ФГБОУ ВО «Ивановский государственный энергетический  
университет имени В.И. Ленина»  
Факультет информатики и вычислительной техники  
Кафедра программного обеспечения компьютерных систем

**Курсовая работа   
по дисциплине   
«Программирование и основы алгоритмизации»   
на тему   
«Разработка программы игры в шашки»**

Выполнил:

студент гр. 1-41 Зверев Е.И. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_

(подпись) (дата)

Руководитель:

доцент каф. ПОКС Алыкова А.Л \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_

(подпись) (дата)

Иваново, 2023

**Содержание**

[1. Задание 6](#_Toc135737901)

[2. Анализ задачи 7](#_Toc135737902)

[3. Математическая модель 7](#_Toc135737903)

[3.1. Основные типы данных 7](#_Toc135737904)

[3.2. Основные алгоритмы 8](#_Toc135737905)

[3.2.1. Выбор средств для решения задачи 9](#_Toc135737906)

[4. Описания основных типов данных и основных алгоритмов 10](#_Toc135737907)

[4.1. Заголовочный файл clcengine\_types.h 10](#_Toc135737908)

[4.1.1. Перечисление BoardCellState 10](#_Toc135737909)

[4.1.2. Перечисление PathMapMarkers 10](#_Toc135737910)

[4.1.3. Перечисление Color 11](#_Toc135737911)

[4.1.4. Перечисление CheckerType 11](#_Toc135737912)

[4.1.5. Перечисление GameType 11](#_Toc135737913)

[4.1.6. Перечисление Difficulty 11](#_Toc135737914)

[4.1.7. Структура Coordinates 11](#_Toc135737915)

[4.1.8. Структура Checker 11](#_Toc135737916)

[4.1.9. Структура Move 12](#_Toc135737917)

[4.1.10. Структура TakingMove 12](#_Toc135737918)

[4.1.11. Структура RegMoveSequence 12](#_Toc135737919)

[4.1.12. Структура TakingSequence 12](#_Toc135737920)

[4.1.13. Структура Board 12](#_Toc135737921)

[4.1.14. Структура GameSituation 13](#_Toc135737922)

[4.1.15. Структура SeqContainer 13](#_Toc135737923)

[4.1.16. Структура Game 14](#_Toc135737924)

[4.1.17. Константные значения 14](#_Toc135737925)

[4.2. Заголовочный файл clcengine\_auxiliary.h 16](#_Toc135737926)

[4.2.1. Функция isCoordinatesEqual 16](#_Toc135737927)

[4.2.2. Функция appendToATakingSequence 16](#_Toc135737928)

[4.2.3. Функция deleteLastFromATakingSequence 16](#_Toc135737929)

[4.2.4. Функция copyToAnotherTakingSequence 17](#_Toc135737930)

[4.2.5. Функция getVictoryEvalFor 17](#_Toc135737931)

[4.2.6. Функция lostByMoves 17](#_Toc135737932)

[4.2.7. Функция copyLevelOneMovesToBuffers 18](#_Toc135737933)

[4.2.8. Функция flushSequenceLists 18](#_Toc135737934)

[4.2.9. Функция replaceConditionFor 19](#_Toc135737935)

[4.2.10. Функция evalQuality 19](#_Toc135737936)

[4.2.11. Функция getVictimMarker 20](#_Toc135737937)

[4.2.12. Функция updateBoardRender 20](#_Toc135737938)

[4.2.13. Функция fillPathMap 20](#_Toc135737939)

[4.2.14. Функция getCheckerIndexByCoordsAndColor 21](#_Toc135737940)

[4.2.15. Функция sideColor 21](#_Toc135737941)

[4.2.16. Функция negateColor 22](#_Toc135737942)

[4.2.17. Функция getCheckerTypeOnBoard 22](#_Toc135737943)

[4.2.18. Функция isAFriend 22](#_Toc135737944)

[4.2.19. Функция isMarkedForDeath 23](#_Toc135737945)

[4.2.20. Функция isAVictim 24](#_Toc135737946)

[4.2.21. Функция isNotOnKBBorder 24](#_Toc135737947)

[4.2.22. Функция longMoveLookingFlag 24](#_Toc135737948)

[4.2.23. Функция regMoveLookingFlag 25](#_Toc135737949)

[4.2.24. Функция findDistanceToVictim 25](#_Toc135737950)

[4.2.25. Функция flushMoveBuffers 26](#_Toc135737951)

[4.2.26. Функция removeChecker 26](#_Toc135737952)

[4.2.27. Функция descendChecker 27](#_Toc135737953)

[4.2.28. Функция ascendChecker 27](#_Toc135737954)

[4.2.29. Функция removeMarkedForDeath 27](#_Toc135737955)

[4.3. Заголовочный файл clcengine\_inits.h 29](#_Toc135737956)

[4.3.1. Функция getNullContainer 29](#_Toc135737957)

[4.3.2. Функция getNullPath 29](#_Toc135737958)

[4.3.3. Функция initiateChecker 29](#_Toc135737959)

[4.3.4. Функция resetPathMap 30](#_Toc135737960)

[4.3.5. Функция createANewBoard 30](#_Toc135737961)

[4.3.6. Функция makeNullGameSituation 31](#_Toc135737962)

[4.3.7. Функция createANewGame 31](#_Toc135737963)

[4.4. Заголовочный файл clcengine\_move\_searhing.h 32](#_Toc135737964)

[4.4.1. Функция findAllKingMovesForOneOnDir 32](#_Toc135737965)

[4.4.2. Функция findAllKingMovesForOne 32](#_Toc135737966)

[4.4.3. Функция findAllRegularMovesForOneOnDir 33](#_Toc135737967)

[4.4.4. Функция findAllRegularMovesForOne 33](#_Toc135737968)

[4.4.5. Функция findAllTakingMovesForOne 34](#_Toc135737969)

[4.4.6. Функция findAllKingTakingMovesForOneOnDir 35](#_Toc135737970)

[4.4.7. Функция findAllKBMovesForOne 36](#_Toc135737971)

[4.4.8. Функция findAllKingTakingMovesForOne 36](#_Toc135737972)

[4.5. Заголовочный файл clcengine\_moving.h 38](#_Toc135737973)

[4.5.1. Функция makeAMove 38](#_Toc135737974)

[4.5.2. Функция makeATakingMove 38](#_Toc135737975)

[4.5.3. Функция cancelAMove 39](#_Toc135737976)

[4.5.4. Функция cancelATakingMove 39](#_Toc135737977)

[4.6. Заголовочный файл clcengine\_sequence\_searching.h 41](#_Toc135737978)

[4.6.1. Функция findRegularMoveSequenceForOne 41](#_Toc135737979)

[4.6.2. Функция findAllRegularMoveSequences 41](#_Toc135737980)

[4.6.3. Функция findAllTakingSequencesForOne 42](#_Toc135737981)

[4.6.4. Функция findAllTakingMoveSequences 43](#_Toc135737982)

[4.6.5. Функция findAllMoves 43](#_Toc135737983)

[4.7. Заголовочный файл clcengine\_sequence\_moving.h 44](#_Toc135737984)

[4.7.1. Функция makeARegMoveSequence 44](#_Toc135737985)

[4.7.2. Функция cancelARegMoveSequence 44](#_Toc135737986)

[4.7.3. Функция makeATakingMoveSequence 44](#_Toc135737987)

[4.7.4. Функция cancelATakingMoveSequence 45](#_Toc135737988)

[4.8. Заголовочный файл clcengine\_analyzer.h 46](#_Toc135737989)

[4.8.1. Функция analyze 46](#_Toc135737990)

[4.8.2. Функция getBestMove 47](#_Toc135737991)

[5. Блок-схемы некоторых алгоритмов 48](#_Toc135737992)

[5.1. Функция copyLevelOneMovesToBuffers 48](#_Toc135737993)

[5.2. Функция evalQuality 49](#_Toc135737994)

[5.3. Функция updateBoardRender 50](#_Toc135737995)

[5.4. Функция fillPathMap 51](#_Toc135737996)

[5.5. Функция findDistanceToVictim 52](#_Toc135737997)

[5.6. Функция removeMarkedForDeath 53](#_Toc135737998)

[5.7. Функция findAllKingMovesForOneOnDir 54](#_Toc135737999)

[5.8. Функция findAllRegularMovesForOneOnDir 55](#_Toc135738000)

[5.9. Функция findAllTakingMovesForOne 56](#_Toc135738001)

[5.10. Функция findAllKingTakingMovesForOneOnDir 58](#_Toc135738002)

[5.11. Функция findAllKBMovesForOne 59](#_Toc135738003)

[5.12. Функция makeAMove 60](#_Toc135738004)

[5.13. Функция makeATakingMove 61](#_Toc135738005)

[5.14. Функция cancelAMove 62](#_Toc135738006)

[5.15. Функция cancelATakingMove 63](#_Toc135738007)

[5.16. Функция findRegularMoveSequenceForOne 64](#_Toc135738008)

[5.17. Функция findAllTakingSequencesForOne 65](#_Toc135738009)

[5.18. Функция findAllTakingMoveSequences 66](#_Toc135738010)

[5.19. Функция analyze 67](#_Toc135738011)

[5.20. Интерфейс приложения: обработка нажатия кнопки создания новой игры против компьютера 69](#_Toc135738012)

[5.21. Интерфейс приложения: обработка нажатия кнопки создания новой игры против человека 70](#_Toc135738013)

[5.22. Интерфейс приложения: обработка нажатия клавиши Enter 71](#_Toc135738014)

[6. Руководство пользователя 73](#_Toc135738015)

[6.1. Руководство по установке приложения 73](#_Toc135738016)

[6.2. Руководство по пользованию интерфейсом приложения 73](#_Toc135738017)

[6.3. Правила игры в русские шашки 74](#_Toc135738018)

[7. Положение об авторском праве 75](#_Toc135738019)

[8. Лицензирование проекта 75](#_Toc135738020)

[9. Приложение 1. Исходный код основных заголовочных файлов 76](#_Toc135738021)

[9.1. Заголовочный файл clcengine\_types.h 76](#_Toc135738022)

[9.2. Заголовочный файл clcengine\_auxiliary.h 78](#_Toc135738023)

[9.3. Заголовочный файл clcengine\_inits.h 83](#_Toc135738024)

[9.4. Заголовочный файл clcengine\_move\_searching.h 85](#_Toc135738025)

[9.5. Заголовочный файл clcengine\_moving.h 89](#_Toc135738026)

[9.6. Заголовочный файл clcengine\_sequence\_searching.h 91](#_Toc135738027)

[9.7. Заголовочный файл clcengine\_sequence\_moving.h 94](#_Toc135738028)

[9.8. Заголовочный файл clcengine\_analyzer.h 95](#_Toc135738029)

[9.9. Заголовочный файл clcengine.h 97](#_Toc135738030)

# Задание

Тема 3. Разработка программы игры в шашки

Игра проводится в соответствии с общепринятыми правилами.

*Требования к функциональным характеристикам:*

* Диалоговый режим работы, диалог инициируется компьютером;
* Отображение игрового поля на экране;
* Выбор в игровом поле шашки и перемещение её с помощью функциональной клавиатуры, помощь игроку по пользованию функциональной клавиатуры;
* Возможность игры двух игроков или игрока с компьютером, начинающий игру выбирается случайно
* Игра компьютера по некоторой стратегии
* Оперативное отображение и контроль ситуации, определение победителя

# Анализ задачи

Основной упор в данной задаче делается на разработку алгоритма определения лучшего хода, который будет нужен для игры против компьютера. За основу берётся минимакс, основная идея которого, по сути, строится на просмотре всех возможных ходов на некоторую глубину и определении лучшего хода для нашей стороны (максимизирующая сторона) и для стороны противника (минимизирующая сторона). Для определения «качества» хода будем оценивать позицию, к которой мы пришли этим ходом.

