Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту на тему

**ИГРОВОЕ ПРОГРАММНОЕ СРЕДСТВО «ШАХМАТЫ»**

**ПО ЛОКАЛЬНОЙ СЕТИ**

БГУИР КП I–40 01 01 421 ПЗ

Выполнил

студент: гр. 851004 Пашкевич А.Л.

Проверил: Красковский П. Н.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и

радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Лапицкая Н.В. 2020 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Пашкевичу Антону Леонтьевичу* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема работы  *Игровое программное средство «Шахматы» по локальной сети* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи законченной работы *10.06.2020 г.*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные к работе *Платформа .NET Framework 4.7.2. Разработать клиент-серверное приложение для игры в шахматы с учётом специфики локальной сети. Для игрового процесса реализовать шахматные правила. Разработать протокол взаимодействия сервера с клиентом.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*1 Анализ предметной области \_*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2 Анализ требований к программному средству\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*3 Проектирование и разработка программного средства*

*4 Тестирование программного средства \_*

*5 Руководство по использованию программного средства \_\_\_\_\_*

*Заключение \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Список использованных источников\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_*Приложения* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*Схема алгоритма в формате А1*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Консультант по курсовой работе  *Красковский П.Н.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

7.Дата выдачи задания *17.02.2020г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объема работы):

*Раздел 1. Введение к 01.03.2020г. – 10 % готовности работы;\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 2 к 01.04.2020г. – 30% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 3 к 02.05.2020г. – 60% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Раздел 4 к 20.05.2020г. – 80% готовности работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Заключение. Приложения к 30.05.2020г. – 90% готовности работы;*

*оформление пояснительной записки и графического материала к 05.06.2020г. – 100% готовности работы.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Защита курсового проекта с 05.06.2020г. по 10.06.2020г.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

РУКОВОДИТЕЛЬ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Красковский П.Н.*

*(подпись)*

Задание принял к исполнению *\_\_\_\_\_\_\_\_\_\_\_\_\_Пашкевич А. Л. 17.02.2020г.*

*(дата и подпись студента)*

Содержание

[Введение 5](#_Toc28965665)

[1 Анализ прототипов и литературных источников 6](#_Toc28965666)

[1.1 Элементы теории логических схем 6](#_Toc28965667)

[1.2 Представление векторного изображения в формате SVG 8](#_Toc28965668)

[1.3 Анализ существующих аналогов 9](#_Toc28965669)

[2 Постановка задачи 12](#_Toc28965670)

[3 Разработка программного средства 13](#_Toc28965671)

[3.1 Структура программного средства 13](#_Toc28965672)

[3.2 Структуры данных программного средства 13](#_Toc28965673)

[3.3 Разработка алгоритмов программного средства 14](#_Toc28965674)

[3.4 Схемы алгоритмов по ГОСТ 19.701-90 20](#_Toc28965675)

[4 Руководство по использованию программного средства 23](#_Toc28965679)

[Заключение 28](#_Toc28965680)

[Список использованной литературы 29](#_Toc28965681)

[Приложение А 30](#_Toc28965682)

Введение

На сегодняшний день однопользовательские компьютерные игры утрачивают свою былую популярность, так как их вытесняют сетевые многопользовательские аналоги. Данная тенденция объясняется появлением спроса у игроков, стремящихся соревноваться с другими людьми и, побеждая их, таким образом доказывать своё превосходство и мастерство игры.

Сетевые игры являются разделом теории игр, который изучает как методы формирования связей между игроками в конфликтно-управляемых системах, так и правила определения выигрышей игроков с учётом этих связей. В основном выделяют три подхода к формированию связей между игроками.

При стратегическом подходе сетевую игру можно рассматривать как классическую игру в нормальной форме: каждый игрок независимо выбирает свою стратегию. В кооперативном варианте сетевой игры основной проблемой является выбор правила распределения суммарного выигрыша игроков между собой при некоторой наперед заданной или сформированной самими игроками сетевой структуре. При рассмотрении динамического подхода предполагается, что сетевая структура может пересматриваться игроками поочередно согласно некоторому порядку очередности ходов.

Данный курсовой проект направлен на создание простого сетевого приложения для игры в шахматы по локальной сети с клиент-серверной архитектурой. Актуальность такого продукта подкрепляется бурным развитием сетевых технологий.

Помимо реализации сетевой составляющей приложения, в данном курсовом проекте будут смоделированы основные шахматные правила и использована международная стандартизированная шахматная нотация.

# Анализ предметной области

## Архитектуры взаимодействия по сети

1.1.1 Одноранговая (пиринговая)

Оверлейная компьютерная сеть, основанная на равноправии участников. Часто в такой сети отсутствуют выделенные серверы, а каждый узел (peer) является как клиентом, так и выполняет функции сервера. В отличие от архитектуры клиент-сервера, такая организация позволяет сохранять работоспособность сети при любом количестве и любом сочетании доступных узлов. Участниками сети являются все пиры.

В сети присутствует некоторое количество машин, при этом каждая может связаться с любой из других. Каждая из этих машин может посылать запросы другим машинам на предоставление каких-либо ресурсов в пределах этой сети и, таким образом, выступать в роли клиента. Будучи сервером, каждая машина должна быть способной обрабатывать запросы от других машин в сети, отсылать то, что было запрошено. Каждая машина также должна выполнять некоторые вспомогательные и административные функции (например, хранить список других известных машин-«соседей» и поддерживать его актуальность).

Любой член данной сети не гарантирует своего присутствия на постоянной основе. Он может появляться и исчезать в любой момент времени. Но при достижении определённого критического размера сети наступает такой момент, что в сети одновременно существует множество серверов с одинаковыми функциями.

1.1.2 Многоранговая (клиент-сервер)

Вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами. Обычно эти программы расположены на разных машинах и взаимодействуют между собой через сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных.

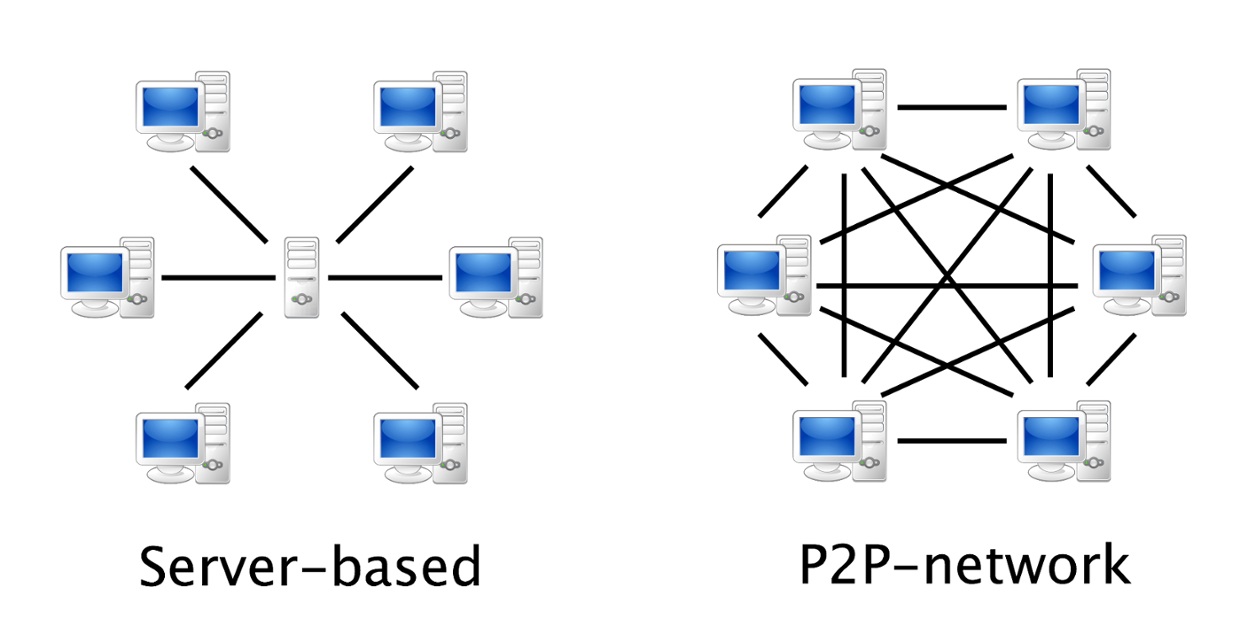


Рисунок 1.1 – Разные сетевые архитектуры

## Основы шахматных правил

Шахматы – настольная логическая игра со специальными фигурами на 64-клеточной доске для двух соперников, сочетающая в себе элементы искусства (в том числе в части шахматной композиции), науки и спорта.

