МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования
«Гомельский государственный технический университет
имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1-40 05 01-12 Информационные системы и технологии (в игровой индустрии)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
по дисциплине «Объектно-ориентированное программирование»

на тему: «Игровое приложение *Windows Form* «Кольцевые гонки» с использованием графики *OpenGL*»

Исполнитель: студент группы ИТИ-21
Ковалёв И.А.
Руководитель: доцент
Курочка К.С.

Дата проверки: _____
Дата допуска к защите: _____
Дата защиты: _____

Оценка работы: _____

Подписи членов комиссии
по защите курсового проекта: _____

Гомель 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные компьютерные технологии предоставляют множество возможностей для создания интерактивных приложений, таких как игры, которые одновременно развлекают и способствуют развитию навыков программирования, алгоритмизации и работы с графикой. Разработка игровых приложений сохраняет свою актуальность благодаря востребованности в сфере образования и индустрии развлечений. Особое внимание привлекают игры, включающие элементы стратегии и взаимодействия между игроками, поскольку они помогают развивать логическое мышление и умение принимать решения в динамичных условиях.

Эта курсовая работа сосредоточена на создании игрового приложения «Кольцевые гонки» для платформы *Windows Forms* с применением графики *OpenGL*. Главная цель – разработать полноценное приложение для двух игроков, где реализована механика гонок на одном экране с появлением призов на трассе. В процессе работы решаются задачи, связанные с созданием алгоритмов для управления автомобилями и взаимодействия объектов, использованием *OpenGL* для обеспечения качественной графики, применением шаблонов проектирования для гибкости кода, а также тестированием и проверкой работоспособности приложения.

Для реализации используются актуальные технологии и инструменты. Язык *C#* в среде *Windows Forms* упрощает разработку интерфейса и логики, а библиотека *OpenGL* обеспечивает плавное и производительное отображение игрового процесса. Применение таких шаблонов проектирования, как «фабричный метод» и «декоратор», делает код модульным и легко расширяемым, что соответствует современным подходам к разработке программного обеспечения.

Созданное приложение может стать полезным примером для изучения основ разработки игр и базой для дальнейших улучшений. В рамках работы анализируются существующие подходы, разрабатывается алгоритмическая основа, реализуется программная часть и проводится тестирование, что позволяет оценить эффективность предложенных решений.

1 ИГРОВЫЕ ПРИЛОЖЕНИЯ И СРЕДСТВА ИХ РАЗРАБОТКИ

1.1 Особенности жанра «Гонки»

Разрабатываемая игра относится к жанру «Гонки», который с первых дней своего появления неизменно привлекает внимание как игроков, так и разработчиков. Первые образцы этого жанра появились в конце 70-х – начале 80-х годов, когда аркадные автоматы с симуляторами гонок занимали центральное место в развлекательных залах торговых центров. Тогда игроки впервые испытали азарт стремительного движения и риск скоростных заездов, что быстро сделало жанр популярным среди широкой аудитории.

Одной из ключевых особенностей гоночных игр всегда было стремление передать реалистичное ощущение скорости и динамики. Разработчики уделяют особое внимание физике движения транспортных средств, тщательно моделируя нюансы ускорения, торможения, управляемости на поворотах и взаимодействия с различными дорожными покрытиями. Такая проработка позволяет игроку почувствовать себя настоящим гонщиком, где даже малейшая ошибка может повлиять на результат заезда.

С развитием технологий жанр «Гонки» претерпел значительные изменения. Современные симуляторы включают не только улучшенную физическую модель, но и продвинутые системы искусственного интеллекта, позволяющие создавать конкурентную среду с динамичным поведением соперников. Кроме того, интеграция технологий виртуальной (*VR*) и дополненной реальности (*AR*) открывает новые горизонты, позволяя полностью погрузиться в атмосферу настоящего заезда и сделать игровой процесс максимально интерактивным и захватывающим.

Немаловажным аспектом является развитие многопользовательских онлайн-соревнований. Благодаря этому игроки со всего мира могут участвовать в турнирах, чемпионатах и индивидуальных заездах, что способствует становлению киберспорта в этом направлении. Регулярное обновление контента, введение новых трасс, автомобилей и игровых режимов позволяют жанру оставаться актуальным и востребованным даже спустя десятилетия с момента его появления.

Современные гоночные игры также предлагают возможность тонкой настройки автомобилей. Игроки могут экспериментировать с характеристиками, модифицировать агрегаты и оптимизировать управляемость транспортного средства, что превращает каждый заезд в уникальное стратегическое испытание. Такой подход не только повышает вовлеченность, но и стимулирует развитие личных навыков, позволяя каждому пользователю находить индивидуальный стиль вождения.

