button->isEnabled()){

button->setStyleSheet(data->getImages()[i+40][0]);

return false;

}

}

}

return false;

}

void MainWindow::roll()

{

srand(data->getSeed());

int roll= 2+rand()%11;

QElapsedTimer timer;

ui->value->setText(QString::number(roll));

ui->rollButton->setDisabled(1);

log(data->getCurrent()->value->getName()+" выбрасывает "+ui->value->text());

for(int i=0;i<roll;i++){

timer.start();

while(true)

{

qApp->processEvents();

if(timer.elapsed() > 200) break;

}

setIcon(data->getCurrent()->value->getPos()->value->getPosX(), data->getCurrent()->value->getPos()->value->getPosY(),"border-image: url(:/images/casual.png) stretch; ", data->getCurrent()->value->getID());

data->getCurrent()->value->setPos(data->getField().getNext(data->getCurrent()->value->getPos()));

setIcon(data->getCurrent()->value->getPos()->value->getPosX(), data->getCurrent()->value->getPos()->value->getPosY(),data->getCurrent()->value->getIcon(), data->getCurrent()->value->getID());

startCheck();

}

data->saveBuffer();

rentCheck();

}

void MainWindow::botCycle()

{

while(!data->getCurrent()->value->isPlayer()){

roll();

}

}

void MainWindow::log(QString info)

{

QVBoxLayout\* vlayout=qobject\_cast<QVBoxLayout\*>(ui->logLayout->layout());

QLabel\* label;

QLabel\* prev\_label;

for(int i=2;i>0;i--){

prev\_label=qobject\_cast<QLabel\*>(vlayout->itemAt(i-1)->widget());

label=qobject\_cast<QLabel\*>(vlayout->itemAt(i)->widget());

label->setText(prev\_label->text());

}

label=qobject\_cast<QLabel\*>(vlayout->itemAt(0)->widget());

label->setText(info);

}

void MainWindow::botAction()

{

//if(data->getCurrent()->value->getPos()->value->isEvent()) data->getCurrent()->value->event();

}

void MainWindow::updateHouseLabel(Cell \*cell,int id)

{

QString houses="";

QAbstractButton\* target;

House\*house;

if(cell==nullptr){

Node<Cell\*>\*temp=data->getField().getFirst()->next;

for(;temp!=data->getField().getFirst();temp=temp->next){

if(temp->value->isHouse()){

house=dynamic\_cast<House\*>(temp->value);

if(house->getOwnerID()==id){

target=getButtonAt(house->getPosX(),house->getPosY());

target->setText("");

}

}

}

return;

}

target=getButtonAt(cell->getPosX(),cell->getPosY());

house=dynamic\_cast<House\*>(cell);

if(house->getHouses()==5){

target->setText("D");

return;

}

else{

for(int i=0;i<house->getHouses();i++) houses+="0";

}

target->setText(houses);

}

void MainWindow::startCheck()

{

if(data->getCurrent()->value->getPos()->value->getPosX()==10&&data->getCurrent()->value->getPos()->value->getPosY()==10){

log(data->getCurrent()->value->getName()+" получает 200 за пересечение старта!");

data->getCurrent()->value->pay(-200);

updateMoney();

}

}

void MainWindow::prisonCheck()

{

if(data->getCurrent()->value->getPos()->value->getPosX()==0&&data->getCurrent()->value->getPos()->value->getPosY()==10){

data->getCurrent()->value->setPos(data->getCellByPos(10,0));

setIcon(0,10,"border-image: url(:/images/casual.png) stretch; ", data->getCurrent()->value->getID());

setIcon(10,0,data->getCurrent()->value->getIcon(), data->getCurrent()->value->getID());

}

if(data->getCurrent()->value->getPos()->value->getPosX()==10&&data->getCurrent()->value->getPos()->value->getPosY()==0){

data->getCurrent()->value->setSkip(1);

log(data->getCurrent()->value->getName()+" попадает в тюрьму!");

finishPressed();

}

}

void MainWindow::rentCheck()

