

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	3
ВВЕДЕНИЕ.....	4
1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ	5
2. ОПИСАНИЕ ПРОГРАММИРУЕМОЙ СИСТЕМЫ.....	10
3. РЕЗУЛЬТАТЫ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА	13
ЗАКЛЮЧЕНИЕ	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	21
ПРИЛОЖЕНИЯ.....	22
Приложение А – Исходный код программы	23

ВВЕДЕНИЕ

Современный рынок разработки видеоигр требует от разработчиков не только мастерства в своей области, но и понимания смежных технологий. Создание современной игры-логики в соответствии с поставленной задачей и темой позволяет приобрести практический опыт разработки современных видеоигр и взглянуть на классические задачи разработки игр под новым углом.

Цель работы: Создание игры-логики для ПК на языке C++.

Задачи работы:

1. Описание концепции игры-логики;
2. Изучение существующих игр-логики;
3. Составление требований к игре;
4. Разработка дизайн-документа игры;
5. Реализация игры в соответствии с требованиями;
6. Проведение тестирования игры;
7. Составление отчета по работе;
8. Презентация отчета и защита его.

Объектом данного исследования является Разработка игр-логики.

Предметов исследования в данной работе является Разработка игры-логики для ПК на языке C++.

В качестве основных методов исследования применены анализ, синтез, сравнение и моделирование. Практическая реализация поставленной задачи соответствует основным парадигмам ООП, применяемым для разработки игр.

Информационной базой исследования являются открытые источники, а также материалы курса «Технологии индустриального программирования», доступные через систему дистанционного обучения РТУ МИРЭА

В данном отчете будет представлен процесс разработки программного продукта, технологическое проектирование и описание системы, а также непосредственно результаты разработки.

1. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Термином компьютерная игра обозначается компьютерная программа, которая служит для организации игрового процесса(геймплея), связи с партнёрами по игре, или сама выступающая в качестве партнёра.

Компьютерные игры классифицируют по нескольким основным признакам:

- жанр;
- количество игроков;
- визуальное представление;
- платформа.

Жанр игры определяется целью и основной механикой игры.

По количеству игроков игры разделяются на два вида:

- однопользовательские;
- многопользовательские.

По визуальному представлению компьютерные игры можно разделить на следующие виды:

- текстовые – минимальное графическое представление, общение с игроком проходит с помощью текста;
- 2D – все элементы отрисованы в виде двумерной графики (спрайтов);
- 3D – все элементы отрисованы в виде трехмерной графики (3D-модели).

По типу платформы:

- персональные компьютеры;
- игровые приставки/консоли;
- мобильные телефоны.

Общий алгоритм разработки компьютерной игры включает в себя 3 больших этапа:

1. Проектирование.
2. Разработка.
3. Издание и поддержка.

На этапе проектирования игры устанавливаются её основная цель и инструменты для разработки. Цель игры формируется через определение её идеи, жанра, сеттинга и механики. Идея — это ключевой замысел, который мотивирует игрока участвовать в игре, и она напрямую связана с жанром. Как только идея становится ясной, жанр игры определяется практически автоматически. После выбора жанра и идеи следующим шагом становится выбор сеттинга — среды, в которой разворачиваются события игры. Сеттинг задаёт место, время и обстоятельства, в которых происходит действие. Механика — это набор правил и взаимодействий, которые определяют, как игрок взаимодействует с игрой. В логических играх механика может включать решение головоломок, перемещение объектов или стратегическое планирование.

К инструментам разработки относятся, в первую очередь, программный код и игровой движок. Их правильный выбор влияет на скорость создания игры и её стабильную работу в будущем. Выбор программного кода зависит от платформы, для которой разрабатывается игра. Для игры «Виселица» можно использовать как простые инструменты (например, Python и Pygame), так и более сложные (например, Unity и C#), в зависимости от целей и требований проекта.

Игровой движок обеспечивает базовые функции, такие как моделирование физики объектов, отрисовку графики и другие технические аспекты. При выборе движка в первую очередь учитывают его доступность и совместимость с уже выбранным языком программирования. Для игры «Виселица» важно последовательно выполнить все шаги, чтобы создать качественный и увлекательный продукт, который понравится игрокам.

После определения цели игры и выбора инструментов для разработки начинается второй этап — непосредственно создание игры. Этот этап, является самым масштабным и продолжительным. Он включает множество важных шагов, без которых невозможно создать готовый и функциональный продукт.

Игровая механика в игре «Виселица» [1] строится вокруг её основной цели — угадать загаданное слово, избежав «казни» персонажа. Механика определяет правила взаимодействия игрока с объектами: буквами, которые он выбирает, и процессом отображения ошибок (например, постепенное появление элементов виселицы). После определения механики и правил начинается этап непосредственной разработки игры, включающий программирование, тестирование и доработку функционала.

При разработке игры «Виселица» сначала создаётся упрощённая схема, на которой обозначаются основные элементы: загаданное слово, доступные буквы для выбора и этапы отображения виселицы. После этого формируется первая версия игрового процесса, которая затем тестируется и дорабатывается для улучшения функциональности и удобства.

