# **ВВЕДЕНИЕ**

Современные технологии и мобильные приложения играют важную роль в повседневной жизни современного человека, предоставляя разнообразные возможности для развлечения, обучения и общения. Одним из классических и популярных жанров в мире игр являются гонки, которые привлекают внимание игроков своей динамикой и возможностью соревноваться с другими пользователями.

Гоночные игры предоставляют игрокам возможность испытать азарт скорости, мастерства в управлении транспортными средствами и тактического мышления в борьбе за первое место. Современные мобильные гоночные приложения предлагают разнообразные трассы, различные виды транспорта и режимы соревнований, создавая захватывающий игровой опыт.

История гоночных игр на мобильных устройствах началась с появлением первых смартфонов и планшетов. С развитием графики, процессоров и возможностей сетевого взаимодействия игроков, гоночные приложения стали более реалистичными и захватывающими. Сетевой режим позволяет игрокам соревноваться друг с другом в режиме реального времени, делая игровой процесс более социальным и захватывающим.

Одним из пионеров в мире мобильных гоночных игр была "Asphalt", которая появилась в начале 2000-х. Эта серия игр стала популярной благодаря качественной графике, реалистичной физике и множеству автомобилей. С течением времени разработчики добавили множество новых возможностей, включая многопользовательские режимы, адаптивное управление и события в реальном времени.

Цель данной курсовой работы заключается в исследовании и разработке гоночного приложения для двух игроков. Проект ориентирован на создание захватывающего и соревновательного опыта, а также рассматривает технические аспекты разработки приложения, включая выбор платформы, использование языков программирования и особенности проектирования пользовательского интерфейса.

В данной пояснительной записке отражены следующие разделы:

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Анализ требований к программному средству и разработка функциональных требований;
3. Проектирование программного средства;
4. Создание (конструирование) программного средства;
5. Тестирование, проверка работоспособности и анализ полученных результатов;
6. Руководство по установке и использованию;

## 1 Анализ литературных источников

## 1.1 История

Игры являются важным компонентом человеческой культуры, и несомненно одной из наиболее исторически насыщенных игр является жанр гонок. Мало какие игры могут похвастаться такой долгой и насыщенной историей, как ранние гоночные приложения. В то время как многие считают гонки просто источником развлечения, они также играли роль в эволюции технических и графических аспектов игровой индустрии.

Первые гоночные игры с механикой геймплея появились задолго до эры 90-х и развития графических интерфейсов. Одним из первых значительных представителей этого жанра была игра "Gran Trak 10", выпущенная компанией Atari в 1974 году. Эта аркадная гонка предлагала игрокам управлять автомобилем, изображенным в виде красной линии, соревнуясь на трассе с другими автомобилями. Несмотря на простоту графики и интерфейса, "Gran Trak 10" стала первым шагом в развитии гоночных видеоигр.

Однако истинным "прародителем" гоночных приложений, как мы их знаем сегодня, можно считать игру "Pole Position", выпущенную в 1982 году. Эта аркадная гоночная игра от Namco предложила игрокам трехмерную графику и возможность управлять автомобилем с помощью руля и педалей, что сделало игровой процесс более реалистичным.

В эпоху игровых автоматов 80-х годов выделялась также увлекательная гонка "Out Run". Эта аркадная классика, выпущенная компанией Sega в 1986 году, стала символом неповторимого игрового опыта. "Out Run" предоставляла игрокам уникальную возможность путешествовать на высокоскоростных автомобилях по живописным трассам, испытывая адреналин и наслаждаясь динамикой

С течением времени, с развитием технологий и появлением персональных компьютеров, гоночные игры стали более сложными и интересными. Однако, в отличие от сапёра, которому суждено было стать частью операционной системы Windows, гоночные приложения оставались независимыми проектами, разрабатываемыми сторонними компаниями.

Таким образом, в истории гоночных игр, ранние 80-е годы выделяются как период зарождения и развития этого жанра. От аркадных автоматов до персональных компьютеров, гоночные приложения продолжали эволюционировать, открывая новые горизонты для игроков и предоставляя возможность погрузиться в захватывающий мир гонок.

## 1.2 Анализ прототипов

1.2.1 Gran Trak 10 (Atari)

Это, вероятно, самая известная версия гоночных аркадных игр — "Gran Trak 10". Она занимает особое место в истории гейминга и стала пионером в жанре гоночных симуляторов. Впервые представленная в 1974 году компанией Atari, "Gran Trak 10" с течением времени стала стандартом для аркадных залов и является вехой в развитии гоночных видеоигр.

Геймплей "Gran Trak 10" остается классическим, позволяя игрокам наслаждаться адреналином гонок. Игроки управляют автомобилем, стремясь пройти трассу за минимальное время. Игра обеспечивает возможность выбора уровня сложности, включая различные трассы и условия для адаптации под разные навыки игроков. Это позволяет каждому наслаждаться игрой в соответствии с их уровнем опыта и мастерства в мире аркадных гонок.[3]

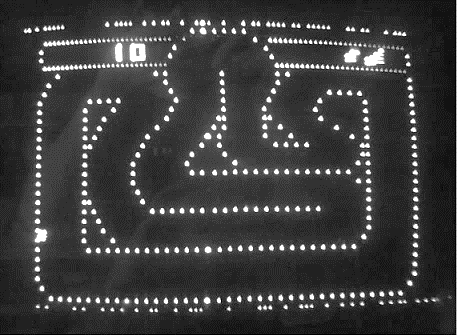


Рисунок 1.2.1 – Gran Trak 10 (Atari)

1.2.2 R.C. Pro-Am

"R.C. Pro-Am" — захватывающая переделка классической аркадной гонки "R.C. Pro-Am" с инновационным многопользовательским режимом, специально разработанная для устройств на платформе Android. Это приложение позволяет вам наслаждаться любимой гоночной аркадой вместе с друзьями или случайными соперниками, поднимая свои гоночные навыки и участвуя в захватывающих соревнованиях в реальном времени.

Игра сохраняет классический геймплей "R.C. Pro-Am", но при этом добавляет инновационный многопользовательский режим. Теперь вы можете погрузиться в уникальный мир гонок "R.C. Pro-Am", соревнуясь с другими игроками через сеть. Помимо этого, предоставляется возможность участвовать в онлайн-турнирах, бороться за лучшие результаты и занимать заслуженное место в рейтинге гонщиков.[4]



Рисунок 1.2.2 – R.C. Pro-Am

**1.3 Постановка задачи**

В своей курсовой я поставил следующие задачи:

1. Проанализировать требования, указанные в задании на курсовое проектирование
2. Изучить прототипы, их достоинства и недостатки
3. Разработать программное средство, удовлетворяющее поставленным функциональным требованиям
4. Отладить и проанализировать программное средство

Языка программирования - С, среда разработки – Visual Studio (Community), работа с окнами и отрисовкой реализована при помощи Windows API.[1][2]

## 2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

**2.1 Описание функциональных требований**

На основе проведенного анализа и с учетом требований, указанных в задании на курсовое проектирование, выделяются следующие основные требования к функционалу программного средства:

1. Реализация базовых функций управления автомобилем.
2. Aвтоматическое размещение точек интереса.
3. Создание интуитивно понятного пользовательского интерфейса.
4. Реализация основных правил игры.
5. Ведение времени, оставшегося до окончания игры.
6. Возможность игры вдвоем.

**2.2 Спецификация функциональных требований**

1. Рeализация базовых функций управления автомобилем:
   1. Возможность изменения направления движения.
   2. Возможность изменения скорости движения.
   3. Изменение отзывчивости управления в зависимости от скорости.
2. Автоматическое размещение точек интереса:
   1. Автоматическое размещение точек, при наезде на которые игроку начисляются очки.
   2. Наличие точек с разной ценностью, а так же увеличение их размера в зависимости от него.
3. Интерфейс пользователя:
   1. Отображение оставшегося времени и очков обоих игроков.
   2. Взаимодействие с игровым полем (наезд на точку интереса).
4. Логика игры:
   1. Обработка события наезда на точку интереса, ее удаление и создание новой.
   2. Отсутствие возможности управления, когда игра на паузе.
   3. Изменение координат автомобиля в случае его выезда за границы поля.
   4. Завершение игры окончании времени или нажатие клавиши для выхода.
   5. Вывод информации о победившем игроке.
5. Статистика и рейтинг:
   1. Отображение времени, оставшегося для игры.
   2. Отображение количества мин, которое нужно отметить.
6. Управление:
   1. Возможность поставить игру на паузу.
   2. Возможность досрочного выхода из игры.

## 3. ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

**3.1 Алгоритм поворота машины**

Изменение угла поворота машины относительно севера на заданную величину, с избеганием переполнения переменной.