{

if(data->getActualRent()!=0){

Estate\*estate=dynamic\_cast<Estate\*>(data->getCurrent()->value->getPos()->value);

Node<Entity\*>\*temp=data->getByID(estate->getOwnerID());

log(data->getCurrent()->value->getName()+" платит аренду на "+data->getCurrent()->value->getPos()->value->getName()+

" в размере "+QString::number(data->getActualRent())+'$');

data->getCurrent()->value->pay(data->getActualRent());

if(temp) temp->value->pay(-data->getActualRent());

updateMoney();

}

}

void MainWindow::eventCheck()

{

if(data->getCurrent()->value->getPos()->value->isEvent()){

Event\* event=dynamic\_cast<Event\*>(data->getCurrent()->value->getPos()->value);

if(event->getChance()==1){

callChance();

finishPressed();

return;

}

else if(event->getSkip()==1){

prisonCheck();

return;

}

else if(event->getGift()<0){

data->getCurrent()->value->pay(-event->getGift());

log(data->getCurrent()->value->getName()+" заплатил налог в размере "+QString::number(-event->getGift())+'$');

finishPressed();

return;

}

}

}

void MainWindow::callChance()

{

srand(data->getSeed());

int chance=1+rand()%10;

switch(chance){

case 1:

data->getCurrent()->value->pay(50);

log(data->getCurrent()->value->getName()+" платит "+QString::number(50)+'$'+" за превышение скорости");

break;

case 2:

data->getCurrent()->value->pay(25);

log(data->getCurrent()->value->getName()+" платит "+QString::number(25)+'$'+" за посещение врача");

break;

case 3:

data->getCurrent()->value->pay(100);

log(data->getCurrent()->value->getName()+" платит "+QString::number(100)+'$'+" по ошибке");

break;

case 4:

data->getCurrent()->value->pay(50);

data->getCurrent()->next->value->pay(-50);

log(data->getCurrent()->value->getName()+" платит "+QString::number(50)+"$ "+data->getCurrent()->next->value->getName()+" по поводу дня рождения");

break;

case 5:

data->getCurrent()->value->pay(50);

data->getCurrent()->prev->value->pay(-50);

log(data->getCurrent()->value->getName()+" платит "+QString::number(50)+"$ "+data->getCurrent()->prev->value->getName()+" по поводу дня рождения");

break;

case 6:

data->getCurrent()->value->pay(-50);

log(data->getCurrent()->value->getName()+" находит "+QString::number(50)+'$'+" на дороге");

break;

case 7:

data->getCurrent()->value->pay(-100);

log(data->getCurrent()->value->getName()+" получает "+QString::number(100)+'$'+" по ошибке банка");

break;

case 8:

data->getCurrent()->value->pay(-200);

log(data->getCurrent()->value->getName()+" получает "+QString::number(200)+'$'+" по наследству");

break;

case 9:

data->getCurrent()->value->pay(-50);

data->getCurrent()->next->value->pay(50);

log(data->getCurrent()->value->getName()+" получает от "+data->getCurrent()->next->value->getName()+" "+QString::number(50)+"$ "+" по поводу дня рождения");

break;

case 10:

data->getCurrent()->value->pay(-50);

data->getCurrent()->prev->value->pay(50);

log(data->getCurrent()->value->getName()+" получает от "+data->getCurrent()->prev->value->getName()+" "+QString::number(50)+"$ "+" по поводу дня рождения");

break;

}

}

QAbstractButton \*MainWindow::getButtonAt(int x, int y)

{

QGridLayout\*grid=qobject\_cast<QGridLayout\*>(ui->gridLayout);

QVBoxLayout\*vlayout=qobject\_cast<QVBoxLayout\*>(grid->itemAtPosition(x,y)->layout());

return qobject\_cast<QAbstractButton\*>(vlayout->itemAt(0)->widget());

}

void MainWindow::updateMoney()

{

QVBoxLayout\*vlayout=qobject\_cast<QVBoxLayout\*>(ui->verticalLayout->layout());

QLabel\*label;

Node<Entity\*>\*entity=data->getPlayers().getFirst();

