МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «ВГТУ»)

Факультет экономики, менеджмента и информационных технологий
(факультет)
Кафедра Систем управления и информационных технологий в строительстве

ЗАДАНИЕ
на курсовой проект

по дисциплине: «Основы программирования и алгоритмизации»

Тема: «Разработка игры головоломки Куромасу»

Студент группы бИСТ-214 Тюленев Данил Вячеславович
Фамилия, имя, отчество

Вариант № А10

Технические условия процессор Intel® Core™ i3-8145U CPU @ 2,10 ГГц, операционная система Windows 10, ОЗУ 8192 МБ

Содержание и объем проекта (графические работы, расчёты и прочее):
74 стр,36 рисунков,3 таб,1 приложение

Сроки выполнения этапов: анализ и постановка задачи (07.10-14.10); разработка пошаговой детализации программы (14.10-21.10); реализация программы (21.10-5.11); тестирование программы (5.11-12.11); оформление пояснительной записки (12.11-26.11).

Срок защиты курсового проекта: _____

Руководитель _____ О.В. Минакова
Подпись, дата Инициалы, фамилия

Задание принял студент _____ Д.В. Тюленев
Подпись, дата Инициалы, фамилия

Содержание

Введение

В настоящее время повышается актуальность головоломок. Часто говорят, что наш мозг перегружен информационным потоком: интернет и телевидение разнообразили получаемую информацию современным человеком (появляется информационное перенасыщение), но такие источники представляют материал пассивного восприятия, мешают концентрироваться на отдельных вещах и никак не развивают умения самостоятельной оценки и отбора получаемой информации.

Решение головоломок приносит огромную пользу человеку на любом этапе его жизни. Головоломки направлены на развитие логического, пространственного и конструктивного мышления. В результате этих игровых упражнений и заданий происходят эффективные тренировки памяти, логики, сообразительности, воображения, находчивости, наблюдательности.

Головоломка — непростая задача, для решения которой, как правило, требуется сообразительность, а не специальные знания высокого уровня.
Существует огромное количество головоломок различного вида и какой-либо общепринятой классификации нет, поэтому можно лишь условно разделить их на несколько групп:

1. Устные головоломки — задачи, полное условие которых может быть сообщено в устной форме, не требующие для решения привлечения никаких дополнительных предметов (загадки, шарады, логические парадоксы).
2. Головоломки с предметами — логические задачи с обычными бытовыми предметами (головоломки с спичками, карточные головоломки).
3. Механические головоломки — предметы, специально изготовленные как головоломки (кубик Рубика, змейки Рубика, проволочные, пятнашки шкатулки с секретом и т. п.).

4. Печатные головоломки — напечатанные или нарисованные «картинки», в которых надо нарисовать какие-либо символы по определённым правилам (кроссворд, ребус, судоку, японский кроссворд и т. п.).

С развитием компьютеров стали активно развиваться компьютерные головоломки. В первую очередь туда попали устные и печатные головоломки, а также стали активно создаваться программы-головоломки: флеш-головоломки, онлайн головоломки, пасьянсы и другие.

Головоломка (видеоигра) — название жанра компьютерных игр, целью которых является решение логических задач, требующих от игрока задействования логики, стратегии и интуиции. Виды головоломок: *Традиционная* (в играх повторяется игровой процесс обычных игр-головоломок) и *Физическая* (решение логических задач путём взаимодействия игровых объектов с помощью физики).

«Куромасу» — (иначе «Куродоко», в переводе «Где чёрные клетки?») японская логическая головоломка с двоичным определением, опубликованная «Николи» (японский издатель, специализирующийся на играх и особенно на логических головоломках), впервые появилась в «Puzzle Communication Nikoli № 34» (июнь 1991 г.). На англоязычном веб-сайте «Nikoli» название переводится с английского как «Где находятся чёрные клетки?».

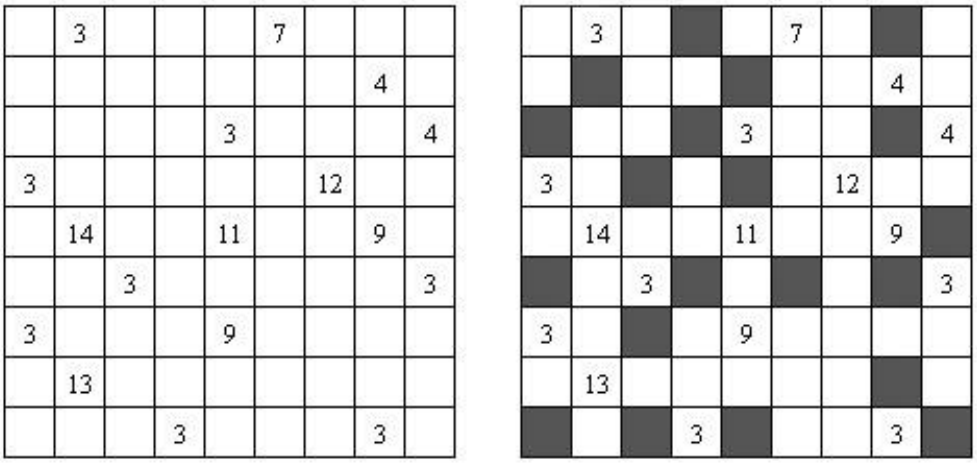

Рисунок 1 — Пример решённого игрового поля

Головоломка представляет собой сетку, некоторые ячейки которой первоначально содержат числа. Каждая ячейка может быть или чёрной или белой. Цель состоит в том, чтобы определить, к какому типу относится каждая ячейка. Следующие правила определяют цвета ячеек:

1. Каждое число на доске представляет количество белых клеток, которые можно увидеть из этой клетки, включая её самого.

2. Ячейку можно увидеть из другой ячейки, если они находятся в той же строке или столбце, и между ними нет чёрных ячеек в этой строке или столбце.

3. Пронумерованные ячейки не могут быть чёрными.

4. Никакие две чёрные клетки не могут быть смежными по горизонтали или вертикали.

5. Все белые клетки должны быть соединены горизонтально или вертикально.

Используемая система программирования: Си (англ. *C*) — компилируемый статически типизированный язык программирования общего назначения, разработанный в 1969—1973 годах сотрудником Bell Labs Деннисом Ритчи. Язык С был разработан для системного программирования ещё в 1970-х, но до сих пор остаётся невероятно популярным. Системные языки рассчитаны на производительность и простоту доступа к внутреннему аппаратному обеспечению, но при этом обладают высокоуровневыми возможностями.

Реализация программы, игры в жанре головоломки, *«Куромасу»* помогает закрепить знания, полученные в процессе изучения дисциплины «Основы программирования и алгоритмизации», приобрести практические навыки самостоятельной разработки программ, структурного программирования и на практике продемонстрировать знания языка программирования Си.

Целью данной работы является создание алгоритма и разработка программы на языке Си для реализации компонентов головоломки: игровой процесс (множество игровых механик), пользовательский интерфейс (оформление экрана для отображения игровых полей и меню; чтение значений,

вводимых пользователем с клавиатуры), структуры уровней с возможностью для редактирования (создание объектов и задания определённых свойств каждому из них).

Разработанная программа служит для организации игрового процесса головоломки «Куромасу», следит за ходом игры и контролирует правильность заполнения значений для игровой сетки.

Основные задачи, которые необходимо решить в рамках проектирования:

1. Изучить предметную область: правила игры «Куромасу» и принцип решения головоломки.
2. Провести анализ задания и постановку задачи.
3. Составить структуру программного продукта, разработать алгоритм рабочего прототипа, пошаговую детализацию программы.
4. Реализовать программный продукт в соответствии с техническим заданием.
5. Провести тестирование готового решения, выявить недоработки и устранить их.

Основная часть

1. Постановка задачи

1.1. Описание игрового поля головоломки

В начале проектирования программы, необходимо определить как будет реализована одна из важных составляющей игры, своеобразный каркас, основа головоломки — представление игрового поля, принцип его хранения и реализации в программе. Игровое поле — квадратная сетка, элементами которой являются клетки. Чтобы в последующем при проверке было удобно обрабатывать отдельные взятые клетки, каждую из них необходимо представить в виде объекта с общими характеристиками: установленное значение («0» — белая (свободная) клетка; «-1» — чёрная (занятая) клетка; любые другие значения — в клетке установленно число видимых клеток), цвет для отрисовки (чтобы наглядно определить видимость каких клеток была ограниченно верно, а где была допущена ошибка), значение видимых клеток по оси «*ox» и «oy»*. Игровое поле будет представлять собой линейное множество таких объектов, которое при отображении будет принимать матричный вид. Чтобы ссылаться к отдельным клеткам поля, необходимо проиндексировать множество данных игрового поля, отвечающее за хранение значений клеток.

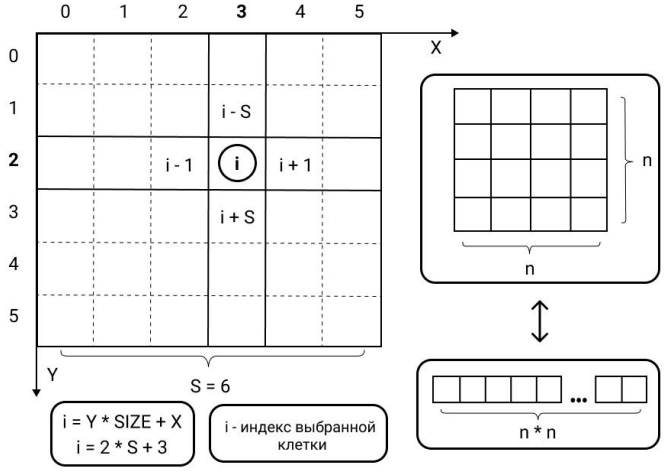

Рисунок 2 — Принцип индексации и хранения игрового поля

В данном случае индекс ячейки имеет вид: $(Y \times SIZE + X)$, где Y — значение вертикальной координаты клетки, X — значение горизонтальной координаты клетки, SIZE — размер строки, которой принадлежит клетка.

Продолжительное решение одного и того же уровня без возможности смены быстро утомит пользователя. Поэтому в программе должна реализованна функция замены карты. Но хранить значения в самой программе не рационально, с точки зрении возможности редактирования и замены предустановленных уровней. Вся информация об уровне будет храниться в отдельном отформатированном файле с расширением «*.txt». При загрузке уровня, данные из файла считываются и на их основе формируется объект игрового поля. Чтобы понимать какой из понимать какой файл из предустановленной директории необходимо прочитать, программа хранить значение имени текущего выбранного поля (имя файла и имя поля совпадают).

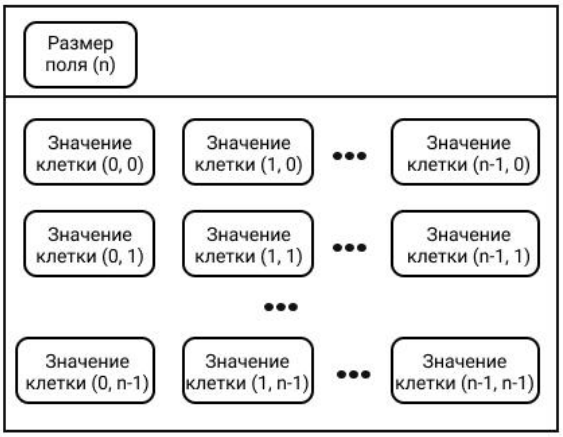

Рисунок 3 — Формат хранения параметров игрового поля внутри файла

1.2. Пользовательский интерфейс программы

Следующей задачей стоит создание механизма пользовательского ввода. Необходимо проанализировать доступные методы управления:

1. Ввод с помощью команды, с указанием координаты клетки и значения, которое будет установленно в клетку.

2. Ввод с помощью курсора: выбор клетки путём перемещения маркера с помощью стрелочной схемы управления, и установки значения в клетку с помощью отдельно предустановленной клавиши.

Изучив допустимые методы управления, было принято решение реализовать взаимодействие пользователя с игрой используя способ №2, поскольку он исключает возможность установки ошибочных значении и является более удобным чем способ №1 и интуитивно понятным для игрока. Пользовательский ввод будет реализоваться с помощью обработчика, который будет отслеживать событие нажатия клавиши клавиатуры, после выполнять действия в зависимости от кода символа считанного с консоли:
за перемещение курсора будет отвечать группа клавиш «← ↑ ↓ →», а за подтверждение и отмену действий — «SPACE» и «ESCAPE» (опционально). Положение курсора описывается с помощью структуры из 2 двух значений, хранящая значение смещения положения метки по осям «*ox*» и «*oy*» относительно начала кадра отрисовки поля (левый верхний угол).

Установленное значение клетки определяется двумя значениями: белая клетки или чёрная; поэтому для определения нового значения клетки можно использовать метод переключения: значение будет сменяться на противоположное при нажатии на клавишу ответственную за

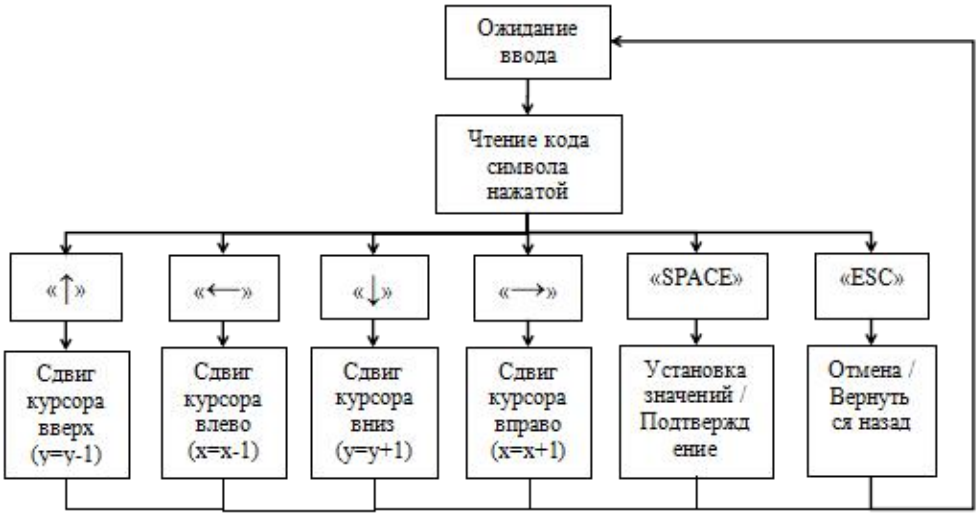

Рисунок 4 — Схема алгоритма управления

Наша программа будет работать через текстовый компьютерный интерфейс, в качестве которого выступает консольное окно, в котором для отображении элементом необходимо использовать комбинации любых печатаемых символов ASCII. Принцип отображения информации будет происходит циклично по следующему принципу: нужная сформированная на печать страница рисуется в консоли → ожидается ввод со стороны пользователя → происходит обработка вводимой команды → предыдущее отрисованное содержимое стирается → рисуется новая страница, основанная на новых введённых пользователем значений.

Необходимо представить, как будет выглядеть консольное окно, определить какая информация должна выводиться, и каким образом её располагать. В верхней части будет отображаться статичная шапка программы. Отображение шапки будет производиться независимо от страницы, элементами которой будет информация о доступных командах (клавишах взаимодействия). После шапки будет выводиться весь основной контент страницы.

Главной для игрока является страница, на которой происходит процесс решения головоломки. В качестве объектов, располагающихся на ней, будут данные о текущих оставшихся попытках, положение курсора (координаты), видимые клетки выбранной ячейки и игровое поле. В процессе отрисовки игровой сетки, необходимо демонстрировать пользователю местоположение курсора, используя заливку выбранной клетки определённым цветом. Также для облегчения процесса решения, использовать заливку нумерованных клеток: зелёных цвет — значение видимых клеток выставлено верно, красный цвет — значение видимых клеток выставлено неверно. Чёрные клетки можно маркировать с помощью символа («X»).

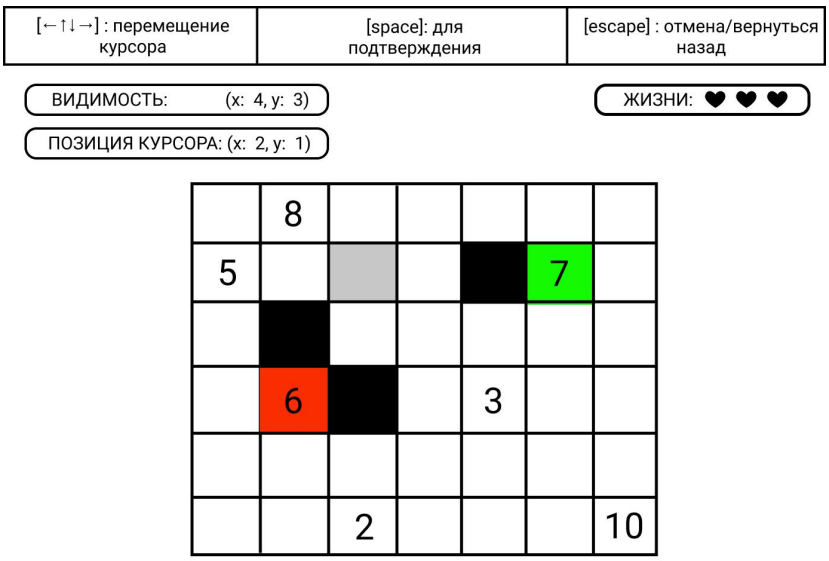

Рисунок 5 — Макет преставления страницы с игровым полем

Другие страницы (главное меню, меню редактора и другие) необходимо оформлять аналогично продемонстрированному примеру.

При запуске программы пользователю будет демонстрироваться страница главного меню, выступающая в качестве главного узла, так как она соединяет остальные страницы и позволяет обратиться к ним.