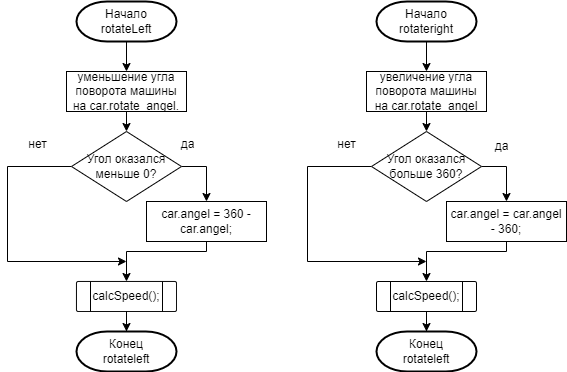


Рисунок 3.1 – Блок-схема алгоритма Поворота машины

**3.2 Алгоритм увеличения скорости**

Разложить линейную скорость машины на 2 составляющие, скорость по оси X и по оси Y, а так же изменить угол поворота машины за единичницу времени в зависимости от скорости.

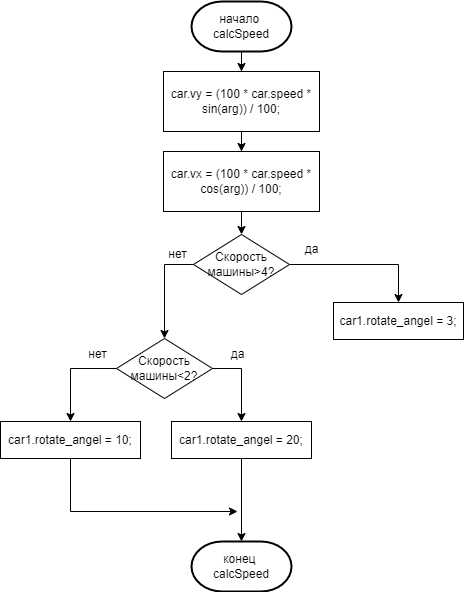


Рисунок 3.2 – Блок-схема алгоритма расставления цифр

**3.3 Алгоритм проверки коллизии машины и точки интереса**

После изменения координат машин необходимо проверить находится ли машина на точке интереса.

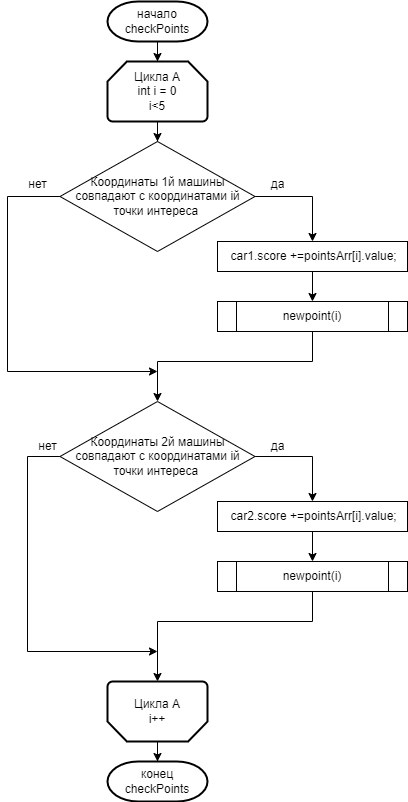


Рисунок 3.3 – проверки коллизии машины и точки интереса

**3.4 Алгоритм действия по срабатыванию таймера**

Изменить время на часах, текущее значение очков у каждого игрока, обработать нажатую клавишу, изменить положения автомобилей и запросить перерисовку окна.

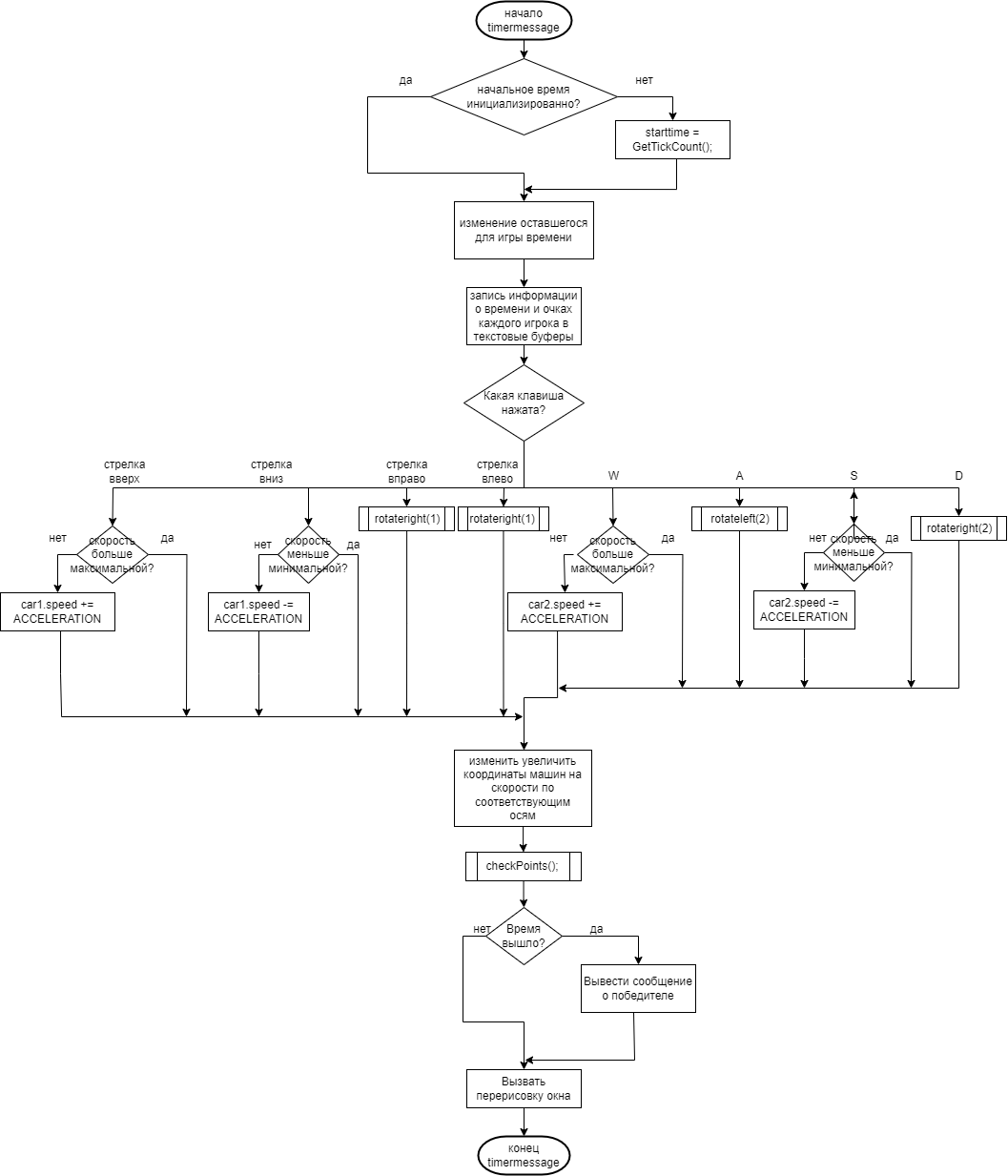


Рисунок 3.4 – Алгоритм действия по срабатыванию таймера

## 4 КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

**4.1 Интерфейс программы**

Программа является однооконной.