Игра происходит на доске, поделённой на равные квадратные клетки, или поля. Размер доски — 8×8 клеток. Вертикальные ряды полей (вертикали) обозначаются латинскими буквами от а до h слева направо, горизонтальные ряды (горизонтали) — цифрами от 1 до 8 снизу вверх; каждое поле обозначается сочетанием соответствующих буквы и цифры. Поля раскрашены в тёмный и светлый цвета (и называются, соответственно, чёрными и белыми) так, что соседние по вертикали и горизонтали поля раскрашены в разные цвета.

У игроков в начале игры имеется по одинаковому набору фигур. Фигуры одного из игроков условно называются «белыми», другого — «чёрными». Сами игроки называются «белые» и «чёрные» по цвету своих фигур.

В каждый комплект фигур входят: король (♔, ♚), ферзь (♕, ♛), две ладьи (♖, ♜), два слона (♗, ♝), два коня (♘, ♞) и восемь пешек (♙, ♟). Белые занимают первую и вторую горизонтали, чёрные — седьмую и восьмую. Пешки расположены на второй и седьмой горизонталях соответственно.

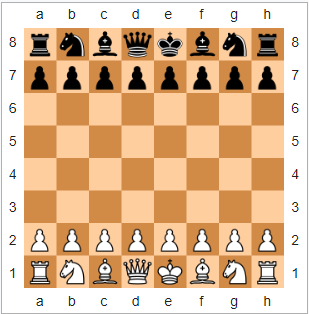


Рисунок 1.2 – Начальные позиции фигур

Игра заключается в том, что игроки поочерёдно делают ходы. Первый ход делают белые. Ход заключается в том, что игрок перемещает одну из своих фигур на другое поле по следующим правилам:

* Фигуры (кроме коня) передвигаются по прямой линии, при этом все промежуточные поля между начальным и конечным должны быть свободны (на них не должно быть своих или чужих фигур). Исключением является ход коня.
* Ход на поле, занятое своей фигурой, невозможен.
* При ходе на поле, занятое чужой фигурой, она снимается с доски (взятие).
* Король ходит на соседнюю клетку по вертикали, горизонтали или диагонали.
* Ферзь ходит на любое расстояние по вертикали, горизонтали или диагонали.
* Ладья ходит на любое расстояние по вертикали или горизонтали.
* Слон ходит на любое расстояние по диагонали.
* Конь ходит «буквой Г»: на две клетки по вертикали и на одну клетку по горизонтали, или наоборот, на две клетки по горизонтали и на одну клетку по вертикали. Конь отличается от остальных фигур также тем, что другие фигуры, стоящие пути коня, не препятствуют его ходу (он через них «перепрыгивает»).
* Пешка может ходить только вперёд (направлением «вперёд» называется направление к восьмой горизонтали для белых или к первой для чёрных): без взятия — на одно поле вперёд по вертикали, а со взятием — по диагонали на одно поле вперёд-вправо или вперёд-влево. Если пешка находится на начальном поле (вторая горизонталь для белых и седьмая для чёрных), то кроме этого, она может сделать ход без взятия на два поля вперёд. Когда пешка ходит на последнюю горизонталь (для белых — на восьмую, для чёрных — на первую), она заменяется по выбору игрока на любую другую фигуру того же цвета, кроме короля (превращение пешки). Превращение пешки является частью того хода, которым она перемещается на последнюю горизонталь.

Поле называется находящимся под ударом или битым, если при своём ходе фигура могла бы взять находящуюся на этом поле фигуру противника (независимо от того, есть ли такая фигура на этом поле). Поле считается битым, даже если фактически ход фигурой туда невозможен, так как ставит под бой собственного короля.

Король, находящийся на битом поле, называется «стоящим под шахом». Сделать ход, после которого король противника оказывается под шахом, значит «объявить шах». Ходы, после которых король сделавшего ход остаётся или оказывается под шахом, запрещены.

Если король игрока находится под шахом и игрок не имеет ни одного хода, позволяющего устранить этот шах, этот игрок называется «получившим мат» и, соответственно, он терпит поражение.

Если игрок при своей очереди хода не имеет возможности сделать ни одного хода по правилам, но король игрока не находится под шахом, такая ситуация называется пат.

Также в шахматах существует два специальных хода:

* Рокировка — если король и одна из ладей того же цвета не двигались с начала игры, то король и эта ладья могут в один ход одновременно сменить положение (рокироваться). При рокировке король сдвигается на две клетки по направлению к ладье, а ладья ставится на поле между начальной и конечной позицией короля.
* Взятие на проходе — когда пешка совершает ход на две клетки через битое поле, находящееся под ударом пешки противника, то ответным ходом она может быть взята этой пешкой противника. При этом пешка противника перемещается на битое поле, а сбитая пешка снимается с доски.

Однако стоит заметить, что данные ходы не будут реализованы в разрабатываемом приложении.

## Нотация Форсайта – Эдвардса

Forsyth–Edwards Notation, FEN — стандартная нотация записи шахматных диаграмм. Предложена в 1883 г. шотландским шахматистом Д. Форсайтом, а позже доработана программистом С. Эдвардсом.

Запись FEN описывает позицию на шахматной доске в виде строки ASCII символов. Поля записи:

* Положение фигур со стороны белых. Позиция описывается цифрами и буквами по горизонталям сверху вниз начиная с восьмой горизонтали и заканчивая первой. Расположение фигур на горизонтали записывается слева направо, данные каждой горизонтали разделяются косой чертой /. Белые фигуры обозначаются заглавными буквами. K, Q, R, B, N, P — соответственно белые король, ферзь, ладья, слон, конь, пешка. k, q, r, b, n, p — соответственно чёрные король, ферзь, ладья, слон, конь, пешка. Цифра задаёт количество пустых полей на горизонтали, счёт ведётся либо от левого края доски, либо после фигуры (8 означает пустую горизонталь).
* Активная сторона: w — следующий ход принадлежит белым, b — следующий ход чёрных.
* Возможность рокировки. k — в сторону королевского фланга (короткая), q — в сторону ферзевого фланга (длинная). Заглавными указываются белые. Невозможность рокировки обозначается «-».
* Возможность взятия пешки на проходе. Указывается проходимое поле, иначе «-».
* Счётчик полуходов. Число полуходов, прошедших с последнего хода пешки или взятия фигуры. Используется для определения применения правила 50 ходов.
* Номер хода. Счётчик увеличивается на 1 после каждого хода чёрных.

Например, FEN для начальной позиции шахматной партии записывается так: rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1

## Обзор существующих аналогов

Самым популярным на данный момент шахматным сервером является сайт <https://lichess.org>. Это бесплатный шахматный сервер с открытым исходным кодом, работающий на добровольных началах и пожертвованиях.

