

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

ПРИЛОЖЕНИЕ, РЕАЛИЗУЮЩЕЕ ИГРУ «Кольцевые
гонки» С ИСПОЛЬЗОВАНИЕМ WINDOWS
FORM И ГРАФИКИ OPENGL

Выполнил: Ковалёв И.А.

Группа: ИТИ-21

Руководитель: Курочка К.С.

Гомель 2025



Актуальность и цель

Актуальность:

Во-первых, игровая индустрия продолжает активно развиваться, и создание гоночных симуляторов остаётся востребованным направлением, особенно в локальном мультиплеере. Во-вторых, использование Windows Forms и OpenGL позволяет изучить как высокоуровневые инструменты для создания интерфейсов, так и низкоуровневую графическую библиотеку, что даёт ценный опыт в работе с разными технологиями.



Цель проекта:

Разработка двухмерной игры «Кольцевые гонки» для двух игроков с использованием C#, Windows Forms и OpenGL.



Постановка задачи

Задачи проекта:

- Использование C#, Windows Forms и OpenGL для создания графического интерфейса и визуализации игрового процесса
- Применение паттернов «Фабричный метод» и «Декоратор»
- Проектирование и разработка игровой логики (физика, призы, коллизии).
- Реализация интерактивного ввода для двух игроков.



Используемые технологии

- Язык программирования: C#
- Пользовательский интерфейс: Windows Forms
- Графика: OpenGL (библиотека OpenTK)



```

classDiagram
    class Frames {
        + JFrame
        + JDialog
    }
    class Dialogs {
        + JFileChooser
        + JColorChooser
        + JColorChooser.Colors
    }
    class Controllers {
        + MainController
        + DialogController
        + ColorController
        + FileController
    }
    class Managers {
        + FileManager
        + FileDialogManager
        + ColorDialogManager
        + FileDialogController
    }
    class Frames2 {
        + JFrame
        + JDialog
    }
    class Dialogs2 {
        + JFileChooser
        + JColorChooser
        + JColorChooser.Colors
    }
    class Controllers2 {
        + MainController
        + DialogController
        + ColorController
        + FileController
    }
    class Managers2 {
        + FileManager
        + FileDialogManager
        + ColorDialogManager
        + FileDialogController
    }
    class Frames3 {
        + JFrame
        + JDialog
    }
    class Dialogs3 {
        + JFileChooser
        + JColorChooser
        + JColorChooser.Colors
    }
    class Controllers3 {
        + MainController
        + DialogController
        + ColorController
        + FileController
    }
    class Managers3 {
        + FileManager
        + FileDialogManager
        + ColorDialogManager
        + FileDialogController
    }
    Frames --> Dialogs
    Dialogs --> Controllers
    Controllers --> Managers
    Managers --> Frames2
    Frames2 --> Dialogs2
    Dialogs2 --> Controllers2
    Controllers2 --> Managers2
    Managers2 --> Frames3
    Frames3 --> Dialogs3
    Dialogs3 --> Controllers3
    Controllers3 --> Managers3
    Managers3 --> Frames
    Frames --> Dialogs
    Dialogs --> Controllers
    Controllers --> Managers
    Managers --> Frames2
    Frames2 --> Dialogs2
    Dialogs2 --> Controllers2
    Controllers2 --> Managers2
    Managers2 --> Frames3
    Frames3 --> Dialogs3
    Dialogs3 --> Controllers3
    Controllers3 --> Managers3
    Managers3 --> Frames
  
```

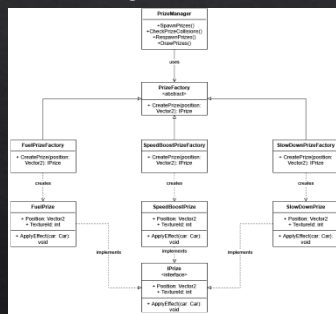
The diagram illustrates the architecture of a Java Swing application. It features several classes organized into groups: **Frames** (containing `JFrame` and `JDialog`), **Dialogs** (containing `JFileChooser`, `JColorChooser`, and `JColorChooser.Colors`), **Controllers** (containing `MainController`, `DialogController`, `ColorController`, and `FileController`), and **Managers** (containing `FileManager`, `FileDialogManager`, `ColorDialogManager`, and `FileDialogController`). The diagram shows a complex set of relationships, including inheritance (indicated by solid lines with hollow triangle heads) and associations (indicated by solid lines). For example, `Frames` is associated with `Dialogs`, which is associated with `Controllers`, which in turn is associated with `Managers`. There are also self-associations and associations between different instances of these classes, suggesting a modular and reusable design.

-

- GameManager
Управление логикой игры, обновление игрового мира
- Car
Машина, которой управляет игрок
- CarDecorator
Изменяет функционал автомобиля
- PrizeManager
Управляет призами на трассе: создание, размещение.
- CollisionMask
Проверяет столкнется ли машина с непроходимой областью трассы



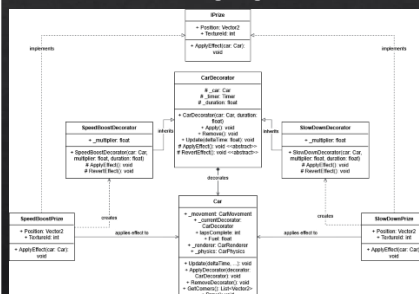
Фабричный метод



Обеспечивает создание различных типов призов (топливо, ускорение, замедление), инкапсулируя логику создания объектов в отдельном классе. Позволяет добавлять новые типы призов, без изменения существующего кода.



Декоратор



Изменяет характеристики машины, например увеличивает/уменьшает максимальную скорость при подборе приза ускорения/замедления соответственно.



Игровые механики

Управление: W,A,S,D; Стрелки.

Физика машин:

- Движение вперед/назад
- Ускорение
- Поворот
- Столкновения с границами трассы

Призы:

- Топливо
- Ускорение
- Замедление



Алгоритм работы

- 1. Запуск приложения**
Отображение главного меню
- 2. Выбор трассы и машин в главном меню**
- 3. Основной игровой цикл**
Обработка действий игроков, обновление игрового мира (машин, коллизий, индикаторов), отрисовка графики
- 4. Завершение и результаты**
Сообщение о победителе
- 5. Возврат в главное меню**



Тестирование, верификация и опытная эксплуатация

Тестирование: тестирование с использованием MSTest подтвердило корректность и стабильность основных компонентов приложения.

Верификация: проверка игровых механик показала соответствие всем функциональным требованиям.

Опытная эксплуатация: опытная эксплуатация показала стабильность и производительность. Критических ошибок выявлено не было.



Заключение

- Разработано игровое приложение «Кольцевые гонки»
- Успешно применены паттерны «Фабричный метод и декоратор»
- Корректность работы подтверждена тестированием, верификацией и эксплуатацией

