# GUI. Вступление

Здесь описаны основные идеи современных архитектур нынешних GUI библиотек.

# Программа в общих чертах. Данные и код.

Есть данные программы, а есть алгоритмы, которые преобразуют эти данные. Совокупность данных программы в определенный момент времени можно назвать состоянием и обозначить S(t0).

Алгоритм – это функция, которая ставит каждому моменту времени новое состояние: S(t)

Здесь мы смоделировали процесс работы программы в виде конечного автомата.

В программе самое важное – это данные, какой алгоритм – это дело второе. Данные можно назвать моделью, но, если они представляют ядро приложения, доменную область.

Основная задача программы – получить данные на вход, обработать, и выдать результат, что эквивалентно переходу из состояния S(t0) к состоянию S(t0+n), где n – время работы. Надо смотреть на чужую программу как на черный ящик.

## Пример:

1. Торрент: запросили скачивание, получили файл. Модель – структура DownloadItem, содержащая массив байт скачиваемого файла, скорость скачивания и другую информацию.

2. Игра: нажали кнопки и таким образом ввели входные данные, после чего персонаж выстрелил, то есть выходные данные – это значение состояния в структуре игрока о том, что он стреляет. Модель – структура игрока, врагов, окружающего мира.

3. Браузер: запросили страницу, то есть скачали её. Как и в торренте. Модель – DOM-дерево, сгенерированное после получения HTML-кода запрошенной страницы.

Таким образом, модель – это данные доменной области приложения, которые постоянно изменяются, как правило, вследствие действий пользователя. Важно здесь получение нового состояния этой модели, а каким образом (какой алгоритм) – уже не так важно, ибо алгоритмов таких может быть бесконечно много. **Данные важнее алгоритма**.

## Поток данных

Человек, работая с ПК, желает видеть то, чего хочет. Он пытается привести программу, с которой он работает, в нужное ему состояние. При этом если возникнет неожиданное состояние, человек приведет его в то, в которое посчитает нужным. Таким образом, человека можно назвать главной программой, которая управляет программой на ПК путем взаимодействия через клавиатуру или мышь.

## Графическое представление модели программы

Модель программы – массив байт на низком уровне, который как-то изменяется. Для пользователя этот массив байт надо представить красиво, по-человечески. Этой задачей занимается отдельная часть программы, называемая GUI. Хотя её можно представить и звуками, и всем тем, что поддается человеческому восприятию благодаря пяти чувствам. По сути, современные ПК представляют человеческий интерфейс людям, чтобы те могли удобно взаимодействовать с массивами байт этих моделей. Например, о стоянии игрового мира можно сообщить не только визуально, но и с помощью звуков и даже тактильных ощущений – вибрации.

Важно понимать, что GUI – это обертка над моделью программы. Если данные модели изменятся, изменится и картинка, характеризующая эти данные. Таким образом, модель управляет представлением. При этом моделью, которая пассивна, управляет другой слой приложения, называемый контроллером. Мы пришли к MVC архитектуре.

# MVC

Проблемная часть в понимании MVC – контроллер. Он нужен, чтобы поменять модель путем взаимодействия человека с ПК. Модель может изменится и без человека сама через системные события (по таймеру, прерыванию и т.д.). Поэтому важно, чтобы модель могла оповещать своего UI-представителя для перерисовки, а не делал бы это контроллер. Контроллер вызывает методы интерфейса модели приложения и больше ничего не делает. Через эти вызовы он меняется состояние модели, и модель сообщает о смене состояния представителю.

**Вопрос**: откуда вызываются функции/методы контроллеров?

# Этапы работы GUI

Модель есть у приложения (структура игрока) и модель у GUI (структура кнопки). Если говорится о MVC графической библиотеки, то подразумевается, что модель – это состояние всех виджетов.

1. Запускается программы, строятся виджеты, то есть строится модель GUI, которая характеризует состояние всех этих виджетов. Виджет с точки зрения программиста – это его расположение на экране, его тип, его связь с другими виджетами, действия, которые должны произойти при изменении состояния виджета.

2. К модели GUI, то есть на все построенные виджеты, составляющие дерево, пристраивается UI-менеджер, задача которого предоставить виджетам средства себя визуально изобразить. Он также рендерит всё это дерево на экране, вызывая у главного виджета (окна) метод paint. Метод paint должен вызваться для всех потомков в классе, при этом в нем идет обращение через UI-менеджер к своим UI-представителям, чтобы те их нарисовали.