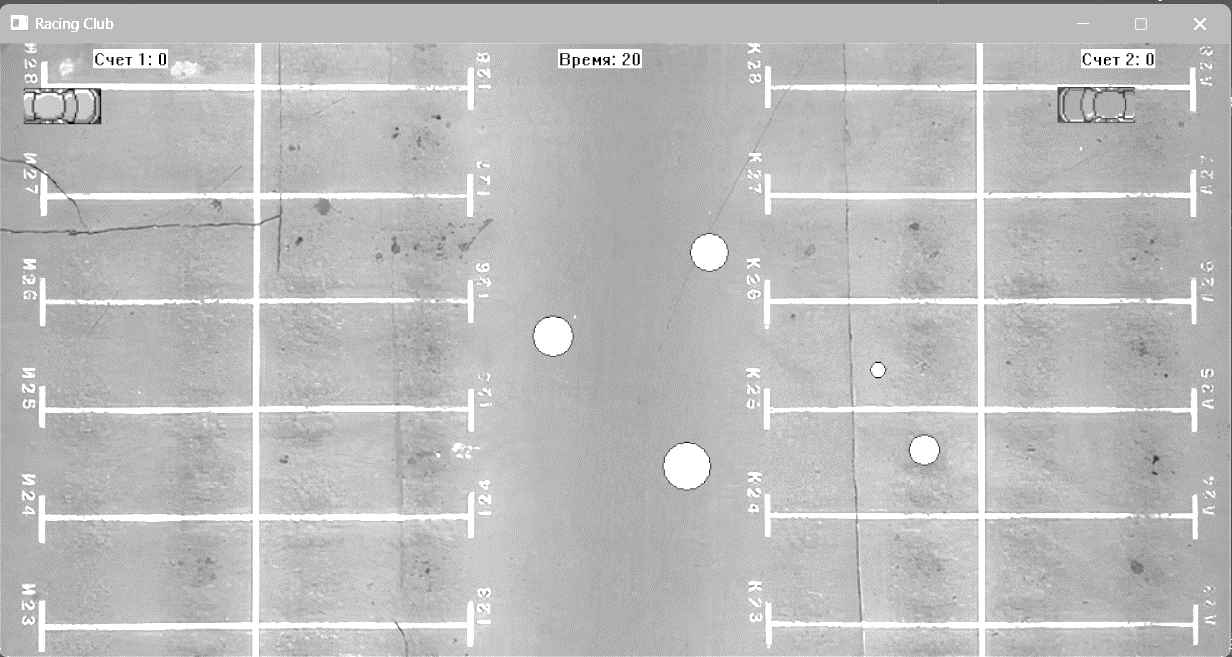


Рисунок 4.1 — Игровое поле

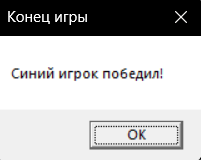


Рисунок 4.2 — Сообщение об окончании игры

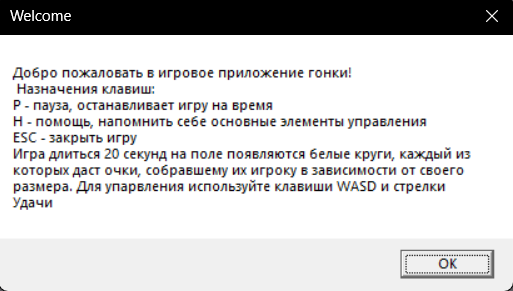


Рисунок 4.3 — Окно с информацией об управлении и правилах

**4.2 Структура модулей программы**

Программа состоит из следующих файлов:

1. Файл реализации main.c входной точки программы;
2. Заголовочный файл windows.h и файл реализации windows.c с стандартными функциями windows;

**4.3 Описание модуля main.c**

Модуль отвечает за основную логику игры, создание окна и таймера.

Таблица 4.3 - Методы модуля main

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| DrawCar | Рисует машину в передаваемых координатах под заданным углом | void DrawCar | HBITMAP hBitmap, double angle, int x, int y | hBitmap – изображение для отрисовки  x,y –координаты для отрисовки  angle - угол поворота |
| DrawBackground | Рисует фоновое изображение | void DrawBackground | - | - |

Продолжение таблицы 4.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| calcSpeed | Устанавливает параметры для уровня сложности, настраиваемого пользователем | void calcSpeed | Int I | Задает номер машины для работы |
| rotateright | Изменяет направление движения автомобиля вправо | void rotateright | Int I | Задает номер машины для работы |
| rotateleft | Изменяет направление движения автомобиля влево | void rotateleft | Int I | Задает номер машины для работы |
| newPoint | Ставит новую точку интереса на карту | void newPoint | Int i | Задает номер точки для изменения |
| checkPoints | Поверяет наехал ли автомобиль на какую-либо точку интереса | void checkPoints | - | - |
| initPoints | Инициализирует все точки интереса на карте | void initPoints | - | - |
| CALLBACK | Вызывается при отправке сообщения окну, и код для их обработки | LRESULT CALLBACK | HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam | Hwnd дескриптор окна, uMsg код сообщения,  wParam и lParam код сообщения |

Продолжение таблицы 4.3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| WINAPI | Точка входа классического приложения windows | int WINAPI | - | - |

**4.4 Описание модуля Graphics**

Модуль windows.c отвечает за работу с операционной системой.

Таблица 4.4 - Методы модуля Graphics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Имя метода** | **Описание** | **Заголовок метода** | **Имя параметра** | **Назначение параметра** |
| SetGraphicsMode | Метод, запускающийся при создании объекта класса (конструктор) | SetGraphicsMode(hdc, GM\_ADVANCED); | hdc | Контекст окна, режим графики |
| ModifyWorldTransform | Устанавливает указатель на игру | ModifyWorldTransform(hdc, &xForm, MWT\_LEFTMULTIPLY); | Hdc,XFORM,MODE | hdc - Контекст окна, XFORM матрица для изменения, MODE способ изменения |
| BitBlt | Загружает картинки | BOOL \_stdCall BitBlt | hdc,top,bot,left,right,HDC | hdc - Контекст окна,top,bot,left,right – координаты для отрисовки  HDC – исходный контекст |
| Ellipse | Рисует на экране элиппс | BOOL \_stdCall Ellipse | \_In\_ HDC hdc, \_In\_ int left, \_In\_ int top, \_In\_ int right, \_In\_ int bottom) | hdc - Контекст окна,top,bot,left,right – координаты для отрисовки |

Продолжение таблицы 4.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CreateCompatibleDC | создает контекст устройства памяти (DC), совместимый с указанным устройством | HDC WINAPI CreateCompatibleDC( \_In\_opt\_ HDC hdc); | \_In\_opt\_ HDC hdc | Дескриптор устройства |
| CreateCompatibleBitmap | оздает совместимый с указанным устройством битмап | HBITMAP WINAPI CreateCompatibleBitmap( \_In\_ HDC hdc, \_In\_ int cx, \_In\_ int cy); | In\_ HDC hdc, \_In\_ int cx, \_In\_ int cy | Дескриптор устройства, ширина и высота |
| SetTimer | Создает таймер | SetTimer(  \_In\_opt\_ HWND hWnd,  \_In\_ UINT\_PTR nIDEvent,  \_In\_ UINT uElapse,  \_In\_opt\_ TIMERPROC lpTimerFunc); | hWnd | Дескриптор окна |
| nIDEvent | Номер таймера |
| uElapse | интервал |
| TIMERPROC | Процедура для обработки сигнала таймера |
| LoadImage | загружает изображение в память и возвращает дескриптор загруженного изображения | LoadImage | HINSTANCE | Рисовальщик |
| \_In\_ LPCWSTR | Путь к файлу |
| \_In\_ UINT | Тип изображения |
| In\_ int cx,  In\_ int cy, | Ширина и высота |

Продолжение таблицы 4.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MessageBox | отображает модальное диалоговое окно, | MessageBoxW | \_In\_opt\_ HWND hWnd,  \_In\_opt\_ LPCWSTR lpText,  \_In\_opt\_ LPCWSTR lpCaption,  \_In\_ UINT uType); | Дескриптор устройства ,  Надпись внутри окна,  Название окна  Параметры окна |
| DrawTextW | рисует отформатированный текст в указанном прямоугольнике. Она форматирует текст в соответствии с указанным методом | DrawTextW | \_In\_ HDC hdc,  \_When\_((format & DT\_MODIFYSTRING), \_At\_((LPWSTR)lpchText, \_Inout\_grows\_updates\_bypassable\_or\_z\_(cchText, 4)))  \_When\_((!(format & DT\_MODIFYSTRING)), \_In\_bypassable\_reads\_or\_z\_(cchText))  LPCWSTR lpchText,  \_In\_ int cchText,  \_Inout\_ LPRECT lprc,  \_In\_ UINT format); | Дескриптор устройства, куда будет размещена надпись, ссылка на текст, прямоугольник для отрисовки текста, параметры текста |

Продолжение таблицы 4.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BeginPaint | которая подготавливает указанное окно для рисования и заполняет структуру PAINTSTRUCT информацией о рисовании. | BeginPaint | \_In\_ HWND hWnd,  \_Out\_ LPPAINTSTRUCT lpPaint); | Дескриптор окна, ссылка на структуру PAINTSTRUCT |
| EndPaint | завершает рисование в указанном окне. Эта функция необходима для каждого вызова функции BeginPaint | EndPaint  \_ | \_In\_ HWND hWnd,  \_In\_ CONST PAINTSTRUCT \*lpPaint); | Дескриптор окна, ссылка на структуру PAINTSTRUCT |
| KillTimer | удаляет указанный таймер | BOOL  WINAPI  KillTimer | (\_In\_opt\_ HWND hWnd,  \_In\_ UINT\_PTR uIDEvent); | Дескриптор окна и номер таймера |
| PostQuitMessage | отправляет сообщение WM\_QUIT в очередь сообщений потока и немедленно возвращает управление | VOID PostQuitMessage | (  \_In\_ int nExitCode); | Код завершения программы |

Продолжение таблицы 4.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| InvalidateRect | добавляет прямоугольник в область обновления указанного окна. Область обновления представляет собой часть клиентской области окна, которую необходимо перерисовать. | \_ BOOL  WINAPI  InvalidateRect | (  \_In\_opt\_ HWND hWnd,  \_In\_opt\_ CONST RECT \*lpRect,  \_In\_ BOOL bErase); | содержащую координаты клиентской области прямоугольника, который нужно добавить в область обновления, и флаг, указывающий, нужно ли стирать фон в области обновления при ее обработке |
| UpdateWindow | обновляет клиентскую область указанного окна, отправляя сообщение WM\_PAINT в окно, если область обновления окна не пуста | BOOL  WINAPI  UpdateWindow | (  \_In\_ HWND hWnd); | Дескриптор окна |

## 5 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

В данном разделе я остановлюсь на тестировании программного средства. В таблицах 5.1, 5.2, 5.3 и 5.4 представлены результаты тестирования программного средства.