Вскоре после создания базовой версии игры «Виселица» собирается первый прототип, называемый альфа-версией. Альфа-тестирование — это важный этап разработки игры, на котором проверяется работоспособность основной механики и функциональности. Он нужен для тестирования основной механики игры и проверки её соответствия задуманным требованиям. Убедиться, что основные элементы игры работают корректно. Например:

- Загаданное слово выбирается случайным образом из списка.
- Игрок может вводить буквы, и система правильно проверяет их наличие в слове.
- Виселица отображается поэтапно в зависимости от количества ошибок.
- Игра завершается при угадывании слова или при превышении допустимого числа ошибок.

Обнаружение и устранение багов, которые мешают нормальному функционированию игры. Например:

- Некорректная обработка ввода (например, ввод цифр или символов вместо букв).
- Ошибки в логике игры (например, неправильный подсчёт ошибок или некорректное отображение букв).

Если альфа-версия успешно проходит тестирование, начинается этап доработки механики и объектов игры. На этом этапе улучшаются правила, добавляются новые функции (например, возможность выбора тематики слов или уровней сложности), а также исправляются ошибки, обнаруженные во время тестирования.

После этого создаётся второй прототип — бета-версия. Она представляет собой почти готовую игру, в которой могут отсутствовать лишь мелкие детали, не влияющие на основной игровой процесс. Бета-версия используется для финального тестирования и поиска возможных недочётов перед выпуском игры.

Во время бета-тестирования выполняется:

- Проверка, что игра не зависает при длительном использовании.
- Проверка работы на разных операционных системах.
- Удобство интерфейса (отображение букв, виселицы).
- Чёткость правил и обратной связи (сообщение о победе или поражении).

Этап бета-тестирования будет считаться завершённым, если:

- Все найденные ошибки исправлены
- Основные функции работают без ошибок.

Игра “Wordle” [2]:

“Wordle” — это популярная современная текстовая игра, в которой игрокам нужно угадать загаданное слово за ограниченное количество попыток. После каждой попытки игрок получает подсказки: буквы, которые есть в слове, подсвечиваются разными цветами в зависимости от того, правильно ли они угаданы и находятся ли на своих местах. Хотя механика “Wordle” отличается от «Виселицы», оба игры объединяет общий элемент — угадывание слов на основе подсказок.

Игра «Крокодил» [3]:

Игра «Крокодил» — это популярная развлекательная игра, в которой один из игроков загадывает или получает слово, а затем должен объяснить его другим участникам, используя только жесты, движения и мимику, без произнесения слов.

«Кроссворды и сканворды» [5]:

«Кроссворды и сканворды» — это популярные словесные головоломки, где игроку предоставляются определения или подсказки, а цель — угадать слова и заполнить ими специальную сетку. В кроссвордах слова пересекаются между собой, создавая взаимосвязанную структуру, а в сканвордах подсказки обычно интегрированы прямо в сетку, что делает процесс решения более интерактивным.

Игра «Поле Чудес» [4]:

Игра «Поле чудес» — это популярная словесная игра, в которой участникам загадывается слово или фраза, скрытые за закрытыми буквами. Игроки по очереди называют буквы, и, если названная буква есть в загаданном слове, она открывается на своих местах. Цель игры — полностью угадать слово или фразу, полагаясь на логику, интуицию и знания.

2. ОПИСАНИЕ ПРОГРАММИРУЕМОЙ СИСТЕМЫ

В таблице 2.1 представлены требования к программируемой системе.

Таблица 2.1 – Требования к продукту

№	Требование	Значение
1	Язык программирования	C++
2	Корректность работы	Приложение запускается и поддерживает стабильный цикл работы от момента старта до завершения
3	Применение принципов объектно-ориентированного программирования	При написании приложения, как минимум, были использованы классы в C++, объектный подход к проектированию системы, а также инкапсуляция
4	Интерфейс пользователя	Создан интерфейс пользователя, поддерживающий корректный пользовательский опыт и содержащий все необходимые пояснения к работе и эксплуатации
5	Инструкция по эксплуатации	Написана инструкция по эксплуатации, содержащая, в том числе, основные рекомендации по использованию и пояснения к возможным ошибкам в программе
6	Генерация слов	Реализована система случайного выбора слов
7	Обратная связь пользователю	После каждой попытки должен изменяться рисунок виселицы: если неверная буква – дорисовывается часть виселицы, если правильная буква – отображается в загаданном слове
8	Логика игры	Реализована логика проверки букв и подсчета ошибок
9	Реализация анимационных эффектов	Текстовые анимации (например, постепенное отображение виселицы и частей тела)
10	Хранение истории использованных букв	В процессе игры должна сохраняться история всех введенных букв и отображать его в течении всей игры.