В 2010 году Тибо Дюплессис начал работу над Lichess как хобби-проект. В начале сайт был прост, даже не проверяя, были ли ходы законными. Постепенно сайт улучшился и собрал энтузиастов-добровольцев, которые помогли разработчику создать и поддерживать сайт. Сегодня пользователи Lichess играют более миллиона игр каждый день. Lichess - один из самых популярных шахматных сайтов в мире, оставаясь на 100% бесплатным.

Однако играть в шахматы - далеко не единственное, что предоставляет Lichess. После окончания игры можно запросить компьютерный анализ с использованием новейшего шахматного движка Stockfish и учиться на своих ошибках или сравнивать свою игру с огромной базой данных игр шахматистов. Сервис предоставляет услугу онлайн просмотра важнейших шахматных партий в мире. Даже чемпионы мира играют на Lichess. Игроки, желающие совершенствоваться, могут нанять тренера и использовать совместные «занятия» в реальном времени для обмена играми, позициями, аннотированными вариантами и чатом.

Данный Интернет-ресурс предоставляет возможность играть с друзьями, а также с искусственным интеллектом, обладающим десятью уровнями мастерства.

Стоит заметить, что это огромная система, создававшаяся долгое время большой командой. Подобные проекты невероятно сложны в разработке и требуют большое количество ресурсов.



Рисунок 1.3 – Игра с компьютером на lichess.org

Самым известным шахматным приложением операционной системы Windows является игра Chess Titans. Chess Titans подчиняется традиционным правилам игры в шахматы. Игровое поле представляет собой шахматную доску, двое соперников управляют традиционными шахматными фигурами. В игре присутствует как однопользовательский режим, в котором противником игрока-человека является компьютер, так и многопользовательский, в котором имеют возможность соревноваться два человека. При игре с компьютером игрок может выбирать цвет своих фигур. Имеется возможность вернуть партию на любое количество ходов назад. При выборе фигуры клетки, на которые она может ходить, подсвечиваются. Также реализованы внутриигровые подсказки. Текущую игру можно сохранить для продолжения при следующем запуске программы; результаты игры могут записываться в файл статистики.

Реализована система из десяти уровней сложности игрока-компьютера. Первый уровень сложности, по мнению обозревателя от ресурса Genius Prophecy Chess, предназначен для новичков, десятый может оказаться сложным даже для лучших игроков.

Графика игры реализована в 3D-режиме, во время многопользовательской игры шахматная доска поворачивается к каждому игроку его частью доски, когда следует ход этого игрока. Вид игрового процесса в данной игре представлен на рисунке 1.4.



Рисунок 1.4 – Игровой процесс в Chess Titans

# Анализ требований к программному средству

Приложение OldChess предназначается для сетевой игры в шахматы и включает в себя три составляющие: серверную часть, клиентскую часть и модуль шахматной логики.

Сервер должен обладать следующим функциями:

* обеспечение надёжного подключения клиента с его регистрацией в качестве пользователя;
* создание игровой сессии одним пользователем;
* подключение пользователя к уже созданной игровой сессии;
* верификация игровой сессии обоими игроками;
* обеспечение игрового процесса с учётом возможности покинуть игру любым из пользователей;
* обеспечение безопасного отключения клиента от сервера в любой момент времени;
* поддержка обслуживания множества игровых сессий единовременно;
* выдача информации о состоянии пользователей и сессий администратору сервера.

Клиент должен обладать следующим функциями:

* предоставление пользователю инструментов подключения к серверу, создания новой игровой сессии и присоединения к уже существующей;
* возможность переподключения к другому работающему серверу;
* предоставление пользователю полной информации об оппоненте;
* обеспечение приятного графического интерфейса в отношении объектов игрового процесса (шахматных фигур и доски).

Модуль шахматной логики должен содержать следующую функциональность:

* математическая формализация и реализация основных шахматных правил в отношении ходов фигур, включая превращение пешки в ферзя, за исключением рокировок и взятия на проходе;
* предоставление списка всех доступных пользователю ходов с учётом невозможности хода под шах и необходимости устранения шаха в свою сторону в обязательном порядке;
* чёткая фиксация конца игры в виде мата или пата.

Для передачи сообщений клиент и сервер используют протокол TCP/IP.

# Проектирование и Разработка программного средства

## Структура программного средства

В соответствие с требованиями программное средство OldChess разбито на 3 части.

Модуль шахматной логики «Chess» представляет собой библиотеку классов для .NET Standard, которая в скомпилированном виде представляет собой .dll файл, который непосредственно используется клиентом. Данный модуль содержит 8 классов:

* Board.cs – класс, описывающий шахматную доску;
* Chess.cs – главный класс, описывающий непосредственно объект шахмат, представляющий собой интерфейс, с помощью которого клиент может взаимодействовать с библиотекой;
* Color.cs – класс, описывающий цвет шахматной фигуры;
* Figure.cs – класс, представляющий непосредственно шахматную фигуру;
* FigureOnSquare.cs – класс, представляющий фигуру в контексте её расположения на доске;
* FigureMoving.cs – класс, описывающий перемещение фигуры;
* Moves.cs – класс, реализующий шахматные правила в отношении ходов фигур;
* Square.cs – класс, описывающий клетку шахматной доски.

Клиентский модуль представляет собой графическое приложение Windows Forms, которое содержит три формы:

* frmMain.cs – главная форма приложения, содержащая инструменты взаимодействия с сервером и непосредственно объекты игрового процесса;
* frmConnect.cs – форма, отвечающая за подключение к серверу;
* frmJoin.cs – форма, отвечающая за подключение к созданной игровой сессии.

Серверный модуль представляет собой консольное приложение и включает три класса:

* ChessServer.cs – класс, непосредственно реализующий функции обработки клиентских запросов;
* User.cs – класс, представляющий пользователя на сервере;
* GameSession.cs – класс, описывающий игровую сессию на сервере.

## Разработка серверной части

В ходе проектирования была выработана схема работы серверной части приложения, представленная на рисунке 3.1.