Таким образом, для нашей задачи необходимо определить все требующиеся структуры для представления шашечной доски, хранилища всех доступных на данный момент ходов, а также функций для совершения ходов, их отмены и анализа и оценки позиции.

# Математическая модель

## Основные типы данных

Для начала определим простую структуру Coordinates, которая будет хранить только две переменные, x и y. Эта структура нам понадобится для хранения координат шашки, а также для упрощения работы с уже существующими шашками и обновлением координат.

Теперь определим структуру Checker, шашка, которая будет хранить свои координаты в структуре Coordinates, цвет типа перечисления Color, вид типа перечисления CheckerType, и булевскую переменную, обозначающую «метку на смерть» (необходима для осуществления выполнения «турецкого правила» в настоящей игре.

Для представления доски определим структуру Board, которая будет содержать в себе массив 8 на 8, состоящий из переменных типа перечисления BoardCellState, который содержит информацию о том, что на данный момент на клетке доски; помимо этого, структура содержит в себе все шашки для обоих сторон игры в виде массива 2 на 12, и количества всех активных («не мёртвых») шашек. Массив необходим для поиска ходов, а также для отрисовки доски, потому необходимо хранить все поля, в том числе и белые, по которым ходов и не происходит. Массив шашек необходим для быстрого удаления уже побитой шашки и для быстрого поиска ходов для всех шашек, исключая полное сканирование доски. В ней также будет хранится вспомогательный массив 8 на 8, который нужен в основном для совершения хода в графическом интерфейсе, сам массив в каких-либо основных алгоритмах участие не принимает.

Для представления обычных ходов воспользуемся структурой Move, которая хранит в себе координаты начала пути, конца пути, и индикатор, обозначающий то, что после этого хода шашка превращается в дамку. (здесь и далее, если переменные обозначают координаты чего-либо, то эти переменные имеют тип Coordinates)

Для представления ходов на взятие определим структуру TakingMove, которая хранит в себе координаты начала пути, конца пути, «жертвы», а также тип «жертвы», её цвет и тип (необходимы отмены хода на взятия в функции поиска лучшего хода), и индикатор, обозначающий то, что после этого хода шашка превращается в дамку.

Можно было бы хранить все возможные ходы в массивах типов Move и TakingMove, однако есть один довольно неприятный момент: если мы будем хранить только доступные ходы по одному, то мы с трудом сможем «отловить» ситуацию, когда после совершения хода на взятие мы можем сделать ещё ход. Это возможно в следующих случаях:

1. После хода шашка превратилась в дамку, тогда она может продолжить ход по правилам, если у неё нет хода на взятие
2. После взятия получается ситуация, когда сходившая шашка может «побить» ещё одну, отличную от той, что она перепрыгнула на предыдущем ходе.

При таком раскладе лучше будет хранить не доступные ходы, в доступные цепочки ходов (последовательности). Тогда определим следующие структуры:

* RegMoveSequence: последовательность обычных ходов (либо один обычный ход или один ход с «превращением» и один обычный ход дамки)
* TakingSequence: последовательность ходов на взятие

Хранение ходов в этих структурах осуществляется в массивах, также в этих структурах хранится и количество ходов в последовательности.

Для хранения доски, всех доступных последовательностей и цвет игрока будем хранить в структуре GameSituation. Но кроме вышеуказанных вещей, в ней мы также будем хранить ходы, временно хранящиеся и необходимые для работы алгоритмов поиска доступных последовательностей.

Определим также структуру SeqContainer, который будет содержать какую-либо последовательность ходов, число, говорящее о том, какую последовательность надо выполнить из этого контейнера, и оценку позиции, к которой мы пришли этой последовательностью. Нужна для алгоритма поиска лучшего хода, а также компьютеру, чтобы потом выполнить найденный лучший ход.

Наконец, определим структуру Game, содержащий саму игровую ситуацию, тип игры и её состояние (не используется).

## Основные алгоритмы

Как уже и было сказано ранее, для поиска лучшего хода воспользуемся алгоритмом «Минимакс». Он основывается на том, чтобы для всех возможных ходов, предугадать ответы соперника, и для каждого ответа — выбрать свой ответ, на который опять предугадать возможные ответы соперника. По сути, перебор всех возможных вариантов на определённую глубину. Данному алгоритму необходимы два элемента: оценка позиции и поиск всех цепочек ходов.

Для оценки позиции будем вычислять для каждой стороны её качество, а потом находить разность между качеством чёрных и качеством белых. Качество же каждой стороны будем рассчитывать по следующим правилам:

* Стоимость дамки равна 250 баллам
* Стоимость обычной шашки равна 100 баллам по умолчанию ,но чем ближе она к дамочным полям, тем она ценнее. (необходимые коэффициенты указаны в массиве qualityFactor)

Поиск обычных цепочек ходов произведём по следующему правилу:

1. Делаем ход
2. Проверяем, есть ли ещё ходы
3. Если есть, то добавляем их в цепочку, а потом – переносим в буфер. Важно! Для каждого нового хода – отдельная цепочка.
4. Если нет, то просто добавляем цепочку в буфер и выходим, отменяя все ранее сделанные ходы.

Поиск цепочек ходов на взятие будем осуществлять так:

1. Создаём переменную пути, в которой пока нет ходов
2. Если мы до этого не ходились, то ищем все доступные ходы на взятие.
3. Начинаем проходится по всем найденным ходам на взятие
4. Делаем ход и добавляем его в путь
5. Проверяем, есть ли ещё ходы на взятие
6. Если да, то рекурсивно идём в шаг 1, говоря, что ищем после хода
7. Если нет, то копируем содержимое пути в буфер
8. Отменяем ход
9. Удаляем последний ход из пути
10. Продолжаем, пока не пройдёмся по всем ходам

### Выбор средств для решения задачи

В качестве среды разработки настоящего курсового проекта была выбрана интегрированная среда разработки CLion версии 2020.2. Для компилирования исходных кодов настоящего проекта использовался компилятор gcc.

Для написания интерфейса использовались методы WinAPI, описанные в заголовочном файле windows.h.

# Описания основных типов данных и основных алгоритмов

Все основные алгоритмы описаны в файлах папки clcengine настоящего проекта, за исключением заголовочного файла clcengine.h, так как данный файл предназначен только для подключения всех остальных заголовочных файлов, а потому не содержит какого-либо кода.

Во всех функциях, описанных далее, происходит манипуляция над полями структур, определённых в следующем разделе. Под «данными» переменными, если не указано иное, подразумевается те переменные, которые были переданы как параметры функций.

Все буферы ходов и последовательностей ходов, если не указано иное, являются соответствующими полями структуры GameSituation.

## Заголовочный файл clcengine\_types.h

Данный заголовочный файл содержит определения всех необходимых типов, задействованных в алгоритмах

### Перечисление BoardCellState

**Назначение:** предназначено для обозначения содержимого клетки доски, используется для отрисовки доски и поиска ходов

|  |  |  |
| --- | --- | --- |
| **Название элемента** | **Значение** | **Назначение** |
| EMPTY\_BLACK | 0 | Обозначает пустую клетку чёрного цвета |
| EMPTY\_WHITE | 1 | Обозначает пустую клетку белого цвета |
| REG\_BLACK | 2 | Обозначает клетку, на которой стоит обычная чёрная шашка |
| REG\_WHITE | 3 | Обозначает клетку, на которой стоит обычная белая шашка |
| KING\_BLACK | 4 | Обозначает клетку, на которой стоит чёрная дамка |
| KING\_WHITE | 5 | Обозначает клетку, на которой стоит белая дамка |

### Перечисление PathMapMarkers

**Назначение:** предназначено для обозначения содержимого «карты путей» (метки того, куда выбранная шашка может пойти). Используется для отрисовки доступных полей для хода, а также жертв взятия в игре

|  |  |  |
| --- | --- | --- |
| **Название элемента** | **Значение** | **Назначение** |
| NoMove | 0 | Обозначает клетку, на которую нельзя пойти |
| Source | 1 | Обозначает клетку, от которой идём (начало пути) |
| Destination | 2 | Обозначает клетку, к которой идём (конец пути) |
| VictimRB | 3 | Обозначает клетку, на которой стоит жертва взятия, являющейся обычной чёрной шашкой |
| VictimRW | 4 | Обозначает клетку, на которой стоит жертва взятия, являющейся обычной белой шашкой |
| VictiomKB | 5 | Обозначает клетку, на которой стоит жертва взятия, являющейся чёрной дамкой |
| VictimKW | 6 | Обозначает клетку, на которой стоит жертва взятия, являющейся белой дамкой |
| MovingLock | 7 | Не используется |

### Перечисление Color

**Назначение:** предназначено для обозначения цвета шашки

|  |  |  |
| --- | --- | --- |
| **Название элемента** | **Значение** | **Назначение** |
| Black | 0 | Чёрный цвет шашки |
| White | 1 | Белый цвет шашки |

### Перечисление CheckerType

**Назначение**: предназначено для обозначения типа шашки

|  |  |  |
| --- | --- | --- |
| **Название элемента** | **Значение** | **Назначение** |
| Regular | 0 | Обозначает обычную шашку |
| King | 1 | Обозначает дамку |

### Перечисление GameType

**Назначение:** предназначено для обозначения типа игры

|  |  |  |
| --- | --- | --- |
| **Название элемента** | **Значение** | **Назначение** |
| RvsR | 0 | Обозначает тип игры «Человек против человека» |
| RvsC | 1 | Обозначает тип игры «Человек против компьютера» |

### Перечисление Difficulty

**Назначение:** предназначено для обозначения уровня сложности, от которого зависит глубина поиска лучшего хода, заданная значением элемента данного перечисления

|  |  |  |
| --- | --- | --- |
| **Название элемента** | **Значение** | **Назначение** |
| Dumbass | 2 | Уровень сложности «Придурок» |
| Easy | 3 | Уровень сложности «Легко» |
| Normal | 4 | Уровень сложности «Нормально» |
| Hard | 6 | Уровень сложности «Сложно» |
| Insane | 7 | Уровень сложности «Безумно» |
| Extreme | 8 | Уровень сложности «Экстремально» |
| Diabolic | 9 | Уровень сложности «Дьявольский», не используется |
| Invincible | 11 | Уровень сложности «Неуязвимый», не используется |

### Структура Coordinates

**Назначение:** предназначена для хранения координат шашки, но иногда используется для хранения пары значений

|  |  |  |
| --- | --- | --- |
| **Название элемента** | **Значение** | **Назначение** |
| X | int | Абсцисса, первый элемент пары |
| Y | int | Ордината, второй элемент пары |

### Структура Checker

**Назначение:** предназначена для хранения самой шашки

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| Coordinates | Coordinates | Координаты шашки |
| Color | Color | Цвет шашки |
| Type | CheckerType | Тип шашки |
| markedForDeath | bool | Индикатор того, что шашка взята и её необходимо убрать с доски |

### Структура Move

**Назначение:** предназначена для хранения обычного хода

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| Source | Coordinates | Координаты начала хода |
| Destination | Coordinates | Координаты конца хода |
| isKingBecomingMove | bool | Индикатор того, превратится ли после этого хода обычная шашка в дамку |

### Структура TakingMove

**Назначение:** предназначена для хранения хода на взятие

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| Source | Coordinates | Координаты начала хода |
| Destination | Coordinates | Координаты конца хода |
| Victim | Coordinates | Координаты жертвы взятия |
| takingSide | Color | Сторона жертвы |
| victimType | CheckerType | Тип жертвы |
| isASpecialMove | bool | Индикатор того, превратится ли после этого хода обычная шашка в дамку |

