Теория и технология программирования

Объектно-ориентированное программирование на языке C++

Лекция 11. Введение в Model-View-Controller

Глухих Михаил Игоревич, к.т.н., доц.
mailto: glukhikh@mail.ru

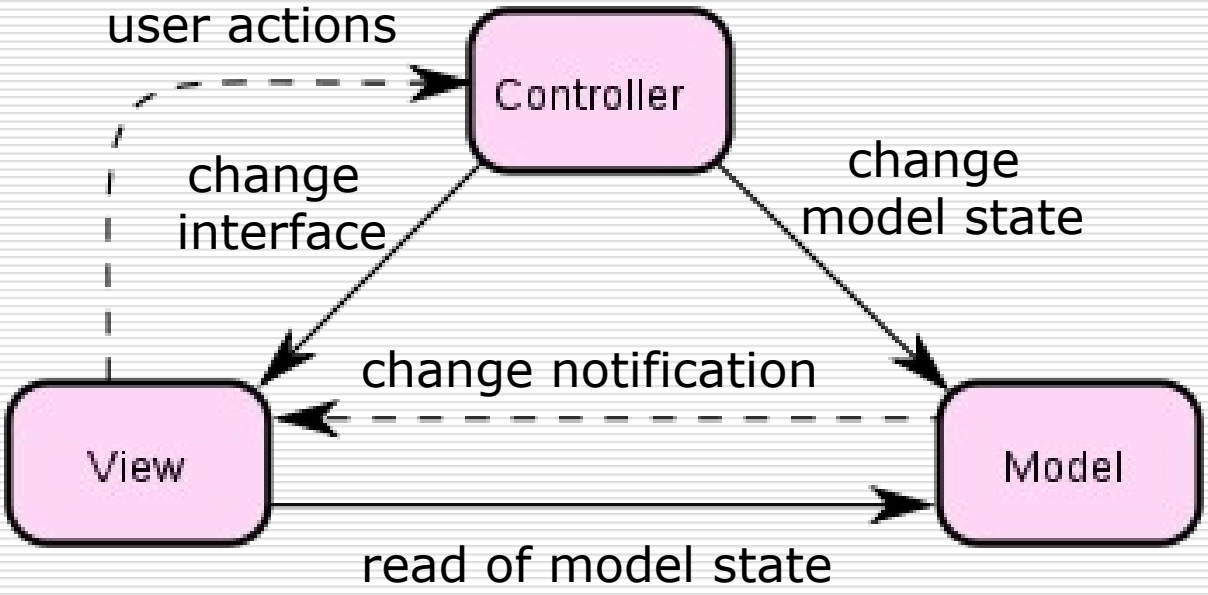
Что такое Model-View-Controller?

- Шаблон проектирования (паттерн, приём)
  - модель описывает внутреннюю структуру объекта (бизнес-логику) и ничего не знает о способах ее представления
  - представление отображает объект, используя его модель
  - контроллер обрабатывает действия пользователя и соответствующим образом изменяет модель и представление

Зачем?

- Приём проектирования MVC позволяет отделить внутреннюю начинку объекта от способа его отображения и от способа взаимодействия с пользователем
- Можно создавать разные представления одного и того же объекта, не трогая сам объект (Model-View)
- Можно по-разному организовать взаимодействие с пользователем, не трогая внешний вид (View-Controller)

Схема взаимодействия MVC (классика)



Упрощенные варианты MVC

- Model-View
  - в этом случае View выполняет также функции контроллера
- Document-View (MFC)
  - примерно то же самое, что Model-View
  - у документа обычно есть возможность записываться на жёсткий диск и считываться с него

MVC внутри Qt

- Таблица
- QAbstractItemModel
  - ряды, колонки, данные...
  - сама таблица может храниться, например, в БД
- QTableView
  - ширина, высота, сетка, стиль рисования...

MVC на примере

- Поиск кратчайшего пути в графе (лекция 4)
- У нас уже есть модель
  - граф
  - путь и искатель пути
- Хотим редактор
  - добавление и удаление вершин и дуг
  - перемещение вершин
  - изменение стоимости (веса) дуг
  - задание начала и конца пути
  - поиск пути

Граф: что нужно добавить?

```
typedef std::set<Node*>::const_iterator
   node_iterator;
class Graph {
   std::set<Node*> nodes;
public:
   void addNode(Node* node);
   void removeNode(Node* node);
   void addEdge(Node* begin, Node* end);
   void removeEdge(Node* begin, Node* end);
   node_iterator begin() const {
      return nodes.begin(); }
   node_iterator end() const { return nodes.end(); }
};
```

Граф: что нужно добавить?