Описание программируемой системы «Игра – Виселица» представляет собой классическую текстовую игру, в которой пользователь угадывает слово по буквам, избегая ошибок, ведущих к проигрышу. Алгоритм работы программы обеспечивает интуитивно понятное взаимодействие с пользователем, включая ввод букв, обработку ответов, отображение текущего состояния игры и завершение сессии. Программное обеспечение реализует классическую игру, в

которой игроку необходимо угадать случайное слово, загаданное системой, посредством попыток с предоставлением обратной связи на каждом шаге. Алгоритм работы программы организован таким образом, что все этапы взаимодействия пользователя с программой, от ввода данных до вывода результатов, выполняются в едином потоке, обеспечивая логичное и связанное течение игрового процесса.

Программа начинает свою работу с инициализации, которая заключается в загрузке необходимых библиотек, установке начальных параметров и генерации случайного слова, выбранного из заранее подготовленного файла. При запуске система считывает конфигурационные данные, устанавливает максимальное количество попыток, а также подготавливает механизмы для обработки пользовательского ввода и вывода информации на экран. В процессе инициализации происходит проверка корректности входных данных, а также настройка интерфейса, который будет использоваться для отображения сообщений и подсказок игроку. Таким образом, на этом этапе создается основа для дальнейшего функционирования приложения, и пользователь получает первичное подтверждение готовности программы к выполнению игровой сессии.

После завершения этапа инициализации начинается основной игровой цикл, в котором программа ожидает ввода буквы от игрока. При получении ввода система анализирует введённую букву, сверяя его с загаданным словом. При неправильном вводе данных пользователем система активирует специальный модуль обработки ошибок, который выполняет последовательную проверку и коррекцию входящих данных перед их передачей в основной игровой цикл. Данный механизм обеспечивает стабильность работы приложения и предотвращает возможные сбои, вызванные неожиданными или ошибочными действиями пользователя. При правильном вводе данные поступают в модуль обработки, где осуществляется сравнение буквы, предложенного пользователем, с загаданным словом системы.

В зависимости от результата сравнения программа генерирует соответствующую обратную связь. Если введенной буквы нет в загаданном слове, пользователю выводится новая часть человечка на виселице. В случае совпадения введенной буквы с загаданным словом система выводит верную букву на табло. Этот момент становится завершающим этапом игровой сессии, после чего пользователю предлагается возможность начать новую игру или завершить работу приложения.

На финальном этапе работы программы осуществляется корректное завершение сессии. Финальный этап включает в себя закрытие активных потоков ввода-вывода, очистку временных данных и подготовку программного модуля к следующему запуску. Такие меры гарантируют отсутствие утечек памяти и стабильное завершение работы, что особенно важно для долгосрочной эксплуатации ПО в разных средах.

В итоге, описанный алгоритм работы программируемой системы «Игра – Виселица» представляет собой минималистичную реализацию, позволяющей пользователю полностью погрузиться в игровой процесс. Программа случайно выбирает слово, скрывая его буквы, а игрок пытается угадать их, вводя по одной букве за ход. При правильном выборе буква открывается, при ошибке — теряется попытка. Игра продолжается до полного отгадывания слова или исчерпания попыток, после чего выводится результат.

Этот алгоритм не только подтверждает, что даже базовые игровые механики можно эффективно реализовать технически, но и является образцом продуманного структурирования процессов. Он органично сочетает генерацию случайных значений, работу с пользовательским вводом, гибкую настройку игровых параметров и безопасное завершение программы. В итоге создается целостное, логически выстроенное и расширяемое решение, которое подходит как новичкам, так и опытным пользователям, гарантируя при этом стабильную работу всех заложенных функций.

3. РЕЗУЛЬТАТЫ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

Реализация требований к системе

1. Язык программирования

В рамках курсовой работы была разработана игра-логика, полностью написанная на языке программирования C++. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

2. Корректность работы

Функционирование приложения стабильно: оно надежно запускается и работает без сбоев в течение всей пользовательской сессии. Подтверждение выполнения этого требования можно найти в разделе (Рисунки 3.1–3.5), где приведены скриншоты рабочего процесса программы.

3. Применение принципов объектно-ориентированного программирования

В ходе разработки игры "Виселица" были применены ключевые принципы объектно-ориентированного программирования (ООП), включая инкапсуляцию, что позволило создать модульную, расширяемую и легко поддерживаемую архитектуру программы. Реализация этих принципов обеспечила четкое разделение функциональности между компонентами игры, такими как обработка пользовательского ввода, управление игровой логикой и отображение состояния игры. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

4. Интерфейс Пользователя

Интерфейс пользователя был разработан с учетом удобства взаимодействия и интуитивной понятности, обеспечивая легкий доступ ко всем функциональным возможностям игры и комфортное управление игровым процессом. Увидеть реализацию данного требования можно обратившись к скриншотам работы программы (Рисунки 3.1–3.5)

5. Инструкция по эксплуатации

Для комфортного освоения игры "Виселица" была разработана подробная инструкция, включающая все необходимые аспекты взаимодействия с программой. Она содержит информацию о начале работы с игрой, правила и цели игры. Инструкция разработана с соблюдением принципов доступности и учитывает потребности пользователей с разным уровнем компьютерной грамотности. (Рисунок 3.5).