### Структура RegMoveSequence

**Назначение:** предназначена для хранения обычной последовательности ходов

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| regularMoves | Массив Move[2] | Обычные ходы, составляющие последовательность |
| rmsCount | int | Количество ходов в последовательности |

### Структура TakingSequence

**Назначение:** предназначена для хранения последовательности ходов на взятие

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| takingMoves | Массив TakingMove[16] | Ходы на взятие, составляющие последовательность |
| tmsCount | int | Количество ходов в последовательности |

### Структура Board

**Назначение:** предназначена для хранения самой игровой доски

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| boardRender | Массив BoardCellState[8][8] | Шаблон доски, который используется для поиска ходов и её отрисовки в игре |
| pathMap | Массив PathMapMarkers[8][8] | «Карта путей» (метки того, куда выбранная шашка может пойти), используется только в реальной игре |
| Checkers | Массив Chekcer[2][12] | Массив всех активных шашек. Первая строка хранит чёрные шашки, вторая – белые |
| checkersCount | Массив int[2] | Количество активных шашек для каждой стороны. Первый элемент – для чёрных, второй – для белых |

### Структура GameSituation

**Назначение:** предназначена для хранения игровой ситуации

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| Board | Board | Игровая доска, привязанная к ситуации |
| playerSide | Color | Цвет игрока |
| regularMovesBuffer | Массив Move[24] | Буфер обычных ходов. Используется при поиске ходов, а также при совершении хода игроком в интерфейсе игры |
| takingMovesBuffer | Массив TakingMove[24] | Буфер ходов на взятие. Используется при поиске ходов, а также при совершении хода игроком в интерфейсе игры |
| tmBufferLen | int | Количество ходов в takingMovesBuffer |
| rmBufferLen | int | Количество ходов в regularMovesBuffer |
| rmsCount | int | Количество последовательностей в regMoveSequences |
| tmsCount | int | Количество последовательностей в takingSequences |
| regMoveSequences | Массив RegMoveSequence[32] | Буфер всех найденных последовательностей обычных ходов, используется при поиске лучшего хода |
| takingSequences | Массив TakingSequence[16] | Буфер всех найденных последовательностей ходов на взятие, используется при поиске лучшего хода |

### Структура SeqContainer

**Назначение:** предназначена для хранения какой-либо последовательности ходов

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| regMoveSequence | RegMoveSequence | Обычная последовательность, хранимая в контейнере |
| takingSequence | TakingSequence | Последовательность на взятия, хранимая в контейнере |
| seqNumberToDo | int | Число, указывающее на то, какая последовательность в контейнере должна быть выполнена |
| Eval | float | Оценка позиции, к которой мы придём, совершив хранимую последовательность ходов. Используется при поиске лучшего хода |

### Структура Game

**Назначение:** предназначена для хранения самой игры

|  |  |  |
| --- | --- | --- |
| **Название поля** | **Тип** | **Назначение** |
| Situation | GameSituation | Игровая ситуация, привязанная к игре |
| Type | GameType | Тип игры |

### Константные значения

|  |  |  |  |
| --- | --- | --- | --- |
| **Название константы** | **Тип** | **Значение** | **Назначение** |
| regularCheckerCost | int | 100 | Стандартная стоимость обычной шашки |
| kingCheckerCost | int | 250 | Стоимость дамки |
| ENABLED\_EMULATION | Bool | true | Индикатор того, что ход, который мы отменяем, был сэмулирован. Используется только для улучшения читаемости кода |
| BLACK\_VICTORY | float | 100000.0 | Оценка позиции для случая, когда чёрные победили |
| WHITE\_VICTORY | float | -100000.0 | Оценка позиции для случая, когда белые победили |
| qualityFactor | Массив float[8][4] | {  {1.2, 1.2, 1.2, 1.2},  {1.15, 1.2, 1.2, 1.15},  {1.15, 1.2, 1.2, 1.13},  {1.0, 1.2, 1.15, 1.0},  {1.0, 1.2, 1.2, 1.0},  {1.0, 1.0, 1.0, 1.0},  {1.0, 1.0, 1.0, 1.0},  {1.0, 1.0, 1.0, 1.0},  } | Таблица коэффициентов для ценности обычной шашки в зависимости от того, насколько она близка к дамочным полям |
| probsOfDrawCPU | Массив int[8] | {999, 800, 500, 250, 80, 0, 0, 0} | Числа, показывающие вероятности принятия предложения ничьи компьютеру. Используется только в игре против компьютера |
| NORTH\_EAST | Coordinates | {1, 1} | Направление «взгляда поиска» северо-восток |
| NORTH\_WEST | Coordinates | {-1, 1} | Направление «взгляда поиска» северо-запад |
| SOUTH\_EAST | Coordinates | {1, -1} | Направление «взгляда поиска» юго-восток |
| SOUTH\_WEST | Coordinates | {-1, -1} | Направление «взгляда поиска» юго-запад |

## Заголовочный файл clcengine\_auxiliary.h

Данный заголовочный файл содержит определения различных вспомогательных функций, которые нужны для упрощения некоторых фрагментов кода.

### Функция isCoordinatesEqual

**Определение:**

bool isCoordinatesEqual(Coordinates arg1, Coordinates arg2)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Coordinates | arg1 | Первая координата на сравнение |
| Coordinates | arg2 | Вторая координата на сравнение |

**Возвращаемое значение:**

true, если координаты равны, иначе false.

**Назначение:**

Проверяет две координаты на равенство друг другу

**Описание:**

Функция попарно проверяет поля x и y структур arg1 и arg2 и после применяет к результатам логическую операцию И, после чего возвращает полученный результат.

### Функция appendToATakingSequence

**Определение:**

void appendToATakingSequence(TakingSequence\* sequence, TakingMove move)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| TakingSequence\* | sequence | Указатель на последовательность взятий, к которой необходимо добавить ход |
| TakingMove | move | Ход на взятие, который добавляется в последовательность |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Необходима для добавления нового хода в последовательность взятий. Используется во время поиска последовательности всех цепочек ходов на взятие.

**Описание:**

Функция добавляет в конец массива ходов на взятие последовательности ходов новый ход, после чего увеличивает количество ходов в последовательности на 1.

### Функция deleteLastFromATakingSequence

**Определение:**

void deleteLastFromATakingSequence(TakingSequence\* sequence)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| TakingSequence\* | sequence | Указатель на последовательности взятий, из которого вы удаляем последний ход |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Необходима для удаления последнего хода из последовательности взятий. Используется во время поиска всех последовательностей ходов на взятие

**Описание:**

Функция уменьшает количество элементов в массиве ходов на взятие последовательности на 1

### Функция copyToAnotherTakingSequence

**Определение:**

void copyToAnotherTakingSequence(TakingSequence\* source, TakingSequence\* destination)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| TakingSequence\* | source |  |
| TakingSequence\* | destination |  |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Копирование содержимого одной последовательности ходов на взятие к другой. Используется во время поиска последовательностей ходов на взятие

**Описание:**

Устанавливает всем элементам массива ходов на взятие последовательности назначения ходы, имеющие тот же индекс в последовательности-источнике, затем присваивает полю tmsCount тоже значение, что и у источника

### Функция getVictoryEvalFor

**Определение:**

float getVictoryEvalFor(Color side)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Color | side | Цвет стороны, для которой мы ищем значение «победной оценки» |

**Возвращаемое значение:**

BLACK\_VICTORY либо WHITE\_VICTORY

**Назначение:**

Используется для инициализации некоторых структур, а также по время поиска лучшего хода

**Описание:**

Функция возвращает значение BLACK\_VICTORY, если был передан параметр side со значением Black, либо WHITE\_VICTORY, если был передан параметр side со значением White.

### Функция lostByMoves

**Определение:**

bool lostByMoves(GameSituation\* situation)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Игровая ситуация, в которой мы проверяем предмет поражения по ходам |

**Возвращаемое значение:**

True, если проиграл по ходам, иначе false

**Назначение:**

Требуется для определения поражения игрока на данном ходе

**Описание:**

Функция суммирует количества найденных последовательностей, и проверяет, является ли эта сумма нулём. Если да, то считается, что игрок проиграл (так как нет доступных ходов), и функция возвращает true, в противном случае возвращает false, так как есть доступные ходы для игрока

### Функция copyLevelOneMovesToBuffers

**Определение:**

void copyLevelOneMovesToBuffers (GameSituation\* situation)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, откуда мы достаём ходы «первого уровня» |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Копирует все ходы «первого уровня», в буферы regularMovesBuffer и takingMovesBuffer переданной игровой ситуации. Требуется для совершения хода игроком в интерфейсе игры.

**Описание:**

Сначала функция проходится по буферу обычных ходов игровой ситуации, и «достаёт» оттуда все первые ходы в каждой цепочке обычных ходов. Тоже самое и для буфера последовательности ходов на взятие.

### Функция flushSequenceLists

**Определение:**

void flushSequenceLists(GameSituation\* situation)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы должны «очистить» списки всех найденных последовательностей ходов |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

«Очищает» все списки последовательностей ходов

**Описание:**

Функция присваивает полям rmsCount и tmsCount нулевое значение.

### Функция replaceConditionFor

**Определение:**

bool replaceConditionFor(float nEval, float oEval, Color side)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| float | nEval | Новое значение оценки позиции |
| float | oEval | Старое значение оценки позиции |
| Color | side | Цвет стороны, для которой нам необходимо проверить «удачность» позиции |

**Возвращаемое значение:**

Результат проверки условия

**Назначение:**

Предназначена для определения того, нашли ли мы лучший ход для данной стороны или нет. Используется в поиске лучшего хода

**Описание:**

Если передан параметр side со значением Black, то возвращается результат проверки на предмет того, что nEval больше oEval (так как чёрные в данной программе – максимизирующая сторона). В случае если передан параметр side со значением White – результат oEval больше nEval.

### Функция evalQuality

**Определение:**

float evalQuality(GameSituation\* situation)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, для которой мы ищем оценку позиции |

**Возвращаемое значение:**

Оценка текущей позиции в данной игровой ситуации

**Назначение:**

Предназначена для оценки текущей позиции в данной игровой ситуации во время поиска лучшего хода

**Описание:**

В начале значению оценки позиции присваивается значение 0

Затем программа проходится по всем шашкам белых, и вычитает из оценки значение kingCheckerCost, если текущая шашка – дамка; если же шашка является обычной, то из оценки позиции вычитается значение regularCheckerCost, умноженное на коэффициент, взятый из массива qualityFactor, который берётся исходя из того, насколько шашка близка к дамочным полям.

То же самое проводится и для чёрных, однако значения уже прибавляются к итоговой оценке.

В конце возвращается итоговое вычисленное значение оценки.

### Функция getVictimMarker

**Определение:**

PathMapMarkers getVictimMarker(Color victimColor, CheckerType victimType)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Color | victimColor | Цвет жертвы, для которой мы получаем метку |
| CheckerType | victimType | Тип жертвы, для которой мы получаем метку |

**Возвращаемое значение:**

Метка жертвы PathMapMarkers

**Назначение:**

Предназначена для заполнения карты путей по ходам на взятие. Метка предназначена для отрисовки «карты» путей в интерфейсе

**Описание:**

Возвращает метку жертвы, определённую параметрами victimColor (белый или чёрный) и victimType (обычная или дамка).