- Доступ к свойствам дуг (вес, начало, конец, стоимость)
- Удобные операции с дугами (перебор, поиск)
- Доп. методы – см. пример, класс Graph

Что войдёт в представление графа?

- Поля?

Что войдёт в представление графа (поля)?

- модель (граф)
- информация о координатах вершин
- информация о построенном пути, его начале и конце
- методы?

Что войдёт в представление графа (методы)?

- рисование графа
- доступ к модели
- добавление вершин/дуг
- перемещение вершин
- удаление вершин/дуг
- поиск вершин/дуг по точкам
- получение и изменение стоимости дуги
- задание начала и конца пути
- расчёт пути
- см. пример, класс GraphView

Поиск вершин/дуг по точкам – как реализовать?

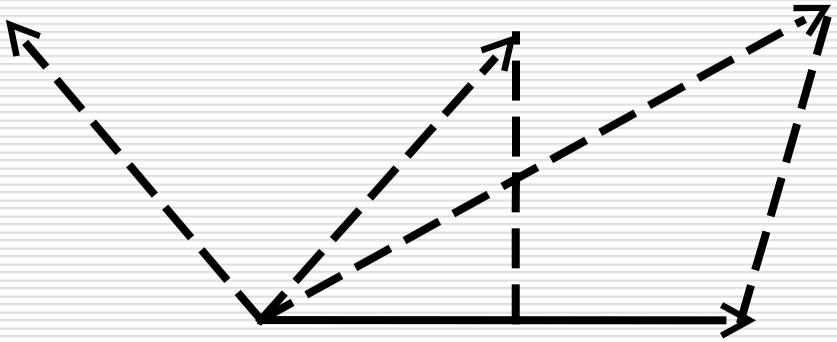
- Пример геометрической задачи, возникающей при создании GUI

Поиск вершин/дуг по точкам

- Вершина – на базе расстояния до центра соответствующей окружности

Поиск вершин/дуг по точкам

- Дуга – на базе расстояния до соответствующего отрезка



Поиск вершин/дуг по точкам

```
static int operator *(const QPoint& p1, const QPoint& p2) {
   return p1.x()*p2.x() + p1.y()*p2.y();
}
static int getSquareDist(const QPoint& p1,
                         const QPoint& p2) {
   return (p2-p1)*(p2-p1);
}
struct Segment {
   QPoint from, to;
   Segment(const QPoint& f, const QPoint& t) :
       from(f), to(t) {}
   QPoint diff() { return to-from; }
};
```

Поиск вершин/дуг по точкам

```
static int getSquareDist(const QPoint& p,
                         const Segment& s) {
    QPoint v = s.diff();
    QPoint w = p - s.from;
    int c1 = v * w;
    if (c1 <= 0) return getSquareDist(p, s.from);
    int c2 = v * v;
    if (c2 <= c1) return getSquareDist(p, s.to);
    double b = double(c1) / c2;
    QPoint pb = s.from + v * b;
    return getSquareDist(p, pb);
}
```

Рисование представления

- См. пример, GraphView::paint
- Демонстрация
- Как лучше выбрать цвета?

Построение пути

- Путь строится, когда указаны старт и финиш
- Иначе путь пуст

```
void GraphView::calcWay() {
    if (start || finish)
        way = Way();
    else {
        Dijkstra dijkstra(model);
        way = dijkstra.shortestWay(start, finish);
    }
}
```

Контроллер

- В данном проекте его функции выполняются классом MainWindow
- Он же выполняет часть функций представления (в частности, события формируются тоже в нём)

Организация интерфейса

- Режимы работы
  - добавление вершин
  - добавление дуг
  - перемещение вершин
  - выбор старта
  - выбор финиша
- Добавление и выбор вершин – щелчком мыши
- Удаление и задание стоимости – через контекстное меню

Группы действий

```
// Можно выбрать только одно
// триггерное действие из группы
modeGroup = new QActionGroup(this);
modeGroup->addAction(ui->actionAddEdge);
modeGroup->addAction(ui->actionAddNode);
modeGroup->addAction(ui->actionMove);
modeGroup->addAction(ui->actionSetStart);
modeGroup->addAction(ui->actionSetFinish);
```

Дефекты архитектуры

- Как реализовать MVC точнее?

Итоги

- Рассмотрена архитектура MVC
- Рассмотрен пример её реализации (поиск кратчайшего пути)