1.3. Модуль проверки решения головоломки

Процесс обработки и проверки правильности заполнения пользователем поля можно реализовать с помощью отдельного модуля, вызываемого в момент после произведённого ввода со стороны пользователя. Глобальный этап проверки разумно разделить на отдельные сегменты проверок различного характера:

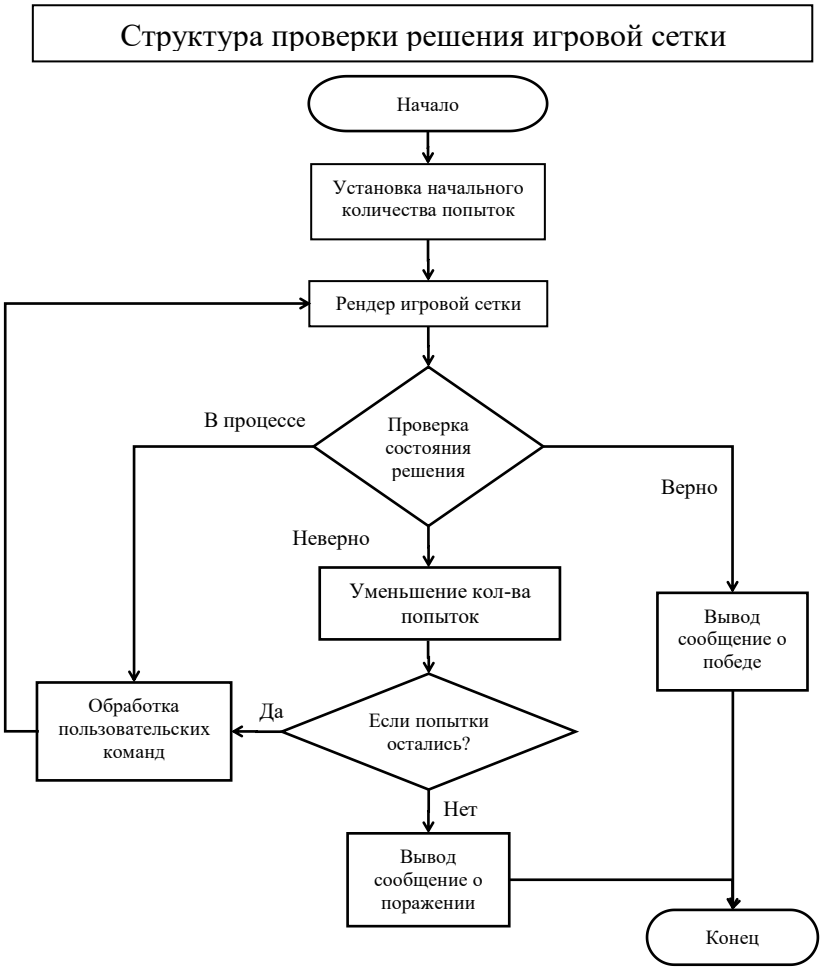

Рисунок 6 — Структура общей проверки решения игровой сетки

1. Проверка установки чёрной метки в клетку со значением. Программа обязана исключить вероятность установки значения соответствующего чёрной клетки в ячейку с предустановленным статическим значением поля. Другими словами, если игрок установил курсор в пронумерованную клетку и совершает попытку смены значения белого цвета на чёрный, модуль проверки должен игнорировать данную команду от пользователя.

2. Исключение касаний клеток с чёрной меткой друг с другом по горизонтали и вертикали. При попытке установки пользователем значений соответствующего чёрной клетки по соседству относительно друг друга, блокировать ввод со стороны пользователя.

3. Проверка, что все белые клетки соединены по горизонтали или вертикали. Нельзя допустить, чтобы пользователь в процессе решения уровня, разорвал

связь между каждой белой клеткой, выставив значения чёрных меток. При совершений ошибки данной группы общий счётчик оставшихся попыток будет декрементирован.

4. Проверка нумерованных клеток, что их предустановленное значение не меньше суммы видимых клеток по двум осям. Важнейший элемент контроля игрового процесса. Если все вышеперечисленные проверки пропустили установленный пользователем чёрный маркер, то происходит перерасчёт значений поля видимых клеток для всех ячеек расположенных на двух ортогональных линиях (oy и ox), имеющих пересечение в точке, где было установлено значение (позиция курсора). Необходимо помнить, что каждая клетка хранит значение видимых клеток по двум линиям. Поэтому для каждой линий совершается одна и та же операция: линия разбивается на несколько отрезков в соответствии с расположением установленных чёрных меток → высчитывается значение длины каждого из отрезков → после для каждой ячейки линий будет перезаписано значение видимых клеток на длину отрезка, которому она принадлежит. Далее для каждой нумерованной клетки производится сравнение установленного значения и уменьшенной на единицу суммы длины двух отрезков (разных линий), которым она принадлежит: если значение суммы меньше чем значение ячейки, то значение установленной пользователем метки возвращается в изначальное состояние и количество оставшихся попыток уменьшается на единицу; если значение суммы равно значению ячейки, то игроку засчитывается победа; иначе решение игровой сетки продолжается.

Общая проверка по всем четырём пунктам производится каждый раз, когда было изменено значение какой-либо клетки игровой сетки. Изначально при загрузки уровня из файла общий счётчик оставшихся попыток устанавливается на основе определённой константы, которая характеризует максимальное количество «жизней игрока».

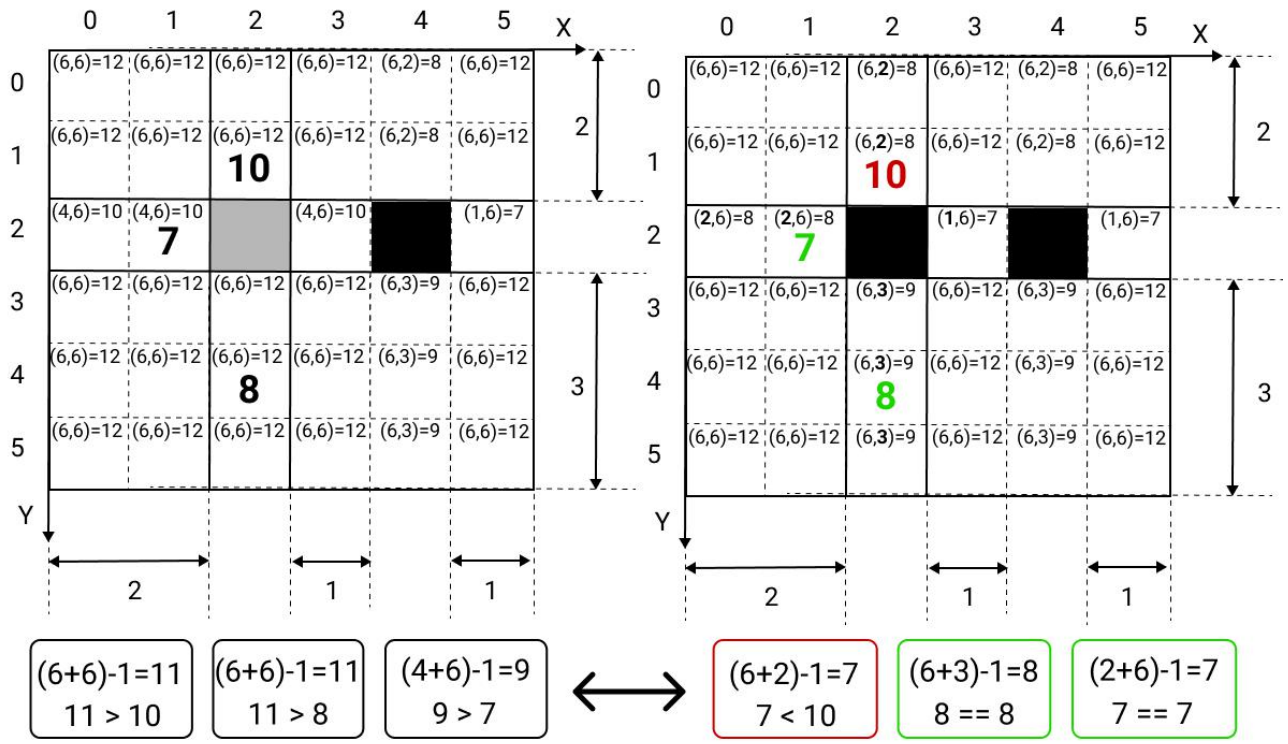

Рисунок 7 — Принцип проверки поля (пункт №4)

1.4. Процесс редактирования и смены полей головоломок

Редактор карт для программы будет представлять группу страниц: страница создания новых полей с последующим сохранением в предустановленной директории, страница выбора карт из директории (для каждой карты предусматривается два действия: выбор в качестве основной карты для игры, редактирование выбранной карты и сохранение в директории), страница с правилами игры (необходимо, чтобы новый пользователь программы смог разобраться с правилами игры).

Процесс создания представляет собой последовательность следующих действий: пользователем устанавливается размер сетки → консольном окне рисуется поле установленного размера, на котором пользователь перемещая курсор и выбирая нужную клетку устанавливает значения «видимости» → полю присваивается название, которое ему даёт игрок, и его значения сохраняются в файл по описанному в разделе «**Описание игрового поля головоломки**» формату.

Страница выбора сохранённых карт представляет список с значениями имён полей. Элементы данной страница формируются на основе названий файлов в предустановленной директории. Пользователь курсором выбирает поле, после чего перед ним встаёт выбор дальнейших действий: выбрать поле в качестве основного или отредактировать его содержимое.

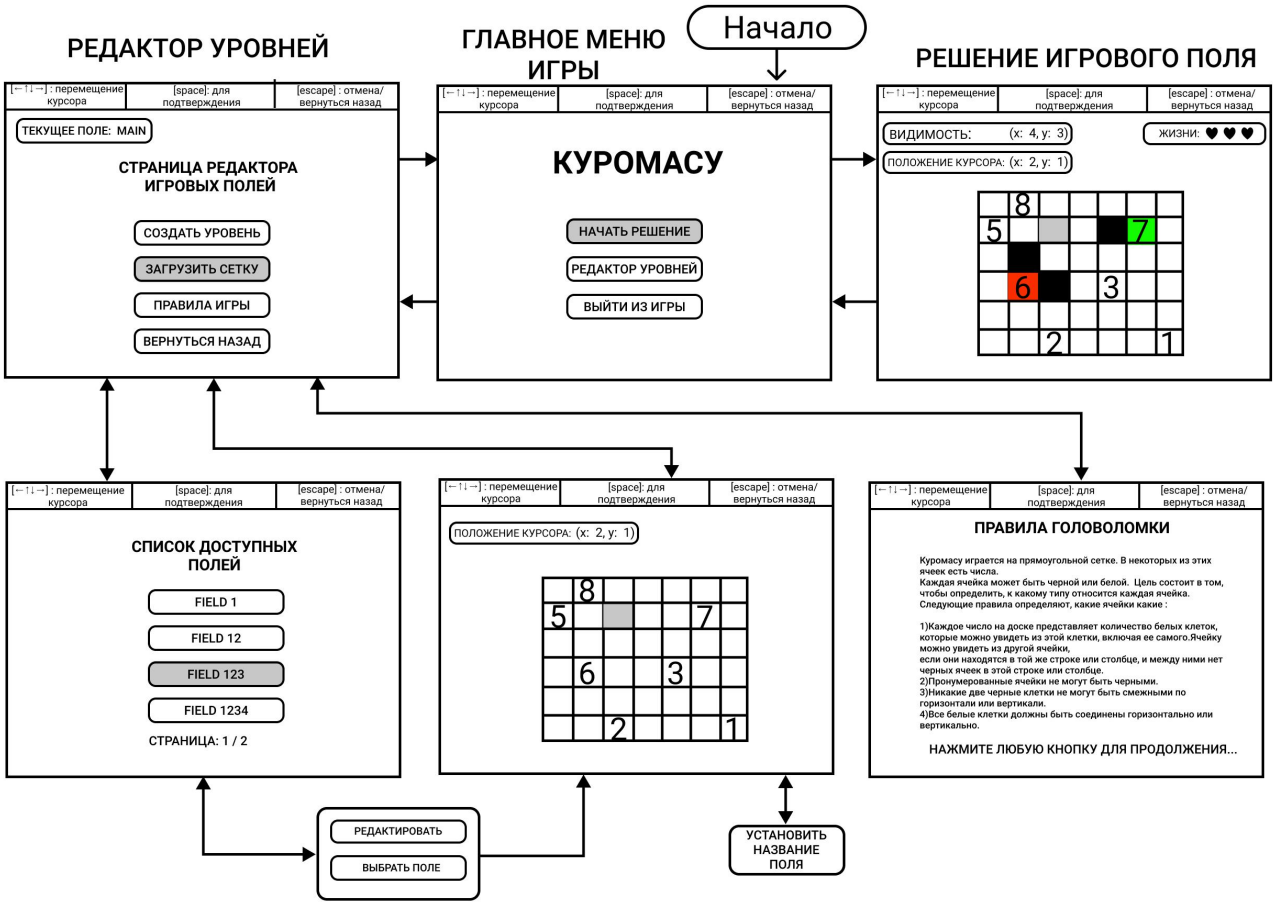

Рисунок 8 — Концепция взаимосвязи и работы страниц

2. Конструирование программы

Прежде чем начинать анализ решения задачи и непосредственно приступать к её выполнению, необходимо для большего понимания и удобства разбить задачу на подзадачи и для каждой расписать функции и определить новые типы данных:

1. Модуль пользовательского интерфейса.
2. Модуль, характеризующий игровой процесс головоломки.
3. Модуль для работы с файловой системой.

2.1. Модуль пользовательского интерфейса

Главной задачей стоит обработка пользовательского ввода. Для её решения разработаем функцию kaction_t get_keyboard_input(tuple_t* pos, tuple_t max) — обрабатывает ввод с клавиатуры через функцию «getch» — читает код нажатой клавиши и проверяет его значение на соответствие определённым константам целочисленного типа с помощью оператора «switch», с помощью которого изменяется значение указателя «pos» ответственного за местоположение курсора; возвращает значение состояния обработки ввода. С помощью переменной «max» будет устанавливаться границы для курсора, за которые он не сможет выйти. Чтобы было проще сопоставлять значение кода клавиши и условного значения для кейса определим несколько констант отвечающий за коды клавиш — для этого объявим перечисление «enum», в котором укажем константные значения для целочисленных значений клавиш. Такая конструкция позволит быстро в случае необходимости изменить клавиши отвечающие за управление.

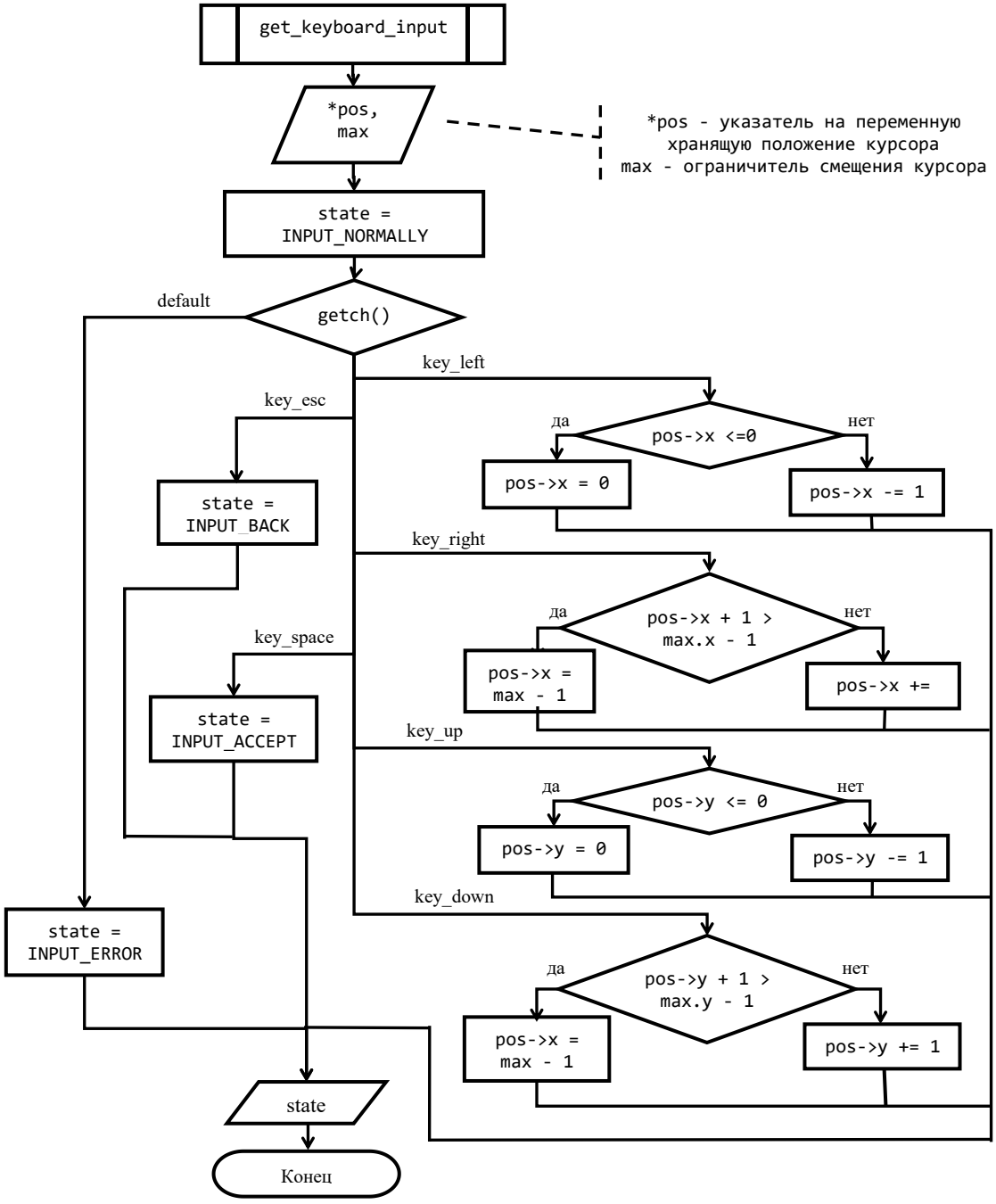

Рисунок 9 — Блок-схема функции обработки ввода с клавиатуры

Для работы со значениями координат (сгруппированных двух переменных в одну с общим именем) определим новый структурный тип «tuple_t», полями которого будут две целочисленные неотрицательные числа - координаты. Данную запись можно использовать при объявлении переменных, в которых необходимо храниться пары сгруппированных значений, относящихся к одной задачи. Также для удобной работы с событиями ввода

определим новые именованные целочисленное констант с помощью ключевого слова «enum»: («INPUT_ERROR» — пользователь нажал неверную клавишу, «INPUT_NORMALLY» — пользователь нажал клавишу для смены положение курсора, «INPUT_ACCEPT» — пользователь нажал клавишу для подтверждения, «INPUT_BACK» — пользователь нажал клавишу "возврата").

Чтобы при объявлении новых переменных агрегированного типа не писать постоянно ключевые слова «struct» и «enum» «тип» применим оператор «typedef» с помощью которого будем переопределять типы, для сокращения записи объявлений.