### Функция updateBoardRender

**Определение:**

void updateBoardRender(Board\* board)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Board\* | board | Указатель на доску, шаблон которой нам надо обновить |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Требуется для обновления шаблона доски после совершения хода и/или удаления всех «помеченных на смерть» шашек, чтобы затем отрисовать её в интерфейсе игры

**Описание:**

Сначала происходит первичное заполнение шаблона доски значениями EMPTY\_BLACK (для клеток, сумма координат которых чётная) и EMPTY\_WHITE (для клеток, сумма координат которых нечётная)

Затем для каждой стороны производится проходка по шашкам, и на основании их координат и типов производится присвоение элементам шаблона доски нужных значений.

### Функция fillPathMap

**Определение:**

void fillPathMap(GameSituation\* situation, Coordinates source)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в доске которой нам необходимо заполнить карту путей |
| Coordinates | source | Координаты начала пути, на основании которой мы заполняем «карту путей» |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для заполнения «карты путей» доски на основании выбранной клетки доски.

**Описание:**

Функция «проходится» по всем ходам на взятие, хранящимся в буфере regularMovesBuffer, если они есть, и если координаты начала пути совпадают с переданными, то функция ставит на место конца хода метку Destination, а на место жертвы – метку, соответствующей жертве, записанной в текущем рассматриваемом ходе. Метка определяется функцией getVictimMarker.

Если же в буфере нет ходов на взятие, то происходит тоже самое, но для обычных ходов. В этом случае жертвы не рассматриваются.

### Функция getCheckerIndexByCoordsAndColor

**Определение:**

int getCheckerIndexByCoordsAndColor(GameSituation\* situation, int cx, int cy, Color color)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Игровая ситуация, в доске которой мы ищем нужную шашку |
| int | cx | Абсцисса шашки, которую необходимо найти |
| int | cy | Ордината шашки, которую необходимо найти |
| Color | color | Цвет шашки, которую необходимо найти |

**Возвращаемое значение:**

Индекс шашки в массиве шашек данного цвета, имеющей данные координаты.

**Назначение:**

Поиск индекса шашки в массиве шашек доски

**Описание:**

Для всех шашек данного цвета сравнивает координаты данной шашки с переданными, и если они равны, то возвращает текущее значение счетчика цикла «проходки».

### Функция sideColor

**Определение:**

Color sideColor(GameSituation\* situation, int cx, int cy)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в доске которой мы ищем нужный цвет |
| int | cx | Абсцисса клетки поля доски |
| int | cy | Ордината клетки поля доски |

**Возвращаемое значение:**

Цвет шашки, которая стоит на указанном поле доски

**Назначение:**

Требуется для определения цвета шашки, который стоит на указанной клетке. Используется в некоторых алгоритмах поиска ходов

**Описание:**

Получает состояние шаблона клетки доски по координатам (cx, cy), и возвращает White, если состояние данной клетки – REG\_WHITE или KING\_WHITE, или Black, если состояние данной клетки равно KING\_BLACK или REG\_BLACK.

### Функция negateColor

**Определение:**

Color negateColor(Color color)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Color | color | Цвет, который необходимо инвертировать |

**Возвращаемое значение:**

Инвертированный цвет переданного цвета

**Назначение:**

Инвертирование переданного цвета

**Описание:**

Возвращает White, если был передан цвет Black, или возвращает Black, если был передан цвет White

### Функция getCheckerTypeOnBoard

**Определение:**

CheckerType getCheckerTypeOnBoard(GameSituation\* situation, int cx, int cy)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в доске которой мы определяем тип шашки |
| int | cx | Абсцисса определяемой шашки |
| int | cy | Ордината определяемой шашки |

**Возвращаемое значение:**

Тип шашки, расположенной на данной клетке доски

**Назначение:**

Используется только в поиске ходов на взятие для записи типа жертвы, чтобы потом, при необходимости, ход на взятие можно было корректно отменить, восстановя жертву взятия.

**Описание:**

Получает состояние шаблона клетки доски по координатам (cx, cy), и возвращает Regular, если состояние данной клетки – REG\_WHITE или REG\_BLACK, или King, если состояние данной клетки равно KING\_BLACK или KING\_WHITE.

### Функция isAFriend

**Определение:**

bool isAFriend(GameSituation\* situation, Color forWhichSide, int tx, int ty)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Игровая ситуация, в доске которой мы осуществляем «проверку на друга» |
| Color | forWhichSide | Цвет, для которого мы осуществляем «проверку на друга» |
| int | tx | Абсцисса гипотетического друга |
| int | ty | Ордината гипотетического друга |

**Возвращаемое значение:**

Результат «проверки на друга»

**Назначение:**

Используется в некоторых алгоритмах поиска ходов, чтобы случайно не съесть «свою» шашку

**Описание:**

Получает состояние шаблона доски по координатам (tx, ty), и возвращает true, если цвет стороны, для которой мы проверяем - белый, и состояние клетки равно REG\_WHITE или KING\_WHITE, или если цвет стороны, для которой мы проверяем - чёрный, и состояние клетки равно REG\_BLACK или KING\_BLACK. В остальных случаях возвращает false

### Функция isMarkedForDeath

**Определение:**

bool isMarkedForDeath(GameSituation\* situation, Color checkerColor, int mx, int my)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Игровая ситуация, в которой мы определяем шашку на предмет наличия метки на смерть |
| Color | checkerColor | Цвет определяемой шашки |
| int | mx | Абсцисса определяемой шашки |
| int | my | Ордината определяемой шашки |

**Возвращаемое значение:**

Результат проверки шашки на наличие «метки на смерть»

**Назначение:**

Используется в алгоритмах поиска последовательностей ходов на взятие и некоторых других функциях, чтобы не съесть ту шашку, которую мы уже перескочили во время взятия.

**Описание:**

Если клетка с координатами (mx, my) пуста, то возвращает false.

Иначе находит индекс шашки с данным цветом и данными координатами, после чего возвращает значение поля isMarkedForDeath для шашки с найденным индексом

### Функция isAVictim

**Определение:**

bool isAVictim(GameSituation\* situation, Color forWhichSide, int tx, int ty)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Игровая ситуация, в которой мы осуществляем «проверку на жертву» |
| Color | forWhichSide | Цвет, для которого мы осуществляем «проверку на жертву» |
| int | tx | Абсцисса гипотетической жертвы |
| int | ty | Ордината гипотетической жертвы |

**Возвращаемое значение:**

Результат «проверки на жертву»

**Назначение:**

Используется в некоторых алгоритмах поиска ходов, а также в функции определения расстояния до жертвы

**Описание:**

Получает состояние шаблона доски по координатам (tx, ty), и возвращает true, если цвет стороны, для которой мы проверяем - чёрный, и состояние клетки равно REG\_WHITE или KING\_WHITE, или если цвет стороны, для которой мы проверяем - белый, и состояние клетки равно REG\_BLACK или KING\_BLACK. В остальных случаях возвращает false

### Функция isNotOnKBBorder

**Определение:**

bool isNotOnKBBorder(int y, Color side)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| int | y | Ордината проверяемой шашки |
| Color | side | Цвет проверяемой шашки |

**Возвращаемое значение:**

Результат проверки условия нахождения на пред-дамочной горизонтали

**Назначение:**

Предназначена для проверки того, не ищем ли мы ходы для обычной шашки, стоя при этом на «пред-дамочной» горизонтали

**Описание:**

Возвращает результат вычисления логического выражения y < 6, если проверяемая шашка – белая, иначе результат выражения y > 1.

### Функция longMoveLookingFlag

**Определение:**

bool longMoveLookingFlag(int x, int y, Coordinates direction)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| int | x | Абсцисса проверяемой координаты |
| int | y | Ордината проверяемой координаты |
| Coordinates | direction | Направление проверки |

**Возвращаемое значение:**

Флаг проверки условия нахождения на границе при движении по данному направлению

**Назначение:**

Используется в некоторых алгоритмах поиска ходов на проверку того, не дошли ли мы до границы поля в ходе поиска

Описание:

* Если направление проверки – NORTH\_EAST, то возвращает результат вычисления логического выражения !(x >= 7 || y >= 7)
* Если направление проверки – NORTH\_WEST, то возвращает результат вычисления логического выражения !(x <= 0 || y >= 7)
* Если направление проверки – SOUTH\_EAST, то возвращает результат вычисления логического выражения !(x >= 7 || y <= 0)
* Если направление проверки – SOUTH\_WEST, то возвращает результат вычисления логического выражения !(x <= 0 || y <= 0)

### Функция regMoveLookingFlag

**Определение:**

bool regMoveLookingFlag(int x, int y, Coordinates direction)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| int | x | Абсцисса проверяемой координаты |
| int | y | Ордината проверяемой координаты |
| Coordinates | direction | Направление проверки |

**Возвращаемое значение:**

Значение флага проверки гипотетического выхода за границы массива шаблона доски.

**Назначение:**

Предназначена для проверки координат клетки на то, не выйдем ли мы в ходе просмотра по данному направлению за пределы массива.

**Описание:**

Если направление проверки – NORTH\_EAST (или SOUTH\_EAST), возвращается результат проверки абсциссы на предмет того, является ли она меньше 7

Иначе если направление проверки – NORTH\_WEST (SOUTH\_WEST), возвращается результат проверки абсциссы на предмет того, является ли она больше 1

### Функция findDistanceToVictim

**Определение:**

int findDistanceToVictim(GameSituation\* situation, int x, int y, Color forWhichSide, Coordinates direction)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Игровая ситуация, в которой мы ищем расстояние до жертвы |
| int | x | Абсцисса начала поиска |
| int | y | Ордината начала поиска |
| Coordinates | direction | Направление поиска |
| Color | forWhichSide | Цвет шашки, для которой мы ищем жертву |

**Возвращаемое значение:**

Расстояние до жертвы для данной шашки, расположенной на данной абсциссе и ординате по данному направлению

**Назначение:**

Предназначена для поиска расстояния до жертвы для данной шашки. Используется для поиска ходов на взятие дамкой.

**Описание:**

Устанавливаем значение сдвига равным -1

Пока флаг (longMoveLookingFlag) активен, делаем следующее:

* Если на клетке, расположенной на расстоянии сдвига по данному направлению, находится «своя» шашка или помеченная на смерть шашка врага, то возвращаем -1
* Если же на этой клетке располагается жертва, то возвращаем величину сдвига
* Если же эта клетка пустая, то увеличиваем величину сдвига на 1

Если флаг был сброшен (был выполнен выход из цикла), то возвращаем -1

### Функция flushMoveBuffers

**Определение:**

inline void flushMoveBuffers(GameSituation\* situation)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Игровая ситуация, в которой мы очищаем буферы ходов |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для «очистки» буферов ходов игровой ситуации.

**Описание:**

Устанавливает полям tmBufferLen и rmBufferLen данной игровой ситуации нулевое значение.

### Функция removeChecker

**Определение:**

void removeChecker(Board\* board, int index, Color color)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Board\* | board | Указатель на доску, из которой мы убираем шашку |
| int | index | Индекс убираемой шашки |
| Color | color | Цвет убираемой шашки |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для удаления шашек с доски

**Описание:**

Если шашка с таким индексом и данным цветом может существовать, то смещает все шашки данного цвета с позиции в массиве на 1 больше, чем индекс данной шашки, на 1 элемент назад, после чего уменьшает количество шашек данного цвета на 1.

### Функция descendChecker

**Определение:**

void descendChecker(Checker\* checker)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Checker\* | checker | Указатель на шашку, которую мы понижаем до обычной шашки |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для «понижения» шашки до обычной

**Описание:**

Устанавливает значение поля type структуры checker равным Regular.

### Функция ascendChecker

**Определение:**

void ascendChecker(Checker\* checker)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Checker\* | checker | Указатель на шашку, которую мы проводим в дамки |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для «повышения» шашки до дамки

**Описание:**

Устанавливает значение поля type структуры checker равным King.

