Государственное образовательное учреждение высшего профессионального образования

«Московский Государственный Технический Университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»
КАФЕДРА «ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ»

КУРСОВОЙ ПРОЕКТ

Модель дорожного движения

Руководитель проекта: _____ / Бородин А.А.
(подпись, дата)

Разработчик проекта: _____ / Кошман А.А.
(подпись, дата)

Москва 2017

Оглавление

ЦЕЛЬ

Создание модели дорожного движения с графическим интерфейсом на алгоритмическом языке программирования Си++ при помощи знаний и навыков, полученных на 1 курсе по специальности «Информационная безопасность». Изучение реального прототипа дорожного движения на примере данного проекта. Возможность решения существующих проблем в системе дорожного движения с помощью ее модели.

ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ

Алгоритмический язык программирования - формальный язык, используемый для записи, реализации и изучения алгоритмов. В отличие от большинства языков программирования, алгоритмический язык не привязан к архитектуре компьютера, не содержит деталей, связанных с устройством машины.

Объектно-ориентированное программирование (ООП) - методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Модель (фр. modèle, от лат. modulus — «аналог, образец») - это система, исследование которой служит средством для получения информации о другой системе; представление некоторого реального процесса, устройства или концепции.

SFML (англ. Simple and Fast Multimedia Library - простая и быстрая мультимедийная библиотека) - свободная кроссплатформенная мультимедийная библиотека, обеспечивающая простой интерфейс для разработки игр и прочих мультимедийных приложений.

CI (англ. Continuous Integration**)** - это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок проекта для скорейшего выявления и решения интеграционных проблем.

Интерфейс (англ. interface) - общая граница между двумя функциональными объектами, требования к которой определяются стандартом. Это совокупность средств, методов и правил взаимодействия между элементами системы.

ВВЕДЕНИЕ

Модель - есть абстрактное представление реальности в какой-либо форме, предназначенное для представления определённых аспектов этой реальности и позволяющее получить ответы на изучаемые вопросы. Таким образом моделирование помогает изучить какую-либо реальную систему на примере подобной ей модели.

Создание модели дорожного движения поможет сделать анализ эффективности при изменении транспортно-эксплуатационных показателей, составить корректный прогноз относительно работы движения автомобильных транспортных средств и управление дорожными системами.

Транспортная система, ее реализация и анализ позволят решать широкий круг задач по организации существующего дорожного движения с использованием информационных и информационно-управляющих процессов.

ТРЕБОВАНИЯ К ПРОЕКТУ

Данная модель должна удовлетворять следующим требованиям:

- отделение данных от логики (основной принцип ООП)
- автономность модели, т.е. работа программы без вмешательства человека
- выполнение основных существующих правил дорожного движения
- возможность добавления действующих объектов в процессе работы программы
- максимальная приближенность к реальности
- поддержка системы Continuous Integration (CI)

ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Система данной модели дорожного движения состоит из 6 классов :

- Класс “карта” (Map)
- Класс “автомобиль” (Car)
- Класс “светофор” (TrafficLight)
- Класс “дорожный знак” (RoadSign)
- Класс “вид” (View)
- Класс “авария” (Crash)

Map – класс, главным полем которого является двумерный вектор типа char, где каждый символ отвечает за ту или иную часть карты . Ниже приведены некоторые маркеры, использующиеся в моем проекте для задания карты:

- **r** - дорога
- **p** - перекресток
- **e** - выезд
- **пробел** - зеленый фон
- **f** - контур карты
- **hl**
  mn - маркеры для сборки пункта выезда/въезда объекта Car

Рисунок 1 иллюстрирует способ задание карты в исходном коде, на Рисунке 2 изображено, как будет выглядеть карта после запуска программы.

```
"fffffffffffffffffffffffffff",
"f                          f",
"fhl pprrrrrrrrrrrrrrrrrrpp f",
"fmnrpprrrrrrrrrrrrrrrrrrpp f",
"f   rr                  rr f",
"f   rr                  rr f",
"f   rr                  rr f",
"fhl pprrrrrrrrrrrrrrrrrrpp f",
"fmnrpprrrrrrrrrrrrrrrrrrpp f",
"f                          f",
"fffffffffffffffffffffffffff"
```