Следующим этапом разработки становиться создание функции для запуска отображения созданных страниц и обработки состояний ввода. Чтобы не повторять одинаковую программную конструкцию (цикл для анализа статуса ввода и отрисовки кадров) при реализации каждой вкладки, для этой задачи создадим отдельную функцию:

void* update_frame (gupdate_t(*action)(void* args, kaction_t action, tuple_t pos), tuple_t max, bool back_use, void* param) — функция загрузки цикличной отрисовки кадров страницы и обработки событий ввода. Чтобы можно было указать на объект, с которым будет производиться взаимодействие, в функции перед открытием цикла обработки инициализируется локальная переменная, ответственная за значение местоположения курсора. После этого открывается цикл, в котором каждую итерацию происходит анализ статуса ввода и отрисовка нужных элементов на странице: происходит вызов функции «action», которая отображает необходимые элементы страницы в зависимости от входных параметров и обрабатывает состояния управления; проверяется возращённое значение функции «action» — состояние текущего кадра (итерации): отрисовка продолжается, вернуться в начало текущей отрисовки или выйти из отрисовки данного кадра и вернуть значения; после вызывается функция «get_keyboard_input», которая ожидает команды со стороны игрока, генерирует состояние нажатия и изменяет значение переменной, ответственной

за положение курсора. С использованием параметра «back_use» можно заблокировать обработку событий нажатия на клавишу «ESCAPE» (необходимо для создания некоторых страниц, которые исключают взаимодействие с использованием клавиши «esc»). Каждый кард отрисовки интерфейса печатается шапка, показывающая информацию о доступных действиях с учётом заблокированных. Принимает в качестве параметров: функция-действие, вызываемая каждый раз, когда обновляется состояние ввода, для каждой итерации цикла - «action», максимальное допустимое смещение курсора по двум направлениям - «tuple_t max», значение отвечающее за отключение обработки нажатия на клавишу «возврата» - «back_use», входной параметр для функции «action» - «void *param». Возвращает результат выполнения функции «action».

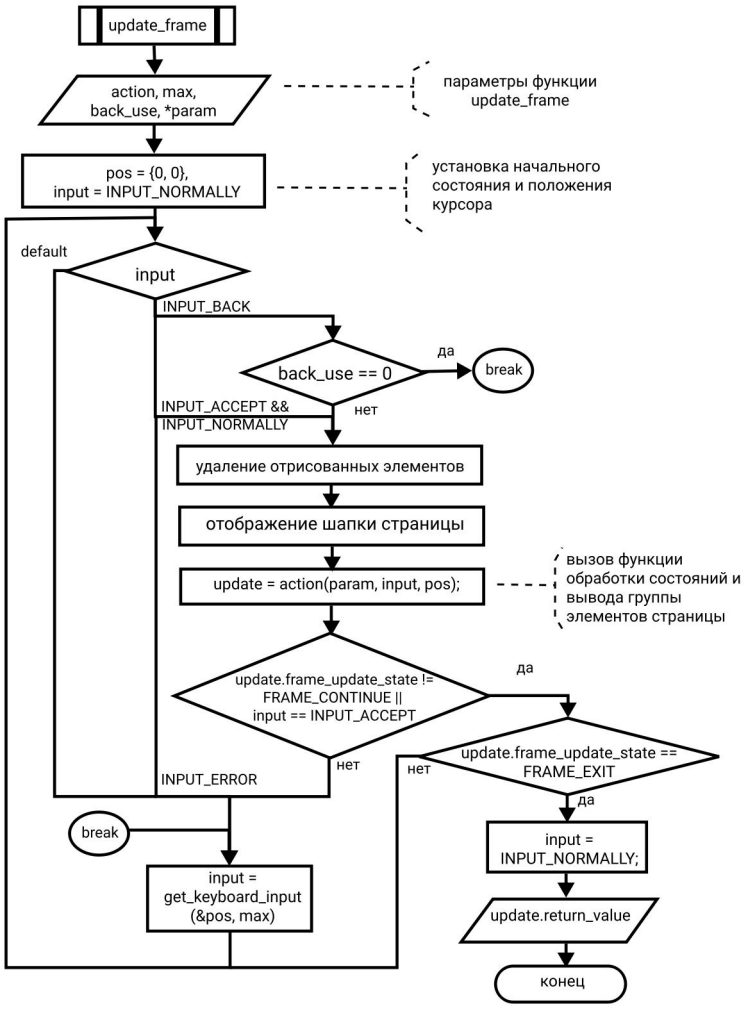

Рисунок 10 — Блок-схема функции загрузки страниц

Каждая функция формы gupdate_t(«название_функции»)(void* args, kaction_t action, tuple_t pos) — функция обработки состояний типа «kaction_t action», передаваемых в качестве параметров из функции загрузки отрисовки страницы, и формирования на основе их систему из элементов для вывода в окно консоли. Возвращает соответствующее значение структурного типа gupdate_t при каждом моменте отображении: состояние кадра предназначенный для функции update_frame (состояние определяет, что делать с циклом отрисовки: закрыть, перезапустить или продолжить) и указатель на значение полученное в результате обработки (указатель может хранить адрес до определённого значения или быть пустым). Принимает в качестве параметров: входные значения для обработки - «void* args», состояние ввода - «kaction_t action», положение курсора - «tuple_t pos». Для связи работы страниц между собой может использоваться вложенная загрузка дополнительных страниц внутри функции.

Входные и итоговые значения функции имеют тип пустого указателя, чтобы можно было реализовать обобщённое описание поведения функции, которые будут вызываться для объектов разных типов. Чтобы вернуться к нужному типу применяется оператор явного приведение типов данных.

Для вывода элементов на экран применяется стандартная функция «printf», но предусматриваются составные отображаемые элементы, вывод которых стоит организовать в виде отдельных функций. Такими функциями будут являться «draw_field» и «draw_list»:

void draw_list(unsigned int cursor, dir_t* param, int begin, int end) — функция для отображения элементов списка из массива «param» путём индексации с использование цикла. Параметры «begin» и «end» необходимы для ограничения видимости элементов (отображаются элементы с индексом от значения «begin» до «end»). Функция при отрисовке пунктов учитывает положение курсора, используя значение «cursor» и показывает его путём закрашивания установленным цветом. Также функция учитывает ошибку

выхода за пределы массива (вместо элемента печатается конструкция со значением «пусто»).

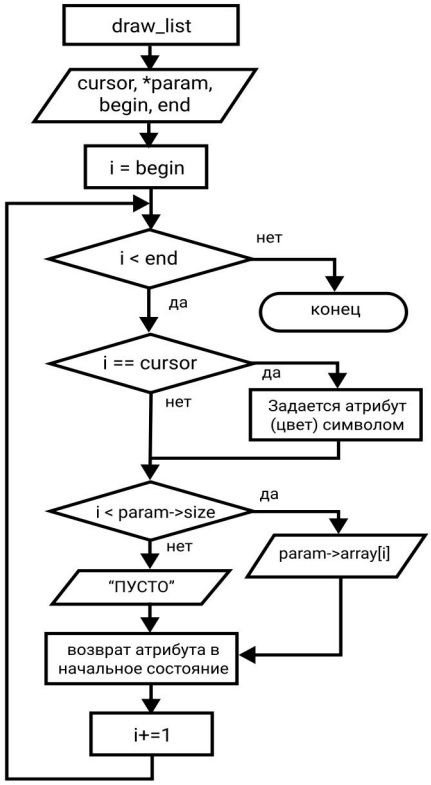

Рисунок 11 — Блок-схема функции печати списка элементов

void draw_field(tuple_t pos, field_t* param) — функция для отображения игрового поля из массива «param» (используется конструкция с вложенным циклом для обработки линейного множества в виде матрицы, поля путём индексации массива по формуле - i=y*size+x). Функция при отображение клеток учитывает положение курсора с помощью параметра «pos» и показывает его путём закрашивания установленным цветом. Перед полем печатается информация о координатах курсора. Для демонстрации сведения об статусе той или иной клетки, в процессе обработки массива меняется атрибут символа (цвет клетки), после чего печатается группа символом, отвечающих за представление клетки в поле, затем в конце значение атрибута возвращается в начальное состояние.

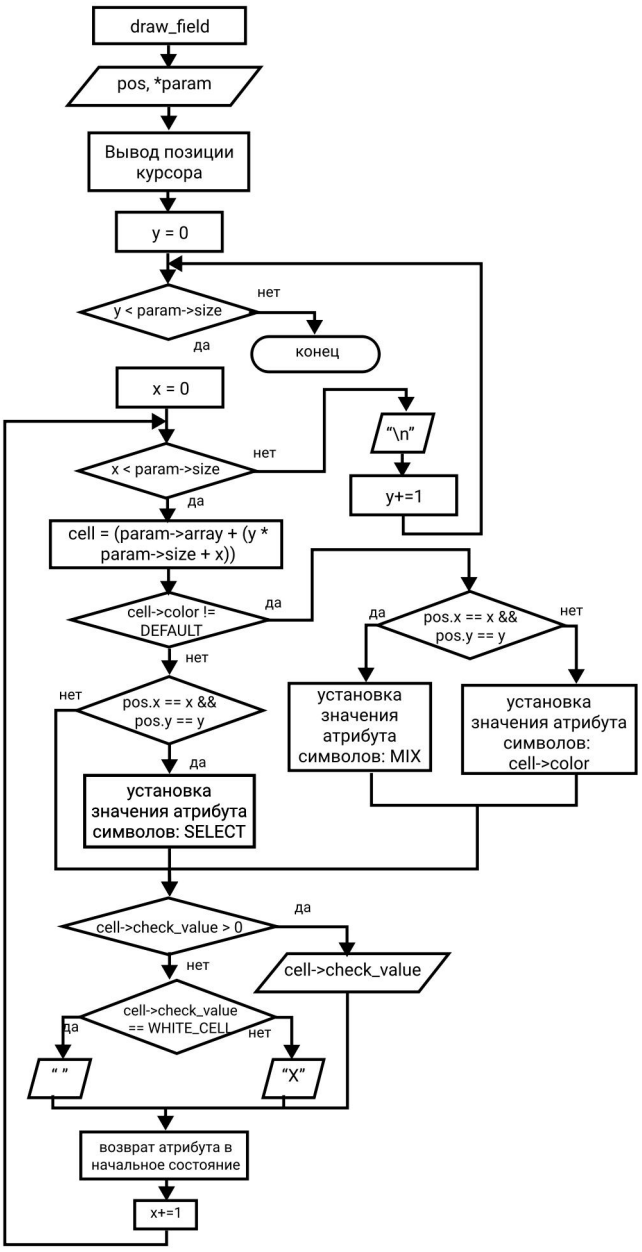

Рисунок 12 — Блок-схема функции печати игрового поля

Для изменения цвета объектов используется функция «SetConsoleTextAttribute» — задаёт атрибуты символов записываемых в буфер экрана консоли (функция влияет на текст, записанный после вызова функции), принимает в качестве параметра: дескриптор буфера экрана консоли («HANDLE hConsoleOutput») и значение атрибута - кода цвета («WORD wAttributes»). Для того чтобы извлечь дескриптор для стандартного устройства вывода используется функция «GetStdHandle», параметром которой

выступает «STD_OUTPUT_HANDLER» — обозначение стандартного устройства вывода (буфер экрана консоли).

2.2. Модуль организации игрового процесса головоломки

Основной частью программы выступает модуль, который реализует игровой процесс головоломки. Для отображения игрового интерфейса и поля реализуем функцию gupdate_t game_loop(void* args, kaction_t action, tuple_t pos) — отображает страницу, через которую складывается игровой процесс. При вызове функции внутри «update_frame» производится обработка событий ввода, отображение количества оставшихся попыток, видимости выбранной клетки, игрового поля через функцию draw_field и проверка состояние решения головоломки. В начале алгоритма происходит устанавливается оставшиеся количество попыток (с помощью использования статической переменной — сохраняет своё значение даже после выхода из блока, в котором она определена), определение состояния проверки решения (начальным значением = «STATE_RUNNING» — обычное состояние) и создание копия игрового поля. В качестве основного аргумента принимает принимает указатель на объект типа «field_t» (необходимо использовать оператор явного преобразования типа из «void» в «field_t») — хранит информацию о загруженном из файла игровом поле. В зависимости от значения состояния ввода совершаются дополнительные действия: при значении соответствующего состоянию «INPUT_ACCEPT» — совершается переключение (инвертирование) значения выбранной клетки, пересчёт видимости клеток (с помощью функции «set_axies») и проверка поля на наличие ошибок решения со стороны игрока (с помощью функции «check_field»); при значение «INPUT_BACK» (обработка клавиши «ESC» включено) — происходит установка значения оставшихся «жизней» в начальное положение, выход из функции путём закрытия цикла отрисовки в «update_frame» (для этого функция возвращает

значение состояния отрисовки кадра равному FRAME_EXIT — выйти из отрисовки данного кадра и вернуть значения). В случае проигрыша возвращает значение «0», в случае победы «1».

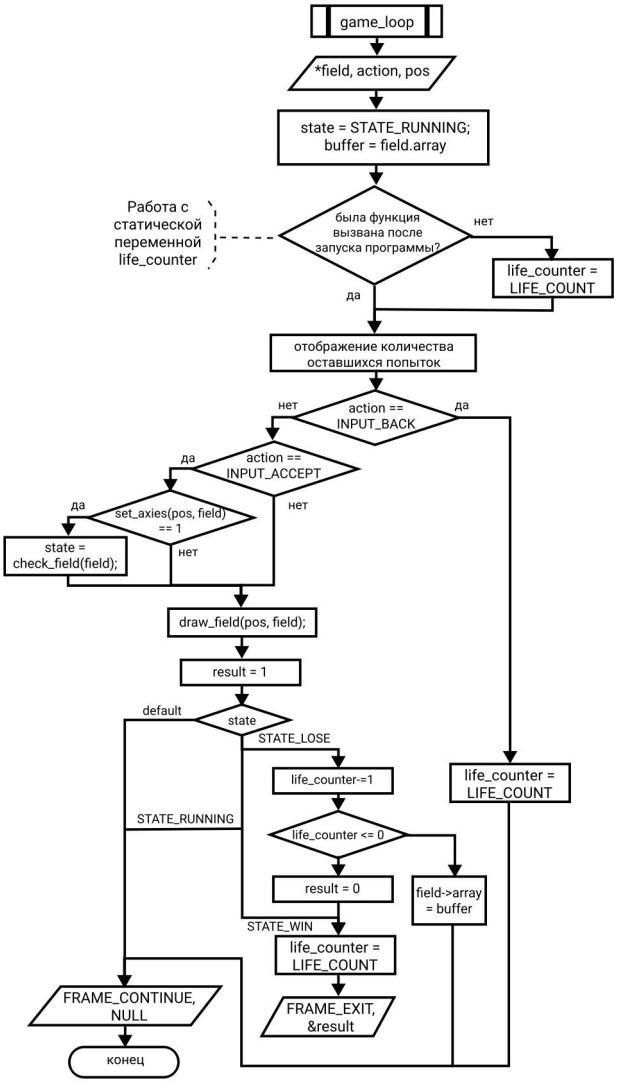

Рисунок 13 — Блок-схема функции определяющая игровой процесс головоломки

Для переключения значения клетки поля (белая ↔ чёрная) используется функция bool set_axies(tuple_t pos, field_t* ptr_param) — функция переключает значение клетки (инвертирует цвет клетки) по позиции выбранной клетки —

«pos», путём индексации массива «field_t* ptr_param» (значением индекса: i = y * size + x). Однако прежде чем изменять значение выбранной клетки происходит проверка следующих правил с использование составного условного выражения:

1. проверка установки чёрной метки в клетку со значением,
2. исключение касаний клеток с чёрной меткой друг с другом по горизонтали и вертикали.

Если одно из правил нарушено, то процесс функции завершается и возвращается значение ошибки — «false» , иначе производится смена значения клетки на соответствующее и изменение видимости клеток, принадлежащих двум ортогональным линиям (имеющих пересечение в месте изменённой клетки).

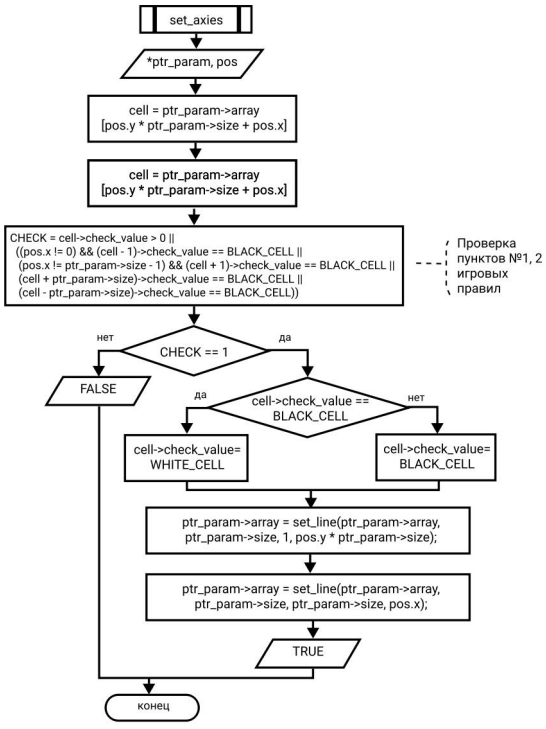

Рисунок 14 — Блок-схема функции для переключения цвета клетки

Чтобы изменить значения видимости клетки по двум направлениям используются двойной вызов функции cell_t* set_line(cell_t* ptr_param, int size, int step, int shift) — производит обновление значений видимости после

установки нового значения клетки поля путём индексации элементов массива «cell_t* ptr_param» через цикл. Значение «step» отвечает за шаг совершаемый при проходе по линии — расстояние между соседними клетками, принадлежащих одной линии (также отвечает за направление линии обновления). Параметр «shift» определяет на сколько клеток смещена линия, в которой обновляются клетки, от параллельной ей оси. Алгоритм функции следующий: инициализируется переменная для хранения индекса последней клетки со значением соответствующим «чёрной клетки» → открывается цикл с диапазоном [0; size], каждую итерацию которого идёт поиск «чёрной клетки» → если «чёрная клетка» найдена, то всем клеткам с индексами в интервале от значения последнего сохранённого положения до найденного присваивается новое значение видимости для выбранного направления — длина интервала; иначе пропуск действий → происходит перезапись значения последнего сохранённого индекса на новое. Функция возвращает указатель на массив с обновлёнными значениями видимости.