Таким образом, жанр «Гонки» продолжает эволюционировать, объединяя в себе элементы аркадного азарта и глубоких симуляций. Технологический прогресс и инновационные подходы делают его не просто развлечением, а полноценной экосистемой, в которой каждая деталь – от реалистичной физики

до детализированного аудиовизуального оформления – способствует созданию по-настоящему захватывающего опыта для игроков всех уровней.

1.2 Игровой процесс в разрабатываемом приложении

Игровой процесс в приложениях, реализующих жанр кольцевых гонок, представляет собой сложную и многогранную систему, в которой сочетаются динамика, стратегия и реакция игрока. В данной курсовой работе разрабатывается игра «Кольцевые гонки» для двух пользователей, играющих на одном экране, где каждая секунда заезда имеет решающее значение.

Разрабатываемое приложение «Кольцевые гонки» представляет собой динамичную игру для двух пользователей, играющих на одном экране, где каждый игрок управляет гоночным автомобилем на кольцевой трассе. Цель игры – первым проехать пять кругов, при этом ключевым элементом становится не только скорость, но и грамотное использование бонусов, а также тактическое планирование расхода топлива. Ниже приведён подробный анализ основных аспектов игрового процесса, реализованных в данном проекте.

В основе приложения лежит идея состязания двух гонщиков на замкнутой трассе, где каждый заезд становится испытанием не только скорости, но и внимательности игрока. Игра начинается с недостаточным запасом топлива, что вынуждает участников активно собирать бонусы, расположенные по всей трассе. Таким образом, успех определяется умением сочетать агрессивное вождение с точным расчетом расхода топлива. Ограничение движения за границы трассы, которое приводит к снижению скорости, добавляет элемент дисциплины и требует от игрока постоянного контроля за положением автомобиля. Тестовое изображение игрового поля (трассы) представлено на рисунке 1.1.

Рисунок 1.1 – Тестовое изображение игрового поля

Одним из центральных элементов игрового процесса является система расхода топлива. При запуске заезда у автомобиля имеется ограниченный запас топлива, которого недостаточно для прохождения полного круга. Это вынуждает

игроков искать и собирать бонусы, появляющиеся в случайных местах трассы. В игре реализованы следующие типы бонусов:

– топливо: пополнение запаса, необходимого для продолжения гонки. Собирая топливо, игрок получает возможность преодолеть критические участки трассы, избегая остановок и вынужденных замедлений;

– ускорение: временное увеличение максимальной скорости автомобиля, что позволяет сделать решающий обгон на прямых участках трассы. Такой бонус может кардинально изменить расстановку сил в заезде;

– замедление: временное снижение скорости соперника или корректировка собственной динамики, что помогает точнее проходить повороты или избегать столкновений.

Эта система бонусов вносит элемент стратегии: игрокам необходимо не только своевременно реагировать на появление бонусов, но и грамотно планировать момент их использования, чтобы оптимизировать расход топлива и максимально увеличить шансы на победу.

Режим игры для двух пользователей на одном экране создает условия для непосредственного противостояния, где каждый игрок видит действия соперника в режиме реального времени. Это исключает задержки, присущие онлайн-соревнованиям, и гарантирует синхронное отображение всех игровых событий. Такая организация игрового процесса способствует более тесному взаимодействию между игроками, повышая уровень адреналина и конкурентоспособности.

При этом система управления, рассчитанная на двух игроков, позволяет каждому участнику полностью контролировать своего автомобиля. Возможность мгновенного реагирования на действия соперника и оперативное использование бонусов превращают каждую гонку в напряженное и захватывающее состязание, где успех определяется не только техническими характеристиками автомобиля, но и стратегией, быстрой реакцией и тактическим мышлением.

1.3 Графическая библиотека *OpenGL*: обзор возможностей и применение в *Windows Form* и *C#*

Графическая библиотека *OpenGL* занимает центральное место в разработке современных графических приложений благодаря своей универсальности, гибкости и высокой производительности. *OpenGL* (*Open Graphics Library*) – это кроссплатформенный *API* для работы с 2*D* и 3*D* графикой, который позволяет разработчикам использовать аппаратное ускорение для создания визуально насыщенных приложений, от научной визуализации до компьютерных игр. В контексте разработки игровых приложений, таких как «Кольцевые гонки», *OpenGL* предоставляет мощный набор инструментов для рендеринга спрайтов, создания динамичных сцен и реализации эффектов, что является критически важным для создания качественного пользовательского опыта.