Рисунок 1 – Исходный код

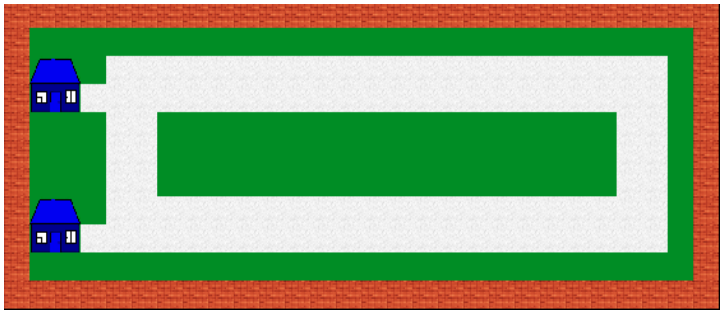

Рисунок 2 – Карта

Car - основной действующий объект, в свою структура включает алгоритмы, которые контролируют перемещение спрайта (sf::Sprite) данного объекта по карте (класс Map). Рассмотрим основные методы:

- freeDirections(const Map &) – метод выбора возможных направлений движения объекта. Основан на просмотре частей карты, находящихся рядом с объектом.
- Блок …..Around() – набор методов, основной функцией которых является определение возможности движения в заданном направлении. Вызов данных методов происходит в нижеописанном методе go(…).
    - carAround(const Map &, const bool);
    - lightAround(const Map &) const;
    - signAround(const Map &);
    - crashAround() const;

- go(sf::RenderWindow &, const Map &, bool &) – основной метод движения.
    - Вызывает метод анимации въезда и выезда объекта на карту (entryExit())
    - Вызывает freeDirections(), затем выбирает одно из предложенных свободных направлений (если их много)
    - вызывает блок методов …..Around(). Если каждый метод разрешает движение, то происходит приращение координат, установка новой позиции спрайта.
    - Также в случае возможности аварии, генерирует объект класса Crash, описанный ниже.
    - При каждом вызове рисует спрайт объекта.

На Рисунке 3 приведена иллюстрация прозвольно принятого задания направлений на карте. К примеру, при движении объекта Car в положительном направлении по оси X, его поле Direction (направление) равно 0.

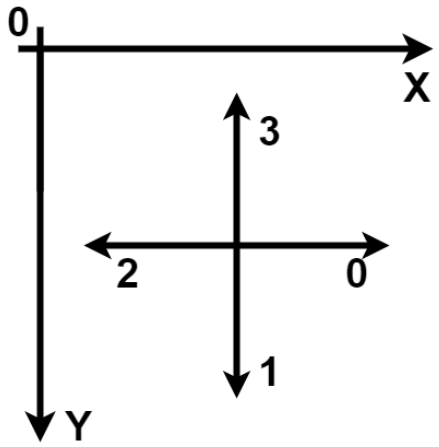

Рисунок 3 – Направления движения

ВЫБОР ТЕХНОЛОГИЙ

Выбор языка программирования

Выбор языка программирования очень важная часть в разработке любого проекта. Нужно подобрать именно тот язык, который будет самым оптимальным для конкретной разработки. Именно поэтому чем больше проект, тем больше стек технологий, который в нем используется. Глобальные проекты состоят из огромного количества подпроектов. Для каждого из подпроектов выбирается тот язык, который будет наилучшем образом справляться с задачами, поставленными в нем.

Важными критериями при выборе технологий являются:

- Размер и тип проекта
- Сложность проекта
- Скорость разработки
- Доступные инструменты разработки
- Наличие готовых решений
- Гибкость решения
- Наличие подробной документации
- Требования к нагрузкам
- Требования к безопасности
- Кроссплатформенность

- Возможность интеграции с другими решениями

Так как данная работа планировалась быть не очень объёмной, а время на разработку было ограничено, то стоял вопрос выбора одного языка, а не нескольких.