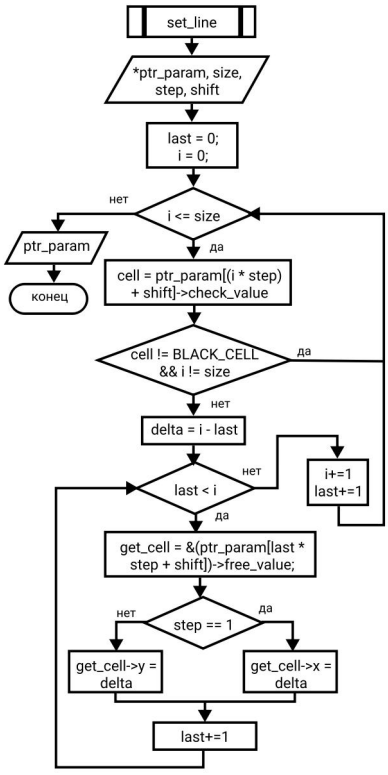

Рисунок 15 — Блок-схема функции для обновления значения видимости

Для всеобщей проверки правильности решения игровой сетки будет реализованна функция gstate_t check_field(field_t* param). В качестве аргумента принимает указатель на значение, описывающее параметры игрового поля (программный тип «field_t»); возвращает значение игрового состояния (обычное состояние, пользователь допустил ошибку во время игры или пользователь заполнил поле правильно). Функция в первую очередь совершает проверку целостности соединения между всеми «белыми» клетками. Для этого происходит поиск клетки со значением «белой» клетки, после нахождения первой из которых совершается процесс подсчёта соединённых с ней клеток того же значения: инициализируется массив для хранения информации о клетках, которые уже были учтены в процессе счёта, и вызывается функция «connection_check» — путём рекурсии (функция, вызывает сама себя) вычисляет количество соединённых друг с другом клеток. После происходит сравнение: если количество соединённых равно общему количеству «белых клеток» — все клетки соединены; иначе между клетками есть разрыв и функция проверки возвращает состояние «допуска ошибки». Если проверка на соединение прошла, то производится проверка нумерованных клеток, что их предустановленное значение не меньше суммы видимых клеток по двум направлениям, путём индексации элементов игрового поля. Если у какой-либо нумерованной клетки условие неверно, то флаг победы переключается в необходимое состояние, возвращаемое значение устанавливается равное «STATE_LOSE» — пользователь допустил ошибку во время игры, иначе «STATE_WIN» — игровое поле решено верно. В во время проверки каждой клетки у неё меняется значение параметра «цвета для отображения».

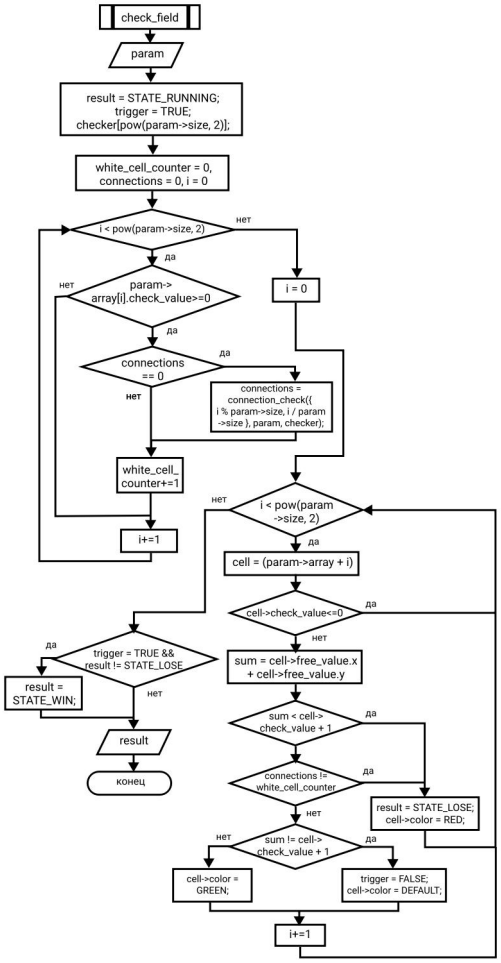

Рисунок 16 — Блок-схема функции проверки правильности решения головоломки

2.3. Модуль для работы с файловой системой

Для того чтобы организовать загрузку и сохранение игровых полей в файл формата «*.txt», реализуем функцию bool file_data(field_t* field, bool readonly) — функция для взаимодействия с файлами в предустановленной (объявленной с помощью директивы #define) директории. В зависимости от параметра (флага) «readonly» производит разные действия: при передаче в качестве значения «1» — производит чтение игрового поля из форматированного файла в переменную характеризующую текущее определённое игровое поле по указателю «field_t* field» (хранит имя поля - файла, чтобы можно было найти и прочитать выбранный файл с полем); при

значение «0» — производится запись данных игрового поля в указанный файл с помощью специального метода компоновки данных. При ошибки чтения или записи данных возвращает (файл не найден, занят другой программой и т.д) «0», если процесс завершился успешно — «1».

Чтобы прочитать доступные файлы из установленной папки, создадим функцию bool read_path(dir_t* param) — производит сбор информации о доступных файлах в указанной папке. В процессе извлечения данных происходит запись имён файлов (без расширения) в качестве строковых элементов массива «param». Возвращает состояние выполнения: «1» — процесс завершился без ошибок, «0» — в процессе чтения произошла ошибка. Функция игнорирует файлы, у которых расширение не текстового документа «*.txt». Основной частью функции является использование стороннего пакета «Dirent», позволяющий работать с указанной директорией системы, при помощи функции «readdir», которая возвращает указатель на прочитанный из открытого потока (папки) файл, который размещён в открытом пути. Если все файлы были прочитаны из открытого потока, то функция вернёт значение null.

Все программы, написанные на языке C, содержат основную функцию, которая должна иметь имя main. Именно с этой функции должна начинаться любая программа и, как правило, ею же и завершаться. Функции в исходном коде программы выполняют одну или несколько конкретных задач. Функция main может вызывать эти функции для выполнения соответствующих задач. Так в главной функции устанавливается локализация, создаётся на основе типа «field_t» новый объект (инициализируется переменная) с установленным по умолчанию начальным именем стартового поля. Также в случае отсутствия директории, где хранятся игровые поля, она будет создана. После чего происходит загрузка корневой страницы всей программы — главного меню программы с помощью функции «update_frame», в которую передаётся указатель на «mainmenu», устанавливаются границы курсора и флаг, отвечающий за наличие обработки клавиши «esc». В дальнейшем остальные

участки программы будут загружаться из страницы «mainmenu» подобным вышеописанным методом через «update_frame».

Весь проектируемый исходных код будет размещаться в разных файлах, используя концепцию многофайлового проекта — как только программы становятся больше, их следует разбивать на несколько файлов, в целях удобства и улучшения функциональности. В главном проектной файле «main.c» (содержит определение стартовой функции «int main(void)») будет подключаться заголовочный файл «kuromasu.h» с объявлением функций, определением типов данных (структур, перечислений) и указанием константных значений. Определение функций будет находится в отдельных файлах, которые будут отвечать за разные аспекты программы: «body.c» — хранит реализацию модулей для чтения и записи игровых полей в файл, чтения информации об файлах с уровнями из установленной директории, отображения основных объектов пользовательского интерфейса (списков, полей), обработки ввода с клавиатуры, обработки состояний ввода и цикличного отображения указанной страницы; «ui.c» — содержит определения методов, правил отрисовки страниц в зависимости от входного состояния; «gamelogic.c» — хранит определение главного модуля приложения, игровых механик, правил контроля решения головоломки.

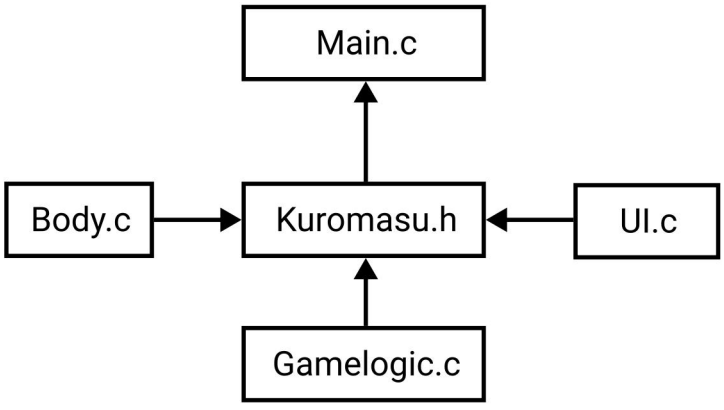

Рисунок 17 — Файловая структура проекта

Таблица 1 — Определение константных значений программы.

Название токена	Значение	Описание константы
Числовые значения для определения цвета клетки.		
WHITE_CELL	0	значение для обозначения белой (пустой) клетки.
BLACK_CELL	-1	значение для обозначения белой (пустой) клетки.
Значения для установки размеров поля.		
MIN_FIELD_SIZE	4	минимальный размер игровой сетки, допустимый для установки при создании уровня.
MAX_FIELD_SIZE	8	максимальный размер игровой сетки, допустимый для установки при создании уровня.
Внутриигровые параметры.		
LIFE_COUNT	3	максимально допустимое количество попыток, устанавливаемое при запуске игры.
Параметры для загрузки уровней из файлов.		
MAX_FILES_IN_DIR	20	максимально число читаемых файлов (сохранённых полей) в директории.
FILES_ON_PAGE	4	количество полей, которые отрисовываются на одной странице выбора уровня.
PATH_NAME	"data"	директория, в которой будут храниться все файлы с сформированными уровнями.

Таблица 2 — Определение собственных типов данных.

Описание определённых новых типов данных		
Псевдоним типа	Описание	Элементы набора; поля
Перечисления (именованные целочисленные константы) («enum»)		
KEY_CODE	Коды клавиш используемых для управления.	key_left = 75, key_right = 77, key_down = 80, key_up = 72, key_space = 32, key_esc = 27.
color_t	Коды атрибутов для маркировки цветов, которые используются при отрисовки интерфейса, значения соответствуют кодам цветов.	RED = 64, GREEN = 32, DEFAULT = 7, (прозрачный) SELECT = 112, (белый) MIX = 95. (фиолетовый)
gstate_t	Необходим для определения текущего внутриигрового состояния.	STATE_RUNNING - обычное состояние. STATE_LOSE - пользователь допустил ошибку во время игры. STATE_WIN - пользователь заполнил поле правильно.
kaction_t	Предназначен для обработки состояний нажатых клавиш пользователем.	INPUT_ERROR - пользователь нажал не верную клавишу. INPUT_NORMALLY - пользователь нажал клавишу для смены положение

		курсора. INPUT_ACCEPT - пользователь нажал клавишу для подтверждения. INPUT_BACK - пользователь нажал клавишу "возврата".
fupdate_t	предназначен для обработки состояния отрисовки интерфейса.	FRAME_CONTINUE - отрисовка продолжается. FRAME_RETURN - вернуться в начало текущей отрисовки . FRAME_EXIT - выйти из отрисовки данного кадра и вернуть значения.
Структуры данных (записи) («struct»)		
gupdate_t	Определяет состояние и действие для текущего кадра	frame_update_state - для хранения состояния. return_value - указатель на возвращаемое значение.
tuple_t	Хранение двух значений объединённых между собой (координат). Кортеж (упорядоченный набор фиксированной длины).	X, Y.
cell_t	описывает свойства клетки игрового поля	free_value - кортеж для хранение видимых клеток по двум осям. check_value - значение клетки.

		color - цвет для отрисовки.
dir_t	описывает значения и размер массива строковых элементов.	array - массив строк. size - размер массива.
field_t	Описывает характеристики игрового поля и хранит множество объектов типа cell_t	array - массив клеток типа cell_t. size - размер поля (массива). name - название поля.

Таблица 3 — Описание собственных функций.

Описание собственных функций		
Наименование	Назначение	Примечание
update_frame	Функция открывает цикл отрисовки страницы, используя для этого функцию action, и принимает пользовательские команды. Каждую итерацию внутреннего цикла происходит обработка введённой клавиши от пользователя, после удаляется прежний кард страницы и формируется новый основанный на введённом значении.	Возвращает указатель пустого типа - результат вызова функции action, который явно преобразуется в указатель нужного типа данных.
get_keyboard_input	обрабатывает вводимые команды от пользователя (клавиши). Изменяет положение курсора по двум	Возвращает состояние ввода: INPUT_ERROR -пользователь нажал не верную клавишу INPUT_NORMALLY -

	направлениям, используя указатель на значения координат.	пользователь нажал клавишу для смены положение курсора INPUT_ACCEPT - пользователь нажал клавишу для подтверждения INPUT_BACK - пользователь нажал клавишу "возврата"
draw_field	Отображает игровое поле	При отображение учитывает цветовые значения каждой клетки и положение курсора.
draw_list	Отображает элементы из списка, передаваемого в качестве аргумента функции.	Учитывает положение курсора и отображает его
set_line	Выполняет пересчёт значений выбранной оси. Направление устанавливается с помощью постоянной переменной ответственной за количество ячеек между двумя клетками. (сколько необходимо пропустить, чтобы от одной клетки перейти к следующей)	Возвращает указатель на массив ячеек поля с пересчитанными значениями видимости. Значение шага: 1. Проходка по линии параллельной оси ОУ = размер поля. 2. Проходка по линий параллельной оси ОХ = 1
print_rules	Выводит на экран правила игры	Ожидает ввода любой клавиши для закрытия
print_message	Выводит на экран сформированное сообщение	Ожидает ввода любой клавиши для закрытия
connection_check	Выполняет проверки целостности соединения	Возвращает количество соединённых клеток друг с

	белых клет ок между собой. Применяется рекурсивный алгоритм.	другом. Необходимо передать указатель на массив, в котором будет храниться информация о проверенных клетках.
check_field	Совершает полную проверку всего поля: проверяет каждую нумерованную клетку на соответствие видимому значению и значению установленного в клетке, и также проверяет наличие разрыва между белыми клетками.	Возвращает игровое состояние (победа, проигрыш, по умолчанию). Изменяет значение цвета клеток в соответствии с проверкой условия видимости и установленного значения.
set_axies	Переключает (инвертирует) значение у выбранной клетки и происходит переписывание значений видимости по двум ортогональным линиям, пересекающихся в точке положения курсора.	Возвращает ошибку, если были нарушены пункты № 1, 2 проверки. Для перезаписи значений используются двойной вызов функции set_line с установкой необходимого шага.
file_data	В зависимости от установленного флага совершает следующие операции: Чтение данных из выбранного файла в структурную переменную игрового поля. Производит запись в	Возвращает ошибку при записи или чтении из файла (файла для чтения нет и т.д.)

	созданный, существующий файл значений из переменной игрового поля по сформированному формату.	
read_path	Считывает все доступные файлы с игровыми полями в директории data в переменную хранящую массив значений имён полей.	Возвращает ошибку при проблеме чтений файлов в директории (такой директории нет).
Функции отрисовки одного кадра страницы и обработки генерируемых событий ввода, получаемых из get_keyboard_input. Для запуска необходимо передать в качестве параметра указатель на нужную функцию в update_frame.		
set_cell_value	Отображает страницу для установки значения клетки при редактировании	Возвращает новое значение для клетки, превращает её в нумерованную.
create_field	Отображает страницу выбора размера уровня и производит создание игрового поля.	Возвращает указатель на созданное поле выбранного размера.
set_field_values	Отображает страницу редактирования поля, где на отображаемом с помощью вызова функции draw_field поле необходимо выставить значения для каждой клетки.	При событии нажатия кнопки «ESCAPE» открывается диалоговое окно (с помощью функции update_frame, в которую передаётся указатель на функцию dialog_box), по результату которого происходит выбор дальнейших действий: продолжить редактирование, покинуть процесс

		редактирования без сохранения или завершить процесс редактирования с сохранением.
dialog_box	Отображает страницу диалогового окна, в котором пользователь должен выбрать один из установленных пунктов дальнейших действий (передаются в качестве аргумента функции, представляют из себя массив строк).	Возвращает номер выбранного пользователем пункта. Для отображения выбираемых элементов использует функцию draw_list.
load_file	Отображает страницу выбора сохранённого поля из списка доступных.	Возвращает имя выбранного поля. Массив доступных элементов формируется при помощи функции read_path, а печать пунктов при помощи draw_list.
mainmenu	Главный узел программы, отображает страницу главного меню игры.	
settings	Отображает страницу редактора.	Возвращает указатель на объект игрового поля.
game_loop	Отображает страницу, через которую складывается игровой процесс. При вызове функции производится обработка событий ввода, и	Возвращает указатель булевого типа. Если указатель пуст, то это означение пользователь остановил процесс игры и вышел в

	отображение поля через функцию draw_field. При событии «INPUT_BACK» пользователь покидает процесс решения, при событии «INPUT_ACCEPT» - вызывается функция установки значения чёрной клетки и происходит проверка поля на наличии ошибок.	главное меню, иначе: истина - победа игрока, ложь - проигрыш.

3. Тестирование программы

При запуске программы пользователя встречает страница главного меню игрового меню, предоставляющая выбор дальнейших разделов. Также ставиться шапка, указывающая активные клавиши для управления (данная система элементов отображается при отрисовке всех страниц).

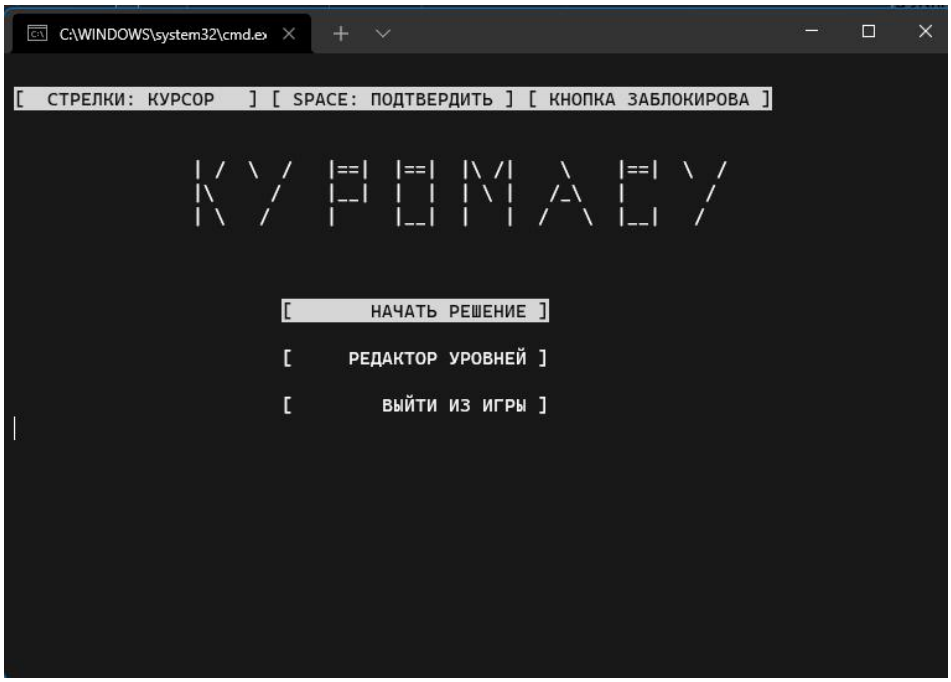

Рисунок 18 — Главное меню (корневая страница)

Используя стрелки на клавиатуре пользователь может определить следующую загруженную страницу. При нажатии на «Начать решение» происходит загрузка дочерней страницы, отвечающей за игровой процесс головоломки. В данном разделе программы отображается необходимая информация: видимость текущей выбранной ячейки поля, позиция курсора относительно начала координат (левый верхний угол: ox - вправо, oy - вниз), игровое поле представляющий множество клеток с установленными или пропущенными числовыми значениями, количество оставшихся попыток для решения. С помощью стрелок происходит перемещение курсора по полю, пробел — переключение значения пустых клеток.