OpenGL был разработан в начале 1990-х годов с целью создания стандартного интерфейса для взаимодействия с графическими процессорами. С течением времени библиотека претерпела значительные изменения – от фиксированного функционального конвейера до полностью программируемого, что позволило разработчикам использовать шейдеры для создания индивидуальных визуальных эффектов. В современных версиях (начиная с *OpenGL* 3.*x* и далее) основное внимание уделяется использованию программируемых шейдеров, что позволяет значительно расширить возможности по управлению светом, тенями, текстурированием и другими эффектами. Схема графического конвейера *OpenGL* представлена на рисунке 1.2. Такая эволюция делает *OpenGL* не только мощным, но и гибким инструментом для реализации самых современных алгоритмов рендеринга.

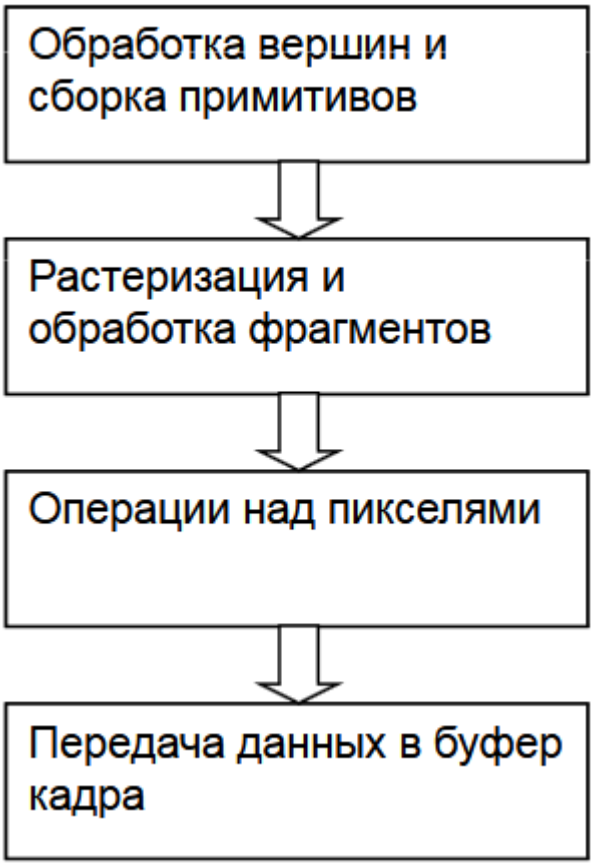

Рисунок 1.2 – Схема графического ковейера

OpenGL предоставляет ряд ключевых возможностей, которые делают его незаменимым инструментом для разработки графики:

– аппаратное ускорение: использование *GPU* позволяет значительно ускорить рендеринг, что особенно важно для динамичных сцен и игр, где требуется высокая частота обновления кадров;

– кроссплатформенность: *OpenGL* поддерживается на различных операционных системах, включая *Windows, macOS* и *Linux*, что облегчает переносимость приложений;

– поддержка 2*D* и 3*D* графики: *API* позволяет работать как с двумерными изображениями (например, спрайтовой графикой), так и с трехмерными объектами, что дает разработчику широкие возможности для творчества;

– гибкость через шейдеры: программируемые шейдеры дают возможность реализовывать индивидуальные визуальные эффекты, варьируя освещение, тени, отражения и прочие параметры в реальном времени;

– оптимизация рендеринга: многочисленные функции *OpenGL* позволяют оптимизировать процесс отрисовки за счет использования буферов, инстансинга и других современных методов.

Эти возможности обеспечивают высокую производительность и визуальную привлекательность конечного продукта, что является основополагающим для реализации игровых приложений с насыщенной графикой.

В контексте разработки приложения «Кольцевые гонки», интеграция *OpenGL* в среду *Windows Form* на языке *C#* является ключевым аспектом проекта. Использование *Windows Form* позволяет создавать удобный и привычный интерфейс для пользователей *Windows*, а применение *OpenGL* гарантирует высокую производительность графики при минимальных задержках и плавной анимации.

Для подключения *OpenGL* к *Windows Form [*4, стр. 72*]* применяются специальные библиотеки-обертки, такие как *OpenTK [*5, стр. 1*], SharpGL* или *OpenGL.Net*, которые предоставляют разработчику удобный интерфейс для вызова функций *OpenGL* из *C#*. Эти библиотеки упрощают процесс настройки контекста рендеринга, создания оконного пространства, а также управления жизненным циклом графических объектов. В рамках проекта «Кольцевые гонки» именно такой подход позволяет объединить преимущества знакомой среды *Windows Form* с мощью *OpenGL*.