**5.1 Тестирование и проверка работоспособности кнопок**

Таблица 5.1 - Тестирование и проверка работоспособности кнопок

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Увеличение скорости движения первого автомобиля | Запустить приложение, нажать стрелку вверх | Увеличение скорости передвижения автомобиля | Тест пройден |
| 2 | Увеличение скорости движения первого автомобиля | Запустить приложение, нажать кнопку W | Увеличение скорости передвижения автомобиля | Тест пройден |
| 3 | Остановка игры | Запустить приложение, нажать кнопку P | Прекращение движения машин и остановка таймера обратного отсчета | Тест пройден |
| 4 | Вывод информации об игре | Запустить приложение, нажать кнопку H | Вывод окна с сообщением о кнопках управления,  Правилах игры, а так же прекращение движения машин и работы таймера | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | Закрытие игры при помощи клавиши ESC | Запустить приложение, нажать кнопку ESC | Завершение работы приложения | Тест пройден |
| 6 | Закрытие игры при помощи клавиши ESC, когда игра на паузе | Запустить приложение, нажать кнопку P, нажать кнопку ESC | Завершение работы приложения | Тест пройден |
| 7 | Отображение окна выигрыша красного игрока | Запустить приложение, набрать, играя за красную машину очков больше, чем у синей, и дождаться окончания игры | Появление окна с сообщением о победе красного игрока | Тест пройден |
| 8 | Отображение окна выигрыша синего игрока | Запустить приложение, набрать, играя за синюю машину очков больше, чем у красной, и дождаться окончания игры | Появление окна с сообщением о победе синего игрока | Тест пройден |
| 9 | Отображение окна с сообщением о ничье | Запустить приложение, набрать на обоих машинах одинаковое количество очков | Появление окна с сообщением ничье | Тест пройден |

**5.2 Тестирование управления игровым процессом**

Таблица 5.2 - Тестирование управления игровым процессом

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Возможность разгона и торможения у первого автомобиля | Запустить приложение, нажать стрелку вперед и продержать так несколько секунд, после чего нажать стрелку назад и продержать ее так несколько секунд, при необходимости повторить несколько раз | Поочередное увеличение и уменьшение скорости красного автомобиля | Тест пройден |
| 2 | Возможность разгона и торможения у второго автомобиля | Запустить приложение, нажать клавишу W вперед и продержать так несколько секунд, после чего нажать клавишу S и продержать ее так несколько секунд | Поочередное увеличение и уменьшение скорости синего автомобиля | Тест пройден |
| 3 | Уменьшение угла поворота при повышении скорости у первой машины | Запустить приложение, нажать стрелку вперед и продержать так несколько секунд, после чего попробовать повернуть | Значительное увеличение радиуса поворота | Тест пройден |

Продолжение таблицы 5.2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | Уменьшение угла поворота при повышении скорости у второй машины | Запустить приложение, нажать клавишу W и продержать так несколько секунд, после чего попробовать повернуть | Значительное увеличение радиуса поворота | Тест пройден |
| 5 | Увеличение угла поворота при понижении скорости красного автомобиля | Запустить приложение, нажать стрелку назад и продержать так несколько секунд, после чего попробовать повернуть | Значительное уменьшение радиуса поворота | Тест пройден |
| 6 | Увеличение угла поворота при понижении скорости синего автомобиля | Запустить приложение, нажать клавишу S и продержать так несколько секунд, после чего попробовать повернуть | Значительное уменьшение радиуса поворота | Тест пройден |
| 7 | Невозможность управления красной машиной на паузе | Запустить приложение, нажать клавишу P, после чего попробовать повернуть | Отсутствие движения красной машины | Тест пройден |
| 8 | Невозможность управления синей машиной на паузе | Запустить приложение, нажать клавишу P, после чего попробовать повернуть | Отсутствие движения синей машины | Тест пройден |

**5.3 Тестирование игровой логики**

Таблица 5.1 - Тестирование игровой логики

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Тестируемая функциональность** | **Последовательность действий** | **Ожидаемый результат** | **Полученный результат** |
| 1 | Увеличение очков красного игрока при наезде на точку интереса | Запустить приложение, управляя красной машиной наехать на точку интереса | Увеличение очков первого игрока | Тест пройден |
| 2 | Увеличение очков синего игрока при наезде на точку интереса | Запустить приложение, управляя синей машиной наехать на точку интереса | Увеличение очков второго игрока | Тест пройден |
| 3 | Корректное отображение победителя | Набрать на красной машине очков больше чем у синей и дождаться окончания игры, повторить но для синей машины | Корректное отображение победителя | Тест пройден |
| 4 | Начало нового раунда после начала прошлого | Дождаться окончания игры и сообщения о исходе, после чего закрыть его | Запуск новой игры | Тест пройден |

**5.5 Итоги тестирования**

Подводя итог, отмечу, что программа отвечает заданным функциональным требованиям, наблюдается стабильность в работе. Вопросов к эстетической части не имеется.

**6. РУКОВОДСТВО ПО УСТАНОВКЕ И ИСОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА**

Для корректной работы программы необходимо распаковать архив в удобную вам папку, чтобы начать использовать программное средство, необходимо запустить файл Racingclub.exe, для корректного отображения текстур, оставить их в папке images.

После открытия программы появляется окно с игровым полем (рис. 6.1)

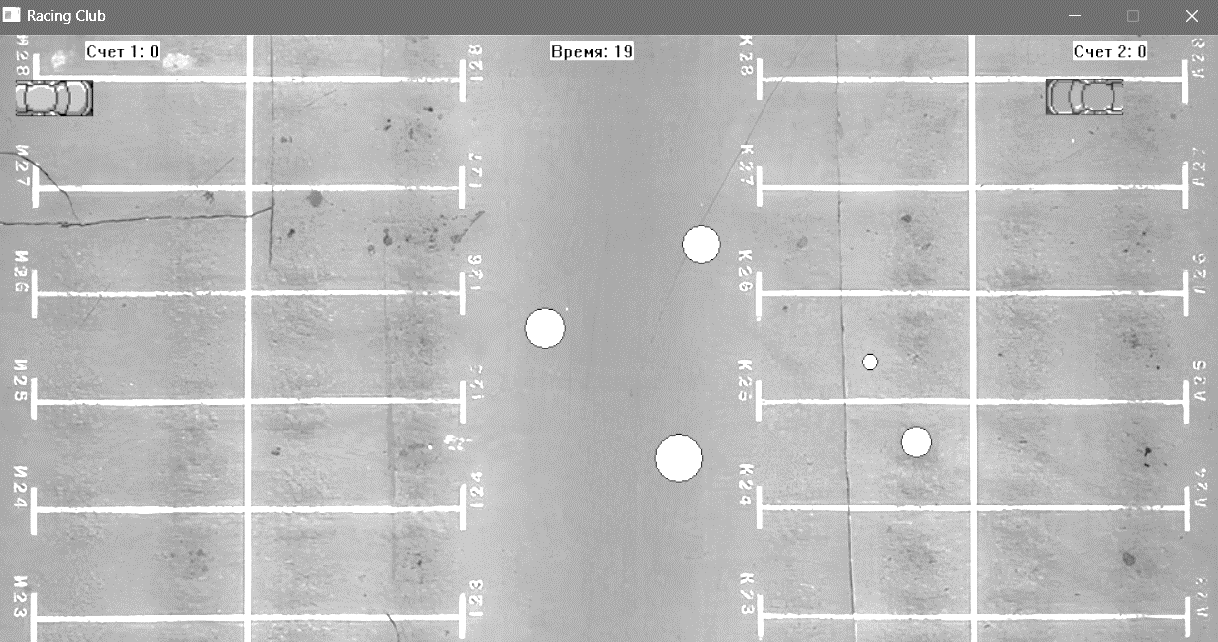


Рисунок 6.1 – Игровое поле

При нажатии кнопки H появляется окно с информацией об управлении и правилах:

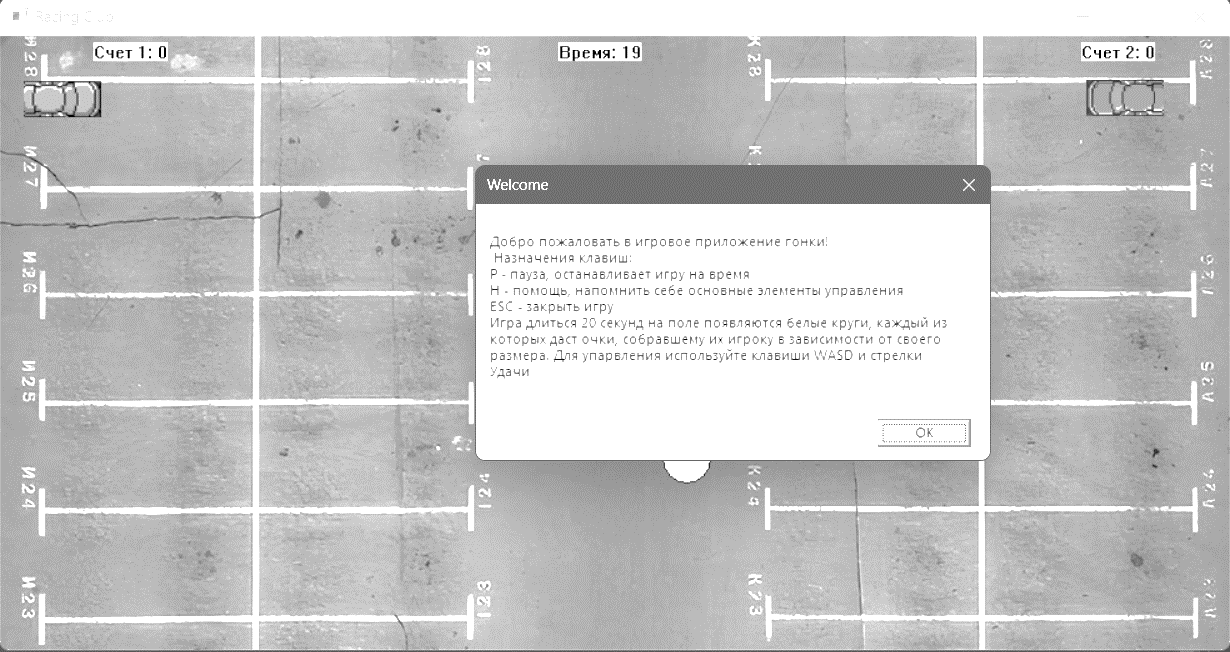


Рисунок 6.2 – Правила игры

При нажатии кнопки P игра ставиться на паузу, и движение машин оканчивается

Изначально машины стоят на месте, но после нажатия какой-либо клавиши для их управления они начнут движение

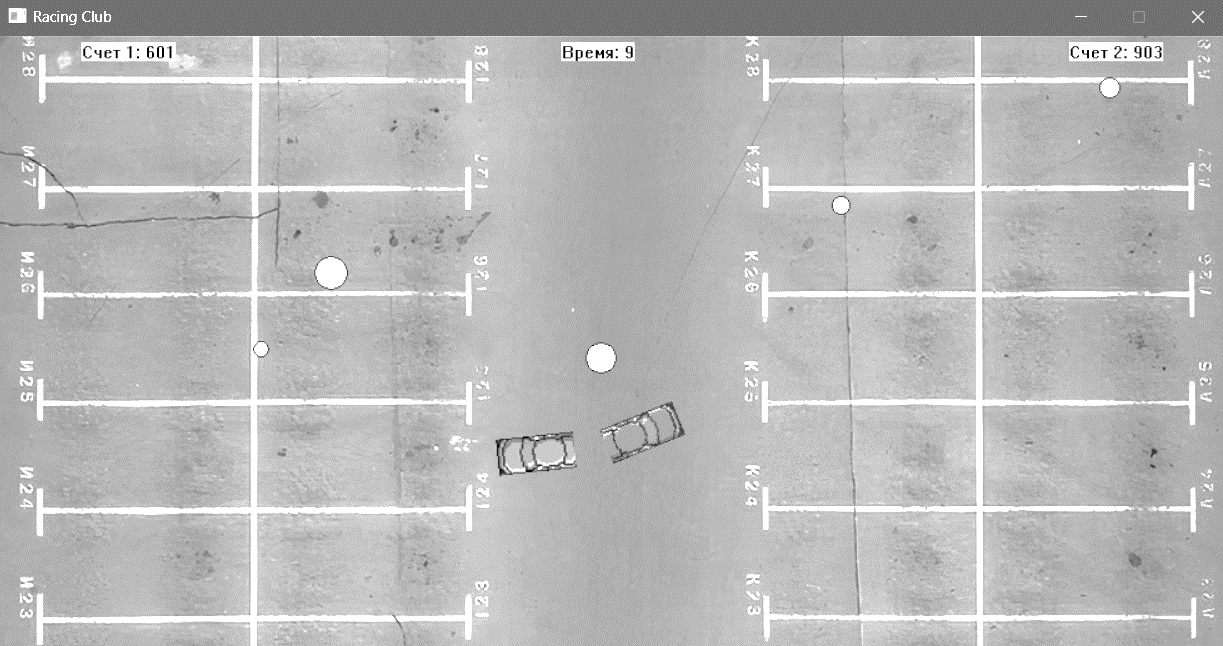


Рисунок 6.4 – Игровое поле после начала двужения

На игровом поле присутствуют белые круги, с черным контуром - это точки интереса, при наезде на них ваше количество очков будет увеличиваться пропорционально их размерам. Соберите больше очков чем ваш оппонент и тогда вы победите:

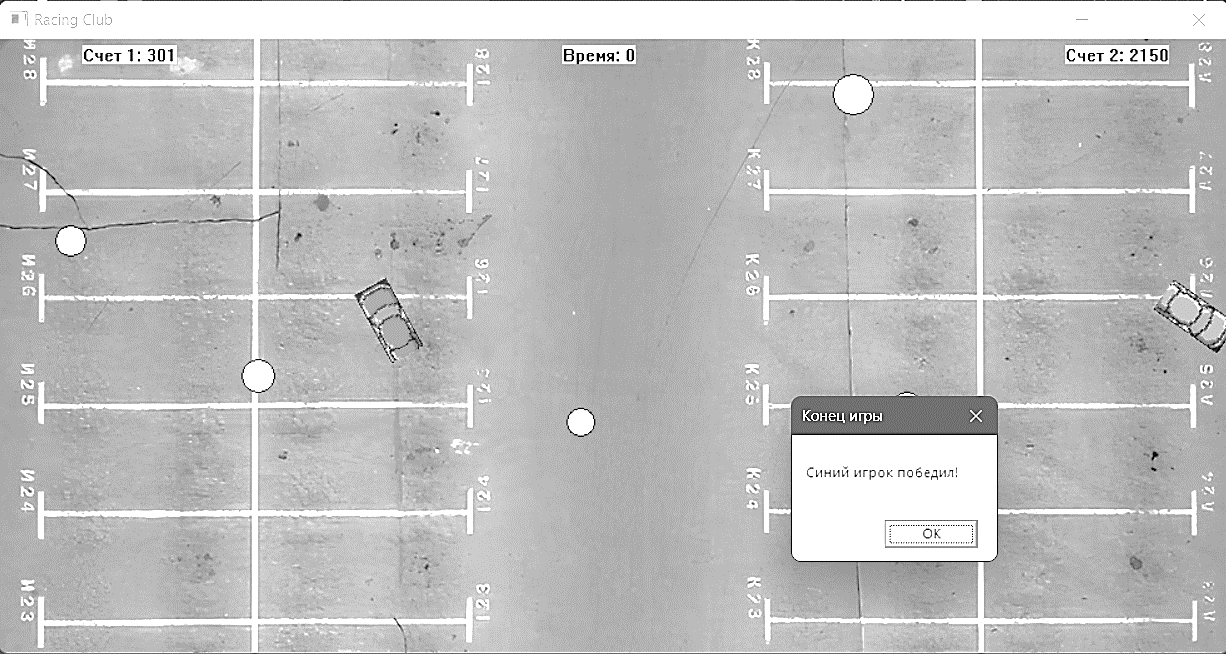


Рисунок 6.5– Игровое поле среднего уровня сложности

После окончания игры, закройте окно победителя, если вы хотите продолжить играть, то игра сама запустится заново, если же вы решите выйти из нее, то нажмите клавишу esc чтобы закрыть приложение.