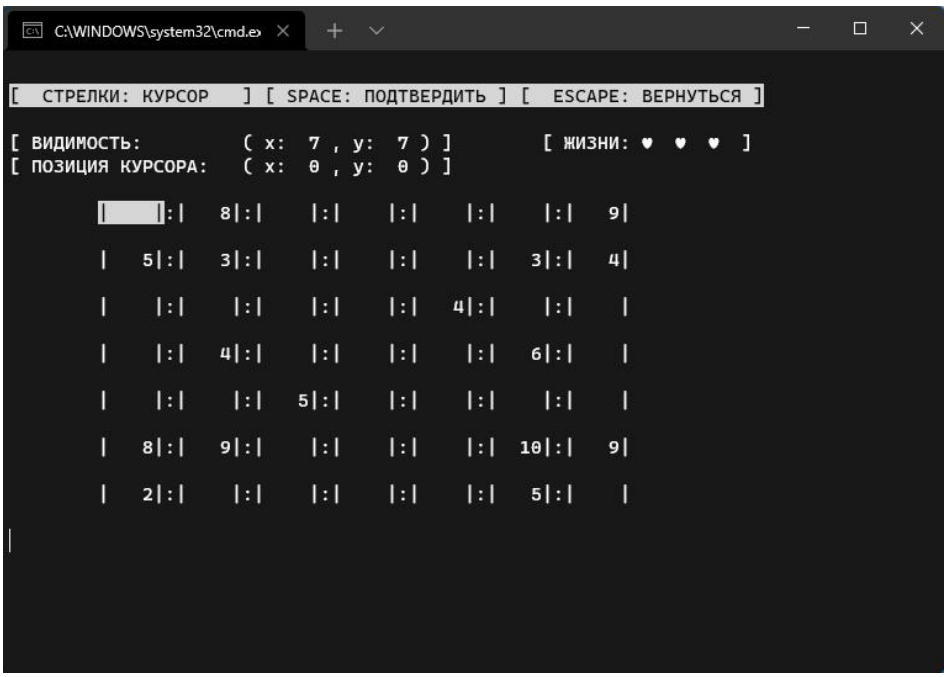

Рисунок 19 — Процесс решения головоломки

Если при запуске невозможно загрузить выбранное поле из файла (не соответствие формата, или файл был повреждён, удалён, или занят в другой программе), пользователю будет показано предупреждение.

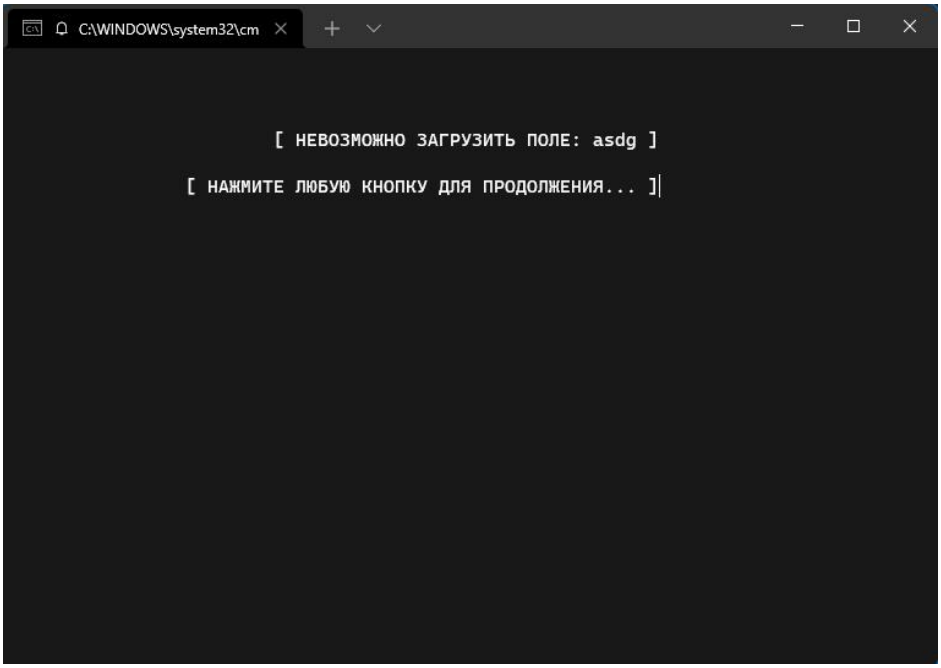

Рисунок 20 — Ошибка загрузки игрового уровня

Если видимость клетки ограниченна верно, то цвет её заднего фона закрашивается зелёным цветом (цвет можно изменить в программе, для этого необходимо изменить значение константы ответственной за данное представление).

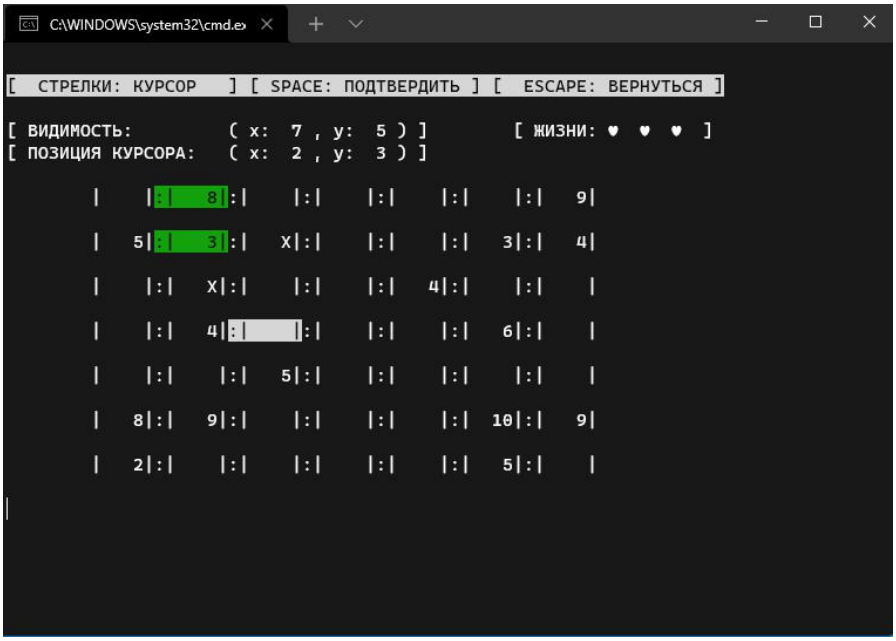

Рисунок 21 — Демонстрация подсказки «правильного ограничения видимости»

В случае верного решения игровой сетки или поражения страница сменится на соответствующую.

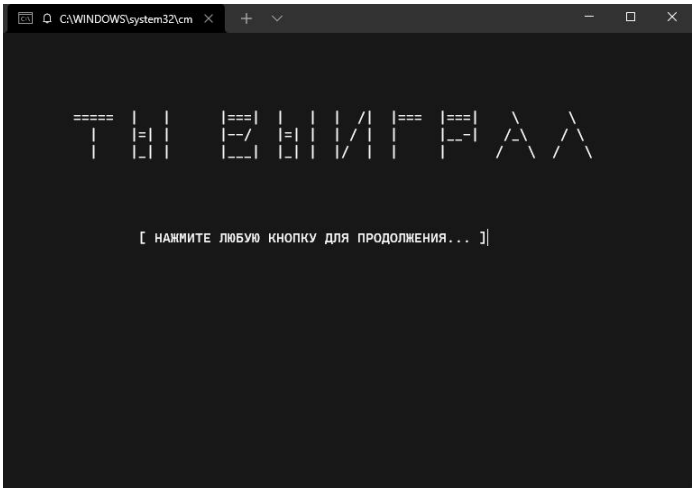

Рисунок 22 — Страница победы пользователя

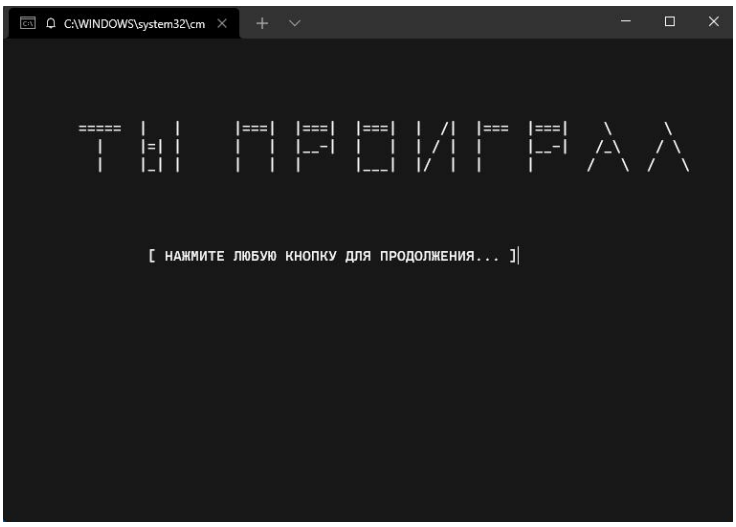

Рисунок 23 — Страница поражения пользователя

Если модуль контроля правильности заполнения игрового поля выявит ошибку, то пользователю будут продемонстрированы место, где она была совершена. Также будут уменьшено количество «жизней».

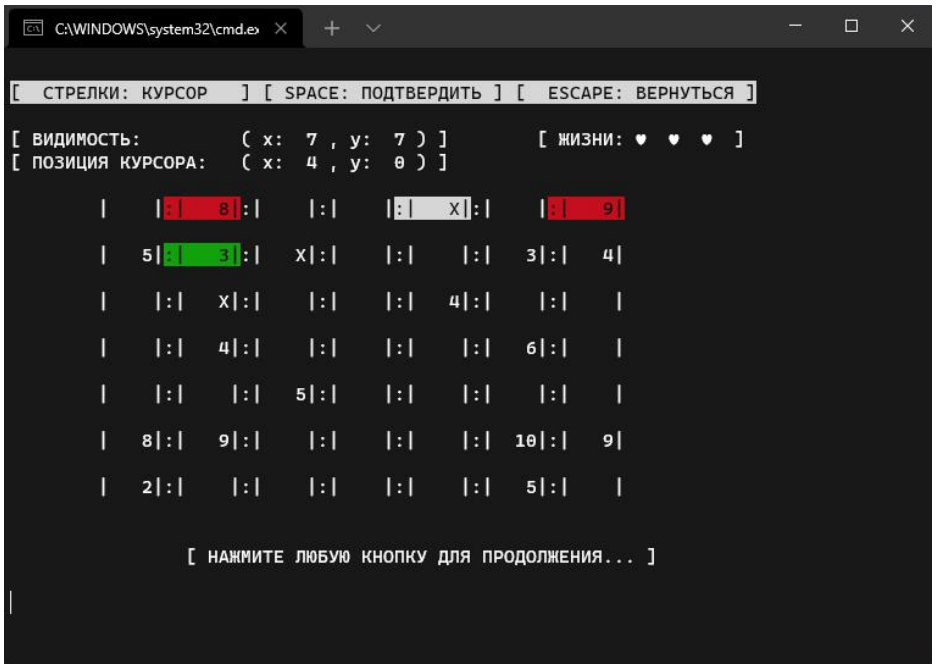

Рисунок 24 — Демонстрация пользователю место допущения ошибки

Если пользователю необходимо завершить процесс игры, то для этого, нажав на клавишу «esc», откроется диалоговая страница, в которой можно выбрать следующие действия: продолжить игру или завершить решение и выйти в главное меню.

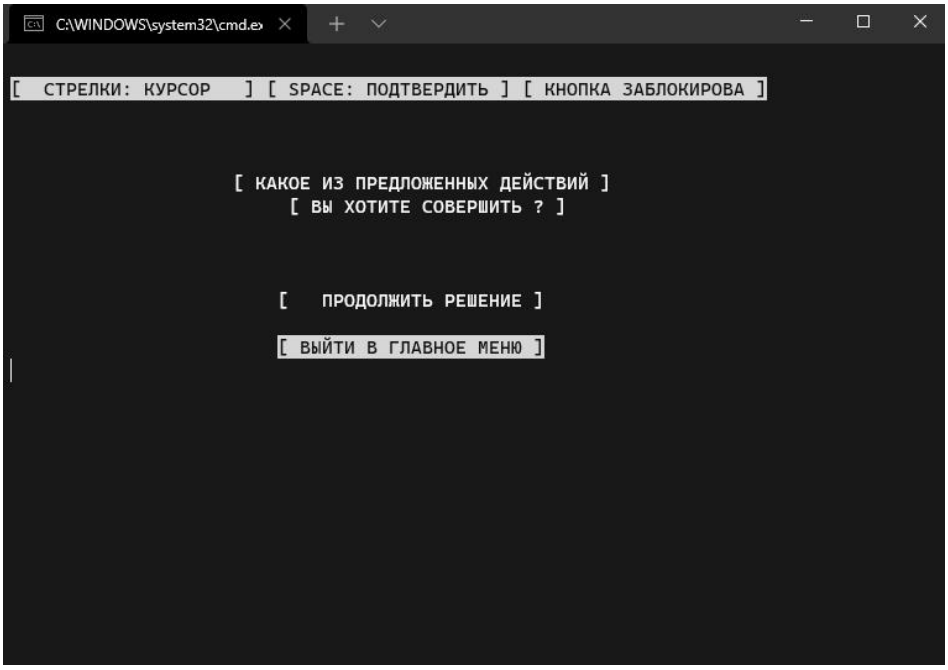

Рисунок 25 — Диалоговая страница, открывающаяся при попытке покинуть приложение

При выборе в главном меню пункта «Редактор уровней» откроется страница, которая предоставляет выбор: начать создание нового собственного уровня головоломки, загрузить новый уровень из установленного каталога, посмотреть правила решения головоломки «Куромасу» или же вернуться на корневую страницу программы. В качестве доступной информации отображается текущее выбранное поле.

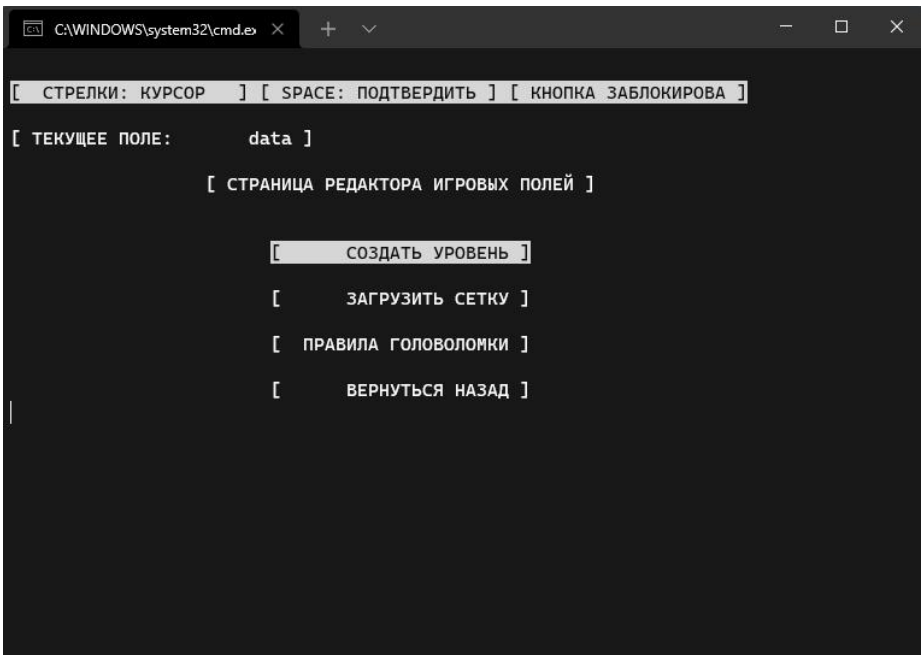

Рисунок 26 — Страница доступных программных инструментов

При выборе пункта «создать уровень» пользователю будет предложен выбор размера создаваемой сетки. Чтобы установить значение используются клавиши: «стрелка вправо» — увеличить на единицу, «стрелка влево» — уменьшить на единицу. (максимальное и минимально значение выставляется в коде программы, при помощи соответствующих констант).

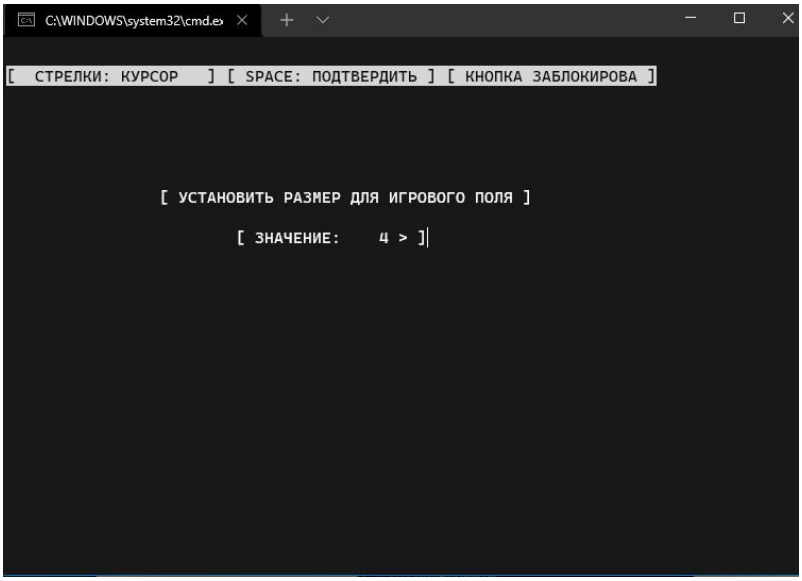

Рисунок 27 — Страница доступных программных инструментов

После подтверждения размера, открывается страница редактирования поля. При помощи клавиш «стрелок» выбирается нужная клетка, при нажатии на пробел открывается страница выбора значения клетки.