Всем известно, что языки программирования разделяются по сфере применения. Основными сферами являются веб-разработка, мобильная и игровая разработка. Самыми популярными языками в веб-разработке на данный момент являются: HTML, CSS, JavaScript, Java , Python и PHP . В разработке мобильных приложений на Android: Java, на iOS: Swift и Objective-C. И наконец в разработке игр: C#, C++, JavaScript, Java, Smalltalk. Данный проект относиться к сфере разработке игр, поэтому выбор был из последнего списка ЯП, технологии которого относятся к объектно-ориентированным языкам программирования.

Рассмотрев достоинства и недостатки каждого из перечисленных языков, я остановилась на выборе C++. Ниже приведено подробное доказательство принятого решения.

C++ - язык общего назначения и задуман для того, чтобы настоящие программисты получили удовольствие от самого процесса программирования. За исключением второстепенных деталей он содержит язык C как подмножество. Язык C расширяется введением гибких и эффективных средств, предназначенных для построения новых типов. Программист структурирует свою задачу, определив новые типы, которые точно соответствуют понятиям предметной области задачи. Такой метод построения программы обычно называют абстракцией данных. Информация о типах содержится в некоторых объектах типов, определенных пользователем. С такими объектами можно работать надежно и просто даже в тех случаях, когда их тип нельзя установить на стадии трансляции. Программирование с использованием таких объектов обычно называют объектно-ориентированным. Если этот метод применяется правильно, то программы становятся короче и понятнее, а сопровождение их упрощается.

Достоинства C++:

- Чрезвычайно мощный язык, содержащий средства создания эффективных программ практически любого назначения.
- Компилируемость со статической типизацией.
- Сочетание высокоуровневых и низкоуровневых средств.

- Реализация ООП.
- Работает максимально быстро.
- Предсказуемое выполнение программ, что является важным для построения систем реального времени.
- Автоматический вызов деструкторов объектов при их уничтожении, причём в порядке, обратном вызову конструкторов. Это упрощает (достаточно объявить переменную) и делает более надёжным освобождение ресурсов (память, файлы, семафоры и т. п.), а также позволяет гарантированно выполнять переходы состояний программы, не обязательно связанные с освобождением ресурсов (например, запись в журнал).
- Пользовательские функции-операторы позволяют кратко и ёмко записывать выражения над пользовательскими типами в естественной алгебраической форме.

- Язык поддерживает понятия физической (const) и логической (mutable) константности. Это делает программу надёжнее, так как позволяет компилятору, например, диагностировать ошибочные попытки изменения значения переменной. Объявление константности даёт программисту, читающему текст программы дополнительное представление о правильном использовании классов и функций, а также может являться подсказкой для оптимизации. Перегрузка функций-членов по признаку константности позволяет определять изнутри объекта цели вызова метода (константный для чтения, неконстантный для изменения). Объявление mutable позволяет сохранять логическую константность при использовании кэшей и ленивых вычислений.
- Поддерживаются различные стили и технологии программирования, включая традиционное директивное программирование, ООП, обобщённое программирование, метапрограммирование (шаблоны, макросы).
- Используя шаблоны, возможно создавать обобщённые контейнеры и алгоритмы для разных типов данных, а также специализировать и вычислять на этапе компиляции.
- Возможность имитации расширения языка для поддержки парадигм, которые не поддерживаются компиляторами напрямую. Например, библиотека Boost.Bind позволяет связывать аргументы функций.
- Возможность создания встроенных предметно-ориентированных языков программирования. Такой подход использует, например библиотека

Boost.Spirit, позволяющая задавать EBNF-грамматику парсеров прямо в коде C++.