QString buffer;

for(int i=0; i<data->getPlayers().getSize();i++){

label=qobject\_cast<QLabel\*>(vlayout->itemAt((entity->value->getID()-1)\*2+1)->widget());

buffer=QString::number(entity->value->getMoney());

label->setText(buffer+'$');

entity=entity->next;

}

}

void MainWindow::press(Node<Cell\*>\* cell)

{

//iterator++;

emit openCellInfo(cell);

}

void MainWindow::rollPressed()

{

roll();

ui->finishButton->setDisabled(0);

ui->rerollButton->setDisabled(0);

eventCheck();

}

void MainWindow::rerollPressed()

{

srand(data->getSeed());

int roll= 2+rand()%11;

data->loadBuffer();

updateMoney();

updateHouseLabel();

setIcon(data->getCurrent()->value->getPos()->value->getPosX(), data->getCurrent()->value->getPos()->value->getPosY(),"border-image: url(:/images/casual.png) stretch; ", data->getCurrent()->value->getID());

data->getCurrent()->value->setPos(data->getField().getPrevN(data->getCurrent()->value->getPos(),roll));

setIcon(data->getCurrent()->value->getPos()->value->getPosX(), data->getCurrent()->value->getPos()->value->getPosY(),data->getCurrent()->value->getIcon(), data->getCurrent()->value->getID());

ui->rerollButton->setDisabled(1);

ui->rollButton->setDisabled(0);

ui->finishButton->setDisabled(1);

log(data->getCurrent()->value->getName()+" решает перебросить!");

}

void MainWindow::finishPressed()

{

if(data->getCurrent()->value->getMoney()<0){

log("У Вас отрицательный баланс, заложите что-нибудь, чтобы продолжить");

return;

}

data->setSeed(QTime::currentTime().msec());

ui->finishButton->setDisabled(1);

ui->rollButton->setDisabled(0);

ui->rerollButton->setDisabled(1);

updateMoney();

updateHouseLabel();

data->setNextCurrent();

}

void MainWindow::forfeitPressed()

{

log(data->getCurrent()->value->getName()+" сдается");

data->updateAfterForfeit(data->getCurrent()->value->getID());

updateHouseLabel(nullptr,data->getCurrent()->value->getID());

setIcon(data->getCurrent()->value->getPos()->value->getPosX(), data->getCurrent()->value->getPos()->value->getPosY(),"border-image: url(:/images/casual.png) stretch; ", data->getCurrent()->value->getID());

data->getPlayers().popValue(data->getCurrent()->value);

finishPressed();

}

void MainWindow::startSettings()

{

setStyleSheet("QMainWindow{border-image:url(:images/background.png); stretch}");

ui->player1name->setStyleSheet("border-image: url(:/images/player1name.png) stretch; ");

ui->player2name->setStyleSheet("border-image: url(:/images/player2name.png) stretch; ");

ui->player3name->setStyleSheet("border-image: url(:/images/player3name.png) stretch; ");

ui->player4name->setStyleSheet("border-image: url(:/images/player4name.png) stretch; ");

ui->player1money->setStyleSheet("border-image: url(:/images/player1money.png) stretch; ");

ui->player2money->setStyleSheet("border-image: url(:/images/player2money.png) stretch; ");

ui->player3money->setStyleSheet("border-image: url(:/images/player3money.png) stretch; ");

ui->player4money->setStyleSheet("border-image: url(:/images/player4money.png) stretch; ");

ui->value->setStyleSheet("border-image: url(:/images/value.png) stretch; ");

ui->aboutButton->setStyleSheet("border-image: url(:/images/about.png) stretch; ");

ui->rollButton->setStyleSheet("border-image: url(:/images/roll.png) stretch; ");

ui->forfeitButton->setStyleSheet("border-image: url(:/images/forfeit.png) stretch; ");

ui->rerollButton->setStyleSheet("border-image: url(:/images/reroll.png) stretch; ");

ui->finishButton->setStyleSheet("border-image: url(:/images/finish.png) stretch; ");

ui->startButton->setStyleSheet("border-image: url(:/images/start.png) stretch; ");