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсового проекта была осуществлена тщательная анализ предметной области, изучены существующие аналоги гоночных приложений, выделены их сильные и слабые стороны. Решено было использовать язык программирования C с использованием библиотеки WinAPI для создания гоночного приложения.

На этапе проектирования были разработаны блок-схемы алгоритмов, учитывающие особенности взаимодействия с WinAPI. Работа с графикой, вводом данных и обработкой событий были внимательно продуманы и представлены в виде детальных планов.

В ходе разработки приложения "Racing Club", использующего WinAPI, был получен ценный опыт работы с низкоуровневыми функциями Windows для обеспечения графического интерфейса и обработки событий. Программа предоставляет увлекательное гоночное взаимодействие, позволяя пользователям наслаждаться геймплеем в реальном времени.

Созданное гоночное приложение помимо развлечения также способствует развитию реакции и моторики пользователя, предоставляя положительный опыт виртуальных гонок. Тестирование программы подтвердило ее стабильную работоспособность и соответствие заявленным техническим требованиям.

Проведено тестирование работоспособности разработанной программной части. Поставленная цель была выполнена в полном объеме, работоспособность подтверждена тестированием программного средства.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

# **Список использованной литературы**

[1] https://ru.wikipedia.org/wiki/Windows\_API

[2] https://learn.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes

[3] https://en.wikipedia.org/wiki/Gran\_Trak\_10

[4] https://ru.wikipedia.org/wiki/R.C.\_Pro-Am

**Приложение А**

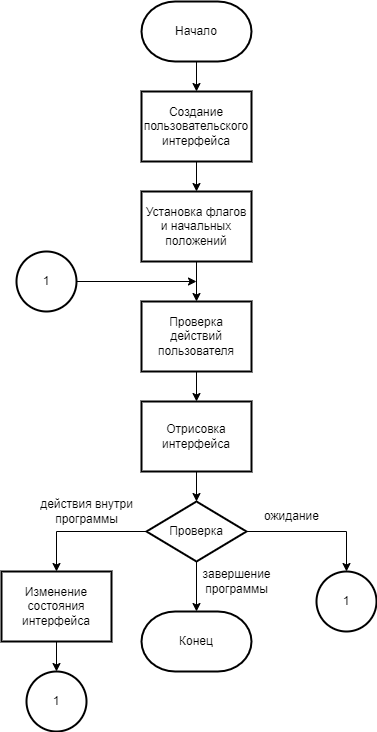
****

Рисунок А.1 – Блок-схема программы

**Приложение Б**

**main.h**

#include <windows.h>

#include <math.h>

#define ID\_TIMER 1

#define START\_VX 5

#define START\_VY 5

#define START\_X1 50

#define START\_Y 50

#define START\_X2 900

#define ACCELERATION 0.2

#define MAX\_SPEED 7

#define MIN\_SPEED 1

#define CAR\_START\_SPEED 3

#define ROTATE\_ANGEL 10

#define RADIAN 57.2958

#define TIMER\_INTERVAL 20

typedef struct cars {

HBITMAP hBitmap;

BOOL RIGHT;

BOOL LEFT;

BOOL UP;

BOOL DOWN;

int rotate\_angel;

double speed;

int angel;

int pos;

double x;

double y;

double vx;

double vy;

int ax;

int ay;

int score;

}carinfo;

typedef struct point {

int x;

int y;

int r;

int value;

}sPoint;

sPoint pointsArr[5];

XFORM xForm;

// Глобальные переменные

HBITMAP hBitmapBackground;

HBITMAP hBitmapcar1U;

HBITMAP hBitmapcar1;

HBITMAP hBitmapcar1D;

HBITMAP hBitmapcar1L;

HBITMAP hBitmapcar1R;

HBITMAP hBitmapcar2;

carinfo car1, car2;

int xPos = 0;

int yPos = 0;

DWORD starttime;

DWORD timerbefor,timerPtime;

BOOL timer;

static RECT rectTimer, rectScore1, rectScore2;

void DrawCar(HDC hdc, HBITMAP hBitmap, double angle, int x, int y) {

BITMAP bm;

GetObject(hBitmap, sizeof(BITMAP), &bm);

HDC hdcMem = CreateCompatibleDC(hdc);

HBITMAP hOldBitmap = (HBITMAP) SelectObject(hdcMem, hBitmap);

// Создаем матрицу трансформации и вращаем ее под нужным углом

xForm.eM11 = (FLOAT)cos(angle);

xForm.eM12 = (FLOAT)sin(angle);

xForm.eM21 = (FLOAT)-sin(angle);

xForm.eM22 = (FLOAT)cos(angle);

xForm.eDx = (FLOAT)x;

xForm.eDy = (FLOAT)y;

SetGraphicsMode(hdc, GM\_ADVANCED);

ModifyWorldTransform(hdc, &xForm, MWT\_LEFTMULTIPLY);

// Рисуем изображение

BitBlt(hdc, 0- 14, 0 - 31, bm.bmWidth + 15, bm.bmHeight + 31, hdcMem, 0, 0, SRCCOPY);

// Возвращаем исходные настройки

ModifyWorldTransform(hdc, NULL, MWT\_IDENTITY);

SetGraphicsMode(hdc, GM\_COMPATIBLE);

SelectObject(hdcMem, hOldBitmap);

DeleteDC(hdcMem);

}

void DrawBackground(HDC hdc) {

// Отображение изображения

HDC hdcMem = CreateCompatibleDC(hdc);

// HDC hdcMem2 = CreateCompatibleDC(hdc);

SelectObject(hdcMem, hBitmapBackground);

/\*switch (car1.pos) {

case 0:

SelectObject(hdcMem2, hBitmapcar1U);

break;

case 1:

SelectObject(hdcMem2, hBitmapcar1R);

break;

case 2:

SelectObject(hdcMem2, hBitmapcar1D);

break;

case 3:

SelectObject(hdcMem2, hBitmapcar1L);

break;

}\*/

BitBlt(hdc, 0, 0, 1000, 500, hdcMem, 0, 0, SRCCOPY);

/\* switch (car1.pos) {

case 0:

case 2:

BitBlt(hdc, car1.x, car1.y, car1.x + 29, car1.y + 62, hdcMem2, 0, 0, SRCCOPY);

break;

case 1:

case 3:

BitBlt(hdc, car1.x-20, car1.y+ 20 , car1.x -20 +62, car1.y + 20 + 29, hdcMem2, 0, 0, SRCCOPY);

break;

}\*/

DeleteDC(hdcMem);

// DeleteDC(hdcMem2);

}

void calcSpeed(int i) {

if(i == 1){

int temp = car1.angel;

double arg = temp / RADIAN;

car1.vy = (100 \* car1.speed \* sin(arg)) / 100;

car1.vx = (100 \* car1.speed \* cos(arg)) / 100;

if (car1.speed > 4)

car1.rotate\_angel = 3;

else

if (car1.speed < 2)

car1.rotate\_angel = 20;

else

car1.rotate\_angel = 10;

}

else {

int temp = car2.angel;

double arg = temp / RADIAN;

car2.vy = (100 \* car2.speed \* sin(arg)) / 100;

car2.vx = (100 \* car2.speed \* cos(arg)) / 100;

if (car2.speed > 4)

car2.rotate\_angel = 3;

else

if (car2.speed < 2)

car2.rotate\_angel = 15;

else

car2.rotate\_angel = 10;

}

}

void calcpos(int i) {

//if ((car1.ax > 0) && (car1.ax >= START\_AX - 1))

// car1.pos = 1;

//if ((car1.ax < 0) && (-1 \* car1.ax >= START\_AX - 1))

// car1.pos = 3;

//if ((car1.ay > 0) && (car1.ay >= START\_AY - 1))

// car1.pos = 2;

//if ((car1.ay < 0) && (car1.ay >= START\_AY - 1))

// car1.pos = 0;

//if (car1.ax > START\_AX / 2)

// car1.pos = 0;

//if (car1.ax <= -1 \* START\_AX / 2)

// car1.pos = 2;

//if (car1.ay > START\_AY / 2)

// car1.pos = 1;

//if (car1.ay <= -1 \* START\_AY / 2)