6. Генерация случайного слова

В игре "Виселица" для выбора случайного слова из списка используется функция `rand()` из стандартной библиотеки `cstdlib`. Чтобы обеспечить различную последовательность слов при каждом запуске программы, применяется функция `srand()` с начальным значением (`seed`), основанным на текущем системном времени (`time(0)`). Слова загружаются из файла `word.txt` (или используется стандартный набор, если файл недоступен). Количество слов зависит от содержимого файла `word.txt`, что делает игру гибкой и расширяемой. Таким образом, механизм выбора слов обеспечивает реиграбельность и вариативность игры.

7. Реализация визуализации виселицы

В игре "Виселица" используется пошаговая анимация виселицы, которая визуализирует прогресс игрока. Реализация выполнена через: 7 стадий рисунка (от пустой виселицы до полного изображения). Каждая неверная буква добавляет новый элемент к рисунку. Такая визуализация позволяет пользователю не теряться и всегда видеть сколько попыток у него осталось.

8. Обработка пользовательского ввода

В игре "Виселица" реализована комплексная система обработки пользовательского ввода, обеспечивающая надежное и удобное взаимодействие с программой. Основной механизм ввода реализован через функцию `inputLetter()`, которая выполняет многоуровневую проверку вводимых данных: система автоматически проверяет, что введенный символ является буквой русского алфавита, преобразует его к нижнему регистру для единообразия обработки, а также отслеживает и предотвращает попытки повторного ввода уже

использованных букв. Для обеспечения понятного взаимодействия реализована система обратной связи - при обнаружении некорректного ввода (цифр, символов или латинских букв) программа выводит соответствующее поясняющее сообщение. Увидеть реализацию данного требования можно обратившись к исходному коду программа (Приложение А).

9. Система хранения игровых данных

Система хранения игровых данных реализована через гибкий механизм работы с внешними файлами, где основной словарь слов загружается из текстового файла words.txt с простой и удобной структурой (каждое слово записано на отдельной строке), при этом в случае отсутствия или недоступности файла автоматически активируется резервный встроенный набор слов, что обеспечивает бесперебойную работу приложения и позволяет легко расширять словарный запас игры путем простого редактирования текстового файла без необходимости внесения изменений в исходный код программы, а сама реализация данной системы, включая обработку исключительных ситуаций и загрузку данных, представлена в функции loadWordsFromFile() (см. исходный код в Приложении А), что гарантирует простоту модификации словаря и стабильную работу приложения при любых условиях.

10. Система обратной связи с игроком

В игре реализован интерактивный механизм взаимодействия с пользователем, который после каждого хода предоставляет детальную текстовую обратную связь: при правильной букве программа визуально отображает её позиции в слове через обновление игрового табло, а при ошибке - добавляет новый элемент к рисунку виселицы и выводит предупреждение о неверном выборе. Система включает интеллектуальную обработку ввода: автоматически фильтрует некорректные символы (цифры, латиницу, спецзнаки), проверяет повторный ввод уже использованных букв (с выводом соответствующего уведомления), а также преобразует регистр букв для единообразия обработки. Дополнительные подсказки в интерфейсе (список использованных букв, текущее состояние слова) помогают игроку анализировать

ситуацию и принимать осознанные решения. Реализованная система предотвращает ошибки взаимодействия, делает игровой процесс прозрачным и увлекательным, что подтверждается интуитивно понятными сообщениями в основном игровом цикле (функции inputLetter() и ShowTable()) и визуализацией прогресса (функция draw()), как показано в исходном коде (Приложение А).

Функциональное тестирование программного продукта

Работа с программой начинается с вывода меню игры, которую запустил пользователь. (Рисунок 3.1).

```
=== ДОБРО ПОЖАЛОВАТЬ В ИГРУ 'ВИСЕЛИЦА' ===  
1. Начать игру  
2. Показать инструкцию  
3. Выйти из игры  
=====
```

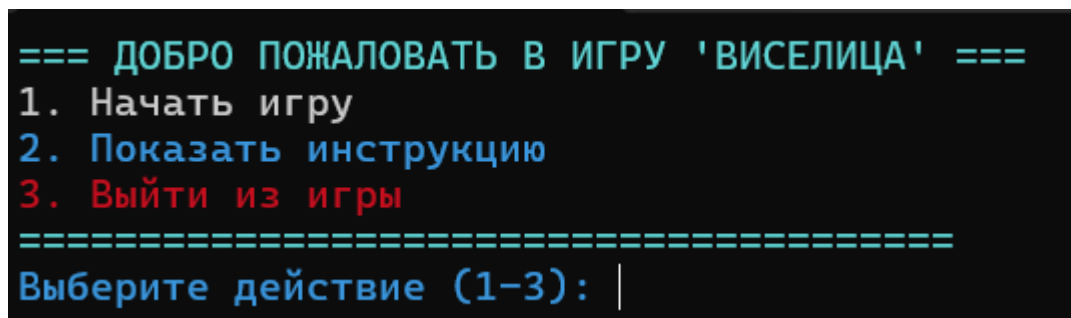


Рисунок 3.1 – меню

В случае введения “1” пользователь программы переходит непосредственно к игровому процессу.

```
=== ИГРАЕМ! ===  
Осталось попыток: 6
```



Рисунок 3.2 – переход к игровому процессу

При вводе буквы программа анализирует её соответствие загаданному слову. Если буква присутствует в слове, она отображается на соответствующих позициях игрового табло. В случае отсутствия буквы в слове программа добавляет элемент к рисунку виселицы, уменьшает количество оставшихся попыток. Игрок получает возможность вводить новые буквы, основываясь на визуализации текущего состояния слова (отображённые угаданные буквы) и списке уже использованных букв. При каждой ошибке обновлённое изображение виселицы демонстрирует прогресс и приближение к поражению, что помогает игроку оценить риск следующих попыток.