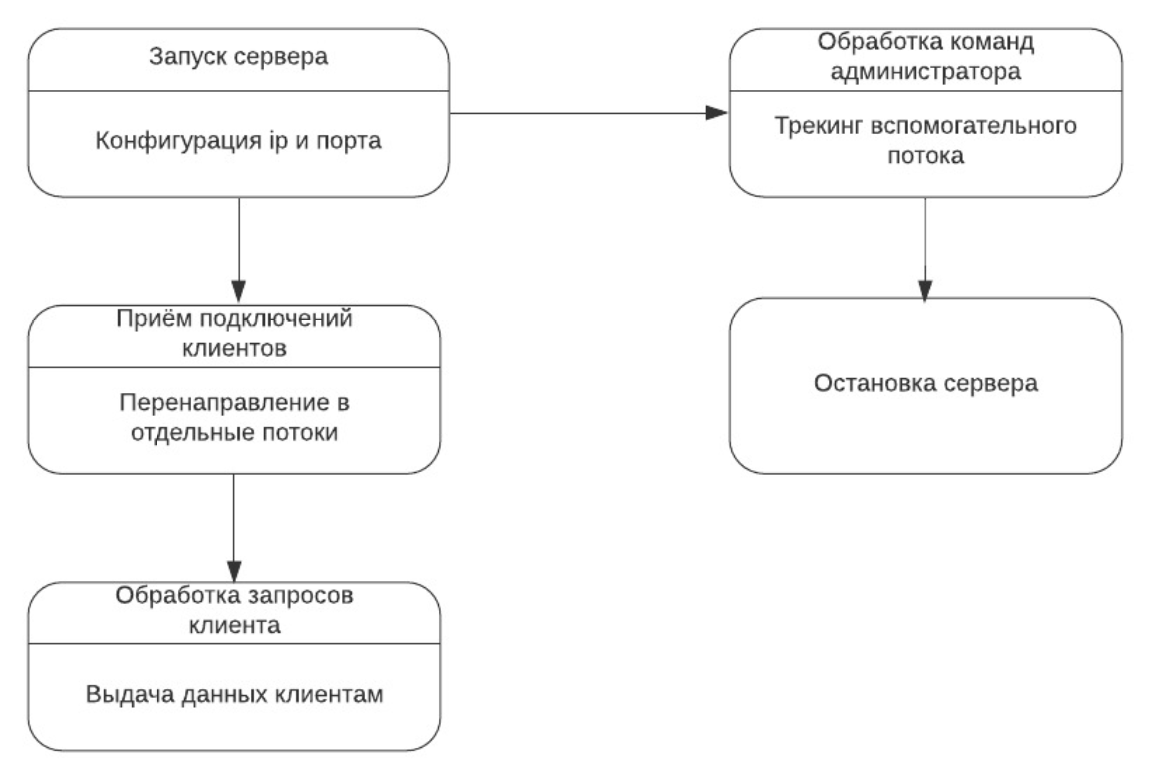


Рисунок 3.1 – Схема работы сервера

Сервер использует класс TcpListener, который находится в пространстве имён System.Net.Sockets. Данный класс позволяет привязать сервер на определённый ip адрес и порт и запустить прослушивание указанного порта. В основом потоке сервер постоянно ожидает подключений и с помощью метода AcceptTcpClient получает объект сокета класса TcpClient, который используется в качестве абстракции клиента, и запускает новый поток с целью обработки запросов подключившегося клиента.

В созданном потоке сервер постоянно ожидает запросы клиента. Список поддерживаемых запросов:

* «CONNECT <UserName>» - зарегистрировать пользователя с уникальным именем <UserName>;
* «DISCONNECT <UserName>» - удалить пользователя;
* «NEWGAME <UserName> <side>» - создать новую игру за сторону <side>;
* «GETAVAIL <UserName>» - получить список доступных игр;
* «JOIN <UserName> <SessionID>» - присоединиться к сессии с идентификатором <SessionID>;
* «ACCEPT <UserName>» - принять игру;
* «REJECT <UserName>» - отклонить игру;
* «MOVE <UserName> <move>» - сделать ход <move>;
* «MOVEWIN <UserName> <move>» - поставить мат ходом <move>;
* «MOVEDRAW <UserName> <move>» - поставить пат ходом <move>;
* «QUITGAME <UserName>» - покинуть игру;
* «CANCELNEW <UserName>» - отменить поиск игры.

Соответственно, сервер отправляет ответы клиентам. Список поддерживаемых ответов:

* «OK» - подтверждение регистрации пользователя;
* «(<SessionID> <OpponentName> <OpponentSide>\n)+» - список доступных игр;
* «NONE» - отсутствие доступных игр;
* «DESTROY» - отмена игровой сессии;
* «GAMEREADY <OpponentName> <OpponentSide>» - оповещение о готовности игры с противником под именем <OpponentName>, играющего за сторону <OpponentSide>;
* «GAMESTART <PlayerSide> <SessionID>» - начало игры в игровой сессии с идентификатором <SessionID> за сторону <PlayerSide>;
* «WIN» - игрок победил в партии;
* «DRAW» - партия закончилась ничьёй;
* «DEFEATED» - игрок проиграл в партии.

Помимо этих ответов, сервер пересылает противнику игрока сообщения его запросов «MOVE», «MOVEWIN» и «MOVEDRAW» в неизменяемом виде. Стоит заметить, что сервер не обрабатывает самостоятельно игровые ходы, а лишь пересылает их оппоненту. Текущая шахматная доска формируется у каждого клиента независимо.

Новый объект пользователя класса User создаётся после запроса CONNECT при условии, что имя пользователя уникально. Для коммуникации с клиентом в данном объекте присутствует соответствующее поле класса TcpClient, в котором и находится объект клиента.

Новый объект игровой сессии класса GameSession создаётся после запроса NEWGAME. Он содержит поля для игрока белых и игрока чёрных и поле состояния. Игровая сессия может быть в одном из четырёх состояний:

* wait – состояние ожидания начала игры;
* AcceptOne – состояние после принятия игры одним из пользователей;
* inProgress – состояние, в котором игра идёт;
* destroyed – состояние, описывающее либо законченную, либо отклонённую игровую сессию.

В соответствие с заявленными требованиями администратор сервера имеет расширяемый список команд, который на текущий момент представляется следующим списком:

* «!stop» - остановить сервер при условии, что к нему не подключен ни один пользователь;
* «!users» - получить список подключённых пользователей и их действия;
* «!sessions» - получить список созданных игровых сессий с их статусом.

## Разработка клиентской части

Системообразующей основы работы клиента является схема переходов состояний, вид которой представлен на рисунке 3.2.