### Функция removeMarkedForDeath

**Определение:**

void removeMarkedForDeath(GameSituation\* situation, Color inWhere)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| Тип параметра | Имя параметра | Пояснение |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы убираем «помеченных на смерть» |
| Color | inWhere | Цвет шашек, среди которых мы ищем «помеченных» |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Убирает все шашки, которые помечены на смерть в данной игровой ситуации

**Описание:**

Проходится по всем шашкам данного цвета, и, если среди них находит помеченную на смерть, удаляет её. После «проходки» происходит обновление шаблона доски

## Заголовочный файл clcengine\_inits.h

Данный заголовочный файл содержит определения тех функций, которые возвращают некоторые нуль-инициализированные («пустые») структуры.

### Функция getNullContainer

**Определение:**

SeqContainer getNullContainer(Color side)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Color | side | Цвет стороны, для которой необходимо создать пустой контейнер |

**Возвращаемое значение:**

Пустой «контейнер последовательности»

**Назначение:**

Предназначена для создания пустого «контейнера последовательности» для данной стороны. Используется во время поиска лучшего хода.

**Описание:**

Создаётся новая структура SeqContainer container, поля которой принимают значения, в соответствии с таблицей

|  |  |
| --- | --- |
| **Имя поля** | **Значение** |
| eval | getVictoryEvalFor(negateColor(side)) |
| seqNumberToDo | -1 |
| takingSequence.tmsCount | 0 |
| regMoveSequence.rmsCount | 0 |

Затем этот контейнер возвращается как результат выполнения функции

### Функция getNullPath

**Определение:**

TakingSequence getNullPath()

**Параметры:**

Без параметров

**Возвращаемое значение:**

Пустая последовательность ходов на взятие

**Назначение:**

Нужна для передачи пустого пути в первом вызове поиска всех последовательностей ходов на взятие для данной шашки

**Описание:**

Создаёт пустую последовательность ходов на взятие, со значением поля tmsCount равным 0, после чего возвращает эту последовательность

### Функция initiateChecker

**Определение:**

void initiateChecker (Checker\* checker, int x, int y, Color color)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Checker\* | checker | Указатель на шашку, которую необходимо инициализировать |
| int | x | Абсцисса инициализируемой шашки |
| int | y | Ордината инициализируемой шашки |
| Color | color | Цвет инициализируемой шашки |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для инициализации шашки с данными координатами и цветом

**Описание:**

Устанавливает полям инициализируемой шашки значения, в соответствии с таблицей

|  |  |
| --- | --- |
| **Имя поля** | **Значение** |
| coordinates.x | x |
| coordinates.y | y |
| color | color; |
| type | Regular; |
| markedForDeath | false; |

### Функция resetPathMap

**Определение:**

void resetPathMap(Board\* board)

Параметры:

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Board\* | board | Указатель на доску, «катру путей» которой мы сбрасываем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для сброса «карты путей» доски

**Описание:**

Устанавливает всем элементам «карты путей» доски значение NoMove

### Функция createANewBoard

**Определение:**

Board createANewBoard()

**Параметры:**

Без параметров

**Возвращаемое значение:**

Новая доска

**Назначение:**

Предназначена для создания новой игровой доски с инициализированными шашками, которые стоять в соответствии с правилами игры в русские шашки

**Описание:**

Создаёт новую структуру Board newBoard.

Устанавливает её полям checkersCount[Black] и checkersCount[White] значение 12

Инициализирует все шашки в соответствии с начальной позицией игры в шашки, затем сбрасывает карту путей доски, обновляет её шаблон, и возвращает её как результат выполнения данной функции.

### Функция makeNullGameSituation

**Определение:**

GameSituation makeNullGameSituation(Color playerSide)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Color | playerSide | Цвет стороны игрока |

**Возвращаемое значение:**

Новая игровая ситуация

**Назначение:**

Предназначена для создания пустой игровой ситуации во время создания новой игры

**Описание:**

Создаётся новая структура GameSituation gameSituation, поля которой принимают значения, в соответствии с таблицей

|  |  |
| --- | --- |
| **Имя поля** | **Значение** |
| board | createANewBoard(); |
| playerSide | playerSide |
| rmBufferLen | 0 |
| tmBufferLen | 0 |
| tmsCount | 0 |
| gameSirmsCoun | 0 |
| takingSequences[i].tmsCount (для i от 0 до 16) | 0 |

Затем эта структура возвращается как результат выполнения функции

### Функция createANewGame

**Определение:**

Game createANewGame(Color playerSide, Color firstMove, GameType type)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| Color | playerSide | Цвет стороны игрока |
| Color | firstMove | Цвет стороны, которая ходит первая |
| GameType | type | Тип игры |

**Возвращаемое значение:**

Новая игра

**Назначение:**

Предназначена для создания новой игры

**Описание:**

Создаёт новую структуру Game newGame, поля которой принимают значения, в соответствии с таблицей

|  |  |
| --- | --- |
| **Имя поля** | **Значение** |
| situation | makeNullGameSituation(playerSide) |
| type | type |
| situation.playerSide | playerSide |

Затем эта игра возвращается как результат выполнения функции

## Заголовочный файл clcengine\_move\_searhing.h

Данный заголовочный файл содержит функции поиска ходов.

### Функция findAllKingMovesForOneOnDir

**Определение:**

void findAllKingMovesForOneOnDir(GameSituation\* situation, Color checkerColor, int checkerIndex, Coordinates directionShift)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем ходы для дамки по данному направлению |
| Color | checkerColor | Цвет шашки, для которой ищем |
| int | checkerIndex | Индекс шашки, для которой ищем |
| Coordinates | directionShift | Направление поиска |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех обычных ходов для дамки данного цвета данного индекса для данной игровой ситуации по данному направлению

**Описание:**

Создаётся новый ход. В координаты его начала записываются координаты шашки, указанной данным цветом и данным индексом, также в него записывается индикатор того, что этот ход не «с превращением»

Устанавливается флаг поиска (longMoveLookingFlag), устанавливается значение сдвига равным 1.

Пока флаг истинен, выполняется следующее:

1. Если клетка, находящееся на расстоянии сдвига по данному направлению, пуста, то в координаты конца созданного хода записываются координаты пустой клетки, находящейся на расстоянии сдвига по данному направлению. Этот ход копируется в конец буфера обычных ходов игровой ситуации. Затем переустанавливается значение флага (longMoveLookingFlag)
2. Иначе значение флага сбрасывается (присваивается значение false)
3. Увеличивается значение сдвига на 1

### Функция findAllKingMovesForOne

**Определение:**

void findAllKingMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем ходы для дамки по данному направлению |
| Color | checkerColor | Цвет шашки, для которой ищем |
| int | checkerIndex | Индекс шашки, для которой ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех обычных ходов для дамки данного цвета данного индекса для данной игровой ситуации по всем направлениям

**Описание:**

Если шашка, определённая данным цветом и данным индексом, является дамкой, то осуществляется последовательный вызов функций findAllKingMovesForOneOnDir с теми же параметрами, но в качестве направления устанавливаются NORTH\_EAST, NORTH\_WEST, SOUTH\_EAST и SOUTH\_WEST соответственно.

### Функция findAllRegularMovesForOneOnDir

**Определение:**

void findAllRegularMovesForOneOnDir(GameSituation\* situation, Color checkerColor, int checkerIndex, Coordinates direction

**Параметры:**

|  |  |  |
| --- | --- | --- |
| Тип параметра | Имя параметра | Пояснение |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем ходы для обычной шашки по данному направлению |
| Color | checkerColor | Цвет шашки, для которой ищем |
| int | checkerIndex | Индекс шашки, для которой ищем |
| Coordiantes | direction | Направление поиска |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех обычных ходов для обычной шашки данного цвета данного индекса в данной игровой ситуации по данному направлению

**Описание:**

Создаётся новый ход. В координаты его начала записываются координаты шашки, указанной данным цветом и данным индексом, также в него записывается индикатор того, что этот ход не «с превращением»

Если значение regMoveLookingFlag – истинно, и шашка не стоит на горизонтали, предшествующей дамочной, что проверяется, пуста ли клетка, стоящая на расстоянии 1 по данному направлению от данной шашки.

Если эта клетка пуста, то в координаты конца хода записываются координаты этой клетки, после чего ход копируется в конец буфера обычных ходов

### Функция findAllRegularMovesForOne

**Определение:**

void findAllRegularMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем ходы для обычной шашки по всем направлениям |
| Color | checkerColor | Цвет шашки, для которой ищем |
| int | checkerIndex | Индекс шашки, для которой ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех обычных ходов шашки, определённой данным цветом и данным индексом в данной игровой ситуации

**Описание:**

Если шашка – белая, то осуществляется последовательный вызов функций findAllRegularMovesForOneOnDir с теми же параметрами, но в качестве направления указываются NORTH\_EAST и NORTH\_WEST. Если чёрная – то же самое, но направления - SOUTH\_EAST SOUTH\_WEST.

### Функция findAllTakingMovesForOne

**Определение:**

void findAllTakingMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем ходы на взятие для данной шашки по всем направлениям |
| Color | checkerColor | Цвет шашки, для которой ищем |
| int | checkerIndex | Индекс шашки, для которой ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех ходов на взятие для обычной шашки, определённой данным индексом и данным цветом для данной игровой ситуации по всем направлениям

**Описание:**

Все следующие действия выполняются в том случае, когда шашка с данным цветом и индексом – не дамка.

Создаётся новый ход на взятие и в координаты его начала записывается координата шашки, определённой данным цветом и данным индексом.

Далее происходит проверка на наличие пустой клетки и клетки с жертвой, координаты которой определяются следующим правилом:

* Если её ордината меньше 6 и её абсцисса меньше 6, то смотреть пустую клетку по координатам с абсциссой и ординатой на 1 больше, чем у текущей шашки, а жертву – на 2 больше
* Если её ордината меньше 6 и её абсцисса больше 1, то смотреть пустую клетку по координатам с абсциссой на 1 меньше и ординатой на 1 больше, чем у текущей шашки, а жертву – на 2 меньше и на 2 больше
* Если её ордината больше 1 и её абсцисса меньше 6, то смотреть пустую клетку по координатам с абсциссой на 1 больше и ординатой на 1 меньше, чем у текущей шашки, а жертву – на 2 больше и на 1 меньше
* Если её ордината больше 1 и её абсцисса больше 1, то смотреть пустую клетку по координатам с абсциссой и ординатой на 1 меньше, чем у текущей шашки, а жертву – на 2 меньше

На каждой ветке, удовлетворяющей условию, производится запись в координаты конца хода координаты найденной пустой клетки, в координаты жертвы – найденной жертвы, тип жертвы, её цвет, противоположный цвету текущей шашки и индикатор того, что это ход «с превращением».

После каждой итерации осуществляется добавление созданного хода в конец буфера ходов на взятие

### Функция findAllKingTakingMovesForOneOnDir

**Определение:**

void findAllKingTakingMovesForOneOnDir(GameSituation\* situation, Color checkerColor, int checkerIndex, Coordinates directionShift

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем ходы на взятие для дамки по данному направлению |
| Color | checkerColor | Цвет шашки, для которой ищем |
| int | checkerIndex | Индекс шашки, для которой ищем |
| Coordiantes | directionShift | Направление поиска |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех ходов на взятие для дамки данного цвета данного индекса для данной игровой ситуации по данному направлению

**Описание:**

Создаётся новый ход на взятие и в координаты его начала записывается координата шашки, определённой данным цветом и данным индексом. В ход также записывается индикатор того, что данный ход – не с «превращением». В сторону жертвы записывается цвет, противоположный цвету данной шашки.

