Ревью на GutyarOvsyannikovHyshovPI42 от ZayvyPolomoshnovSergeevPI42

Достоинства:

+1 Единый стиль именования и визуального разделения переменных от методов, констант и классов

```
// вещь
class Item : public QObject
{
    Q_OBJECT
    public:
        explicit Item(QObject *parent = 0);
        explicit Item(QString _id, QObject *parent = 0);
        virtual void Use(); // использовать
        QString getID();
        void prepare(); // подготовить к новому исп-ю
        bool isDisposeRequired(); // полностью ли исп-но
    protected:
        float status; // состояние
        QString id; // ID/Название
        bool clean; // чистота
    signals:
        void isUsed();
        void isFinal();
    public slots:
};
```

+2 Использование стандартных классов Qt

```
QList<bool> freeCells;------// свободные клетки
QPointF clickCoord;-----// координата клетки
QPair<int, int> roomSize;-------// размер помещения
QList<Dish *> dishesAvailable;------// меню
QList<Entity *> entities;-------// объекты карты
QPoint entrance;--------// точка входа
QList<Person *> employees;------// персонал
QList<Person *> customers;------// посетителм
QList<Person *> customerQueue;--// очередь посетителей
QList<QPoint> queuePath;--------// путь очереди
QList<QThread *> threadPool;--------// простр-во потоков
QList<int> countersShortcuts;-------// ссылки на рабочие объекты
```

+3 Использование объектно-ориентированного подхода к написанию кода

+4 Выделение цветовых констант для многоразового использования

```
#define color_cashier QColor(255, 242, 0)
#define color_cashier_cap QColor(34, 177, 76)
#define color_cook QColor(195, 195, 195)
#define color_cook_cap QColor(255, 255, 255)
#define color_dishwasher QColor(0, 162, 232)
#define color_dishwasher_cap QColor(255, 255, 255)
```

Недостатки:

-1 Ненужная проверка на NULL

```
Dish * DialogDish::getDish()
{
    return (dish != NULL) ? dish : NULL;
}
```

-2 Отсутствие комментариев в теле методов

```
QList<QPoint> Model::findFreeNeighbors(QPoint src)
{
    QList<QPoint> neighbors;
    for (int i = src.y() - (src.y() > 0 ? 1 : 0); i <= src.y() + (src.y() < roomSize.second - 1 ? 1 : 0); i++)
        for (int j = src.x() - (src.x() > 0 ? 1 : 0); j <= src.x() + (src.x() < roomSize.first - 1 ? 1 : 0); j++)
            if (freeCells.at(i * roomSize.first + j) && QPoint(j, i) != src)
                neighbors.push_back(QPoint(j, i));
    return neighbors;
}
```

-3 Методы больших размеров

```
QList<QPoint> Model::findPath(QPoint src, QPoint dest)
{
    QPoint a = src;
    QList<QPoint> openedList, closedList, parents;
    QList<QPoint> path;
    QList<int> f, g, h;
    fillList(&f, 0, roomSize.first * roomSize.second);
    fillList(&g, 0, roomSize.first * roomSize.second);
    fillList(&h, 0, roomSize.first * roomSize.second);
    for (int i = 0; i < roomSize.second; i++)
        for (int j = 0; j < roomSize.first; j++)
            parents.push_back(QPoint(j, i));
    openedList.push_back(a);
    f[a.y() * roomSize.first + a.x()] = h[a.y() * roomSize.first + a.x()] = (dest - src).manhattanLength();
    while(!openedList.isEmpty())
    {
        a = openedList.front();
        int fMin = f[openedList.front().y() * roomSize.first + openedList.front().y()];
        for (int i = 1; i < openedList.size(); i++)
            if (f[openedList.front().y() * roomSize.first + openedList.front().y()] < fMin)
            {
                a = openedList[i];
                fMin = f[openedList.front().y() * roomSize.first + openedList.front().y()];
            }
        closedList.push_back(a);
        openedList.removeAt(openedList.indexOf(a));
        QList<QPoint> neighbors = findFreeNeighbors(a);
        foreach (QPoint neighbor, neighbors) {
            if (closedList.indexOf(neighbor) == -1)
            {
                if (openedList.indexOf(neighbor) == -1)
                {
                    openedList.push_back(neighbor);
                    g[neighbor.x() + neighbor.y() * roomSize.first] = g[a.x() + roomSize.first * a.y()] + 1;
                    h[neighbor.x() + neighbor.y() * roomSize.first] = (dest - neighbor).manhattanLength();
                    f[neighbor.x() + neighbor.y() * roomSize.first] = g[neighbor.x() + neighbor.y() * roomSize.first] + h[neighbor.x() + neighbor.y() * roomSize.first];
                    parents[neighbor.x() + neighbor.y() * roomSize.first] = a;
                    if (neighbor == dest)
                    {
                        QPoint currPoint = neighbor;
                        while (currPoint != src)
                        {
                            path.push_front(currPoint);
                            currPoint = parents[currPoint.x() + roomSize.first * currPoint.y()];
                        }
                        return path;
                    }
                }
                else
                {
                    if ((g[a.x() + roomSize.first * a.y()] + 1) < g[neighbor.x() + neighbor.y() * roomSize.first])
                    {
                        g[neighbor.x() + neighbor.y() * roomSize.first] = g[a.x() + roomSize.first * a.y()] + 1;
                        f[neighbor.x() + neighbor.y() * roomSize.first] = g[neighbor.x() + neighbor.y() * roomSize.first] + h[neighbor.x() + neighbor.y() * roomSize.first];
                        parents[neighbor.x() + neighbor.y() * roomSize.first] = a;
                    }
                }
            }
        }
    }
    QList<QPoint> failPath;
    failPath.push_back(point_fail);
    return failPath;
}
```

-4 Закомментированные участки кода

```
void Model::getMessage(QString msg)
{
    // перенаправляем сообщение от объекта
    message(msg);
}
/*
void Model::cookDishes()
{
    needToCookMore = true;
}
*/
```

Итоговая оценка: 5/10