- Используя шаблоны и множественное наследование можно имитировать классы-примеси и комбинаторную параметризацию библиотек. Такой подход применён в библиотеке Loki, класс SmartPrt которой позволяет, управляя всего несколькими параметрами времени компиляции, сгенерировать около 300 видов «умных указателей» для управления ресурсами.
- Кроссплатформенность: стандарт языка накладывает минимальные требования на ЭВМ для запуска скомпилированных программ. Для определения реальных свойств системы выполнения в стандартной библиотеке присутствуют соответствующие возможности (например, std::numeric_limits). Доступны компиляторы для большого количества платформ, на языке C++ разрабатывают программы для самых различных платформ и систем.
- Эффективность. Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы. Ни одна из языковых возможностей, приводящая к дополнительным накладным расходам, не является обязательной для использования — при необходимости язык позволяет обеспечить максимальную эффективность программы.
- Имеется возможность работы на низком уровне с памятью, адресами.
- Высокая совместимость с языком Си, позволяющая использовать весь существующий Си-код (код на Си может быть с минимальными переделками скомпилирован компилятором C++; библиотеки, написанные на Си, обычно могут быть вызваны из C++ непосредственно без каких-либо дополнительных затрат, в том числе и на уровне функций обратного вызова, позволяя библиотекам, написанным на Си, вызывать код, написанный на C++).

В совокупности с вышеописанным, можно сделать вывод, что в данной разработки программного обеспечения C++ - оптимальный выбор.

Выбор используемых библиотек

Существует много графических библиотек на C++. Основными являются OpenGL, QT, SFML и SDL.

В разработке данного программного обеспечения требовалась быстрая и простая библиотека. Так как у QT реализована очень медленная работа с графикой, она не рассматривалась как вариант для данного проекта.

По этой же причине не подходила библиотека SDL, которая помимо медленной работы не поддерживает концепцию ООП.

OpenGL в отличии от остальных чисто графическая библиотека. И так как в ней нет никаких средств для создания окна, ввода с клавиатуры и отрисовки кнопок, она не удовлетворяла требуемым условиям.

SFML – являлся самым оптимальным вариантом графической библиотеки в разработке данного проекта.

Преимущества SFML:

- Большой, очень простой и понятный фреймворк над разносторонними библиотеками.
- Имеет лицензию ‘zlib/png license’, что означает возможность использования в коммерческих целях.
- Наличие понятной документации с примерами.
- Обеспечивает простой интерфейс для различных компонентов ПК.
- Полностью открытый исходный код.
- Содержание ряда модулей (Audio, Window, Graphics, Main, System) для простого программирования игр и мультимедиа приложений.
- Все библиотеки независимы друг от друга.
- Поддержка концепции объективно-ориентированного программирования
- Компиляция и запуск в самых распространенных операционных системах: Windows, Linux, Mac OS X.
- Имеет официальную привязку к языкам C.

ОПИСАНИЕ ТЕХНИЧЕСКИХ РЕШЕНИЙ

В ходе написания курсового проекта возникли следующие проблемы:

1. Разрушение объекта текстуры (sf::Texture) при выходе из функции. Данная проблема была решена путем расширения времени существования текстуры до времени выполнения всей программы.
2. Создание нескольких объектов Car и наложение их текстур друг на друга за одно нажатие пробела (команда создания и добавления нового объекта Car), в связи с тем, что цикл окна длится одну микросекунду, а время нажатия пробела составляет больше, чем 1 микросекунда. Был введен условный период между добавлением новой машины, что решило данную проблему.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта были получены навыки работы с графической библиотекой SFML, освоены принципы объектно-ориентированного программирования. Была реализована модель дорожного движения, работающая автономно и максимально приближенная к реальному дорожному движению.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация: Стандартные библиотеки C++ [Электронный курс]. URL: www.cplusplus.com/reference/ (дата обращения: 05.10.2017)
2. Документация: Официальная документация по языку программирования C++ [Электронный курс]. URL: http://ru.cppreference.com/w/ (дата обращения: 10.10.2017)
3. Документация: Официальная документация по фреймворку SFML [Электронный курс]. URL: https://www.sfml-dev.org/ (дата обращения: 15.10.2017)
4. Гулицкий А. Создание игры на SFML [Электронный курс]. URL: https://habrahabr.ru/post/149071/ (дата обращения: 18.10.2017)
5. Официальный репозиторий графической библиотеки SFML [Электронный курс]. URL: https://github.com/SFML/SFML/ (дата обращения: 20.10.2017)