При использовании *OpenGL* в *Windows Form* необходимо создать специальное окно или панель *[*4, стр. 126*]* (например, элемент управления *GLControl* в *OpenTK*), в котором будет происходить отрисовка графики. Это окно интегрируется в основное приложение, позволяя совместно с другими элементами интерфейса обеспечить удобное управление игрой. При этом разработчик должен учитывать необходимость управления ресурсами *GPU,* правильной инициализации контекста *OpenGL*, а также синхронизации обновлений экрана с игровым циклом, что особенно важно для динамичных игровых сцен.

Интеграция *OpenGL* в приложение на базе *Windows Form* и *C#* представляет собой мощное решение, объединяющее высокую производительность, гибкость и масштабируемость. Применение этой графической библиотеки позволяет не только реализовать качественную визуализацию игровых объектов и динамичных эффектов, но и интегрировать передовые архитектурные решения, такие как использование шаблонов

«фабричный метод» и «декоратор». Это, в свою очередь, способствует созданию стабильного, производительного и визуально привлекательного приложения «Кольцевые гонки», которое удовлетворяет современным требованиям как к функциональности, так и к эстетике. Использование *OpenGL* в сочетании с *Windows Form* и *C#* открывает широкие возможности для дальнейшего развития проекта и демонстрирует потенциал современных технологий в области компьютерной графики и игрового программирования.

1.4 Сравнительный анализ *OpenGL* и *DirectX*

OpenGL изначально разработан как кроссплатформенный *API*, что позволяет использовать его на различных операционных системах, включая *Windows, macOS* и *Linux*. Это делает *OpenGL* предпочтительным выбором для приложений, требующих работы на разных платформах. В отличие от него, *DirectX* является продуктом *Microsoft* и предназначен исключительно для операционных систем *Windows*. Это ограничивает его применение в средах, где требуется поддержка нескольких платформ.

OpenGL основан на клиент-серверной модели, где приложение (клиент) отправляет команды на сервер (графический драйвер), который обрабатывает их и возвращает результат. Это обеспечивает гибкость и расширяемость *API*. *DirectX*, в частности его компонент *Direct2D [*3, стр. 141*]*, использует другую архитектуру, где управление ресурсами возлагается на приложение, что предоставляет разработчикам больший контроль над процессом рендеринга.

Вопрос производительности между *OpenGL* и *DirectX* долгое время был предметом обсуждений. Ранние версии *DirectX* подвергались критике за сложность использования и низкую производительность. Однако с развитием *API Microsoft* улучшила *DirectX*, и современные версии сравнимы с *OpenGL* по производительности. Некоторые разработчики отмечают, что *DirectX* предоставляет более прямой доступ к функциям оборудования, что может положительно сказываться на производительности в определенных сценариях.

OpenGL имеет репутацию *API* с более простой и интуитивно понятной архитектурой, что облегчает его изучение и использование. *DirectX*, особенно в ранних версиях, считался более сложным для освоения из-за своей архитектуры и необходимости управления ресурсами на уровне приложения. Однако с выходом новых версий *DirectX Microsoft* упростила *API*, сделав его более доступным для разработчиков.

OpenGL позволяет производителям оборудования добавлять собственные расширения, что обеспечивает быструю поддержку новых функций графических карт. Однако это может привести к фрагментации и сложности в обеспечении совместимости между различными устройствами. *DirectX*, напротив, имеет более строгий контроль над реализацией функций, что обеспечивает большую согласованность, но может замедлить внедрение новшеств.

Хотя оба *API* изначально разрабатывались для работы с 3*D*-графикой, они также поддерживают 2*D*-графику. В случае *OpenGL* для работы с 2*D*-графикой используются ортографические проекции и соответствующие функции

рендеринга. *DirectX* включает компонент *Direct2D*, специально предназначенный для работы с 2*D*-графикой, что упрощает разработку 2*D*-приложений на платформе *Windows*. Сравнение *OpenGL* и *DirectX* представлено в таблице 1.1.

Таблица 1.1 – Сравнение *API*

Характеристика	*OpenGL*	*DirectX*
Кроссплатформенность	*Windows, macOS, Linux*	*Windows, Xbox*
Оптимизация	Хорошая, но требует дополнительной настройки	Глубокая интеграция с ОС, высокая производительность
Аппаратное ускорение	Есть, но зависит от драйверов	Гарантированное аппаратное ускорение
Сообщество	Активное, много учебных материалов	Хорошая документация от *Microsoft*

Выбор между *OpenGL* и *DirectX* зависит от конкретных требований проекта. Если необходимо обеспечить кроссплатформенность и гибкость, *OpenGL* является предпочтительным выбором. Для проектов, ориентированных исключительно на *Windows* и требующих глубокого взаимодействия с системой, *DirectX* может предоставить более оптимальные возможности.