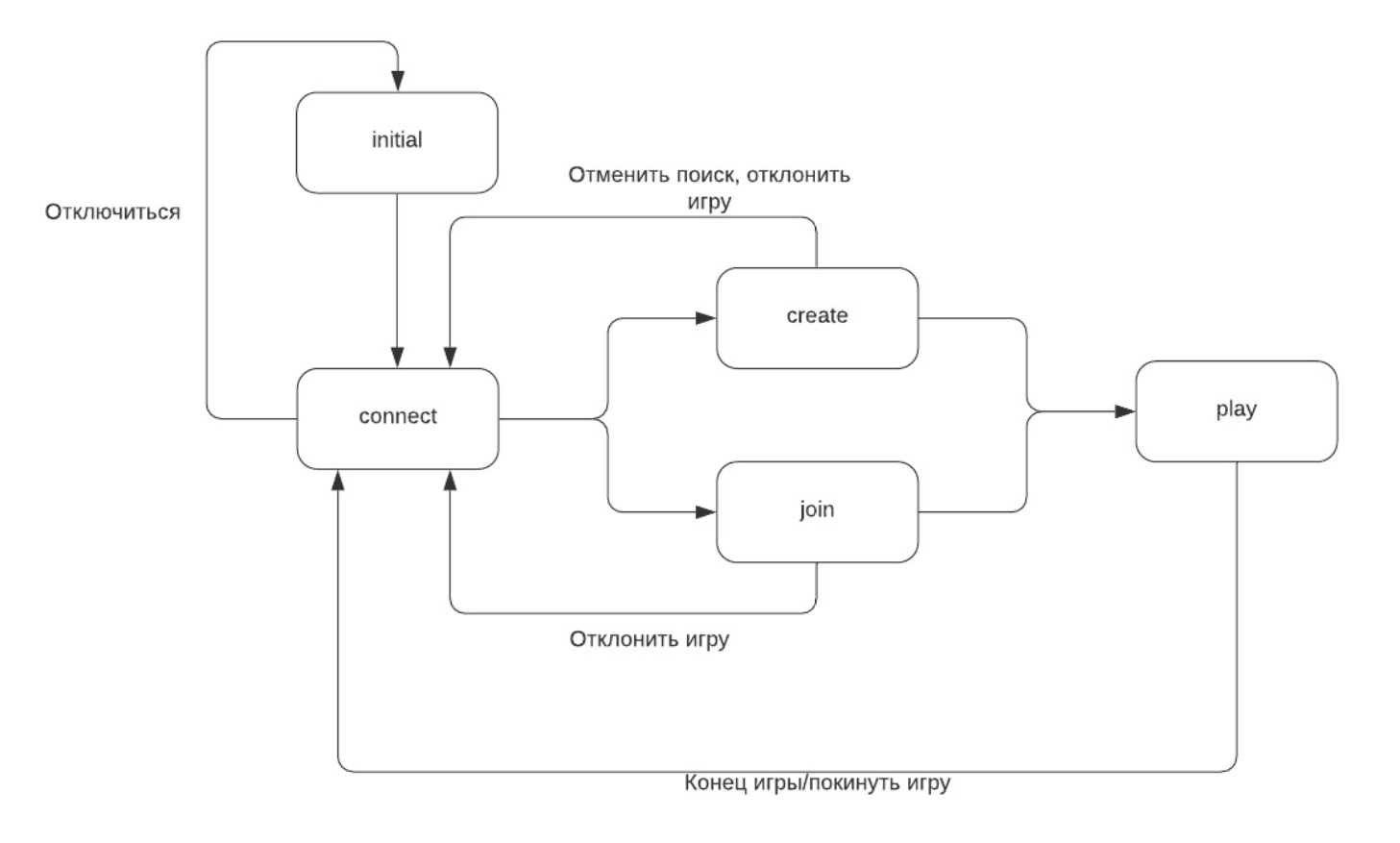


Рисунок 3.2 – Схема работы клиента

Начальное состояние initial является стартовым для клиента. Далее он двигается вперёд к connect, что означает подключение к определённому серверу. Затем клиенту предоставляется возможность либо создать игру, либо присоединиться к уже созданной. Конечное состояние play означает, что клиент находится в состоянии игры. Обратим внимание, что на схеме есть дополнительные обратные переходы, которые подписаны условиями, при которых они происходят. Таким образом обеспечивается возможность проведения нескольких игр пользователем и переподключения к разным серверам.

За счёт переходов между состояниями, на каждом этапе пользователю предоставляются соответствующие состоянию опции взаимодействия с сервером, что отражается в изменениях графического интерфейса, а также в отображении вспомогательной информации.

Для реализации клиента используется класс TcpClient, который находится в пространстве имён System.Net.Sockets. В его конструктор передаются IP-адрес сервера и номер его открытого порта. Далее клиент общается с сервером помощью методов SendMessage и GetServerResponse. Они в свою очередь используют поток объекта клиента класса NetworkStream для записи и чтения данных оттуда. При общении с сервером используются запросы, описанные ранее.

Следует заметить, что клиент устроен так, что пользователь управляет приложением (в том числе и играет) в основном потоке. При создании новой игровой сессии, стартует поток, в котором происходит ожидание игры, её принятие пользователем и инициализация шахматной партии. Этот подход не блокирует пользовательский интерфейс и является оптимальным для подобных приложений. Также новый поток создаётся в методе инициализации партии. В нём происходит обработка получаемых от сервера сообщений.

Следует обратить внимание на то, что изменять компоненты формы может лишь основной поток. Для отображения изменений из потока обработки ответов сервера используется метод Invoke, который применяется к компонентам формы класса Control. В данный метод передаётся лямбда-выражение, которое по сути своей является анонимным делегатом, выполняющемся в основном потоке. Пример использования метода Invoke:

menuStrip1.Invoke(new Action(() => RefreshServerControlTools()));

Для графического интерфейса шахматных фигур и доски используются компоненты класса Panel, которые могут окрашиваться в разные цвета и могут содержать в себе изображение. Шахматные фигуры были добавлены в ресурсы проекта в виде PNG изображений.

Отличительной особенностью шахматного клиента в данном курсовом проекте является подсветка для пользователя всех возможных фигур для хода, а также всех клеток, куда можно сделать ход текущей фигурой. При этом учитывается текущая ситуация на доске (находится ли король под шахом, может ли король походить под шах и т.п.). Данный аспект обеспечивается шахматной логикой, разработанной в классе Moves.cs библиотеки «Chess».