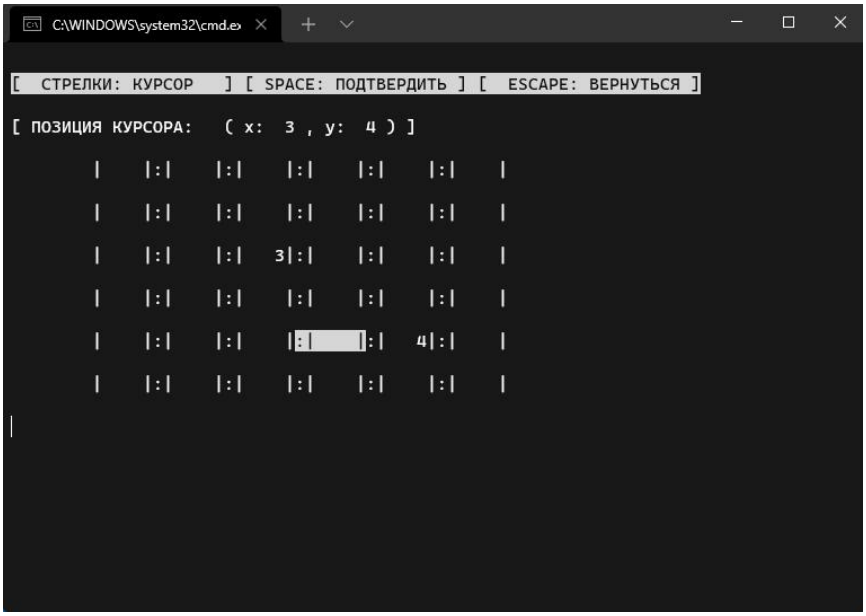

Рисунок 28 — Страница редактирования уровня

Процесс установки значения для клетки аналогичен процессу установки размера игровой сетки.

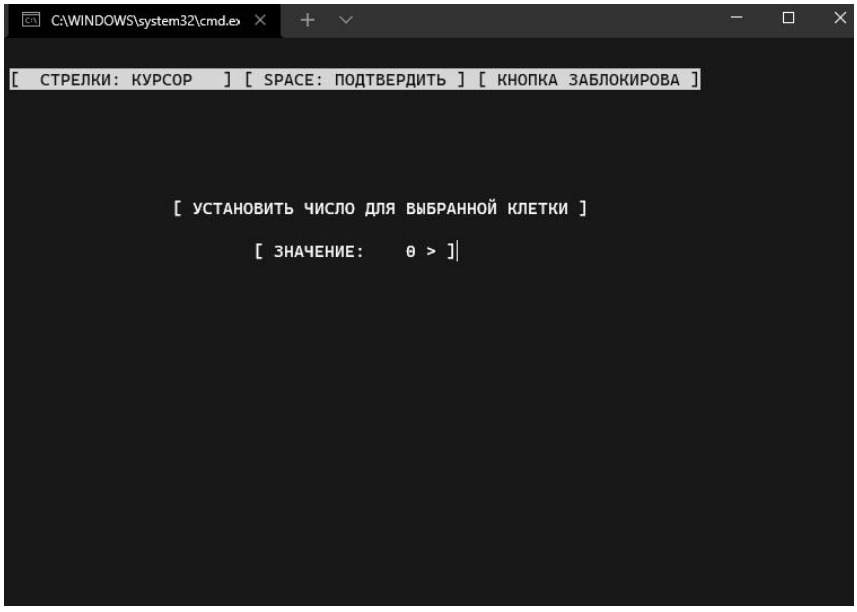

Рисунок 29 — Страница установки значения для клетки игрового поля

При нажатии «esc» в процессе редактирования, откроется диалоговая страница, предлагающая вернуться на страницу редактирования, выйти без сохранения уровня или сохранить поле в файл.

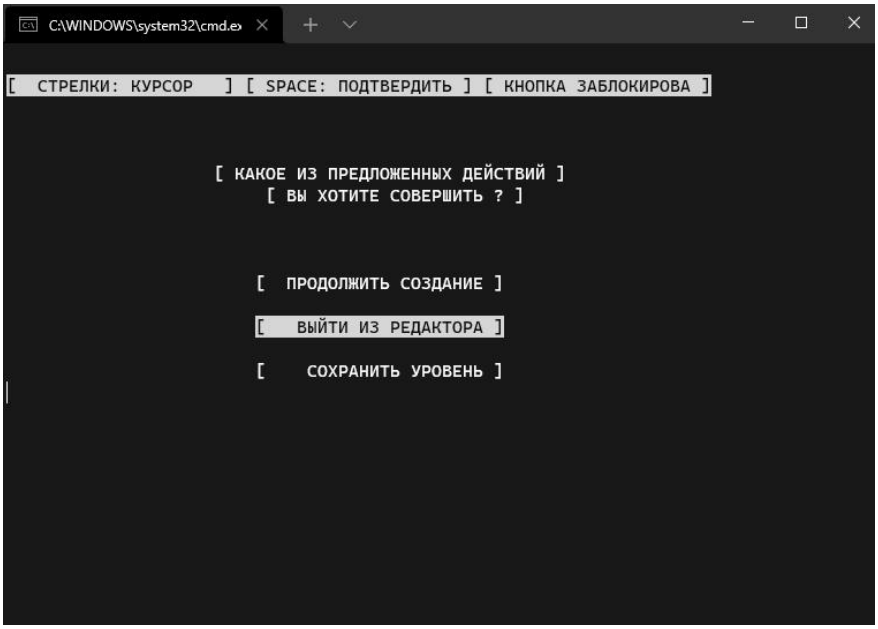

Рисунок 30 — Диалоговая страница выбора действия с отредактированным уровнем

Если пользователь выбрал сохранить созданный уровень, то ему будет предложено ввести имя для уровня (файла).

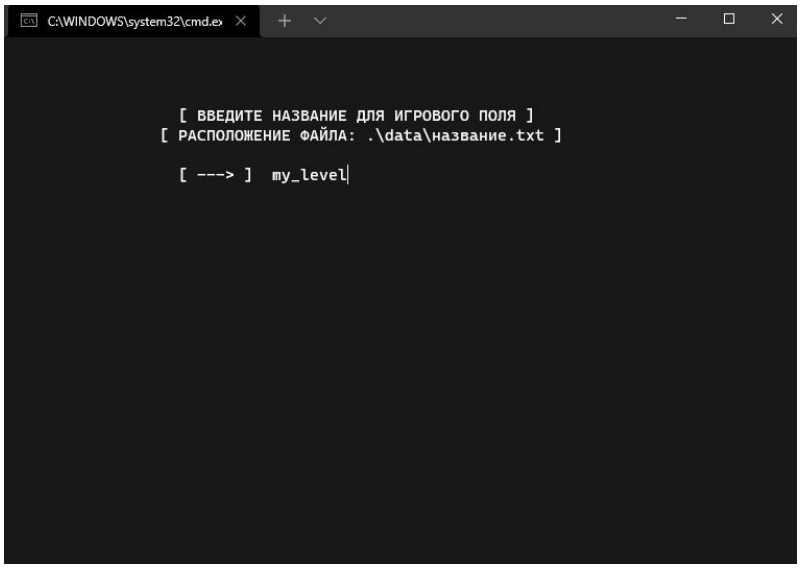

Рисунок 31 — Установка имени игрового поля

Если было введено имя несоответствующего формата или допущена ошибка в процессе редактирования (пустое поле), то пользователю будет выведено оповещение об ошибке.

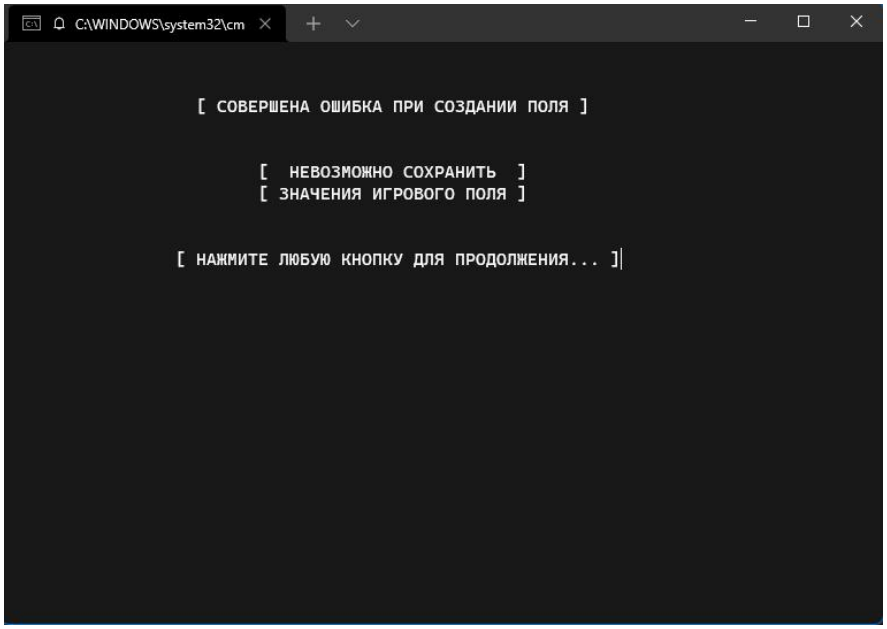

Рисунок 32 — Ошибка при сохранение созданного уровня

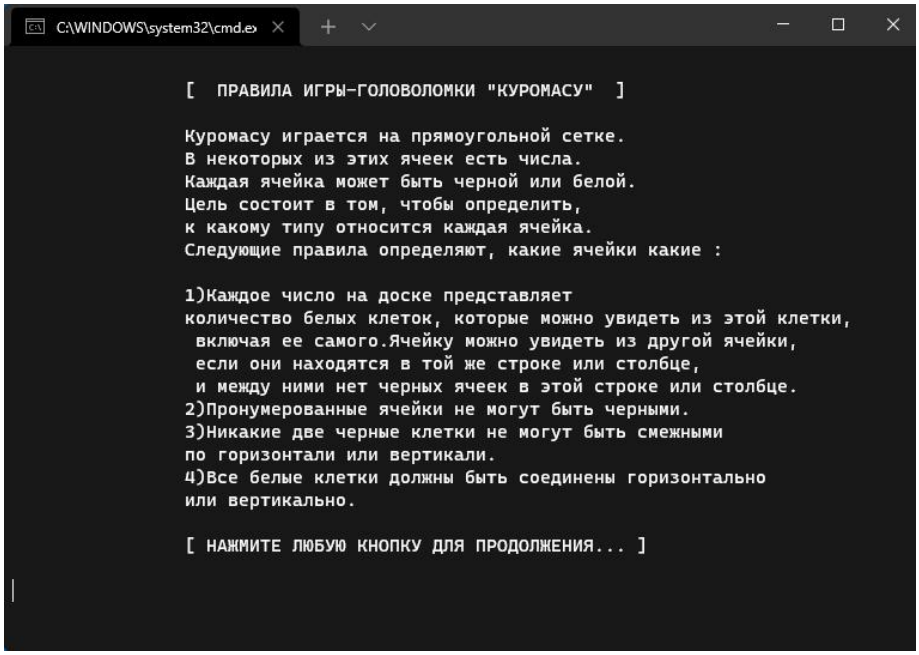

Рисунок 33 — Страница игровых правил головоломки

Если пользователю необходимо загрузить сохранённое поле, то он может перейти на соответствующий раздел из страницы «программных инструментов». Пользователю будут продемонстрированы доступные игровые поля из установленной папки (файлы с расширение текстового документа). Чтобы переключиться между страницами каталога с полями, используются клавиши «вправо» и «влево», а чтобы выбрать уровень на текущей странице, используются клавиши «вниз» и «вверх».

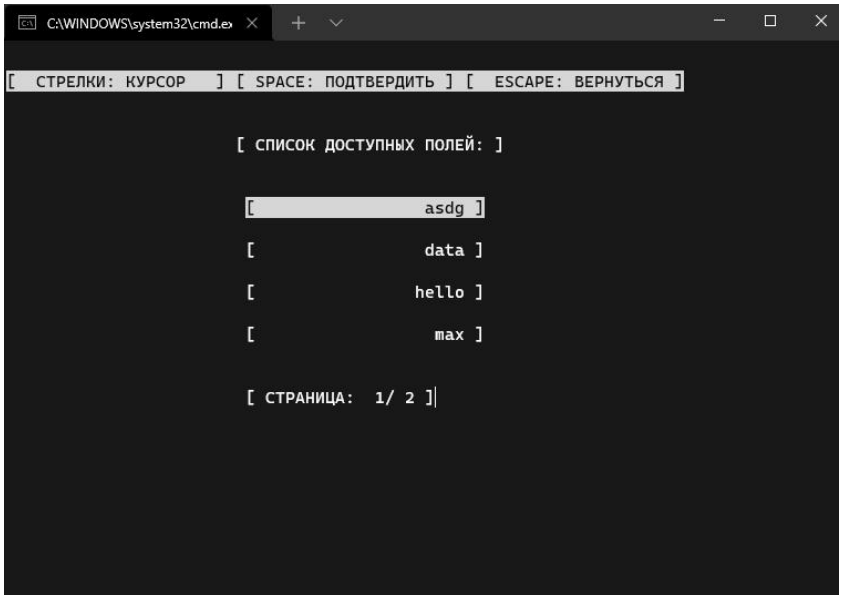

Рисунок 34 — Страница выбора игровой сетки из ранее сохранённых

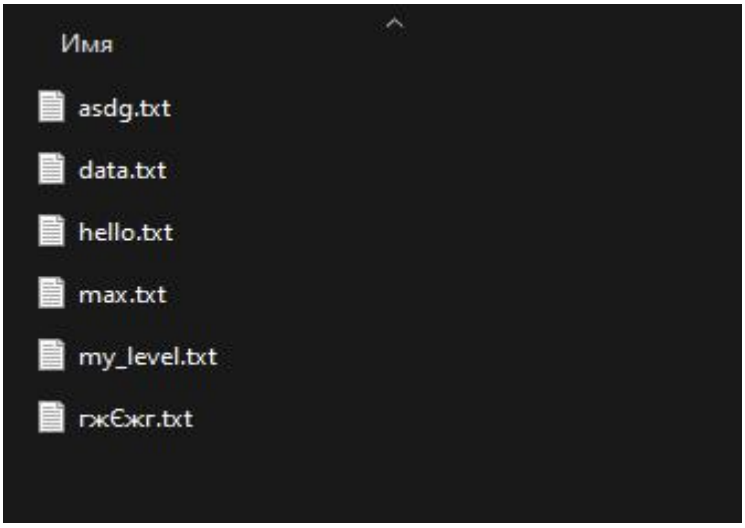

Рисунок 35 — Содержимое каталога, хранящего файлы с информацией о уровнях

При выборе уровня из каталога, пользователю будет предложено установить выбранное поле в качестве основного, совершить редактирование или вернуться на страницу выбора сохранённых уровней.

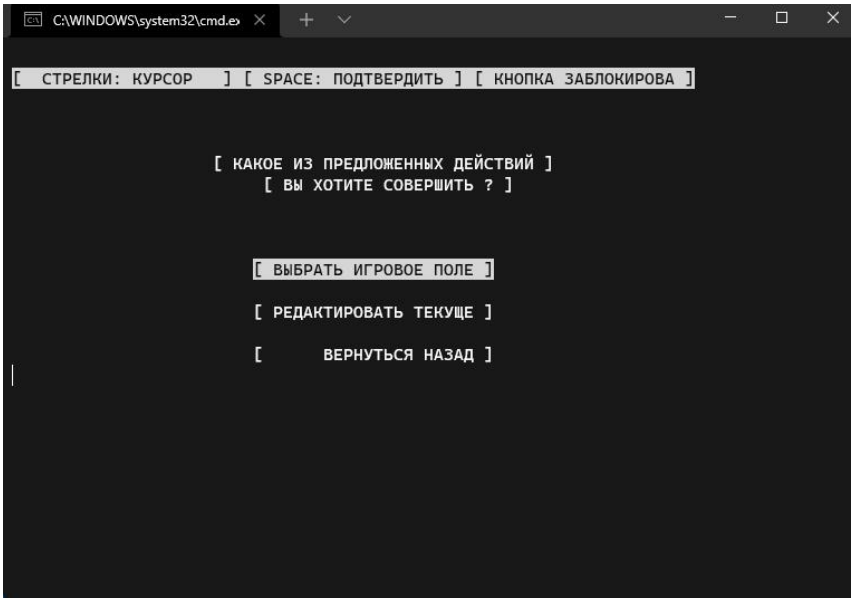

Рисунок 36 — Диалоговая страница определения действия с выбранным уровнем

Заключение

В ходе курсового проектирования на языке С с использованием среды разработки Visual Studio 2019 была реализована программа - игра в жанре головоломка «Куромасу».

Разработанная программа должна предоставлять пользователю систему с удобным пользовательским интерфейсом для решения головоломки, организовать контроль процесса заполнения выставленной игровой сетки. В программе был создан инструмент для редактирования уровней. Реализованна смена текущего выбранного игрового поля, путём загрузки из форматированного текстового файла.

Все требования к программе выполнены успешно. Реализованы следующие аспекты: обработка команд со стороны пользователя в виде состояний нажатия на установленные управляющие клавиши, отображение информации в формате страниц, составлен и реализован алгоритм контроля процесса решения головоломки пользователем с помощью отдельных установленных функций. Дополнительно разработан инструмент для создания и редактирования собственных игровых сеток.

Список литературы

1. Курипта О.В. Основы программирования и алгоритмизации: практикум / О.В.Курипта, О.В. Минакова, Д.К. Проскурин; Воронежский ГАСУ. – Воронеж, 2015.

2. Подбельский В.В., Фомин С.С. Программирование на языке Си. М.: ФиС.

3. Неземский В.И. Процедуры и функции [Электронный ресурс]: методические указания/ Неземский В.И., Орешкина О.А.— Электрон. текстовые данные.— М.: Московский государственный технический университет имени Н.Э. Баумана, 2009.

4. Баженова И.Ю. Введение в программирование [Электронный ресурс]: учебное пособие/ Баженова И.Ю., Сухомлин В.А.— Электрон. Текстовые данные.— М.: БИНОМ. Лаборатория знаний, Интернет- Университет Информационных Технологий (ИНТУИТ), 2007.

5. Шишкин А.Д. Программирование на языке Си [Электронный ресурс]: учебное пособие/ Шишкин А.Д.— Электрон. текстовые данные.— СПб.: Российский государственный гидрометеорологический университет, 2003.

6. Солдатенко И.С. Основы программирования на языке Си: учебное

7. пособие/ Тверской государственный университет, 2017.

8. Конова Е. А. Алгоритмы и программы. Язык С++: учебное пособие

9. 2-е изд./ Поллак Г. А — СПб.: Издательство «Лань», 2017.

10. Столяров А.В. Введение в язык С++: учебное пособие 4-е изд. – М.:МАКС. Пресс, 2018.

11. Васильев А. Н. Самоучитель С++ с примерами и задачами. 4-е издание (переработанное). — СПб.: Наука и Техника, 2016.

12. Литвиненко Н.А. Технология программирования на С++. — СПб.: БХВ-Петербург, 2010.

