МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА «ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АВТОМАТИЗИРОВАННЫХ СИСТЕМ»

ОТЧЁТ
«ТВОРЧЕСКАЯ РАБОТА»

Дисциплина: «Программирование»

Выполнил:
Студент группы ИВТ-21-2б
Безух Владимир Сергеевич

Проверил:
Доцент кафедры ИТАС
Полякова Ольга Андреевна

Пермь, 2022

Содержание

1. Постановка задачи

Работа состоит из двух подзадач: сделать калькулятор и предложить решение задачи коммивояжёра. Причём в обоих случаях необходимо использовать средства графической визуализации.

2. Описание методов и инструментов

Решение любой задачи по созданию программного обеспечения состоит из нескольких этапов. Я выделю ключевые из них, а именно: этап проектирования, этап производства, этап тестирования и этап упаковки продукта для его дальнейшей реализации. Наглядную демонстрацию работы обеих программ можно посмотреть в видеоверсии отчёта на сервисе Youtube.

2.1. Калькулятор

2.1.1. Этап проектирования

На этапе проектирования мне было необходимо определиться с набором инструментов, которые я буду использовать, выделить ключевые смысловые решения, расставить приоритеты и задать сроки исполнения.

Для того, чтобы сделать пользовательский интерфейс калькулятора, я выбрал *графическую библиотеку SFML*, включающую в себя классы по отрисовки графических примитивов и классы по обработке событий.

В контексте производственной задачи я бы выбрал другой инструмент. Например, *Windows Forms* или *Qt*, — мне уже доводилось писать программы с их помощью. Тем не менее, в рамках творческой работы, по этой же причине я отказался от использования знакомых инструментов. Мне хотелось проявить свою креативность, попробовать в работе новые инструменты и, что называется, *to think out of the box*[1].

В своей работе я сконцентрировался на создании системы с точки зрения кода и его дальнейшей поддержки. Код программы написан с применением современных принципов проектирования и программирования сложных систем, что позволяет комфортно делегировать процесс масштабирования калькулятора.

[1] To think out of the box — мыслить нестандартно, выходить за рамки.

2.1.2. Этап производства

Весь проект декомпозирован и разбит на файлы:

— несколько удобных структур для абстракции данных (Point2D, Size2D);

— операции вычислений калькулятора;

— пользовательский интерфейс, UI;

— обработка пользовательского опыта, UX;

Я реализовал иерархию классов для создания элементов графического пользовательского интерфейса (рис. 1).

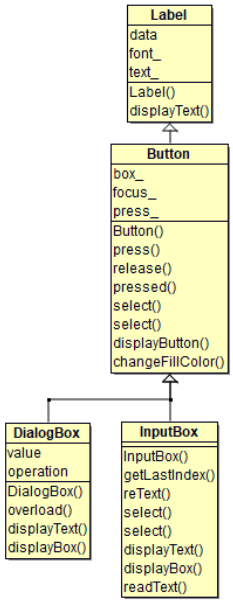

Рисунок 1 — иерархия классов элементов графического пользовательского интерфейса

Также в качестве отдельной сущности я выделил класс калькулятора, что позволило намного глубже декомпозировать обработчик событий, выделить отдельный метод отрисовки кадра. Все эти методы, разумеется, инкапсулированы, чтобы внешний пользователь не смог нарушить инвариант калькулятора.

Для того, чтобы не расставлять кнопки калькулятора вручную, я воспользовался динамическим программированием. В этом мне помог

современный метод создания констант через пространства имён. В константах я объявил и проинициализировал всевозможные величины. В том числе, количество кнопок, которые нужно отрисовать. Это позволило решить *проблему magic numbers*[2] — непонятных без контекста литералов, которые делают код менее понятным и менее гибким.

Код программы доступен в моём репозитории на Github.

2.1.3. Этап тестирования

В ходе разработки программы я уделил отдельное время тестированию программы. Калькулятор корректно обрабатывает разные сценарии поведения пользователя. Из-за ограниченного времени, тестирование носило локальный характер. Тем не менее, при разработке крупных проектов очень важно уделять внимание тестированию и «ловле ошибок» в коде.

2.1.4. Этап упаковки продукта

Презентовать продукт в качестве набора исходных файлов не самое удачное решение. Основное неудобство — динамическая линковка библиотеки SFML. Необходимо, чтобы все файлы проекта находились рядом с исполняемым файлом.

Если у пользователя есть возможность скомпилировать исходные файлы, то можно воспользоваться CMake. Мне доводилось писать небольшие CMake-скрипты ранее, но я посчитал, что в моей ситуации это займёт неоправданно много времени.

Для создания инсталлирующего файла я воспользовался инструментом Inno Setup. Это позволило аккуратно и удобно упаковать программу: установщик

[2] Роберт Мартин. Книга «Чистый код. Создание, анализ и рефакторинг». Страница 339, рекомендация G25: Заменяйте «волшебные числа» именованными константами.