Находится дистанция до жертвы от места данной шашки по данному направлению.

Если дистанция не равна -1, производятся следующие действия:

1. В координаты жертвы хода записываются координаты клетки, находящейся от клетки текущей шашки на расстоянии, равном расстоянию до жертвы. В ход также записывается и её тип.
2. Устанавливается флаг поиска (longMoveLookingFlag) и значение сдвига равным 1
3. Пока флаг истинен:
   1. Если клетка, находящаяся на расстоянии, равном сумме расстоянии до жертвы и сдвига, не пуста, то записываем её координаты в координаты конца хода, добавляем его в конец буфера ходов на взятие и переустанавливаем флаг поиска.
   2. Иначе сбрасываем флаг поиска
   3. Увеличиваем величину сдвига на 1

### Функция findAllKBMovesForOne

**Определение:**

void findAllKBMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем ходы для дамки по данному направлению |
| Color | checkerColor | Цвет шашки, для которой ищем |
| int | checkerIndex | Индекс шашки, для которой ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех ходов «с превращением» для обычной шашки данного цвета данного индекса для данной игровой ситуации

**Описание:**

Все следующие действия выполняются в том случае, когда шашка с данным цветом и индексом – не дамка.

Создаётся новый ход. В координаты его начала записываются координаты шашки, указанной данным цветом и данным индексом, также в него записывается индикатор того, что этот ход «с превращением»

Далее происходит тоже самое, что и при поиске обычных ходов, однако дополнительно перед этим делается проверка координат шашки: если шашка белая и её ордината равна 6 или если шашка чёрная и её ордината равна 1

### Функция findAllKingTakingMovesForOne

**Определение:**

void findAllKingTakingMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем ходы для дамки по данному направлению |
| Color | checkerColor | Цвет шашки, для которой ищем |
| int | checkerIndex | Индекс шашки, для которой ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех ходов на взятие для дамки данного цвета данного индекса для данной игровой ситуации по всем направлениям

**Описание:**

Если шашка, определённая данным цветом и данным индексом, является дамкой, то осуществляется последовательный вызов функций findAllKingTakingMovesForOneOnDir с теми же параметрами, но в качестве направления устанавливаются NORTH\_EAST, NORTH\_WEST, SOUTH\_EAST и SOUTH\_WEST соответственно.

## Заголовочный файл clcengine\_moving.h

Данный заголовочный файл содержит функции совершения различных ходов.

### Функция makeAMove

**Определение:**

int makeAMove(GameSituation\* situation, Move move)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Указатель на игровую ситуацию, в которой мы делаем ход |
| Move | move | Совершаемый ход |

**Возвращаемое значение:**

0, 1 или 2, не используется?

**Назначение:**

Предназначена для совершения обычного хода

**Описание:**

Сначала функция ищет цвет и индекс шашки, которая стоит на месте, указанном в координатах начала хода, затем шашке найденного цвета и найденного индекса присваивается значение координат, указанное в координате конца хода. Если в данных хода указывается, что после хода должно быть превращение в дамку, то для данной шашки осуществляется превращение.

Обновляется шаблон доски, очищаются все буферы ходов, и если после хода должно быть превращение в дамку (в противном случае возвращается 0), то ищутся все ходы на взятие для этой шашки, если их нет, то ищутся обычные ходы и возвращается 2. Если же есть, то буфер ходов на взятие всё равно сбрасывается и возвращается 1.

### Функция makeATakingMove

**Определение:**

int makeATakingMove(GameSituation\* situation, TakingMove move)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Указатель на игровую ситуацию, в которой мы делаем ход |
| TakingMove | move | Совершаемый ход |

**Возвращаемое значение:**

0, либо длина буфера ходов на взятие

**Назначение:**

Предназначена для совершения хода на взятие

**Описание:**

Сначала функция ищет цвет и индекс перемещаемой шашки, которая стоит на месте, указанном в координатах начала хода, то же самое совершается и для жертвы, указанной в данных хода. Затем перемещаемой шашке найденного цвета и найденного индекса присваивается значение координат, указанное в координате конца хода. Если в данных хода указывается, что после хода должно быть превращение в дамку, то для данной шашки осуществляется превращение. Для найденной шашки-жертвы в её данных указывается, что она «помечена на смерть»

Обновляется шаблон доски, очищаются все буферы ходов.

Если сходила дамка, то ищутся все ходы на взятие для этой дамки, иначе (при условии также, что ход не предполагает превращения в дамку) ищутся все ходы на взятие для этой обычной шашки. Иначе возвращается 0.

После всего, возвращается длина буфера ходов на взятие.

### Функция cancelAMove

**Определение:**

int cancelAMove(GameSituation\* situation, Move move)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Указатель на игровую ситуацию, в которой мы отменяем ход |
| Move | move | Отменяемый ход |

**Возвращаемое значение:**

-1 или 0

**Назначение:**

Предназначена для отмены обычного хода

**Описание:**

Сначала функция ищет цвет и индекс шашки, которая стоит на месте, указанном в координатах конца хода, затем шашке найденного цвета и найденного индекса присваивается значение координат, указанное в координате начала хода. Если в данных хода указывается, что после хода должно быть превращение в дамку, то для данной шашки осуществляется её «понижение» до обычной шашки.

Обновляется шаблон доски.

Если в данных хода указано, что после хода должно быть превращение в дамку, то возвращается -1, иначе 0.

### Функция cancelATakingMove

**Определение:**

int cancelATakingMove(GameSituation\* situation, TakingMove move, bool emulated = false)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Указатель на игровую ситуацию, в которой мы делаем ход |
| TakingMove | move | Совершаемый ход |
| bool | emulated | Флаг того, не является ли отменяемый ход сэмулированным (используется только во время поиска последовательностей ходов на взятие). По умолчанию false |

**Возвращаемое значение:**

-1 или 0

**Назначение:**

Предназначена для отмены хода на взятие

**Описание:**

Сначала функция ищет цвет и индекс шашки, которая стоит на месте, указанном в координатах конца хода, затем шашке найденного цвета и найденного индекса присваивается значение координат, указанное в координате начала хода. Если в данных хода указывается, что после хода должно быть превращение в дамку, то для данной шашки осуществляется её «понижение» до обычной шашки.

Если отменяемый ход не сымитирован, то происходит восстановление жертвы из данных хода и её добавление в массив шашек той стороны, жертву которой мы восстанавливаем. Иначе снимаем «метку смерти» с той шашки, которая стоит на месте жертвы, указанной в данных хода.

Обновляется шаблон доски.

Если в данных хода указано, что после хода должно быть превращение в дамку, то возвращается -1, иначе 0.

## Заголовочный файл clcengine\_sequence\_searching.h

Данный заголовочный файл содержит функции поиска последовательностей (цепочек) ходов.

### Функция findRegularMoveSequenceForOne

**Определение:**

void findRegularMoveSequenceForOne(GameSituation\* situation, Color checkerColor, int checkerIndex)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем последовательность обычных ходов |
| Color | checkerColor | Цвет шашки, для которой мы ищем |
| int | checkerIndex | Индекс шашки, для которой мы ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех последовательностей обычных ходов для одной шашки

**Описание:**

Сначала функция ищет для данной шашки все возможные ходы.

Если буфер обычных ходов непустой, то всё содержимое буфера копируется в новый буфер, равно как и их количество, после чего запускается «проходка» по всем ходам в буфере:

1. Если этот ход «с превращением», то…
   1. Очищается буфер обычных ходов (НЕ НОВЫЙ!)
   2. Этот ход совершается
   3. Если нет ходов на взятие и буфер обычных ходов не пустой, то в буфер последовательности обычных ходов добавляются все последовательности вида «хо с «превращением» - «ход дамки после превращения» для всех найденных обычных ходов дамки
   4. Иначе в буфер последовательностей обычных ходов добавляется последовательность, состоящая только из одного этого хода
2. Иначе совершается то же самое, что и в пункте 1.4.

После этого созданный буфер удаляется

### Функция findAllRegularMoveSequences

**Определение:**

void findAllRegularMoveSequences(GameSituation\* situation, Color forWhichSide)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем последовательность обычных ходов |
| Color | forWhichSide | Цвет стороны, для которой мы ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех последовательностей обычных ходов для всех шашек некоторого цвета

**Описание:**

Для всех шашек данного цвета на доске осуществляется поиск всех обычных последовательностей ходов

### Функция findAllTakingSequencesForOne

**Определение:**

void findAllTakingSequencesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex, TakingSequence\* currentPath, bool afterMove = false)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем последовательность ходов на взятие |
| Color | checkerColor | Цвет шашки, для которой мы ищем |
| int | checkerIndex | Индекс шашки, для которой мы ищем |
| TakingSequence\* | currentPath | Текущий путь «взятий» |
| bool | afterMove | Индикатор того, не вошли ли мы в функцию после совершения хода |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех последовательностей ходов на взятие для одной шашки

**Описание:**

Если функция была вызвана не после совершения хода, то для данной шашки ищутся все ходы на взятие

Все найденные ходы на взятие копируются в новый буфер, равно как и их количество.

Для каждого хода на взятие в новом буфере делается следующее:

1. Он добавляется в конец текущего пути «взятий»
2. Совершается текущий ход на взятие, результат выполнения соответствующей функции записывается в отдельную переменную
3. Если результат нулевой, то копируем содержимое текущего пути «взятий» в конец буфера последовательностей ходов на взятие
4. Иначе рекурсивно входим в findAllTakingSequencesForOne, с теми же параметрами, то параметр afterMove принимает значение true
5. Отменяется совершённый ход на взятие с указанием того, что этот ход сымитирован
6. Удаляем этот ход из конца текущего пути

После этого удаляем созданный буфер

### Функция findAllTakingMoveSequences

**Определение:**

void findAllTakingMoveSequences(GameSituation\* situation, Color forWhichSide)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем последовательность обычных ходов |
| Color | forWhichSide | Цвет стороны, для которой мы ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех последовательностей ходов на взятие для всех шашек некоторого цвета

**Описание:**

Для всех шашек данного цвета на доске осуществляется поиск всех последовательностей ходов на взятие

### Функция findAllMoves

**Определение:**

inline void findAllMoves(GameSituation\* situation, Color forWhichSide)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituation\* | situation | Указатель на игровую ситуацию, в которой мы ищем последовательность обычных ходов |
| Color | forWhichSide | Цвет стороны, для которой мы ищем |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для поиска всех возможных последовательностей ходов на взятие для всех шашек данного цвета.

**Описание:**

Сначала осуществляется поиск всех последовательностей ходов на взятие для данной игровой ситуации и для всех шашек данного цвета.

Если последовательности ходов на взятие не были найдены, то осуществляется поиск всех последовательностей обычных ходов для данной игровой ситуации и для всех шашек данного цвета.

## Заголовочный файл clcengine\_sequence\_moving.h

Данный заголовочный файл содержит функции для совершения последовательности ходов.