## Схемы алгоритмов по ГОСТ 19.701-90

### Схема алгоритма is\_cycle\_exist

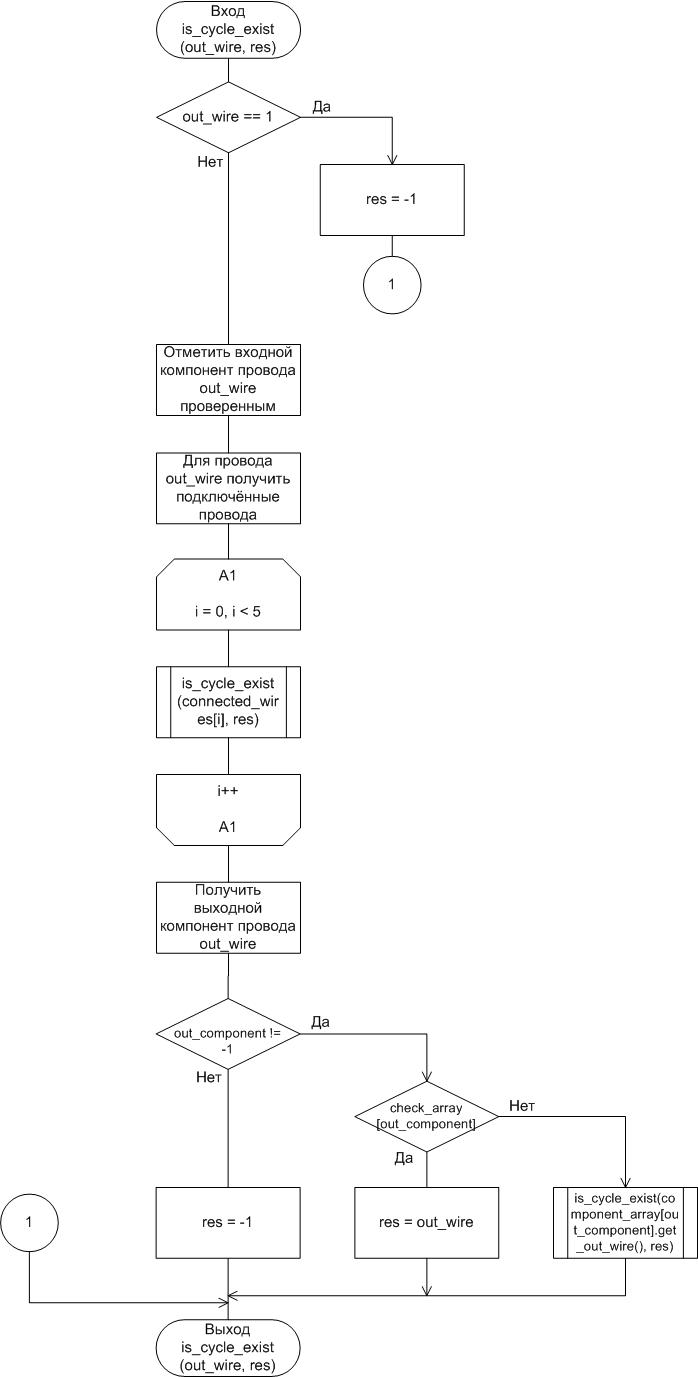


Рисунок 3.6 – Схема алгоритма is\_cycle\_exist

### Схема алгоритма delete\_component

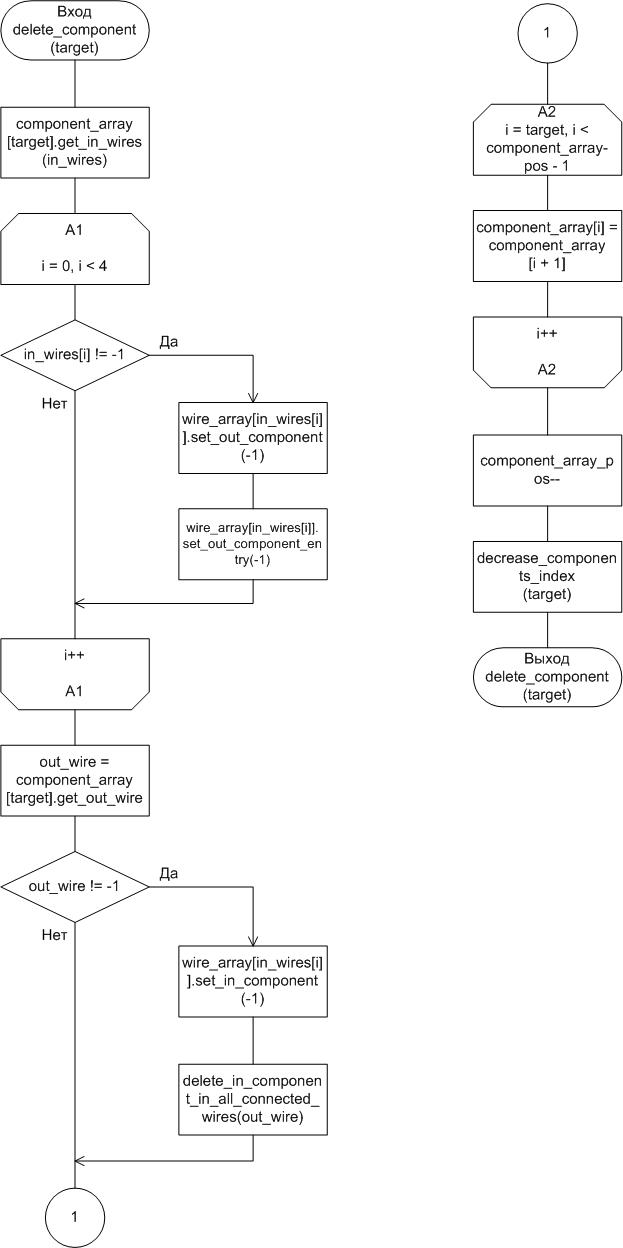


Рисунок 3.7 – Схема алгоритма delete\_component

### Схема алгоритма valid\_wire\_end

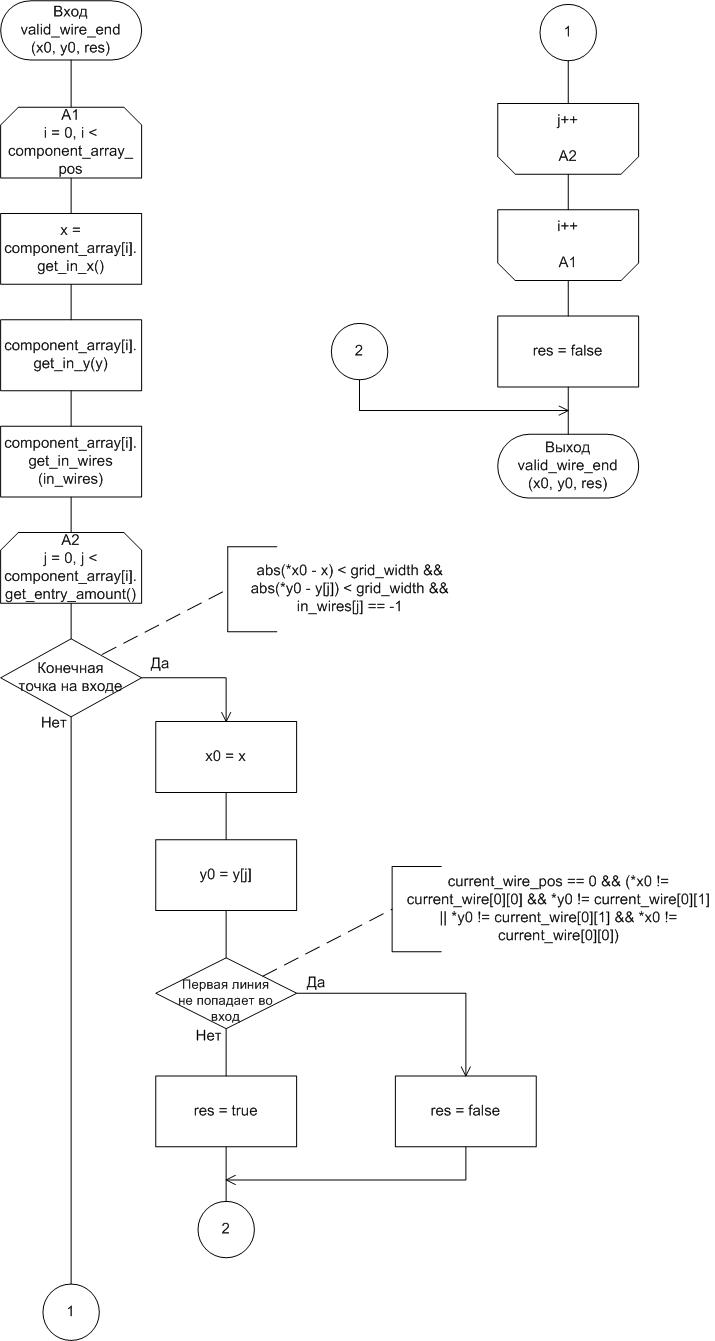


Рисунок 3.8 – Схема алгоритма valid\_wire\_end

# Руководство по использованию программного средства

Приложение LogicBuilder предназначается для построения логических схем линейных булевых функций и моделирования процесса их функционирования.

Процесс работы в приложении разбит на несколько режимов:

* cursor mode – режим курсора, в котором на полотне размещаются компоненты, производится выделение, перемещение и удаление компонентов и проводов;
* model mode – режим моделирования работы построенной схемы, позволяющий пользователю менять значения на источнике и следить за изменениями на схеме;
* wire mode – режим построения провода для соединения компонентов;
* branch wire mode – режим построения ответвляющегося провода, стартовой точкой которого является другой провод.

Режим курсора включается нажатием клавиши «C» и является активированным по умолчанию при открытии приложения. В данном режиме двойным щелчком мыши по названию компонента из списка слева выбирается желаемый компонент. Далее пользователь может свободно располагать компоненты этого типа по полотну одинарным щелчком мыши. При этом, последний добавленный компонент остаётся выделенным, что позволяет сразу же его перемещать.

Перемещение происходит с помощью управляющих клавиш «W», «A», «S», «D». По умолчанию шагом перемещения является значение 10 пикселей, что является удвоенным значением ширины сетки полотна. Для изменения этого параметра используется клавиша «shift». При её нажатии шаг перемещения изменяется циклически по схеме 10-15-20-5. Таким образом, компоненты можно расположить с любой необходимой точностью. Заметим, что перемещается лишь один выделенный компонент. Процесс работы в режиме курсора отображён на рисунке 4.1.

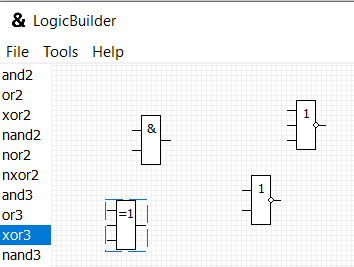


Рисунок 4.1 – Режим курсора

Также в режиме курсора происходит выделение и удаление проводов. Вид выделенного провода представлен на рисунке 4.2.

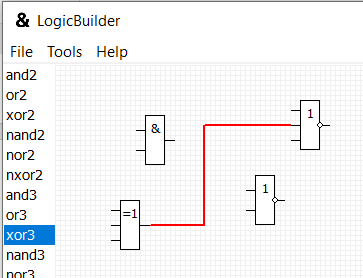


Рисунок 4.2 – Выделенный провод

Выделение и компонентов, и проводов происходит в результате щелчка по объекту. При щелчке по любому свободному месту полотна выделение пропадает. Также стоит упомянуть возможность включения и отключения сетки полотна клавишей «G». К слову, данная опция работает во всех режимах.