ui->brown1Button->setStyleSheet("border-image: url(:/images/brown1.png) stretch; ");

ui->chance1Button->setStyleSheet("border-image: url(:/images/chance.png) stretch; ");

ui->chance2Button->setStyleSheet("border-image: url(:/images/chance.png) stretch; ");

ui->chance3Button->setStyleSheet("border-image: url(:/images/chance.png) stretch; ");

ui->chance4Button->setStyleSheet("border-image: url(:/images/chance.png) stretch; ");

ui->chance5Button->setStyleSheet("border-image: url(:/images/chance.png) stretch; ");

ui->chance6Button->setStyleSheet("border-image: url(:/images/chance.png) stretch; ");

ui->brown2Button->setStyleSheet("border-image: url(:/images/brown2.png) stretch; ");

ui->tax1Button->setStyleSheet("border-image: url(:/images/tax1.png) stretch; ");

ui->tax2Button->setStyleSheet("border-image: url(:/images/tax2.png) stretch; ");

ui->train1Button->setStyleSheet("border-image: url(:/images/train1.png) stretch; ");

ui->train2Button->setStyleSheet("border-image: url(:/images/train2.png) stretch; ");

ui->train3Button->setStyleSheet("border-image: url(:/images/train3.png) stretch; ");

ui->train4Button->setStyleSheet("border-image: url(:/images/train4.png) stretch; ");

ui->prisonButton->setStyleSheet("border-image: url(:/images/prison.png) stretch; ");

ui->gotoprisonButton->setStyleSheet("border-image: url(:/images/gotoprison.png) stretch; ");

ui->parkingButton->setStyleSheet("border-image: url(:/images/parking.png) stretch; ");

ui->blue1Button->setStyleSheet("border-image: url(:/images/blue1.png) stretch; ");

ui->blue2Button->setStyleSheet("border-image: url(:/images/blue2.png) stretch; ");

ui->blue3Button->setStyleSheet("border-image: url(:/images/blue3.png) stretch; ");

ui->pink1Button->setStyleSheet("border-image: url(:/images/pink1.png) stretch; ");

ui->pink2Button->setStyleSheet("border-image: url(:/images/pink2.png) stretch; ");

ui->pink3Button->setStyleSheet("border-image: url(:/images/pink3.png) stretch; ");

ui->orange1Button->setStyleSheet("border-image: url(:/images/orange1.png) stretch; ");

ui->orange2Button->setStyleSheet("border-image: url(:/images/orange2.png) stretch; ");

ui->orange3Button->setStyleSheet("border-image: url(:/images/orange3.png) stretch; ");

ui->red1Button->setStyleSheet("border-image: url(:/images/red1.png) stretch; ");

ui->red2Button->setStyleSheet("border-image: url(:/images/red2.png) stretch; ");

ui->red3Button->setStyleSheet("border-image: url(:/images/red3.png) stretch; ");

ui->yellow1Button->setStyleSheet("border-image: url(:/images/yellow1.png) stretch; ");

ui->yellow2Button->setStyleSheet("border-image: url(:/images/yellow2.png) stretch; ");

ui->yellow3Button->setStyleSheet("border-image: url(:/images/yellow3.png) stretch; ");

ui->green1Button->setStyleSheet("border-image: url(:/images/green1.png) stretch; ");

ui->green2Button->setStyleSheet("border-image: url(:/images/green2.png) stretch; ");

ui->green3Button->setStyleSheet("border-image: url(:/images/green3.png) stretch; ");

ui->darkblue1Button->setStyleSheet("border-image: url(:/images/darkblue1.png) stretch; ");

ui->darkblue2Button->setStyleSheet("border-image: url(:/images/darkblue2.png) stretch; ");

ui->electricityButton->setStyleSheet("border-image: url(:/images/electricity.png) stretch; ");

ui->watersupplyButton->setStyleSheet("border-image: url(:/images/watersupply.png) stretch; ");

ui->finishButton->setDisabled(1);

ui->rerollButton->setDisabled(1);