### Функция makeARegMoveSequence

**Определение:**

void makeARegMoveSequence(GameSituation\* situation, RegMoveSequence regMoveSequence)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Игровая ситуация, в которой мы совершаем последовательность ходов |
| RegMoveSequence | regMoveSequence | Совершаемая последовательность ходов |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для совершения последовательности обычных ходов

**Описание:**

Для каждого хода, хранящегося в последовательности, от первого до последнего, осуществляется его «совершение»

### Функция cancelARegMoveSequence

**Определение:**

void cancelARegMoveSequence(GameSituation\* situation, RegMoveSequence regMoveSequence)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Игровая ситуация, в которой мы совершаем последовательность ходов |
| RegMoveSequence | regMoveSequence | Отменяемая последовательность ходов |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для отмены последовательности обычных ходов

**Описание:**

Для каждого хода, хранящегося в последовательности, от последнего до первого, осуществляется его отмена

### Функция makeATakingMoveSequence

**Определение:**

void makeATakingMoveSequence(GameSituation\* situation, TakingSequence takingSequence)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Игровая ситуация, в которой мы отменяем последовательность ходов |
| TakingSequence | takingSequence | Совершаемая последовательность ходов на взятие |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для совершения последовательности ходов на взятие

**Описание:**

Для каждого хода, хранящегося в последовательности, от первого до последнего, осуществляется его «совершение»

### Функция cancelATakingMoveSequence

**Определение:**

void cancelATakingMoveSequence(GameSituation\* situation, TakingSequence takingSequence)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Игровая ситуация, в которой мы отменяем последовательность ходов на взятие |
| TakingSequence | takingSequence | Отменяемая последовательность ходов на взятие |

**Возвращаемое значение:**

Не возвращает

**Назначение:**

Предназначена для отмены последовательности ходов на взятие

**Описание:**

Для каждого хода, хранящегося в последовательности, от последнего до первого, осуществляется его отмена

## Заголовочный файл clcengine\_analyzer.h

Данный заголовочный файл содержит саму функцию поиска лучшего хода методом «Минимакс»

### Функция analyze

**Определение:**

SeqContainer analyze(GameSituation\* situation, Color side, int currentDepth, Difficulty maxDepth)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation\* | situation | Игровая ситуация, в которой мы ищем лучший ход |
| Color | side | Цвет стороны, для которой мы ищем лучший ход |
| int | currentDepth | Текущая глубина поиска |
| Difficulty | maxDepth | Максимальная глубина поиска |

**Возвращаемое значение:**

Лучшая последовательность ходов, хранимая в SeqContainer (вместе с её кодом)

**Назначение:**

Предназначена для поиска лучшего хода для данной игровой ситуации. Используется в игре против компьютера. Параметр maxDepth зависит от уровня сложности, на котором играет пользователь

**Описание:**

Сначала создаётся пустой SeqContainer toReturn, в котором будет хранится лучшая последовательность ходов. Затем инициализируется верхняя граница оценки topBorder, которая в начале равна «победному» значению оценки для противоположной стороны.

Производится поиск ходов. Если ходы не найдены, то производится возврат из функции структуры topBorder, в поле eval которой записывается topBorder.

Если же мы достигли максимальной глубины поиска, то присваиваем полю eval структуры topBorder значение, равное текущей оценке позиции, после чего возвращаем эту структуры.

Если последовательностей на взятие не было обнаружено, то осуществляется «проходка» по всем найденным обычным последовательностям ходов, иначе – по всем последовательностям ходов на взятие.

В ходе каждой «проходки» все найденные последовательности и их количество сохраняются в буфер, по которому в последствии осуществляется «проходка». Для каждой последовательности в буфере делаем следующее:

1. Совершаем текущую последовательность ходов
2. Очищаем все списки найденных последовательностей (flushSequenceLists)
3. Рекурсивно запускаем функцию analyze с параметрами текущей игровой ситуации, цветом, противоположным текущему, текущей глубиной на 1 больше, чем на момент вызова и той же максимальной глубиной поиска. Результат выполнения записываем в структуру SeqMoveContainer deepMoveAnalyzed
4. Отменяем совершённую последовательность ходов
5. Проверяем оценку позиции, хранящуюся в deepMoveAnalyzed. Если она лучше, чем текущая в toReturn, то записываем в этот контейнер нашу оценку и текущую рассматриваемую позицию. Если же равна, то случайно выбираем, оставить ли содержимое toReturn тем же, или заменить на текущую последовательность
6. Удаляем буфер
7. Возвращаем toReturn

Аналогично и для последовательностей ходов на взятие

### Функция getBestMove

**Определение:**

SeqContainer getBestMove(GameSituation situation, Color side, Difficulty depth)

**Параметры:**

|  |  |  |
| --- | --- | --- |
| **Тип параметра** | **Имя параметра** | **Пояснение** |
| GameSituiation | situation | Игровая ситуация, для которой мы ищем лучший ход |
| Color | side | Цвет стороны, для которой мы ищем ход |
| Difficulty | depth | Максимальная глубина поиска |

**Возвращаемое значение:**

SeqContainer с лучшим найденным ходом и кодом, говорящий компьютеру, которую последовательность ему стоит выполнить

**Назначение:**

Является обёрткой для функции analyze. Предназначена для поиска лучшего хода для данной игровой ситуации. Используется в игре против компьютера. Параметр maxDepth зависит от уровня сложности, на котором играет пользователь

**Описание:**

Создаётся копия игровой ситуации, а затем возвращается результат выполнения функции analyze, вызванной с параметрами указателя на копию игровой ситуации, тем же цветом, что был передан в getBestMove, 0 (текущая глубина) и depth (максимальная глубина)