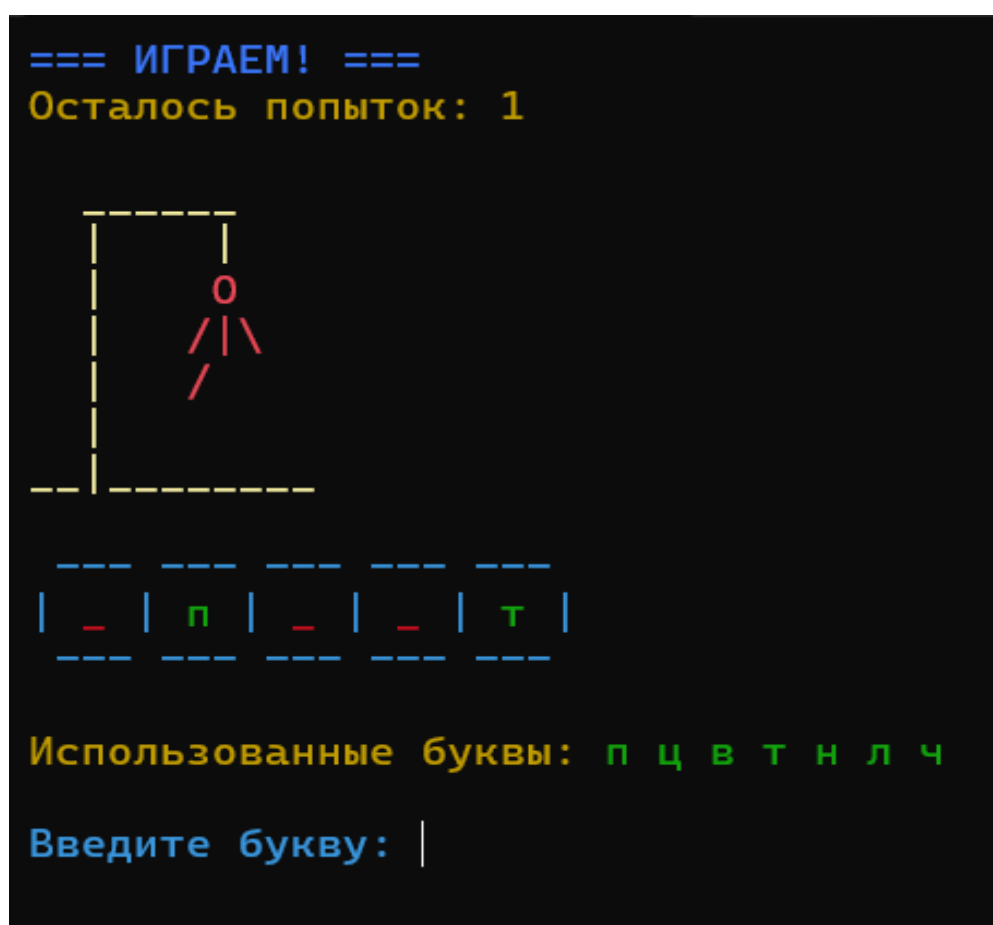


Рисунок 3.3 – игровой процесс

Игровой раунд завершается в одном из двух случаев: когда игрок полностью угадывает загаданное слово (все буквы открыты на табло) или когда исчерпаны все попытки (нарисована полная виселица).



Рисунок 3.4 – конец игры

В случае введения “2” пользователь программы переходит непосредственно к инструкции и правилам игры.

```

=== ИНСТРУКЦИЯ ПО ИГРЕ 'ВИСЕЛИЦА' ===
1. Цель игры: угадать загаданное слово, вводя буквы по одной.
2. У вас есть 6 попыток, после чего игра завершается.
3. Если буква есть в слове: она откроется на табло (зеленым цветом).
4. Если буквы нет в слове: к виселице добавится часть тела (красным цветом).
5. Использованные буквы: отображаются желтым цветом.
6. Для выхода: используйте пункт меню 3 или Ctrl+C
=====
Нажмите Enter для возврата в меню...|

```

Рисунок 3.5 – Инструкция по эксплуатации

Инструкция по эксплуатации

После запуска программы выводится меню игры, где ему выводится приветствие с названием игры, и выбором действий пользователя. Для начала игры ему необходимо ввести “1”. Для просмотра правил игры, ему необходимо ввести “2”. Для выхода из игры ему необходимо ввести “3”.