// car1.pos = 3;

if (i == 1) {

if (car1.angel >= 45 && car1.angel < 135)

car1.pos = 1;

if (car1.angel >= 135 && car1.angel < 225)

car1.pos = 2;

if (car1.angel >= 225 && car1.angel < 315)

car1.pos = 3;

if (car1.angel >= 315 || car1.angel < 45)

car1.pos = 0;

car1.pos = car1.pos + 1;

car1.pos %= 4;

}

else {

if (car2.angel >= 45 && car2.angel < 135)

car2.pos = 1;

if (car2.angel >= 135 && car2.angel < 225)

car2.pos = 2;

if (car2.angel >= 225 && car2.angel < 315)

car2.pos = 3;

if (car2.angel >= 315 || car2.angel < 45)

car2.pos = 0;

car2.pos = car2.pos + 1;

car2.pos %= 4;

}

}

void rotateleft(int i) {

/\* if (car1.ax >= 0 && car1.ay >= 0) {

if (car1.ax > -1 \* START\_AX - 1)

car1.ax--;

if (car1.ay < START\_AY + 1)

car1.ay++;

}

if (car1.ax >= 0 && car1.ay < 0) {

if (car1.ax < START\_AX + 1)

car1.ax++;

if (car1.ay < START\_AY + 1)

car1.ay++;

}

if (car1.ax < 0 && car1.ay < 0) {

if (car1.ax < START\_AX + 1)

car1.ax++;

if (car1.ay > -1 \* START\_AY - 1)

car1.ay--;

}

if (car1.ax < 0 && car1.ay >= 0) {

if (car1.ax > -1 \* START\_AX - 1)

car1.ax--;

if (car1.ay > -1 \* START\_AY - 1)

car1.ay--;

}\*/

if(i==1){

car1.angel = (car1.angel - car1.rotate\_angel);

if (car1.angel < 0)

car1.angel = 360 - car1.angel;

calcSpeed(1);

}

else {

car2.angel = (car2.angel - car2.rotate\_angel);

if (car2.angel < 0)

car2.angel = 360 - car2.angel;

calcSpeed(2);

}

// calcpos();

}

void rotateright(int i) {

/\*if (car1.ax >= 0 && car1.ay >= 0) {

if (car1.ax < START\_AX + 1)

car1.ax++;

if (car1.ay > -1 \* START\_AY - 1)

car1.ay--;

}

if (car1.ax >= 0 && car1.ay < 0) {

if(car1.ax > -1 \* START\_AX - 1)

car1.ax--;

if(car1.ay > -1 \* START\_AY - 1)

car1.ay--;

}

if (car1.ax < 0 && car1.ay < 0) {

if (car1.ax > -1 \* START\_AX - 1)

car1.ax--;

if (car1.ay < START\_AY + 1)

car1.ay++;

}

if (car1.ax < 0 && car1.ay >= 0) {

if (car1.ax < START\_AX + 1)

car1.ax++;

if(car1.ay < START\_AY + 1)

car1.ay++;

}\*/

if(i==1){

car1.angel = (car1.angel + car1.rotate\_angel) % 361;

calcSpeed(1);

// calcpos();

}

else {

car2.angel = (car2.angel + car2.rotate\_angel) % 361;

calcSpeed(2);

}

}

void newPoint(int i) {

pointsArr[i].r = rand() % 15 + 5;

pointsArr[i].x = rand() % 950 + 25;

pointsArr[i].y = rand() % 360 + 20;

pointsArr[i].value = pointsArr[i].r \* 10 + rand() % 10;

}

void initPoints() {

for (int i = 0; i < 5; i++)

{

newPoint(i);

}

}

void checkPoints() {

for (int i = 0; i <5; i++)

{

if((car1.x + 50 > pointsArr[i].x ) && (car1.x - 50 < pointsArr[i].x))

if ((car1.y + 50 > pointsArr[i].y) && (car1.y - 50 < pointsArr[i].y))

{

car1.score += pointsArr[i].value;

newPoint(i);

}

if ((car2.x + 50 > pointsArr[i].x) && (car2.x - 50 < pointsArr[i].x))

if ((car2.y + 50 > pointsArr[i].y) && (car2.y - 50 < pointsArr[i].y))

{

car2.score += pointsArr[i].value;

newPoint(i);

}

}

}

void drawPoints(HDC hdc, int i) {

// Ellipse(hdc, pointsArr[4].x - pointsArr[4].r, pointsArr[4].x - pointsArr[4].r, pointsArr[4].x + pointsArr[4].r, pointsArr[4].x + pointsArr[4].r);

// Ellipse(hdc, pointsArr[3].x - pointsArr[3].r, pointsArr[3].x - pointsArr[3].r, pointsArr[3].x + pointsArr[3].r, pointsArr[3].x + pointsArr[3].r);

// Ellipse(hdc, pointsArr[2].x - pointsArr[2].r, pointsArr[2].x - pointsArr[2].r, pointsArr[2].x + pointsArr[2].r, pointsArr[2].x + pointsArr[2].r);

// Ellipse(hdc, pointsArr[1].x - pointsArr[1].r, pointsArr[1].x - pointsArr[1].r, pointsArr[1].x + pointsArr[1].r, pointsArr[1].x + pointsArr[1].r);

// Ellipse(hdc, pointsArr[0].x - pointsArr[0].r, pointsArr[0].x - pointsArr[0].r, pointsArr[0].x + pointsArr[0].r, pointsArr[0].x + pointsArr[0].r);

Ellipse(hdc, pointsArr[i].x - pointsArr[i].r, pointsArr[i].y - pointsArr[i].r, pointsArr[i].x + pointsArr[i].r, pointsArr[i].y + pointsArr[i].r);

}

// Обработчик сообщений окна

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam) {

int timeleft;

static HDC hdcBuffer;

static HBITMAP hbmBuffer;

hdcBuffer = CreateCompatibleDC(NULL);

hbmBuffer = CreateCompatibleBitmap(GetDC(hwnd), 1000,500);

SelectObject(hdcBuffer, hbmBuffer);

static TCHAR szBuffer[100];

static TCHAR szBufferscore1[100];

static TCHAR szBufferscore2[100];

switch (uMsg) {

case WM\_CREATE:

rectTimer.left = 410;

rectTimer.right = 550;

rectTimer.top = 5;

rectTimer.bottom = 20;

rectScore1.top = 5;

rectScore1.bottom = 20;

rectScore1.left = 30;

rectScore1.right = 180;

rectScore2.top = 5;

rectScore2.bottom = 20;

rectScore2.left = 1000-180;

rectScore2.right = 1000-30;

car1.x = START\_X1;

car1.y = START\_Y;

car2.x = START\_X2;

car2.y = START\_Y;

car1.speed = CAR\_START\_SPEED;

car2.speed = CAR\_START\_SPEED;

car1.rotate\_angel = ROTATE\_ANGEL;

car2.rotate\_angel = ROTATE\_ANGEL;

car2.angel += 180;

SetTimer(hwnd, ID\_TIMER, TIMER\_INTERVAL, NULL);

timer = TRUE;

initPoints();

// Загрузка изображения задника

hBitmapBackground = (HBITMAP)LoadImage(NULL, L"C:/MYSTUFF/Kursach/images/backgrnd.bmp", IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

if (hBitmapBackground == NULL) {

MessageBox(hwnd, L"Failed to load image!", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

car1.hBitmap = (HBITMAP)LoadImage(NULL, L"C:/MYSTUFF/Kursach/images/carU.bmp", IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

if (car1.hBitmap == NULL) {

MessageBox(hwnd, L"Failed to load image!", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

car2.hBitmap = (HBITMAP)LoadImage(NULL, L"C:/MYSTUFF/Kursach/images/car2U.bmp", IMAGE\_BITMAP, 0, 0, LR\_LOADFROMFILE);

if (car1.hBitmap == NULL) {

MessageBox(hwnd, L"Failed to load image!", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

// Рисуем на скрытом буфере

DrawBackground(hdcBuffer);

DrawText(hdcBuffer, szBuffer, -1, &rectTimer, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);

DrawText(hdcBuffer, szBufferscore1, -1, &rectScore1, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);

DrawText(hdcBuffer, szBufferscore2, -1, &rectScore2, DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER);

DrawCar(hdcBuffer, car1.hBitmap, (double)(car1.angel + 90) / RADIAN, car1.x, car1.y);

DrawCar(hdcBuffer, car2.hBitmap, (double)(car2.angel + 90) / RADIAN, car2.x, car2.y);

for (int i = 0; i < 5; i++) {

drawPoints(hdcBuffer, i);

}

// Копируем скрытый буфер на видимый

BitBlt(hdc, 0, 0, 1000, 500, hdcBuffer, 0, 0, SRCCOPY);

EndPaint(hwnd, &ps);