Приложение

1. kuromasu.h

```
#pragma once//ver 1.5
#define _CRT_SECURE_NO_WARNINGS
/*
* подключаемые заголовочные файлы
*/

#include <stdio.h>
#include <locale.h>
#include <math.h>
#include <malloc.h>

#include <stdlib.h>
#include <Windows.h>
#include <string.h>
#include <dirent.h>

/*
* константы для настройки работы приложения
*/

#define WHITE_CELL 0
#define BLACK_CELL -1
#define LIFE_COUNT 3
#define MAX_FILES_IN_DIR 20
#define MIN_FIELD_SIZE 4
#define MAX_FIELD_SIZE 8
#define FILES_ON_PAGE 4
#define clear_frame() system("cls")
#define path_name "data"

/*
* константы для отрисовки баннеров
*/

#define WIN_LABEL "\t=====  |   |      |===|  |   |  |  /|  |===  |===|
\\      \\ \n\t  |    |=| |      |--/   |=| |  | / |  |     |__-|   /_\\
/ \\ \n\t  |    |_| |      |___|  |_| |  |/  |  |     |      /   \\  /
\\ \n"
#define LOSE_LABEL  "\t=====  |   |      |===|  |===|  |===|  |  /|  |===
|===|    \\      \\ \n\t  |    |=| |      |   |  |__-|  |   |  | / |  |
|__-|   /_\\    / \\\n\t  |    |_| |      |   |  |     |___|  |/  |  |
|      /   \\  /   \\ \n"
#define NAME_LABEL  "\t\t| /  \\  /   |==|  |==|  |\\ /|    \\    |==|
\\  /\n\t\t|\\    /    |__|  |  |  | \\ |   /_\\   |       /\n\t\t| \\
/    |     |__|  |   |  /   \\  |__|   /"
```

```
/*
* создание типа KEY_CODE для работы с клавишами
* значения присваиваются в соответствии с кодами клавиатурных символов
*/

//typedef enum
//{
//    key_a = 97,
//    key_d = 100,
//    key_s = 115,
//    key_w = 119,
//    key_space = 32,
//    key_esc = 27
//} KEY_CODE;

typedef enum
{
      key_left = 75,
      key_right = 77,
      key_down = 80,
      key_up = 72,
      key_space = 32,
      key_esc = 27
} KEY_CODE;


/*
* определение типа color_t для маркировки цветов, которые используются
при отрисовки интерфейса
* значения соответствуют кодам цветов.
*/

typedef enum
{
      RED = 64,
      GREEN = 32,
      DEFAULT = 7,
      SELECT = 112,
      MIX = 95,
} color_t;

/*
* тип gstate_t необходим для работы с текущим внутриигровым состоянием:
*     STATE_RUNNING - обычное состояние
*     STATE_LOSE - пользователь допустил ошибку во время игры
*     STATE_WIN - пользователь заполнил поле правильно
*/
typedef enum { STATE_RUNNING, STATE_WIN, STATE_LOSE } gstate_t;

/*
```

```
* тип kaction_t необходим для обработки состояний нажатых клавишь
пользователем:
*     INPUT_ERROR - пользователь нажал не верную клавишу
*     INPUT_NORMALLY - пользователь нажал клавишу для смены положение
курсора
*     INPUT_ACCEPT - пользователь нажал клавишу для потверждения
*     INPUT_BACK - пользователь нажал клавишу "возврата"
*/
typedef enum { INPUT_ERROR, INPUT_NORMALLY, INPUT_ACCEPT, INPUT_BACK }
kaction_t;
/*
* тип fupdate_t предназначен для обработки состояния отрисовки интерфейса
*     FRAME_CONTINUE - отрисовка продолжается
*     FRAME_RETURN - вернуться в начало отрисовки
*     FRAME_EXIT - выйти из отрисовки данного кадра и вернуть значения
*/
typedef enum { FRAME_CONTINUE, FRAME_RETURN, FRAME_EXIT } fupdate_t;

typedef char name_t[261]; //строковый тип определенный для работы с
названиями
typedef _Bool bool; //опредение булевого типа

/*
* структура gupdate_t определяет состояние и действие для текущего кадра
*     frame_update_state - для хранения состояния
*     return_value - указатель на возвращаемое значение
*/

typedef struct gupdate_t
{
     fupdate_t frame_update_state;
     void* return_value;
} gupdate_t;
//gupdate_t gupdate_c(fupdate_t frame_update_state, void* return_value);
// конструктор для структуры gupdate_t


/*
* структура tuple_t хранение двух значений (координат)
*/

typedef struct tuple_t { unsigned int x, y; } tuple_t;
//tuple_t tuple_c(int x, int y); // конструктор для структуры tuple_t

/*
* структура cell_t описывает свойства клетки игрового поля
*     free_value - кортеж для хранение видимых клеток по двум осям
*     check_value - значение клетки
*     color - цвет для отрисовки
*/
```

```
typedef struct cell_t
{
	tuple_t free_value;
	int check_value;
	color_t color;
} cell_t;
//cell_t cell_c(int check, int free, color_t color);// конструктор для
структуры cell_t

/*
* структура dir_t описывает значения и размер массива строк
*	array - массив строк
*	size - размер массива
*/

typedef struct dir_t
{
	name_t* array;
	unsigned int size;
} dir_t;
//dir_t dir_c(name_t* dir, int size); // конструктор для структуры dir_t

/*
* структура field_t описывает игровое поле
*	array - массив клеток типа cell_t
*	size - размер поля (массива)
*	name - название поля
*/

typedef struct field_t
{
	cell_t* array;
	name_t name;
	unsigned int size;
}field_t;
//field_t field_c(cell_t* array, name_t name, int size); // конструктор
для структуры field_t

/*
* определение типа указателя на функцию, вызываемой при отрисовки кадра
страницы.
* функция - возвращает состояние fupdate_t и указатель на переменную
* args - указатель входное значение
* action - пользовательская команда
* pos - положение курсора
*/
typedef gupdate_t(*update_action_function)(void* args, kaction_t action,
tuple_t pos);

/*
* функция set_line - пересчет значений выбранной оси
```

```
* функция - возвращает указатель на массив ячеек поля
* ptr_param - указатель на массив ячеек поля
* size - длина линий
* step - шаг, необходимый чтобы из одной ячейки линий перейти в другую
* shift - смещение точки начала линии перерасчета, относительно начала
координат
*/
cell_t* set_line(cell_t* ptr_param, int size, int step, int shift);

/*
* функция draw_list - отрисовывает список
* cursor - индекс выбранного пункта
* param - список для отрисовки
* begin - индекс первого объекта для отрисовкаparam[]
* end - индекс последнего объекта для отрисовка
*/
void draw_list(unsigned int cursor, dir_t* param, int begin, int end);

/*
* функция draw_list - отрисовывает игровое поле
* pos - координаты курсора
* param - указатель на структуру, характеризующее поле
*/
void draw_field(tuple_t pos, field_t* param);

/*
* update_frame - функция открывает цикл отрисовки и сохраняет
пользовательские команды
*     возращает указатель пустого типа - результат вызова функции action,
который явно преобразуется в указаетль нужного типа данных
* action - указатель на функцию, которая вызывается каждый кадр (кадр
меняется при польовательском вводе) для отрисовки и обработки
*     пользовательских команд
* max - значения границ для курсора по двум осям
* back_use - true=клавиша escape будет обрабатываться в функции action;
false=игнорирование нажатие
* param - указатель на значение для передачи в функцию action в качестве
параметра
*/
void* update_frame(update_action_function action, tuple_t max,
     bool back_use, void* param);
/*
* print_rules - отрисовка правил игры
*/
void print_rules(void);
void print_messenge(char* str[3]);

int connection_check(tuple_t pos, field_t* field, bool* checker);

gupdate_t set_cell_value(void* args, kaction_t action, tuple_t pos); //
значения для выбранной клетки
```

```
gupdate_t create_field(void* args, kaction_t action, tuple_t pos); //
значение размера поля
gupdate_t set_field_values(void* args, kaction_t action, tuple_t pos); //
отрисовка поля для редактирования
gupdate_t dialog_box(void* args, kaction_t action, tuple_t pos); //
диалоговое окно

gupdate_t load_file(void* args, kaction_t action, tuple_t pos); //
доступные поля в папке data
gupdate_t mainmenu(void* args, kaction_t action, tuple_t pos);
gupdate_t settings(void* args, kaction_t action, tuple_t pos);
gupdate_t game_loop(void* args, kaction_t action, tuple_t pos);

/*
* функция check_field - выполняет проверку поля
*  возвращает ошибку
*  param - указатель на игровое поле
*/

gstate_t check_field(field_t* param);

/*
* функция get_keyboard_input - обрабатывает вводимые команды от
пользователя (клавиши).
*    возращает состояние ввода: тип kaction_t
*
*  pos - положение курсора.
*    max - границы для курсора
*/

kaction_t get_keyboard_input(tuple_t* pos, tuple_t max);

/*
* функция set_axies - переключает (инвертирует) значение у выбранной
клетки,
*    перезаписывает видимые клетки у кажкой клетки, которая пренадлежит
линий, в которой лежит выбранная клетка.
*    возвращает ошибку.
*  pos - положение курсора
*  ptr_param - указатель на игровое поле
*/

bool set_axies(tuple_t pos, field_t* ptr_param);

/*
* функция file_data - считывает поле из выбранного файла или записывает
из field в файл
*  возвращает ошибку
*  field - указатель, который будет хранить значения клеток поля
*    readonly - true=считавание в field из файла, false=считывание из
field в файл.
```

```
*/

bool file_data(field_t* field, bool readonly);

/*
* функция read_path - считывает все доступные файлы с игровыми полями в
директории data
*  возвращает ошибку
*  param - указатель, который будет хранить название файлов
*/

bool read_path(dir_t* param);
```

2. main.c

```
#include "src/kuromasu.h"

int main(void)
{
	setlocale(LC_ALL, "rus");

	char* setup_path = (char*)calloc(261, sizeof(char));
	sprintf(setup_path, "mkdir %s", PATH_NAME);
	system(setup_path);

	update_frame(mainmenu, (tuple_t) { 0, 3 }, FALSE, &(field_t){ .name
= "data" });

	return 0;
}
```

3. body.c

```
#include "kuromasu.h"

void* update_frame(update_action_function action, tuple_t max,
	bool back_use, void* param)
{
	kaction_t input = INPUT_NORMALLY;
	tuple_t pos = (tuple_t){ 0, 0 };
	HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);

	while (1)
	{
		switch (input)
		{
		case INPUT_BACK: if (!back_use) break;
		case INPUT_ACCEPT:
		case INPUT_NORMALLY:
			clear_frame();
			SetConsoleTextAttribute(console, SELECT);
```

```
                printf("\n[  СТРЕЛКИ: КУРСОР   ] [ SPACE: ПОДТВЕРДИТЬ ]
[ %18.18s ]\n\n",
                      (back_use) ? "ESCAPE: ВЕРНУТЬСЯ" : "КНОПКА
ЗАБЛОКИРОВАНА");
                SetConsoleTextAttribute(console, DEFAULT);

                gupdate_t update = action(param, input, pos);
                if (update.frame_update_state || input == INPUT_ACCEPT)
                {
                      input = INPUT_NORMALLY;
                      if (update.frame_update_state == FRAME_EXIT) return
update.return_value;
                      else continue;
                }
           case INPUT_ERROR: break;
           }
           input = get_keyboard_input(&pos, max);
     }
}

void draw_list(unsigned int cursor, dir_t* param, int begin, int end)
{
     HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
     for (unsigned int i = begin; i < end; i++)
     {
           if (cursor == i)SetConsoleTextAttribute(console, SELECT);
           printf("\n\t\t\t[ %20.20s ]\n", i < param->size ? *(param-
>array + i) : "ПУСТО");
           SetConsoleTextAttribute(console, DEFAULT);
     }
}

gupdate_t dialog_box(void* args, kaction_t action, tuple_t pos)
{
     dir_t* items = (dir_t*)args;
     if (action == INPUT_ACCEPT)
     {
           unsigned int select = (pos.y);
           return (gupdate_t) { FRAME_EXIT, & select };
     }

     printf("\n\n\t\t%38s\n\t\t%34s\n\n\n", "[ КАКОЕ ИЗ ПРЕДЛОЖЕННЫХ
ДЕЙСТВИЙ ]", "[ ВЫ ХОТИТЕ СОВЕРШИТЬ ? ]");
     draw_list(pos.y, items, 0, items->size);
     return (gupdate_t) { FRAME_CONTINUE, NULL };
}

void draw_field(tuple_t pos, field_t* param)
{
     cell_t* cell;
     HANDLE console = GetStdHandle(STD_OUTPUT_HANDLE);
```

```
	printf("[ %-18s ( x: %2d , y: %2d ) ]\n\n", "ПОЗИЦИЯ КУРСОРА:",
pos.x, pos.y);
	for (unsigned int y = 0; y < param->size; y++)
	{
		for (unsigned int x = 0; x < param->size; x++)
		{
			cell = (param->array + (y * param->size + x));
			if (cell->color != DEFAULT)
SetConsoleTextAttribute(console, (pos.x == x && pos.y == y) ? MIX : cell-
>color);
			else SetConsoleTextAttribute(console, (pos.x == x &&
pos.y == y) ? SELECT : DEFAULT);

			char shift = (x == 0) ? '\t' : ':';
			if (cell->check_value > 0)printf("%c|%4d|", shift, cell-
>check_value);
			else printf("%c|%4c|", shift, cell->check_value ==
WHITE_CELL ? ' ' : 'X');

			SetConsoleTextAttribute(console, DEFAULT);
		}
		printf("\n\n");
	}
}

void print_rules(void)
{
	clear_frame();
	printf("\n\t\t[  ПРАВИЛА ИГРЫ-ГОЛОВОЛОМКИ \"КУРОМАСУ\"  ]\n\n");
	printf("\t\tКуромасу играется на прямоугольной сетке. \n\t\tВ
некоторых из этих ячеек есть числа. \n\t\tКаждая ячейка может быть черной
или белой. \n\t\tЦель состоит в том, чтобы определить, \n\t\tк какому
типу относится каждая ячейка.\n\t\tСледующие правила определяют, какие
ячейки какие :\n\n\t\t1)Каждое число на доске представляет
\n\t\tколичество белых клеток, которые можно увидеть из этой
клетки,\n\t\t включая ее самого.Ячейку можно увидеть из другой
ячейки,\n\t\t если они находятся в той же строке или столбце,\n\t\t и
между ними нет черных ячеек в этой строке или
столбце.\n\t\t2)Пронумерованные ячейки не могут быть
черными.\n\t\t3)Никакие две черные клетки не могут быть смежными \n\t\tпо
горизонтали или вертикали.\n\t\t4)Все белые клетки должны быть соединены
горизонтально \n\t\tили вертикально.\n");
	printf("\n\t\t[ НАЖМИТЕ ЛЮБУЮ КНОПКУ ДЛЯ ПРОДОЛЖЕНИЯ... ]\n\n");
	getch();
}

void print_messenge(char* str[3])
{
	clear_frame();
	printf("\n\n\t\t%40s\n\n\n", str[0]);
```

```
	printf("\t\t\t%20s\n\t\t\t\t%20s\n\a\n\n\t\t[ НАЖМИТЕ ЛЮБУЮ КНОПКУ
ДЛЯ ПРОДОЛЖЕНИЯ... ]", str[1], str[2]);
	getch();
}

bool read_path(dir_t* param)
{
	DIR* directory = opendir(".\\data");
	struct dirent* ptr_dirent_value;
	*param = (dir_t){ (name_t*)calloc(1, sizeof(name_t)), 1 };

	if (!directory) return FALSE;
	int file_name_size = 0;
	while ((ptr_dirent_value = readdir(directory)) != NULL && param-
>size < MAX_FILES_IN_DIR)
	{
		if (strstr(ptr_dirent_value->d_name, ".txt") == NULL) continue;

		file_name_size = abs(ptr_dirent_value->d_name -
strstr(ptr_dirent_value->d_name, ".txt"));
		ptr_dirent_value->d_name[file_name_size] = '\0';
		param->array = (name_t*)realloc(param->array, sizeof(name_t) *
(param->size + 1));
		strncpy(param->array[param->size++], ptr_dirent_value->d_name,
file_name_size + 1);

	}
	closedir(directory);
	*param = (dir_t){ param->array+1, param->size -1};

	return TRUE;
}

bool file_data(field_t* field, bool readonly)
{
	if (!readonly)
	{
		bool checker = FALSE;
		for (unsigned int i = 0; i < pow(field->size, 2); i++)
		{
			if ((field->array + i)->check_value > 0)
			{
				checker = TRUE;
				break;
			}
		}
		if (!checker) return FALSE;
	}

	name_t dirname;
	FILE* file;
```

```
	sprintf(dirname, "./data/%s.txt", field->name);
	if ((file = fopen(dirname, readonly ? "rt" : "wt"))==NULL) return
FALSE;

	if (readonly)
	{
		unsigned int index = 0;
		if (!fscanf(file, "%d", &(field->size))) return FALSE;
		field->array = (cell_t*)calloc(pow(field->size, 2),
sizeof(cell_t));

		for (unsigned int i = 0; i < pow(field->size, 2); i++)
		{
			(field->array + i)->free_value.x = (field->array + i)-
>free_value.y = field->size;
			(field->array + i)->color = DEFAULT;
		}
		while (!feof(file))
		{
			if (!fscanf(file, "%d", &((field->array + (index++))-
>check_value))) continue;
		}
	}
	else
	{
		fprintf(file, "%d\n", field->size);
		for (unsigned int i = 0; i < field->size; i++)
		{
			for (unsigned int k = 0; k < field->size; k++)
				fprintf(file, "%d\t", (field->array + (i * field-
>size + k))->check_value);
			fprintf(file, "\n");
		}
	}
	if(!fclose(file)) return FALSE;
	return TRUE;
}

kaction_t get_keyboard_input(tuple_t* pos, tuple_t max) // стрелочное
управление WASD & SPACE
{
	kaction_t state = INPUT_NORMALLY;
	switch (getch())
	{
	case key_left: pos->x = (pos->x <= 0 ? 0 : pos->x - 1); break;
	case key_right:
		pos->x = (pos->x + 1 > max.x - 1 ? max.x - 1 : pos->x + 1);
		break;
	case key_up: pos->y = (pos->y <= 0 ? 0 : pos->y - 1); break;
	case key_down:
		pos->y = (pos->y + 1 > max.y - 1 ? max.y - 1 : pos->y + 1);
```

```
            break;
        case key_space: state = INPUT_ACCEPT; break;
        case key_esc: state = INPUT_BACK; break;
        default: state = INPUT_ERROR; break;
        }
        return state;
}
```