При введении “1” запускается непосредственно игровой процесс. При вводе буквы пользователю доступны следующие подсказки:

1. «Буква введенная пользователем отображается на табло» - означает, что введенная буква присутствует в загаданном слове, и ее позиция отображается на табло.
2. «Появляется новая часть на виселице» - означает, что введенная буква пользователем отсутствует в загаданном слове.
3. «Вы уже вводили букву!» - означает, что введенная буква пользователем уже была использована.
4. «Поздравляем! Вы угадали слово!» - означает, что пользователь смог угадать слово, которое загадала программа.
5. «Вы проиграли! Загаданное слово: [СЛОВО]» - означает, что у пользователя не получилось отгадать слово, которое загадала программа.

Конечной целью игрового процесса является угадывание загаданного слова программой, с возможным улучшением своего лучшего результата в дальнейшем. Для прохождения игры пользователю необходимо ориентироваться на подсказки, сделанные в виде визуального отображения после его попыток, и избегать повторения вводимых им же букв. При повторении предыдущей буквы пользователем, программа выдаст такую же подсказку. Игра заканчивается, когда пользователь смог угадать слово, которое загадала программа или если у пользователя не получилось угадать загаданное слово.

После окончания непосредственно игрового процесса пользователь может либо начать новую игру, либо выйти в меню игры откуда он может выйти из игры.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была описана программируемая система, рассмотрены существующие решения-аналоги по теме, сформированы требования к системе, спроектированы диаграммы состояний, классов и последовательности самой системы. Сам программный продукт был разработан в соответствии с требованиями, протестирован, а также была написана инструкция по его эксплуатации.

Проект отражает понимание объектно-ориентированного программирования и практические навыки работы с C++, включая консольный ввод-вывод, обработку ошибок и проектирование программной структуры. Разработанное приложение отличается стабильностью, продуманным пользовательским интерфейсом и качественной визуализацией. [6]

В процессе разработки были успешно решены сложные задачи, связанные с проектированием программы и отработкой ошибок, что позволило реализовать полноценный игровой цикл, удобное взаимодействие с пользователем, соблюдение принципов ООП, работу с файлами, и устойчивость к ошибкам ввода. Структурированная организация кода, логичное разделение функций и ясность реализации обеспечили простоту поддержки.

Проект показал значительный потенциал для дальнейшего развития и улучшения, включая добавление новых функций и уровней, что может стать основой для будущих исследований и разработок в области создания игр. Работа над курсовой работой также способствовала повышению профессиональных компетенций в области программирования и дизайна игр.

Полученный опыт и знания будут служить твердой основой для дальнейшего обучения и профессионального роста в области разработки программного обеспечения и игровой индустрии.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Игра «Виселица» / [Электронный ресурс] // Википедия: [сайт]. — URL: [https://ru.wikipedia.org/wiki/Виселица_\(игра\)](https://ru.wikipedia.org/wiki/Виселица_(игра))
2. Игра “Wordle” / [Электронный ресурс] // Википедия: [сайт]. — URL: <https://ru.wikipedia.org/wiki/Wordle>
3. Игра «Крокодил» / [Электронный ресурс] // Википедия: [сайт]. — URL: [https://ru.wikipedia.org/wiki/Крокодил_\(игра\)](https://ru.wikipedia.org/wiki/Крокодил_(игра))
4. Игра «Поле чудес» / [Электронный ресурс] // Википедия: [сайт]. — URL: https://ru.wikipedia.org/wiki/Поле_чудес
5. «Кроссворды и сканворды» / [Электронный ресурс] // Википедия: [сайт]. — URL: <https://ru.wikipedia.org/wiki/Кроссворд>
6. «Объективно-ориентированное программирование» / [Электронный ресурс] // Википедия: [сайт]. — URL: https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование

ПРИЛОЖЕНИЯ

Приложение А – Исходный код программы

Приложение А – Исходный код программы

Листинг А.1 – Исходный код программы

```
#define NOMINMAX
#include <iostream>
#include <string>
#include <vector>
#include <cstdlib>
#include <fstream>
#include <algorithm>
#include <cctype>
#include <limits>
#include <locale>
#include <signal.h>

// Кросс-платформа
#ifdef _WIN32
#include <windows.h>
#define ENABLE_VIRTUAL_TERMINAL_PROCESSING 0x0004
#else
#include <locale>
#endif

using namespace std;

// Цветовые коды ANSI
const string RESET = "\033[0m";
const string RED = "\033[31m";
const string GREEN = "\033[32m";
const string YELLOW = "\033[33m";
const string BLUE = "\033[34m";
const string WHITE = "\033[37m";
const string CYAN = "\033[36m";
const string BOLD = "\033[1m";

// Цветной вывод
void enableColors() {
#ifdef _WIN32
    HANDLE hOut = GetStdHandle(STD_OUTPUT_HANDLE);
    DWORD dwMode = 0;
    GetConsoleMode(hOut, &dwMode);
    dwMode |= ENABLE_VIRTUAL_TERMINAL_PROCESSING;
    SetConsoleMode(hOut, dwMode);
#endif
}

// Очистка экрана
void clearScreen() {
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
}

// Функция для настройки локали
void setRussianLocale() {
#ifdef _WIN32
    SetConsoleCP(1251);
    SetConsoleOutputCP(1251);
    setlocale(LC_ALL, "Russian");
#endif
}
```

```
#else
    setlocale(LC_ALL, "ru_RU.UTF-8");
#endif
}

// Проверки русской буквы
bool isRussianLetter(char c) {
    c = tolower(c);
    return (c >= 'а' && c <= 'я') || c == 'ё';
}

// Загрузка слов из файла
vector<string> loadWordsFromFile(const string& filename) {
    vector<string> words;
    ifstream file(filename);

    if (file.is_open()) {
        string word;
        while (getline(file, word)) {
            if (!word.empty()) {
                words.push_back(word);
            }
        }
        file.close();
    }
    else {
        words = { "программирование", "алгоритм", "компьютер",
                  "виселица", "разработка", "интерфейс" };
    }

    return words;
}

// Рисование линии для табло
void drawLine(int n) {
    cout << CYAN;
    for (int i = 0; i < n; ++i)
        cout << " ---";
    cout << RESET << endl;
}

// Отображение табло
void ShowTable(const string& table) {
    drawLine(table.size());
    cout << CYAN << "|" << RESET;
    for (char c : table) {
        if (c == '_') {
            cout << " " << RED << c << RESET << " " << CYAN << "|" << RESET;
        }
        else {
            cout << " " << GREEN << c << RESET << " " << CYAN << "|" << RESET;
        }
    }
    cout << endl;
    drawLine(table.size());
}

// Отображение использованных букв
void ShowUsedLetters(const vector<char>& usedLetters) {
    if (!usedLetters.empty()) {
        cout << YELLOW << "Использованные буквы: " << RESET;
```

```
        for (char c : usedLetters) {
            cout << GREEN << c << RESET << " ";
        }
        cout << endl;
    }
}

// Ввод буквы с проверкой
char input(vector<char>& usedLetters) {
    char c;
    bool validInput = false;

    while (!validInput) {
        cout << CYAN << "Введите букву: " << RESET;
        cin >> c;

        if (cin.eof()) {
            cin.clear();
            cout << RED << "Пожалуйста, введите букву русского алфавита.\n" << RESET;
            continue;
        }

        if (!isRussianLetter(c)) {
            cout << RED << "Ошибка! Пожалуйста, введите букву русского алфавита.\n"
<< RESET;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            continue;
        }

        c = tolower(c);

        if (find(usedLetters.begin(), usedLetters.end(), c) != usedLetters.end()) {
            cout << BLUE << "Буква уже была использована! Попробуйте другую.\n" <<
RESET;
            continue;
        }

        validInput = true;
    }

    usedLetters.push_back(c);
    return c;
}

// Открытие букв на табло
bool openLetters(const string& word, string& table, char c) {
    bool found = false;
    for (int i = 0; i < word.size(); ++i) {
        if (word[i] == c) {
            table[i] = c;
            found = true;
        }
    }
    return found;
}

//Рисунок Виселицы
void draw(int err) {
    cout << BOLD << YELLOW;
    switch (err) {
```

```

case 0:
    cout << "      \n";
    cout << " |      | \n";
    cout << " | \n";
    cout << " | \n";
    cout << " | \n";
    cout << " | \n";
    cout << " __|      \n";
    break;
case 1:
    cout << "      \n";
    cout << " |      | \n";
    cout << " |      " << RED << "O" << YELLOW << " \n";
    cout << " | \n";
    cout << " | \n";
    cout << " | \n";
    cout << " __|      \n";
    break;
case 2:
    cout << "      \n";
    cout << " |      | \n";
    cout << " |      " << RED << "O" << YELLOW << " \n";
    cout << " |      " << RED << "|" << YELLOW << " \n";
    cout << " | \n";
    cout << " | \n";
    cout << " __|      \n";
    break;
case 3:
    cout << "      \n";
    cout << " |      | \n";
    cout << " |      " << RED << "O" << YELLOW << " \n";
    cout << " |      " << RED << "/|" << YELLOW << " \n";
    cout << " | \n";
    cout << " | \n";
    cout << " __|      \n";
    break;
case 4:
    cout << "      \n";
    cout << " |      | \n";
    cout << " |      " << RED << "O" << YELLOW << " \n";
    cout << " |      " << RED << "/|\" << YELLOW << " \n";
    cout << " | \n";
    cout << " | \n";
    cout << " __|      \n";
    break;
case 5:
    cout << "      \n";
    cout << " |      | \n";
    cout << " |      " << RED << "O" << YELLOW << " \n";
    cout << " |      " << RED << "/|\" << YELLOW << " \n";
    cout << " |      " << RED << "/" << YELLOW << " \n";
    cout << " | \n";
    cout << " __|      \n";
    break;
case 6:
    cout << "      \n";
    cout << " |      | \n";
    cout << " |      " << BLUE << "O" << YELLOW << " \n";
    cout << " |      " << BLUE << "/|\" << YELLOW << " \n";
    cout << " |      " << BLUE << "/ \" << YELLOW << " \n";
    cout << " | \n";