// for(int i=0;i<10;i++){

// for(int j=1;j<5;j++) setIcon(i,0,"border-image: url(:/images/casual.png) stretch; ",j);

// }

// for(int i=1;i<11;i++){

// for(int j=1;j<5;j++) setIcon(0,i,"border-image: url(:/images/casual.png) stretch; ",j);

// }

// for(int i=1;i<11;i++){

// for(int j=1;j<5;j++) setIcon(i,10,"border-image: url(:/images/casual.png) stretch; ",j);

// }

// for(int i=0;i<10;i++){

// for(int j=1;j<5;j++) setIcon(10,i,"border-image: url(:/images/casual.png) stretch; ",j);

// }

QVBoxLayout\*vlayout=qobject\_cast<QVBoxLayout\*>(ui->verticalLayout->layout());

QLabel\*label;

Node<Entity\*>\*entity=data->getPlayers().getFirst();

QString buffer;

for(int i=0; i<data->getPlayers().getSize();i++){

label=qobject\_cast<QLabel\*>(vlayout->itemAt(i\*2)->widget());

label->setText(entity->value->getName());

label=qobject\_cast<QLabel\*>(vlayout->itemAt(i\*2+1)->widget());

buffer=QString::number(entity->value->getMoney());

label->setText(buffer+'$');

setIcon(10,10,entity->value->getIcon(),i+1);

entity=entity->next;

}

for(int i=data->getPlayers().getSize();i<4;i++){

label=qobject\_cast<QLabel\*>(vlayout->itemAt(i\*2)->widget());

label->hide();

label=qobject\_cast<QLabel\*>(vlayout->itemAt(i\*2+1)->widget());

label->hide();

}

}

void MainWindow::setIcon(int cellPosX, int cellPosY, QString icon, int playerID)

{

QGridLayout\*grid=qobject\_cast<QGridLayout\*>(ui->gridLayout);

QVBoxLayout\*vlayout=qobject\_cast<QVBoxLayout\*>(grid->itemAtPosition(cellPosX,cellPosY)->layout());

QHBoxLayout\*hlayout=qobject\_cast<QHBoxLayout\*>(vlayout->itemAt(1)->layout());

QLabel\*label=qobject\_cast<QLabel\*>(hlayout->itemAt(playerID-1)->widget());

label->setStyleSheet(icon);

}

#include "player.h"

Player::Player(int id,int money,Node<Cell\*>\*pos,int skip, QString icon,QString name):Entity(id,money,pos,skip,icon,name)

{

}

Player::Player(const Player &player):Entity(player)

{

}

Player::~Player()

{

}

void Player::buy()

{

Estate\* estate=dynamic\_cast<Estate\*>(getPos()->value);

estate->setOwner(id);

money-=estate->getCost();

}

bool Entity::isPlayer()

{

Player\* player;

if(player=dynamic\_cast<Player\*>(this)) return 1;

return 0;

}

#include "startwindow.h"

#include "ui\_startwindow.h"

StartWindow::StartWindow(QWidget \*parent) :

QMainWindow(parent),

ui(new Ui::StartWindow)

{

ui->setupUi(this);

players=1;

bots=0;

mainWindow=new MainWindow();

startSettings();

}

StartWindow::~StartWindow()

{

delete ui;

}

void StartWindow::startSettings()

{

ui->start->installEventFilter(this);

setStyleSheet("QMainWindow{border-image:url(:images/startwindowbackground.png); stretch}");

ui->start->setStyleSheet("border-image: url(:/images/continue.png) stretch; ");

ui->playercount->setStyleSheet("border-image: url(:/images/spinbox.png) stretch; ");

ui->botcount->setStyleSheet("border-image: url(:/images/spinbox.png) stretch; ");

ui->name1edit->setStyleSheet("border-image: url(:/images/lineedit.png) stretch; ");

ui->name2edit->setStyleSheet("border-image: url(:/images/lineedit.png) stretch; ");

ui->name3edit->setStyleSheet("border-image: url(:/images/lineedit.png) stretch; ");

ui->name4edit->setStyleSheet("border-image: url(:/images/lineedit.png) stretch; ");

ui->name2->hide();