автоматически разместит файлы в удобной для пользователя папке, создаст ярлык и файл деинсталляции.

В инсталляторе наглядно выводится информация о лицензии и авторских правах на программу. Как известно, защита интеллектуальной собственности, это неотъемлемый атрибут информационной эпохи.

Инсталлятор доступен в моём репозитории на Github.

2.2. Задача коммивояжёра

2.2.1. Этап проектирования

За 10 дней, которые у меня были на всю творческую работу, я реализовал далеко не все задумки.

Основная система была максимально обобщена. Это сделано намерено, чтобы не загонять систему в рамки даже самых креативных, но частных случаев. Условно говоря, у меня нет никакого коммивояжёра, который развозит товары по городам, как, впрочем, и нет самих городов. В коде всё строго: обезличенные вершины и ребра.

Тем не менее, в планах был и частный случай, реализованный на основе общей системы. Я представлял этот частный случай следующим образом: есть карта России, некоторые локации, которые нужно посетить, и у каждого существующего пути между двумя локациями, есть расстояние, время и стоимость перемещения между ними. В такой модели можно найти не только самые короткие, быстрые и дешёвые пути, но и некоторый оптимальный путь, поиск которого основывается в затратах денег и времени на количество преодолённого расстояния.

В процессе создания творческой работы, я ознакомился с различными научными работами на тему исследования задачи коммивояжёра. Оказывается, многие алгоритмы решения задачи коммивояжёра ищут не точное, а

оптимальное решение с определенной погрешностью. Причём прослеживается зависимость между скоростью работы алгоритма и величиной погрешности.

Например, перебор всех вариантов гарантированно даёт точный результат, но очень медленно работает, эвристические алгоритмы работают быстро, но со значительной погрешностью, есть алгоритмы, вроде метода ветвей и границ, позволяющие находить оптимальные решения за оптимальное время, также есть генетические алгоритмы, позволяющие искать решение, которое стремится к заданной точности. В зависимости от постановки задачи коммивояжёра, можно и нужно применять разные алгоритмы и подходы для её решения.

2.2.2. Этап производства

Я воспользовался ранее созданным для калькулятора кодом и улучшил его. На его основе создал новый элемент интерфейса, позволяющий уменьшать или увеличивать значения. Для задачи коммивояжёра реализованы классы вершин, рёбер и путей.

У меня по-прежнему есть наборы констант, которые делают код гибче. Динамическое программирование позволило мне реализовать систему смены графических тем. Более того, т.к. всё реализовано через своего рода config-файл, если я захочу попросить дизайнера сделать мне новые темы оформления для приложения, то не составит особого труда объяснить ему, как работает такой config-файл, чтобы он уже самостоятельно, даже не владея навыками программирования, смог настраивать значения и тестировать новые графические решения.

Код программы доступен в моём репозитории на Github.

2.2.3. Этап тестирования

В процессе тестирования последовательно устранялись мелкие ошибки.

2.2.4. Этап упаковки продукта

Всё аналогично упаковки калькулятора. Инсталлятор доступен в моём репозитории на Github.

3. Мои поводы для гордости

В кратчайшие сроки я изучил абсолютно новый для себя инструмент графической визуализации SFML. В процессе работы применил множество инструментов для решения поставленных задач.

Среди этих инструментов:

— офисные приложения для проектирования и документирования проекта;

— система контроля версия GitHub;

— непосредственно язык программирования C++ и среда разработки;

— программы для редактирования изображений;

— мастер по созданию инсталлятора;

— OBS для записи экрана;

— Shotcut для видеомонтажа;

— Audacity для записи и обработки звука;

— социальная сеть Youtube для публикации видеоотчёта.

Я горжусь тем, что отношусь к своей работе профессионально. Любое программное обеспечение создаётся человеком. И от того, как человек относится к созданию программы, зависит и её итоговое качество. Важно не только быть теоретиком, но и важно быть практиком. Важно уважать других программистов и писать чистый и грамотный код. Важно в силу своей квалификации прилагать максимум усилий, чтобы не занижать планку качества. Важно быть дисциплинированным и находить идеальный *work-life balance*, чтобы не выгореть морально и физически на долгой дистанции. У меня получилось стабильно работать в течение этого мини-марафона.

4. Заключение

Не комментируйте плохой код — перепишите его.

Брайан У. Керниган и П. Дж. Плауэр

В моём коде вы не найдёте большого количества комментариев. Мы вынуждены их использовать, когда нам не удаётся выразить свои мысли через синтаксические средства языка программирования. Большое количество комментариев — не повод для гордости.

Уместный комментарий доходчиво разъясняет ситуацию, но бессодержательные и безапелляционные комментарии приносят много вреда. Комментарии, потерявшие актуальность, и вовсе становятся источником дезинформации.

Чистый код является единственным источником по-настоящему достоверной информации.