```



```

        cout << "__|_____\n";
        break;
    }
    cout << RESET;
}

// Инструкция
void ShowInstructions() {
    clearScreen();
    cout << BOLD << CYAN << "=== ИНСТРУКЦИЯ ПО ИГРЕ 'ВИСЕЛИЦА' ===\n" << RESET;
    cout << GREEN << "1. Цель игры: " << RESET << "угадать загаданное слово, вводя
буквы по одной.\n";
    cout << GREEN << "2. У вас есть 6 попыток, " << RED << "после чего игра
завершается.\n" << RESET;
    cout << GREEN << "3. Если буква есть в слове: " << RESET << "она откроется на
табло " << GREEN << "(зеленым цветом).\n" << RESET;
    cout << GREEN << "4. Если буквы нет в слове: " << RESET << "к виселице добавится
часть тела " << RED << "(красным цветом).\n" << RESET;
    cout << GREEN << "5. Используемые буквы: " << RESET << "отображаются " <<
YELLOW << "желтым цветом.\n" << RESET;
    cout << GREEN << "6. Для выхода: " << RESET << "используйте пункт меню 3 или "
<< RED << "Ctrl+C\n" << RESET;
    cout << BOLD << CYAN << "=====\n\n" << RESET;
    cout << "Нажмите Enter для возврата в меню...";
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cin.get();
}

// Главное меню
void ShowMainMenu() {
    clearScreen();
    cout << BOLD << CYAN << "=== ДОБРО ПОЖАЛОВАТЬ В ИГРУ 'ВИСЕЛИЦА' ===\n" << RESET;
    cout << WHITE << "1. " << WHITE << "Начать игру\n";
    cout << CYAN << "2. " << CYAN << "Показать инструкцию\n";
    cout << RED << "3. " << RED << "Выйти из игры\n";
    cout << BOLD << CYAN << "=====\n" << RESET;
    cout << CYAN << "Выберите действие (1-3): " << RESET;
}

// Обработчик Ctrl+C
void handleCtrlC(int signal) {
    cout << "\n" << RED << "Игра завершена по запросу пользователя (Ctrl+C).\n" <<
RESET;
    exit(0);
}

int main() {
    enableColors();
    setRussianLocale();
    signal(SIGINT, handleCtrlC);

    srand(time(0));
    vector<string> words = loadWordsFromFile("words.txt");
    if (words.empty()) {
        cout << RED << "Не удалось загрузить слова из файла." << RESET << endl;
        return 1;
    }

    bool exitProgram = false;
    while (!exitProgram) {
        ShowMainMenu();
    }
}

```

```

int menuChoice;
while (!(cin >> menuChoice) || (menuChoice < 1 || menuChoice > 3)) {
    if (cin.eof()) {
        cin.clear();
        ShowMainMenu();
        continue;
    }
    cin.clear();
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << RED << "Ошибка! Введите число от 1 до 3: " << RESET;
}

switch (menuChoice) {
case 1: {
    bool playAgain = true;
    while (playAgain) {
        string word = words[rand() % words.size()];
        string table(word.size(), '_');
        vector<char> usedLetters;
        int err = 0;

        while (table != word && err < 6) {
            clearScreen();
            cout << BOLD << BLUE << "=== ИГРАЕМ! ===\n" << RESET;
            cout << YELLOW << "Осталось попыток: " << (6 - err) << "\n\n"
<< RESET;

            draw(err);
            cout << endl;
            ShowTable(table);
            cout << endl;
            ShowUsedLetters(usedLetters);
            cout << endl;
            char c = input(usedLetters);
            if (!openLetters(word, table, c))
                ++err;
        }

        clearScreen();
        draw(err);
        cout << endl;
        ShowTable(table);
        cout << endl;

        if (table == word) {
            cout << GREEN << "Поздравляем! Вы угадали слово! " << BOLD <<
word << RESET << endl;
        }
        else {
            cout << RED << "Вы проиграли! Загаданное слово: " << BOLD <<
word << RESET << endl;
        }

        cout << YELLOW << "\nПопробуете еще раз? (1 - Да, 0 - Нет): " <<
RESET;

        int choice;
        while (!(cin >> choice) || (choice != 0 && choice != 1)) {
            if (cin.eof()) {
                cin.clear();
                cout << YELLOW << "Пожалуйста, введите 1 или 0: " << RESET;
                continue;
            }
        }
    }
}

```

Окончание Листинга A.1

```
        }
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << YELLOW << "Пожалуйста, введите 1 или 0: " << RESET;
    }
    playAgain = (choice == 1);
}
break;
}
case 2:
    ShowInstructions();
    break;
case 3:
    exitProgram = true;
    break;
}
}

cout << BOLD << GREEN << "\nСпасибо за игру! До свидания!\n" << RESET;
return 0;
}
```