Режим провода активируется нажатием клавиши «X». Провода строятся строго из выходов компонентов во входы других компонентов. При этом, при наведении на доступные стартовые точки для провода, они подсвечиваются. Например, как на рисунке 4.3.

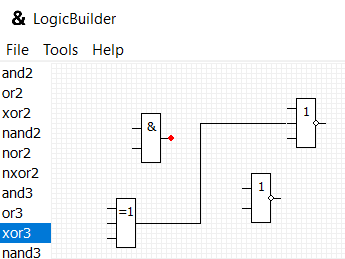


Рисунок 4.3 – Стартовая точка провода

При этом занятые входы или выходы не доступны для построения провода. Далее начинается собственно построение провода и его временное отображение. Для того чтобы сбросить построение, необходимо перейти в режим курсора и щёлкнуть по свободному месту на полотне. Пример построения провода показан на рисунке 4.4.

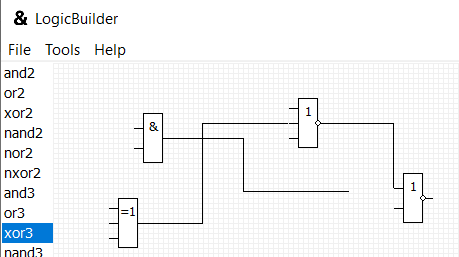


Рисунок 4.4 – Построение провода

Для окончания провода во входе компонента необходимо нажать клавишу «Alt». После этого активируются доступные конечные точки. Данная ситуация отображена на рисунке 4.5.

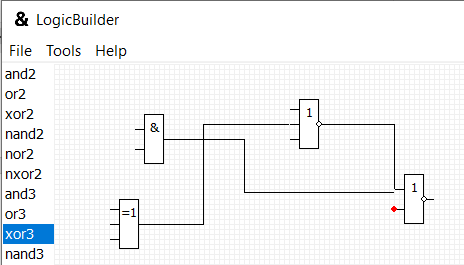


Рисунок 4.5 – Конечная точка провода

Заметим, что произойдёт автоматическая коррекция высоты провода и он будет подключён ко входу компонента.

В режиме branch wire mode стартовой точкой провода служит другой провод, а активируется он нажатием клавиши «Z». Вся остальная механика построения такая как и в wire mode. Пример ситуации показан на рисунке 4.6.

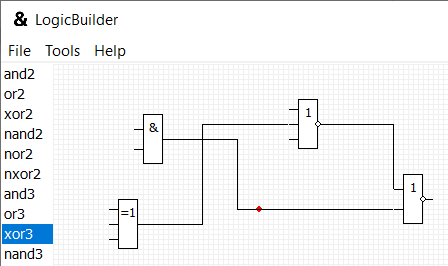


Рисунок 4.6 – Branch wire mode

Режим моделирования активируется и деактивируется клавишей «V». При этом стандартными значениями на источниках являются единицы. Сразу после включения схема начинает работать. Пользователь может нажимать на источники, изменяя их выходные значения. При обнаружении в схеме цикла приложение сообщает о невозможности обработки такой схемы. Пример режима моделирования иллюстрирует рисунок 4.7.

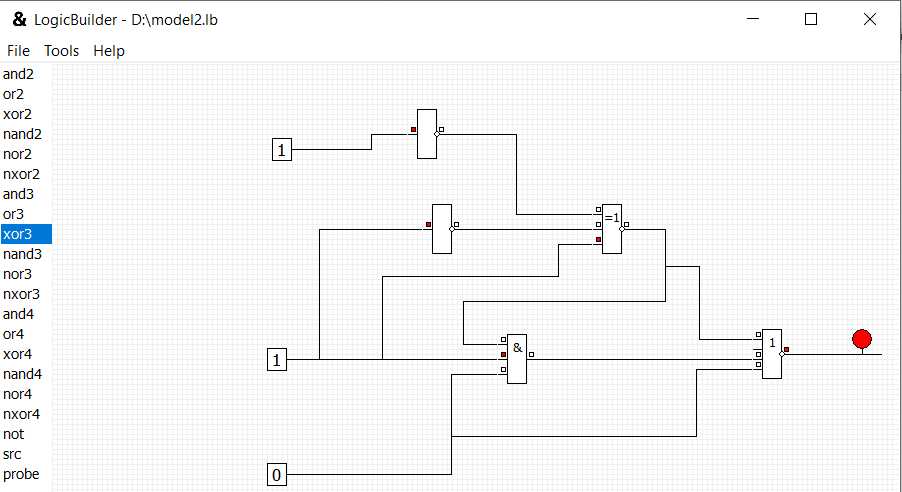


Рисунок 4.7 – Режим моделирования

Все управляющие действия находятся также на вкладке главного меню «Tools». Вид вкладки на рисунке 4.8.

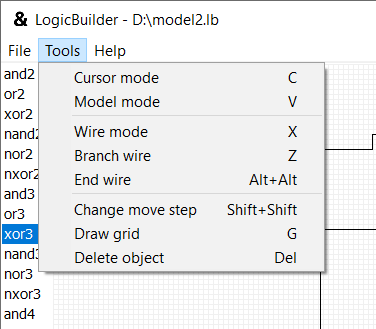


Рисунок 4.8 – Вкладка инструментов

На вкладке главного меню «File» находятся действия, отвечающие за сохранение и открытие файла, создание нового файла, а также экспорт изображения в формат SVG. Вид данной вкладки представлен на рисунке 4.9.

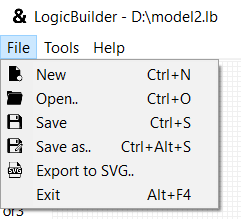


Рисунок 4.9 – Вкладка файла в главном меню

Также в главном меню есть вкладка со справочной информацией по использованию приложения. Открывающееся при этом окно показано на рисунке 4.10.

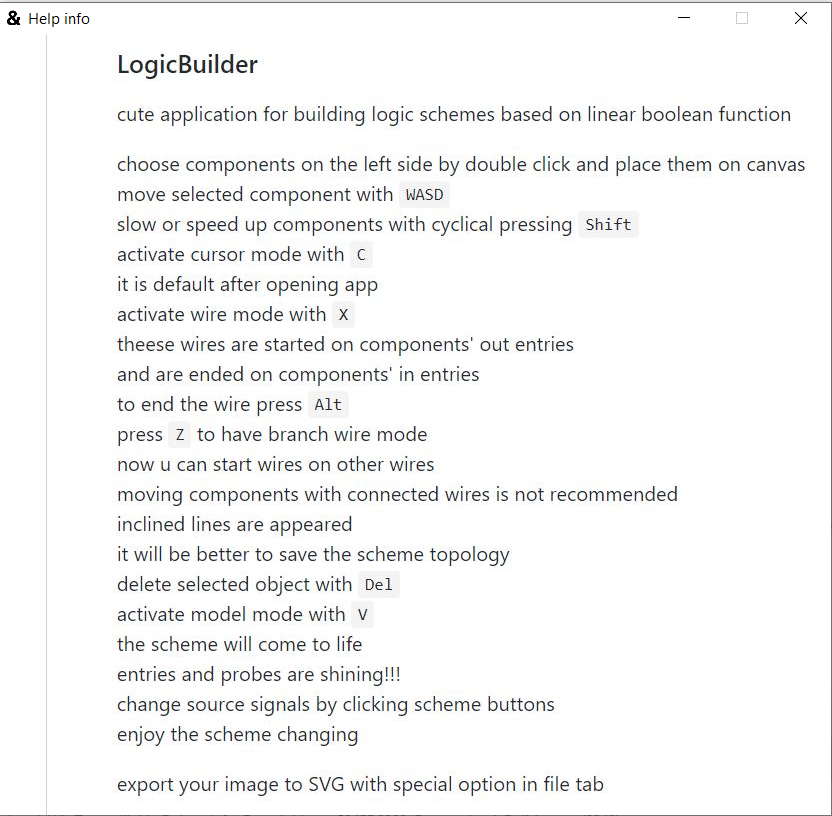


Рисунок 4.10 – Окно справочной информации

Заключение

В ходе разработки было создано приложение для построения логических схем, основанных на линейных булевых функциях. Помимо выполнения своего основного функционала, оно предоставляет пользователю возможность смоделировать процесс работы схемы, проследить изменение входных и выходных сигналов на каждом узле схемы при подаче на неё различных стартовых наборов. Таким образом, пользователь получает способ для проверки правильности построения схемы.