}

break;

case WM\_TIMER:

if (starttime == 0)

starttime = GetTickCount();

timeleft = (int)(20 - ((GetTickCount() - starttime - timerPtime) / 1000));

wsprintf(szBuffer, TEXT("Время: %d"), timeleft);

wsprintf(szBufferscore1, TEXT("Счет 1: %d"), (int)car1.score);

wsprintf(szBufferscore2, TEXT("Счет 2: %d"), (int)car2.score);

if (GetAsyncKeyState(VK\_UP) & 0x8000) {

if (timer) {

if (car1.speed < MAX\_SPEED)

car1.speed += ACCELERATION;

calcSpeed(1);

}

}

if (GetAsyncKeyState(VK\_DOWN) & 0x8000) {

if (timer) {

if (car1.speed > MIN\_SPEED)

car1.speed -= ACCELERATION;

calcSpeed(1);

}

}

if (GetAsyncKeyState(VK\_RIGHT) & 0x8000) {

if (timer) {

rotateright(1);

car1.x += car1.vx;

car1.y += car1.vy;

}

}

if (GetAsyncKeyState(VK\_LEFT) & 0x8000) {

if (timer) {

rotateleft(1);

car1.x += car1.vx;

car1.y += car1.vy;

}

}

// 2 car

if (GetAsyncKeyState('W') & 0x8000) {

if (timer) {

if (car2.speed < MAX\_SPEED)

car2.speed += ACCELERATION;

calcSpeed(2);

}

}

if (GetAsyncKeyState('S') & 0x8000) {

if (timer) {

if (car2.speed > MIN\_SPEED)

car2.speed -= ACCELERATION;

calcSpeed(2);

}

}

if (GetAsyncKeyState('D') & 0x8000) {

if (timer) {

rotateright(2);

car2.x += car2.vx;

car2.y += car2.vy;

}

}

if (GetAsyncKeyState('A') & 0x8000) {

if (timer) {

rotateleft(2);

car2.x += car2.vx;

car2.y += car2.vy;

}

}

//car1

car1.x += car1.vx;

if (car1.x > 1000){

car1.x -= 1000;

}

if (car1.x < 0) {

car1.x += 1000;

}

car1.y += car1.vy;

if (car1.y > 500){

car1.y -= 500;

}

if (car1.y < 0)

car1.y += 500;

//car2

car2.x += car2.vx;

if (car2.x > 1000) {

car2.x -= 1000;

}

if (car2.x < 0) {

car2.x += 1000;

}

car2.y += car2.vy;

if (car2.y > 500) {

car2.y -= 500;

}

if (car2.y < 0)

car2.y += 500;

checkPoints();

if (timeleft <= -1) {

KillTimer(hwnd, ID\_TIMER);

if(car1.score>car2.score)

MessageBox(hwnd, L"Красный игрок победил!", L"Конец игры", MB\_OK );

else

if(car1.score<car2.score)

MessageBox(hwnd, L"Синий игрок победил!", L"Конец игры", MB\_OK );

else

MessageBox(hwnd, L"Ничья", L"Конец игры", MB\_OK);

wchar\_t szModule[MAX\_PATH];

GetModuleFileName(NULL, szModule, MAX\_PATH);

// Перезапускаем программу

ShellExecute(NULL, NULL, szModule, NULL, NULL, SW\_SHOWNORMAL);

PostQuitMessage(0);

}

InvalidateRect(hwnd, NULL, TRUE);

UpdateWindow(hwnd);

break;

case WM\_KEYDOWN:

// Обработка клавиш для перемещения изображения

switch (wParam) {

/\*case VK\_LEFT:

car1.pos = 3;

car1.LEFT = TRUE;

if (car1.pos == 3)

{

if(car1.DOWN == FALSE && car1.UP == false)

car1.x -= 10;

if (car1.DOWN == TRUE)

{

car1.x -= 5;

car1.y += 5;

}

if (car1.UP == TRUE)

{

car1.x -= 5;

car1.y -= 5;

}

}

else{

car1.pos = 3;

car1.LEFT = TRUE;

}

break;

case VK\_RIGHT:

car1.pos = 1;

car1.RIGHT = TRUE;

if (car1.pos == 1) {

if (car1.DOWN == FALSE && car1.UP == false)

car1.x += 10;

if (car1.DOWN == TRUE)

{

car1.x += 5;

car1.y += 5;

}

if (car1.UP == TRUE)

{

car1.x += 5;

car1.y -= 5;

}

}

else{

car1.pos = 1;

car1.RIGHT = TRUE;

}

break;

case VK\_UP:

car1.pos = 0;

car1.UP = TRUE;

if (car1.pos == 0) {

if(car1.LEFT == FALSE && car1.RIGHT == FALSE)

car1.y -= 10;

if (car1.LEFT == TRUE)

{

car1.y -= 5;

car1.x -= 5;

}

if (car1.RIGHT == TRUE)

{

car1.y -= 5;

car1.x += 5;

}

}

else{

car1.pos = 0;

car1.UP = TRUE;

}

break;

case VK\_DOWN:

car1.pos = 2;

car1.DOWN = TRUE;

if (car1.pos == 2)

{

if (car1.LEFT == FALSE && car1.RIGHT == FALSE)

car1.y += 10;

if (car1.LEFT == TRUE)

{

car1.y += 5;

car1.x -= 5;

}

if (car1.RIGHT == TRUE)

{

car1.y += 5;

car1.x += 5;

}

}

else{

car1.pos = 2;

car1.DOWN = TRUE;

}

break;

}\*/

/\*case VK\_UP:

if(timer){

if(car1.speed < MAX\_SPEED)

car1.speed += ACCELERATION;

calcSpeed();

}

break;

case VK\_DOWN:

if (timer) {

if (car1.speed > MIN\_SPEED)

car1.speed -= ACCELERATION;

calcSpeed();

}

break;

case VK\_RIGHT:

if (timer) {

rotateright();

car1.x += car1.vx;

car1.y += car1.vy;

}

break;

case VK\_LEFT:

if (timer) {

rotateleft();

car1.x += car1.vx;

car1.y += car1.vy;

}

break;\*/

case 0x48 :

KillTimer(hwnd, ID\_TIMER);

timerbefor = GetTickCount();

MessageBox(NULL, L"Добро пожаловать в игровое приложение гонки!\n Назначения клавиш: \nP - пауза, останавливает игру на время\nH - помощь, напомнить себе основные элементы управления \nESC - закрыть игру\nИгра длиться 20 секунд на поле появляются белые круги, каждый из которых даст очки, собравшему их игроку в зависимости от своего размера. Для упарвления используйте клавиши WASD и стрелки\nУдачи", L"Welcome", MB\_OK );

timerPtime += GetTickCount() - timerbefor;

SetTimer(hwnd, ID\_TIMER, TIMER\_INTERVAL, NULL);

break;

case 0x50 : // P

if(timer)

{

timerbefor = GetTickCount();

KillTimer(hwnd, ID\_TIMER);

timer = FALSE;

}

else

{

SetTimer(hwnd, ID\_TIMER, TIMER\_INTERVAL, NULL);

timer = TRUE;

timerPtime += GetTickCount() - timerbefor;

}

break;

case VK\_ESCAPE:

KillTimer(hwnd, ID\_TIMER);

DeleteObject(hBitmapBackground);

PostQuitMessage(0);

break;

}

// Принудительное перерисование окна

InvalidateRect(hwnd, NULL, TRUE);

UpdateWindow(hwnd);

break;

case WM\_DESTROY:

// Освобождение ресурсов при закрытии окна

if (hdcBuffer) {

DeleteDC(hdcBuffer);

DeleteObject(hbmBuffer);

}

KillTimer(hwnd, ID\_TIMER);

DeleteObject(hBitmapBackground);

DeleteObject(car1.hBitmap);

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, uMsg, wParam, lParam);

}

return 0;

}

// Точка входа

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {

// Регистрация класса окна

WNDCLASS wc = { 0 };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = L"ImageWindowClass";

RegisterClass(&wc);

// Создание окна

HWND hwnd = CreateWindowEx(

0,

L"ImageWindowClass",

L"Racing Club",

WS\_OVERLAPPEDWINDOW & ~WS\_MAXIMIZEBOX & ~WS\_SIZEBOX,

CW\_USEDEFAULT, CW\_USEDEFAULT, 1000, 530,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL) {

MessageBox(NULL, L"Window creation failed!", L"Error", MB\_OK | MB\_ICONERROR);

return -1;

}

// Отображение окна

ShowWindow(hwnd, nCmdShow);

// Цикл обработки сообщений

MSG msg;

while (GetMessage(&msg, NULL, 0, 0)) {

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}