1.5 Шаблоны проектирования в разработке игровых механизмов

Современная игровая разработка сталкивается с необходимостью создания гибких и масштабируемых систем, способных адаптироваться к изменяющимся требованиям и усложнению функционала. Шаблоны проектирования, как проверенные решения распространённых архитектурных проблем, играют ключевую роль в организации кода, обеспечивая его модульность, читаемость и устойчивость к ошибкам. В контексте игровых приложений, где механики часто взаимодействуют друг с другом, а логика должна обрабатывать множество динамических состояний, применение паттернов становится не просто рекомендацией, а необходимостью. Например, такие задачи, как генерация случайных объектов на сцене, модификация характеристик персонажей в реальном времени или управление сложными цепочками событий, требуют подходов, которые минимизируют жёсткие зависимости между компонентами и позволяют расширять функционал без переписывания существующего кода.

Одним из фундаментальных шаблонов, используемых в игровой разработке, является «фабричный метод». Его применение актуально в сценариях, где требуется создавать объекты с общим интерфейсом, но различной реализацией. Например, в играх с системой бонусов или врагов, которые появляются на уровне случайным образом, фабричный метод позволяет делегировать процесс инстанцирования специализированным классам-фабрикам. Это не только упрощает добавление новых типов объектов, но и централизует управление их параметрами. Если рассматривать генерацию

призов на трассе – таких как ускорение, топливо или временные модификаторы, – каждая категория может быть инкапсулирована в отдельную фабрику. Это обеспечивает контроль над частотой появления определённых типов бонусов, их начальными свойствами и даже условиями спауна (например, привязка к определённым участкам трассы). Кроме того, фабричный метод естественным образом вписывается в парадигму инъекции зависимостей, что упрощает тестирование отдельных компонентов игры в изоляции.

Ещё одним критически важным паттерном для динамических игровых систем выступает «декоратор». В играх, где объекты могут приобретать временные состояния или комбинировать эффекты, наследование становится непрактичным из-за экспоненциального роста числа подклассов. Декоратор решает эту проблему, позволяя оборачивать объекты в слои дополнительной функциональности. Например, автомобиль игрока, обладающий базовой скоростью и управлением, может быть динамически модифицирован декораторами, добавляющими эффекты ускорения (увеличение скорости). Каждый декоратор реализует тот же интерфейс, что и исходный объект, что обеспечивает прозрачность для клиентского кода. Это особенно полезно в многопользовательских играх, где синхронизация состояний между клиентами требует чёткого разделения логики и визуализации.

Важным аспектом является также взаимодействие шаблонов с графическими библиотеками, такими как *OpenGL*. Например, декораторы, изменяющие визуальное состояние объекта (цвет, прозрачность, анимацию), могут инкапсулировать вызовы функций рендеринга, отделяя логику эффектов от низкоуровневых операций с графическим конвейером. Это соответствует принципу разделения интерфейсов (*ISP*), где клиентский код зависит не от деталей реализации *OpenGL*, а от абстракций, предоставляемых декораторами.

Таким образом, применение шаблонов проектирования в игровых механизмах – это не просто следование стандартам, а стратегический выбор, направленный на создание устойчивой архитектуры. Их использование позволяет разработчикам сосредоточиться на творческих аспектах геймдизайна, не опасаясь, что технические долги затруднят реализацию новых идей. В условиях, когда игровая индустрия стремительно эволюционирует, умение выбирать и комбинировать паттерны становится ключевым компетенцией для любого программиста, работающего в этой области.

Список использованных источников

1. Краснов, А. В. *OpenGL*: Руководство разработчика. – М.: Диалектика, 2019. – 320 с.

2. Петров, Б. С. Программирование графики в *C#* с использованием *OpenGL*. – СПб.: Питер, 2020. – 256 с.

3. Сидоров, В. Н. Графические библиотеки и их применение в *.NET*. – Новосибирск: НГТУ, 2018. – 198 с.

4. Иванова, Г. Л. Трёхмерная графика в *Windows Forms*. — Казань: Казанский университет, 2021. – 274 с.

5. *OpenTK Documentation [*Электронный ресурс*]* // Официальный сайт *OpenTK*. *URL*: *https*://*opentk.net*/*learn*/*documentation* (дата обращения: 14.03.2025).