Данное приложение может быть улучшено посредством добавления в режим моделирования предиктивных вычислений и, как следствие, циклов в схеме. Это обстоятельство позволит моделировать такие устройства, как синхронные и асинхронные RS-триггеры, одно- и двухступенчатые D-триггеры и т.п. В процесс построения схемы может быть добавлено множественное выделение объектов, локальный буфер обмена, а также определённая гибкость по отношению к редактированию построенных проводов. Однако стоит заметить, что уже в настоящем своём виде приложение LogicBuilder хорошо справляется со своей задачей.

В ходе написания данного курсового проекта были получены базовые знания по основам объектно-ориентированного программирования, а также навыки модульной декомпозиции в языке C++. Приобретённые знания будут широко использоваться в дальнейшем. Также было сформировано представление о внутреннем устройстве формата векторной графики SVG.

Список использованной литературы

[1] Bjarne Stroustrup, The C++ Programming Language/ Addison-Wesley Professional; 4 edition (May 19, 2013)

[2] Шевелев Ю. П. Дискретная математика. Ч.1: Теория множеств. Булева алгебра: Учебное пособие — Томск: гос. ун-т систем управления и радиоэлектроники, 2003. — 118 с.

[3] Лысиков Б.Г., Арифметические и логические основы цифровых автоматов: учебник для вузов. 2-е изд., перераб. и доп. — Минск: Высш.шк., 1980 – 336 с.

[4] Нарышкин А. К. Цифровые устройства и микропроцессоры: Учеб. пособие для студ. вузов — М.: Издательский центр «Академия», 2006. – 320 с.

[5] В.А. Скляров. Язык C++ и объектно-ориентированное программирование: Справочное издание. - Минск: Вышэйшая школа, 1997. 480с.

[6] Прата, Стивен Язык программирования C++. Лекции и упражнения / Стивен Прата. - М.: Вильямс, 2015.

[7] Х. Дейтел, П. Дейтел. Как программировать на C++: Пер. с англ. - Москва: ЗАО "Издательство БИНОМ", 1998. 1024с

[8] w3.org [Электронный ресурс] – Режим доступа:

<https://www.w3.org/Graphics/SVG/1.1/DTD/svg1.dtd> – Дата доступа: 17.12.2019.

Приложение А

(обязательное)

Исходный код программы

bool valid\_local\_line\_is\_alone(int target\_wire, int mode){

if (wire\_array[target\_wire].

get\_connected\_wires\_amount() != 0) {

int connected\_wires[5];

wire\_array[target\_wire].get\_connected\_wires(

connected\_wires);

int x01, y01, x02, y02;

if (mode == 1)

wire\_array[target\_wire].get\_first\_line(&x01, &y01,

&x02, &y02);

else

wire\_array[target\_wire].get\_last\_line(&x01, &y01,

&x02, &y02);

for (int j = 0; j < wire\_array[target\_wire].

get\_connected\_wires\_amount(); j++) {

int x1, y1, x2, y2;

wire\_array[connected\_wires[j]].get\_first\_line(

&x1, &y1, &x2, &y2);

if ((y1 >= y01 && y1 <= y02 ||

y1 <= y01 && y1 >= y01) && (x1 >= x01 &&

x1 <= x02 || x1 <= x01 && x1 >= x01)) {

return false;

}

}

}

return true;

}

void add\_wire(int item[10][4]){

if (wire\_array\_pos == 300)

return;

int start;

int end[2] = {-1, -1};

if (!branch\_wire\_mode)

start = find\_start\_component(item[0][0], item[0][1]);

else {

start = wire\_array[parent\_wire].get\_in\_component();

int temp = wire\_array[parent\_wire].

get\_connected\_wires\_amount();

if (temp == 5)

return;

wire\_array[parent\_wire].set\_connected\_wire(

wire\_array\_pos, temp++);

wire\_array[parent\_wire].

set\_connected\_wires\_amount(temp);

}

find\_end\_component(end, item[current\_wire\_pos][2],

item[current\_wire\_pos][3]);

if (!branch\_wire\_mode) {

component\_array[start].set\_out\_wire(wire\_array\_pos);

wire\_array[wire\_array\_pos].set\_parent\_wire(-1);

}

else

wire\_array[wire\_array\_pos].

set\_parent\_wire(parent\_wire);

component\_array[end[0]].set\_in\_wire(wire\_array\_pos, end[1]);

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

wire\_array[wire\_array\_pos].set\_connected\_wire(-1, i);

}

wire\_array[wire\_array\_pos].set\_connected\_wires\_amount(0);

wire\_array[wire\_array\_pos].set\_in\_out\_component(start,

end[0]);

wire\_array[wire\_array\_pos].set\_out\_component\_entry(end[1]);

wire\_array[wire\_array\_pos].set\_lines(item);

wire\_array[wire\_array\_pos++].set\_lines\_amount(

++current\_wire\_pos);

}

int generate\_out\_charge(int in\_charges[4],

int (\*bool\_func)(int, int)){

int res = -1;

for (int i = 0 ; i < 4; i++){

if (res == -1 && in\_charges[i] != -1){

res = in\_charges[i];

continue;

}

if (in\_charges[i] != -1)

res = (\*bool\_func)(res, in\_charges[i]);

}

return res;

}

bool is\_trash\_component(ModelComponent entity){

int in\_wires[4];

entity.get\_in\_wires(in\_wires);

for (int i = 0; i < 4; i++)

if (in\_wires[i] != -1)

if (wire\_array[in\_wires[i]].

get\_in\_component() != -1)

return false;

return true;

}

ВЕДОМОСТЬ ДОКУМЕНТОВ

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнительные сведения* | |
|  | | | | | Текстовые документы | | | |  | |
|  | | | | |  | | | |  | |
| БГУИР КП 1–40 01 01 421 ПЗ | | | | | Пояснительная записка | | | | 32 с. | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | | Графические документы | | | |  | |
|  | | | | |  | | | |  | |
| ГУИР.851004-01 СА | | | | | Режим моделирования работы логической схемы | | | | Формат А1 | |
|  | | | | | Схема алгоритма | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  | | | | |  | | | |  | |
|  |  |  |  |  | БГУИР КП I- 40 01 01 421 ПЗ | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | Программное средство для построения логических схем  Ведомость курсового  проекта | Литера | | | Лист | Листов |
| Разраб. | | Пашкевич А.Л. |  | 27.12 | Т |  |  | 32 | 32 |
| Провер. | | Варфоломеев А.В. |  | 28.12 | Кафедра ПОИТ  гр. 851004 | | | | |
|  | |  |  |  |
|  | |  |  |  |
|  | |  |  |  |