Главное преимущество: возможность менять без проблем UI-менеджера, меняя вид всего приложения.

3. От ОС поступают низкоуровневые события, которые адресованы к текущему окну пользователя. Среди них: перемещение мыши, клики, стук клавишей, перемещение или изменение размеров окна. Благодаря им можно осуществить взаимодействие с GUI. Для этого событие перемещения мыши помещается в очередь событий. Далее оно обрабатывается, в результате в каждом виджете вызывается метод dispatchEvent, задача которого определить, принадлежат ли координаты курсора области данного виджета, если да, то отправить это низкоуровневое событие для слушателей. Также надо отправить на обработку своим виджетам-потомкам. Если курсор над кнопкой, то она должна перейти в состояние hovered, то есть поменяется модель, сгенерится высокоуровневое событие hovered. UI-представитель может подписаться на это событие и обработать его, изменив цвет фона кнопки.

**Итого**:

Событие перемещения мыши -> очередь -> обработка -> проверка всех виджетов в дереве -> рассылка низкоуровневого события слушателям. **ИЛИ** -> генерация высокоуровневого события hovered -> оповещение слушателей, в частности UI-представителя -> другой цвет фона кнопки.

Важно заметить, что UI-представитель тоже содержит данные, например о цвете фона кнопки. Но для основной модели кнопки цвет не важен, зато важная информация о расположении на экране с помощью менеджера компоновки, причем не в конкретных координатах, ибо это относится к контексту рисования. Конкретных координат важно избегать в модели GUI.

Низкоуровневое событие можно перехватить, когда оно идет от предков к потомкам с целью не дать его обрабатывать потомками *(использование прозрачной панели)*. С высокоуровневым такое не прокатит, потому что он генерится сразу для всех слушателей, которые хранятся в виде списка и имеют равный приоритет.

Важно отделять процесс рисования и процесс обработки события, ибо обработка события часто меняет состояние модели GUI.

Тут можно ответить на выше заданный вопрос о вызове контроллера.

Изменение модели GUI влечет вызов контроллера, который в свою очередь изменяет модель приложения через её интерфейс, которая оповещает модель GUI, которая изменяется и перерисовывается.

**Итого**:

Высокоуровневое событие нажатия кнопки -> удаление функции в SDA -> оповещение из модели приложения об этом GUI -> изменение модели GUI -> перерисовка.

Таким образом, мы не должны в контроллере пытаться удалить самостоятельно элемент из виджета (строку в списке), она сама должна удалится. Обсудим самое проблемное место – списки, деревья, таблицы и т.д.

# Список, дерево, таблица

Тут нужно отдельно использовать модель и виджет-представитель этой модели. Несколько виджетов могут ссылаться на одну модель. Виджет здесь – представитель. Есть также модель выделения. Модель – это какая-то структура данных: список, массив или дерево. Здесь важен элемент, который хранится в ячейках этих структур. Его нужно индексировать, можно сделать последовательный итератор, общий для вектора, списка, карты. Модель списка может быть прокси-моделью модели приложения, например тупо хранить в своем списке ссылки на функции, причем такую прокси-модель можно фильтровать или сортировать. Прокси-модель должна уметь подгружать необходимый контент из модели приложения с учетом выставленной модели фильтра из GUI, чтобы потом отобразить отфильтрованный и отсортированный список функций.

Эта прокси-модель должна являться слушателем модели приложения. Например, менеджер функций может предоставить обратные вызовы о добавлении/удалении/изменении функции. Прокси модель оповещается и перегружается полностью. Никаких своих добавлений или удалений в эту прокси-модель!

**Важно**: сначала сортируем, потом фильтруем.

Прокси-модель тоже может быть прослушиваема.

Могут быть совмещенные “модель + представление” списки, куда удобно добавить статические элементы.

# Редактирование модели приложения

Упрощенно это выглядит так:

1. Загружаются данные из модели приложения в модель виджета GUI

2. Состояние модели виджета меняется в результате действий пользователя

3. Нажимается кнопка “Save”, и все изменения отправляются из модели виджета в модель приложения

# Передача зависимости из модели приложения в модель GUI

# Модификация виджетов

# Unit тестирование GUI-интерфейса

Есть класс Form, где несколько полей и кнопка. Мы генерим события специально для публичных полей и кнопки. Высокоуровневое событие ввода с клавиатуры и событие нажатия кнопки.