4. gamelogic.c

```
#include "kuromasu.h"

gupdate_t game_loop(void* args, kaction_t action, tuple_t pos)
{
        field_t* field = (field_t*)args;

        // формирование копии поля.
        cell_t* buffer = (cell_t*)calloc(pow(field->size, 2),
sizeof(cell_t));
        for (unsigned int i = 0; i < pow(field->size, 2); i++) *(buffer + i)
= *(field->array + i);

        static unsigned int life_counter = LIFE_COUNT; // статическая
переменная для счета жизни, объявляется и инициализируется при первом
вызове функции game_loop
        gstate_t state = STATE_RUNNING;

        //символьный массив life_line содержит символьное представление
количества попыток для решения головоломки, которые остались у
пользователя
        char life_line[LIFE_COUNT * 3 + 1];
        for (unsigned int i = 0; i < LIFE_COUNT; i++)
        {
                life_line[i*3] = (i < life_counter) ? '\x03' : ' ';
                life_line[i*3 + 1] = life_line[i*3 + 2] = ' ';
        }

        cell_t current_cell = *(field->array + (field->size * pos.y +
pos.x));
        printf("[ %-18s ( x: %2d , y: %2d ) ]\t[ ЖИЗНИ: %8.8s ]\n",
"ВИДИМОСТЬ:", current_cell.free_value.x, current_cell.free_value.y,
life_line);

        if (action == INPUT_BACK) // обработка события нажатия на клавишу
escape
        {
                dir_t items = (dir_t){ (name_t[]){ "ПРОДОЛЖИТЬ РЕШЕНИЕ",
"ВЫЙТИ В ГЛАВНОЕ МЕНЮ" }, 2 }; // формируется объект items отрисовки
диалогового окна
```

```
            switch (*(int*)update_frame(dialog_box, (tuple_t) { 0, 2 },
FALSE, &items)) // вызов отрисовки и обработка возращаемого пользователе
результата
            {
            case 0: return (gupdate_t) { FRAME_RETURN, NULL };
            case 1:
                  life_counter = LIFE_COUNT;
                  return (gupdate_t) { FRAME_EXIT, NULL };
            }
      }
      else if (action == INPUT_ACCEPT) // обработка события нажатия на
клавишу space
      {
            //переключение значения клетки и пересчет видимых клеток
            if (set_axies(pos, field))state = check_field(field); //
проверка значений видимых клеток для каждой клетки поля
      }

      draw_field(pos, field); //  отрисовка поля
      bool result = TRUE; // инициализация переменной для фиксации
поражения или победы

      switch (state) // обработка результата проверки
      {
      case STATE_WIN:
            life_counter = LIFE_COUNT; //возврат статической переменной в
начальное положение
            return (gupdate_t) { FRAME_EXIT, &result };
      case STATE_LOSE:
            printf("\n\t\t[ НАЖМИТЕ ЛЮБУЮ КНОПКУ ДЛЯ
ПРОДОЛЖЕНИЯ... ]\n\n\a");
            getch(); // пауза, чтобы пользователь смог увидеть допущенную
ошибку
            if (--life_counter <= 0)
            {
                  result = FALSE;
                  life_counter = LIFE_COUNT;
                  return (gupdate_t) { FRAME_EXIT, &result };// выход из
отрисовки и возврат значения
            }
            for (unsigned int i = 0; i < pow(field->size, 2); i++)
*(field->array + i) = buffer[i];// возврат к прошлой версии поля.
      case STATE_RUNNING: break;
      }
      free(buffer);
      return (gupdate_t) { FRAME_CONTINUE, NULL }; // продолжение
отрисовки
}

int connection_check(tuple_t pos, field_t* field, bool *checker)
{
```

```
	int result = 1;
	cell_t *current_cell = (field->array + (pos.y * field->size +
pos.x));

	bool * current_checker = (checker + (pos.y * field->size + pos.x));
	*current_checker = TRUE;

	if (pos.x > 0 && (current_cell - 1)->check_value >= 0 &&
*(current_checker - 1) != TRUE)
		result += connection_check((tuple_t) { pos.x - 1, pos.y },
field, checker);
	if (pos.x < field->size - 1 && (current_cell + 1)->check_value >= 0
&& *(current_checker + 1) != TRUE)
		result += connection_check((tuple_t) { pos.x + 1, pos.y },
field, checker);
	if (pos.y > 0 && (current_cell - field->size)->check_value >= 0 &&
*(current_checker - field->size) != TRUE)
		result += connection_check((tuple_t) { pos.x, pos.y - 1 },
field, checker);
	if (pos.y < field->size - 1 && (current_cell + field->size)-
>check_value >= 0 && *(current_checker + field->size) != TRUE)
		result += connection_check((tuple_t) { pos.x, pos.y + 1 },
field, checker);

	return result;
}

cell_t* set_line(cell_t* ptr_param, int size, int step, int shift)
{
	unsigned int last = 0;
	tuple_t* cell;
	for (unsigned int i = 0; i <= size; i++)
	{
		if ((ptr_param + (i * step) + shift)->check_value !=
BLACK_CELL && i != size) continue;
		for (int delta = i - last; last < i; last++)
		{
			cell = &(ptr_param + (last * step) + shift)->free_value;
			if (step == 1) cell->x = delta;
			else cell->y = delta;
		}
		last++;
	}
	return ptr_param;
}

bool set_axies(tuple_t pos, field_t* ptr_param)
{
	// указатель на клетку с курсором
	cell_t* cell = (ptr_param->array + (pos.y * ptr_param->size +
pos.x));
```

```
	// проверка  - 1.Никакие две черных ячейки не могу быть связаны по
горизонтали иливертикали.
	//                 2.У клетки отсутствует значение.
	if (cell->check_value > 0 ||
		((pos.x != 0) && (cell - 1)->check_value == BLACK_CELL ||
			(pos.x != ptr_param->size - 1) && (cell + 1)-
>check_value == BLACK_CELL ||
			(cell + ptr_param->size)->check_value == BLACK_CELL ||
			(cell - ptr_param->size)->check_value == BLACK_CELL))
		return FALSE;

	// переключение значения клетки
	cell->check_value = (cell->check_value == BLACK_CELL ? WHITE_CELL :
BLACK_CELL);
	/*
	* пересчет видимый клеток в каждой клетки двух линий, которые
пересекаются в выбранной клетке
	* проверяют две линий X и У
	* если при проходе по линии встречается черная клетка или конец
поля, значения free_value.(выбранная ось)
	* от клетки с индексом last.(выбранная ось) до клетки где сработало
условие пересчитывается:
	* присваивается значение разности межку началом и концом просчета
	*/
	ptr_param->array = set_line(ptr_param->array, ptr_param->size, 1,
pos.y * ptr_param->size);
	ptr_param->array = set_line(ptr_param->array, ptr_param->size,
ptr_param->size, pos.x);

	return TRUE;
}

gstate_t check_field(field_t* param)
{
	gstate_t result = STATE_RUNNING;
	bool *checker = (bool*)calloc(pow(param->size, 2), sizeof(bool)),
trigger = TRUE;

	int white_cell_counter = 0, connections = 0;
	for (unsigned int i = 0; i < pow(param->size, 2); i++)
	{
		if ((param->array + i)->check_value >= 0)
		{
			if (!connections) connections =
connection_check((tuple_t) { i% param->size, i / param->size }, param,
checker);
			white_cell_counter++;
		}
	}
	free(checker);
```

```
	for (unsigned int i = 0; i < pow(param->size, 2); i++)
	{
		cell_t* cell = (param->array + i);
		// если клетка не содержит цифру то пропуск итерации
		if (cell->check_value <= 0) continue;

		// если сумма свободных клеток по двум осям проверяемой клетки
меньше чем значение клетки
		if ((cell->free_value.x + cell->free_value.y < cell-
>check_value + 1) || (connections != white_cell_counter))
		{
			result = STATE_LOSE;
			cell->color = RED;
			continue;
		}

		// если сумма свободных клеток по двум осям проверяемой клетки
не одинаковы со значение клетки до переключаем тригер победы
		if (cell->free_value.x + cell->free_value.y != cell-
>check_value + 1)
		{
			trigger = FALSE;
			cell->color = DEFAULT;
		}
		else cell->color = GREEN;


	}
	if (trigger && result != STATE_LOSE) result = STATE_WIN;
	return result;
}
```

5. ui.c

```
#include "kuromasu.h"

gupdate_t set_cell_value(void* args, kaction_t action, tuple_t pos)
{
	int* field_size = (int*)args;
	if (action == INPUT_ACCEPT)
	{
		int value = pos.x;
		return (gupdate_t) { FRAME_EXIT, & value };
	}
	else
	{
		printf("\n\n\n\n\t\t[ УСТАНОВИТЬ ЧИСЛО ДЛЯ ВЫБРАННОЙ
КЛЕТКИ ]\n\n\t\t\t[ ЗНАЧЕНИЕ: %c %2d %c ]", (pos.x == 0) ? ' ' : '<',
pos.x,
			(pos.x == *field_size * 2 - 1) ? ' ' : '>');
```

```
	}
	return (gupdate_t) { FRAME_CONTINUE, NULL };
}

gupdate_t create_field(void* args, kaction_t action, tuple_t pos)
{
	if (action == INPUT_ACCEPT)
	{
		field_t field;

		field.size = pos.x + MIN_FIELD_SIZE;
		field.array = (cell_t*)calloc(pow(field.size, 2),
sizeof(cell_t));

		for (int i = 0; i < field.size; i++)
		{
			for (int k = 0; k < field.size; k++)
				*(field.array + (i * field.size + k)) =
(cell_t){ (tuple_t) {0,0}, 0, DEFAULT };
		}
		return (gupdate_t) { FRAME_EXIT, & field };
	}

	printf("\n\n\n\n\t\t[ УСТАНОВИТЬ РАЗМЕР ДЛЯ ИГРОВОГО
ПОЛЯ ]\n\n\t\t\t[ ЗНАЧЕНИЕ: %c %2d %c ]", (pos.x == 0) ? ' ' : '<',
		pos.x + MIN_FIELD_SIZE, (pos.x == MAX_FIELD_SIZE -
MIN_FIELD_SIZE) ? ' ' : '>');

	return (gupdate_t) { FRAME_CONTINUE, NULL };
}

gupdate_t set_field_values(void* args, kaction_t action, tuple_t pos)
{
	field_t* field = (field_t*)args;
	if (action == INPUT_ACCEPT)
	{
		(field->array + (pos.y * field->size + pos.x))->check_value =
			*(int*)update_frame(set_cell_value, (tuple_t) { field-
>size * 2, 0 }, FALSE, & (field->size));
	}
	else if (action == INPUT_BACK)
	{
		dir_t items = (dir_t){ (name_t[]) { "ПРОДОЛЖИТЬ СОЗДАНИЕ",
"ВЫЙТИ ИЗ РЕДАКТОРА", "СОХРАНИТЬ УРОВЕНЬ" }, 3 };
		switch (*(int*)update_frame(dialog_box, (tuple_t) { 0, 3 },
FALSE, & items))
		{
		case 0: break;
		case 1: return (gupdate_t) { FRAME_EXIT, NULL };
		case 2: return (gupdate_t) { FRAME_EXIT, field };
		}
```

```
	}

	draw_field(pos, field);
	return (gupdate_t) { FRAME_CONTINUE, NULL };
}

gupdate_t load_file(void* args, kaction_t action, tuple_t pos)
{
	pos.y = pos.y + pos.x * FILES_ON_PAGE;
	dir_t* param = (dir_t*)args;
	if (action == INPUT_ACCEPT && pos.y < param->size)
	{
		name_t filename;
		strcpy(filename, param->array + pos.y);

		dir_t items = (dir_t){ (name_t[]) { "ВЫБРАТЬ ИГРОВОЕ ПОЛЕ",
"РЕДАКТИРОВАТЬ ТЕКУЩЕЕ", "ВЕРНУТЬСЯ НАЗАД" }, 3};
		switch (*(unsigned int*)update_frame(dialog_box, (tuple_t) { 0,
3 }, FALSE, &items))
		{
		case 0: return (gupdate_t) { FRAME_EXIT, filename };
		case 1:;
			field_t read_field;
			strcpy(read_field.name, filename);
			if (!file_data(&read_field, TRUE))
			{
				print_messenge((char* []) { "[ СОВЕРШЕНА ОШИБКА ПРИ
РЕДАКТИРОВНИИ ПОЛЯ ]", "[  НЕВОЗМОЖНО ОТКРЫТЬ  ]", "[ ФАЙЛ ДЛЯ ИГРОВОГО
ПОЛЯ ]" });
				break;
			}
			field_t* save_field =
(field_t*)update_frame(set_field_values, (tuple_t) { read_field.size,
read_field.size }, TRUE, &read_field);
			if (!file_data(&read_field, FALSE))
				print_messenge((char* []) { "[ СОВЕРШЕНА ОШИБКА ПРИ
РЕДАКТИРОВНИИ ПОЛЯ ]", "[  НЕВОЗМОЖНО ОТКРЫТЬ  ]", "[ ФАЙЛ ДЛЯ ИГРОВОГО
ПОЛЯ ]" });

		case 2: break;
		}

	}
	else if (action == INPUT_BACK) return (gupdate_t) { FRAME_EXIT,
NULL };
	else
	{
		printf("\n\t\t%34s\n\n", "[ СПИСОК ДОСТУПНЫХ ПОЛЕЙ: ]");
		draw_list(pos.y, param, FILES_ON_PAGE * pos.x, FILES_ON_PAGE *
(pos.x + 1));
```

```
            printf("\n\n\t\t\t[ СТРАНИЦА: %2d/%2d ]", pos.x + 1,
(int)ceil(param->size / (double)FILES_ON_PAGE));
      }
      return (gupdate_t) { FRAME_CONTINUE, NULL };
}

gupdate_t settings(void* args, kaction_t action, tuple_t pos)
{
      field_t* field = (field_t*)(args);
      if (action == INPUT_ACCEPT)
      {
            switch (pos.y)
            {
            case 0:;
                  field_t edit_field =
*(field_t*)update_frame(create_field, (tuple_t) { fabs(MAX_FIELD_SIZE-
MIN_FIELD_SIZE)+1, 0 }, FALSE, NULL);
                  field_t* save_field =
(field_t*)update_frame(set_field_values, (tuple_t) { edit_field.size,
edit_field.size }, TRUE, & edit_field);
                  clear_frame();
                  printf("\n\n\n\t\t%40s\n\t\t%30s\n\n\t\t%10s
","[ ВВЕДИТЕ НАЗВАНИЕ ДЛЯ ИГРОВОГО ПОЛЯ ]",
                        "[ РАСПОЛОЖЕНИЕ
ФАЙЛА: .\\data\\название.txt ]","[ ---> ]");

                  if (scanf("%20s", save_field->name) && save_field !=
NULL)
                  {
                        if (!file_data(save_field, FALSE))
                              print_messenge((char* []) { "[ СОВЕРШЕНА
ОШИБКА ПРИ СОЗДАНИИ ПОЛЯ ]", "[  НЕВОЗМОЖНО СОХРАНИТЬ  ]", "[ ЗНАЧЕНИЯ
ИГРОВОГО ПОЛЯ ]" });
                        else *field = *save_field;
                  }
                  break;
            case 1:;
                  dir_t dir;
                  if (read_path(&dir))
                  {
                        unsigned int border = (unsigned int)(ceil(dir.size
/ 3.f) == 0) ? 1 : ceil(dir.size / 3.f);
                        name_t* filename = (name_t*)update_frame(load_file,
(tuple_t) { border, FILES_ON_PAGE }, TRUE, & dir);
                        if (filename != NULL)strcpy(field->name, *filename);
                  }
                  break;
            case 2: print_rules();  return (gupdate_t) { FRAME_CONTINUE,
NULL };
            case 3: return (gupdate_t) { FRAME_EXIT, field };
            }
```

```
	}
	else if (action == INPUT_BACK) return (gupdate_t) { FRAME_EXIT,
field };

	printf("[ ТЕКУЩЕЕ ПОЛЕ: %10s ]\n\n\t\t%38s\n\n", field->name,
"[ СТРАНИЦА РЕДАКТОРА ИГРОВЫХ ПОЛЕЙ ]");
	dir_t items = (dir_t){ (name_t[]) { "СОЗДАТЬ УРОВЕНЬ", "ЗАГРУЗИТЬ
СЕТКУ", "ПРАВИЛА ГОЛОВОЛОМКИ", " ВЕРНУТЬСЯ НАЗАД" }, 4 };
	draw_list(pos.y, &items, 0, 4);

	return (gupdate_t) { FRAME_CONTINUE, NULL };
}

gupdate_t mainmenu(void* args, kaction_t action, tuple_t pos)
{
	field_t* field = (field_t*)args;

	if (action == INPUT_ACCEPT)
	{
		switch (pos.y)
		{
		case 0:;
			bool check = file_data(field, TRUE);
			if (!check)
			{
				clear_frame();
				printf("\n\n\n\t\t\t[ НЕВОЗМОЖНО ЗАГРУЗИТЬ
ПОЛЕ: %s ]\n\a\n\t\t[ НАЖМИТЕ ЛЮБУЮ КНОПКУ ДЛЯ ПРОДОЛЖЕНИЯ... ]", field-
>name);
				getch(); break;
			}
			bool* game_result = (bool*)update_frame(game_loop,
(tuple_t) { field->size, field->size }, TRUE, field);
			if (game_result != NULL)
			{
				bool decode = *game_result;
				clear_frame();
				printf("\n\n\n\n%s\n\a", decode ? WIN_LABEL :
LOSE_LABEL);
				getch();
			}

			break;
		case 1:;
			*field = *(field_t*)update_frame(settings, (tuple_t) { 0,
4 }, FALSE, field);
			break;
		case 2:;
			dir_t items = (dir_t){ (name_t[]) { "ВЕРНУТЬСЯ В МЕНЮ",
"ВЫЙТИ ИЗ ПРОГРАММЫ" }, 2 };
```

```
                switch (*(unsigned int*)update_frame(dialog_box,
(tuple_t) { 0, 2 }, FALSE, &items))
                {
                case 0: return (gupdate_t) { FRAME_RETURN, NULL };
                case 1: return (gupdate_t) { FRAME_EXIT, NULL };

                }
            }
        }

        printf("\n%s\n\n\n", NAME_LABEL);

        dir_t items = (dir_t){ (name_t[]) { "НАЧАТЬ РЕШЕНИЕ", "РЕДАКТОР
УРОВНЕЙ", "ВЫЙТИ ИЗ ИГРЫ" }, 3 };
        draw_list(pos.y, &items, 0, 3);

        return (gupdate_t) { FRAME_CONTINUE, NULL };

}
```