ui->name3->hide();

ui->name4->hide();

ui->name2edit->hide();

ui->name3edit->hide();

ui->name4edit->hide();

connect(ui->playercount,&QSpinBox::valueChanged,this,&StartWindow::showLineEdit);

connect(ui->botcount,&QSpinBox::valueChanged,this,&StartWindow::botSpinBox);

ui->start->installEventFilter(this);

ui->start->setDisabled(1);

connect(this,&StartWindow::sendData,mainWindow,&MainWindow::receiveData);

connect(ui->start,&QAbstractButton::clicked,this, &StartWindow::emitData);

connect(ui->start,&QAbstractButton::clicked,this,&QWidget::hide);

connect(ui->start,&QAbstractButton::clicked,mainWindow,&QWidget::show);

}

bool StartWindow::eventFilter(QObject \*object, QEvent \*event)

{

if(object==ui->start&&event->type()==QEvent::Enter&&ui->start->isEnabled()) ui->start->setStyleSheet("border-image: url(:/images/continueenter.png) stretch; ");

else if(object==ui->start&&event->type()==QEvent::Leave&&ui->start->isEnabled()) ui->start->setStyleSheet("border-image: url(:/images/continue.png) stretch; ");

return false;

}

int &StartWindow::getPlayers()

{

return players;

}

int &StartWindow::getBots()

{

return bots;

}

QFormLayout \*StartWindow::getFormLayout()

{

return ui->formLayout;

}

void StartWindow::showLineEdit()

{

QFormLayout\* flayout=qobject\_cast<QFormLayout\*>(ui->formLayout->layout());

QLineEdit\* lineedit;

QLabel\* label;

if(bots+ui->playercount->value()>4){

ui->playercount->setValue(ui->playercount->value()-1);

return;

}

if(players<ui->playercount->value()){

lineedit=qobject\_cast<QLineEdit\*>(flayout->itemAt(ui->playercount->value()-1,QFormLayout::FieldRole)->widget());

label=qobject\_cast<QLabel\*>(flayout->itemAt(ui->playercount->value()-1,QFormLayout::LabelRole)->widget());

players++;

lineedit->show();

label->show();

}

else if(players>ui->playercount->value()){

players--;

lineedit=qobject\_cast<QLineEdit\*>(flayout->itemAt(ui->playercount->value(),QFormLayout::FieldRole)->widget());

label=qobject\_cast<QLabel\*>(flayout->itemAt(ui->playercount->value(),QFormLayout::LabelRole)->widget());

lineedit->hide();

label->hide();

}

if(players+bots<2&&ui->start->isEnabled()) ui->start->setDisabled(1);

else if(players+bots>=2&&!ui->start->isEnabled()) ui->start->setDisabled(0);

}

void StartWindow::botSpinBox()

{

if(players+ui->botcount->value()>4){

ui->botcount->setValue(ui->botcount->value()-1);

}

else if(bots>ui->botcount->value()) bots--;

else if(bots<ui->botcount->value()) bots++;

if(players+bots<2&&ui->start->isEnabled()) ui->start->setDisabled(1);

else if(players+bots>=2&&!ui->start->isEnabled()) ui->start->setDisabled(0);

}

void StartWindow::emitData()

{

std::vector<QString> names;

QFormLayout\* flayout=getFormLayout();

QLineEdit\* lineedit;

for(int i=0;i<players;i++){

lineedit=qobject\_cast<QLineEdit\*>(flayout->itemAt(i,QFormLayout::FieldRole)->widget());

names.push\_back(lineedit->text());

}

emit sendData(players,bots,names);

}

#include "startwindow.h"

#include <QApplication>

#include <QLocale>

#include <QTranslator>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

QTranslator translator;

const QStringList uiLanguages = QLocale::system().uiLanguages();

for (const QString &locale : uiLanguages) {

const QString baseName = "monopoly\_" + QLocale(locale).name();

if (translator.load(":/i18n/" + baseName)) {

a.installTranslator(&translator);

break;

}

}

StartWindow startWindow(nullptr);

startWindow.show();

return a.exec();

}