# Блок-схемы некоторых алгоритмов

## Функция copyLevelOneMovesToBuffers

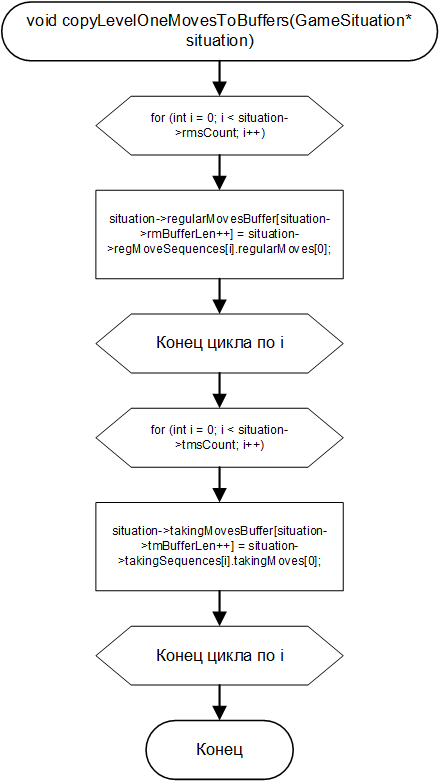


Рисунок 5.1.1. Блок схема функции copyLevelOneMovesToBuffers

## Функция evalQuality

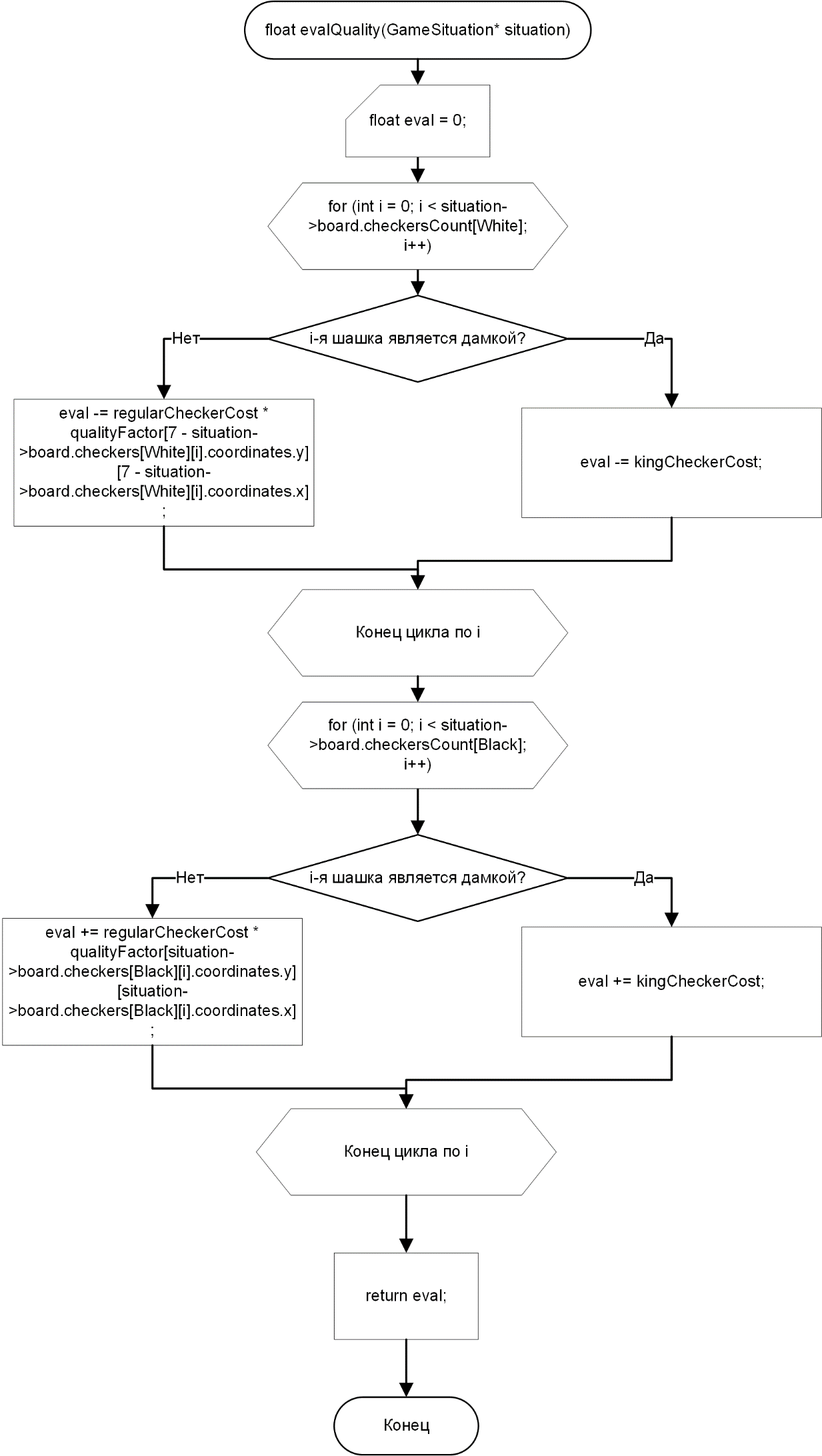


Рисунок 5.2.1. Блок-схема функции evalQuality

## Функция updateBoardRender

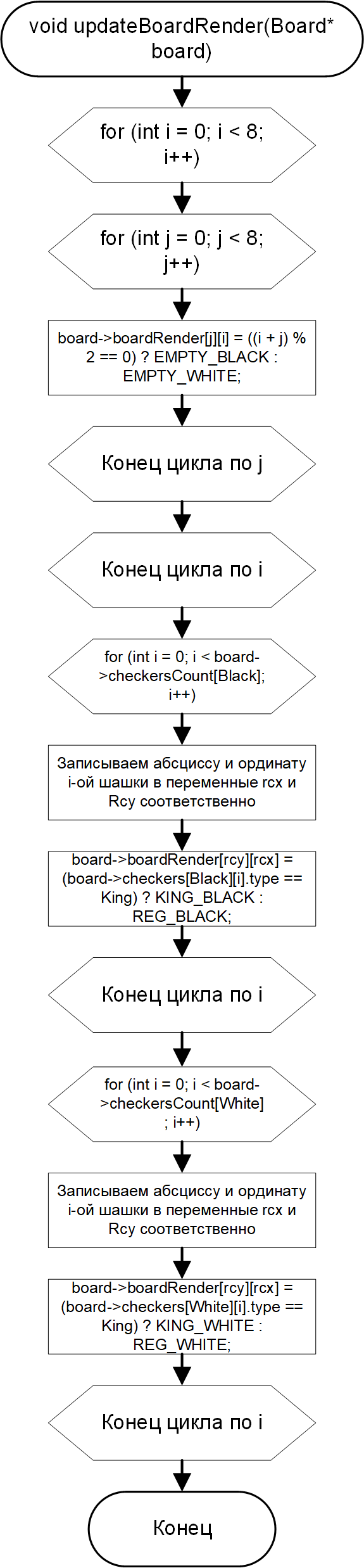


Рисунок 5.3.1. Блок-схема функции updateBoardRender

## Функция fillPathMap

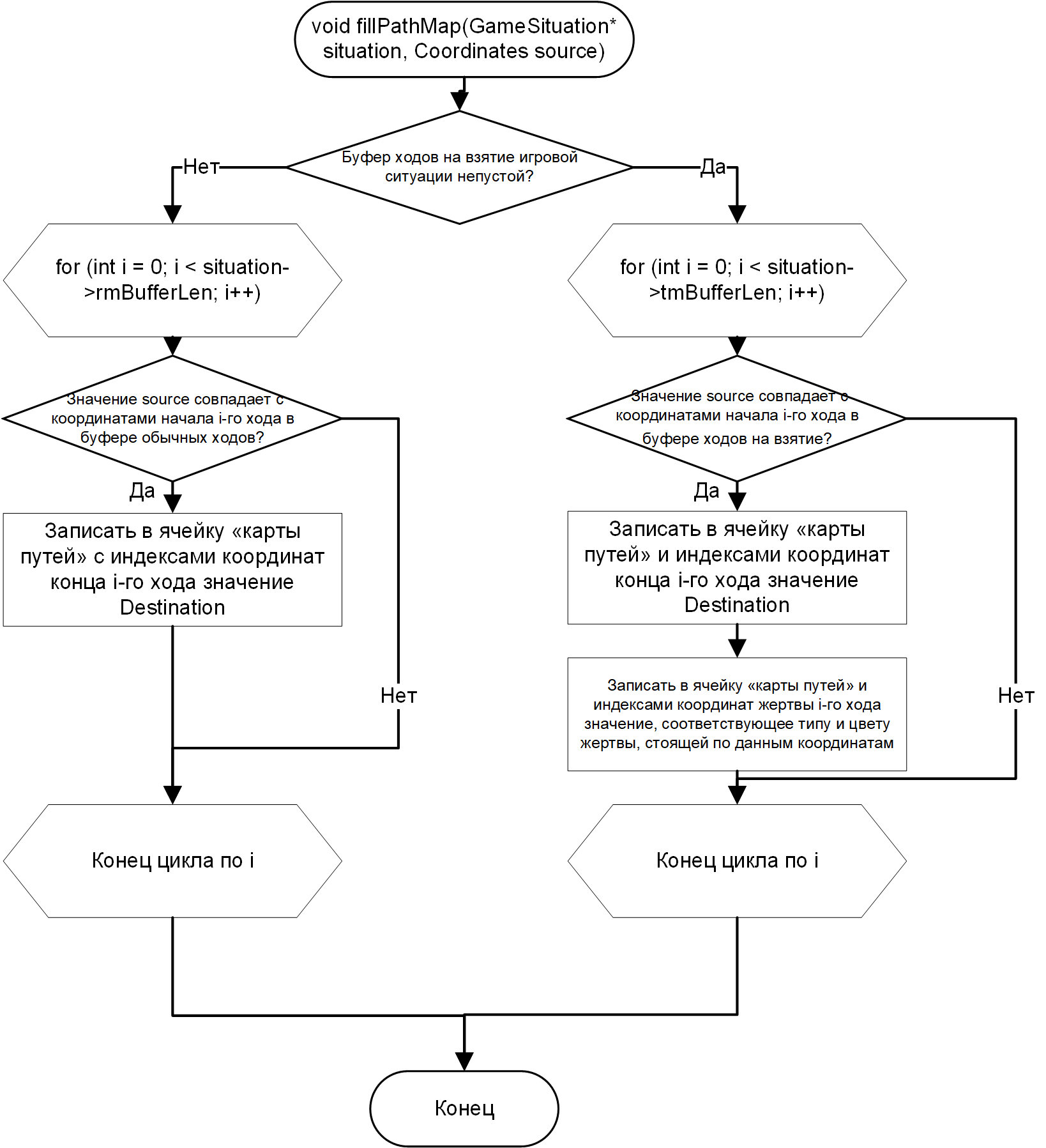


Рисунок 5.4.1. Блок-схема функции fillPathMap

## Функция findDistanceToVictim

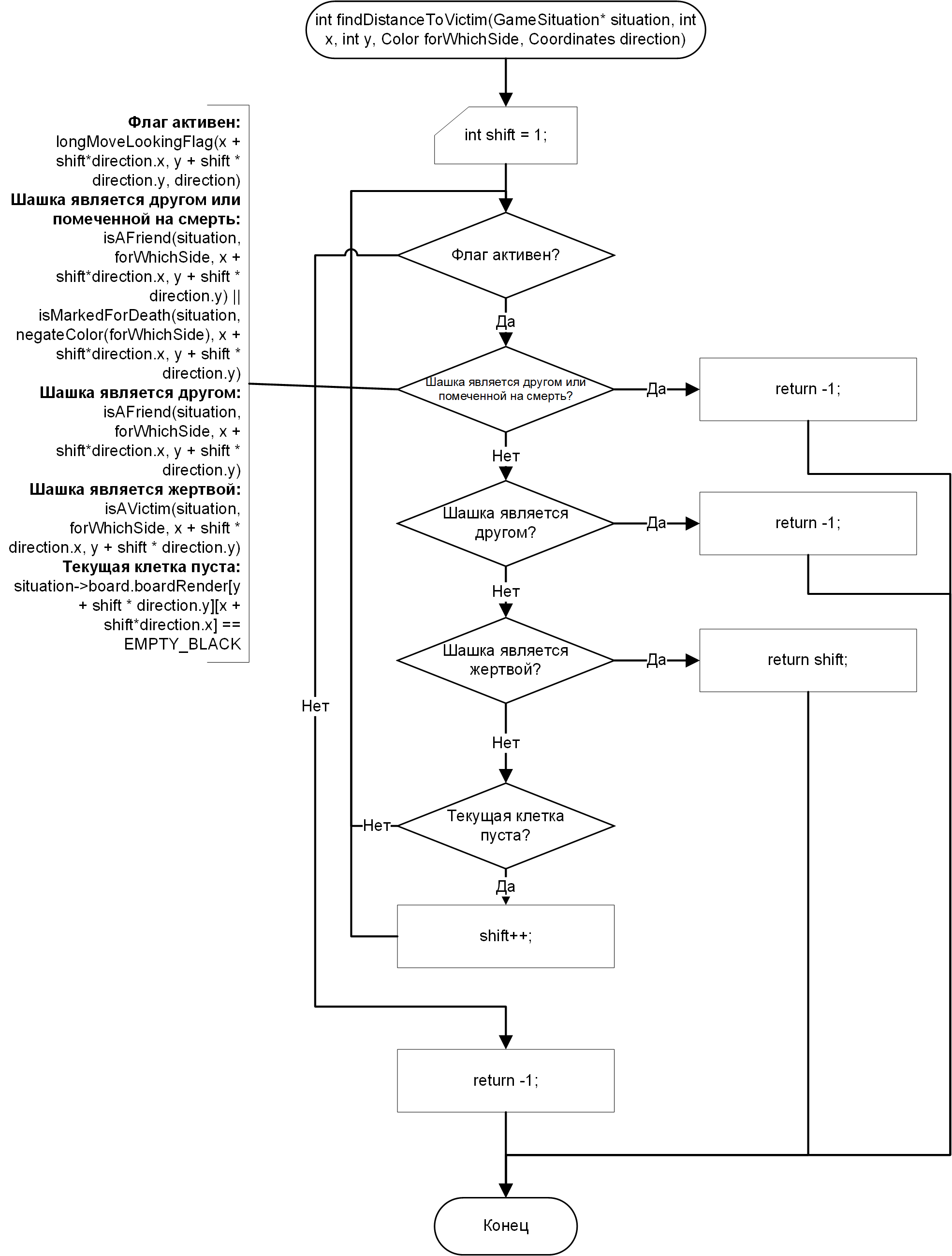


Рисунок 5.5.1. Блок-схема функции findDistanceToVictim

## Функция removeMarkedForDeath

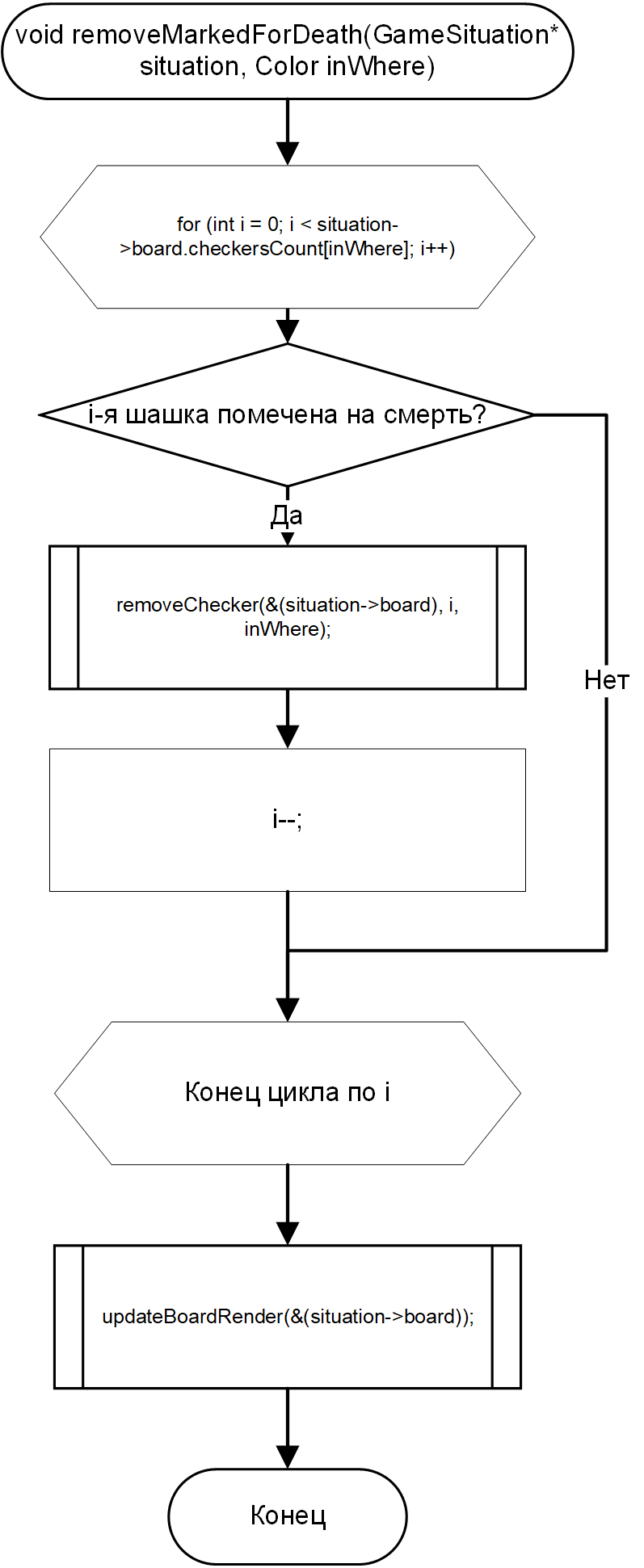


Рисунок 5.6.1. Блок-схема функции removeMarkedForDeath

## Функция findAllKingMovesForOneOnDir

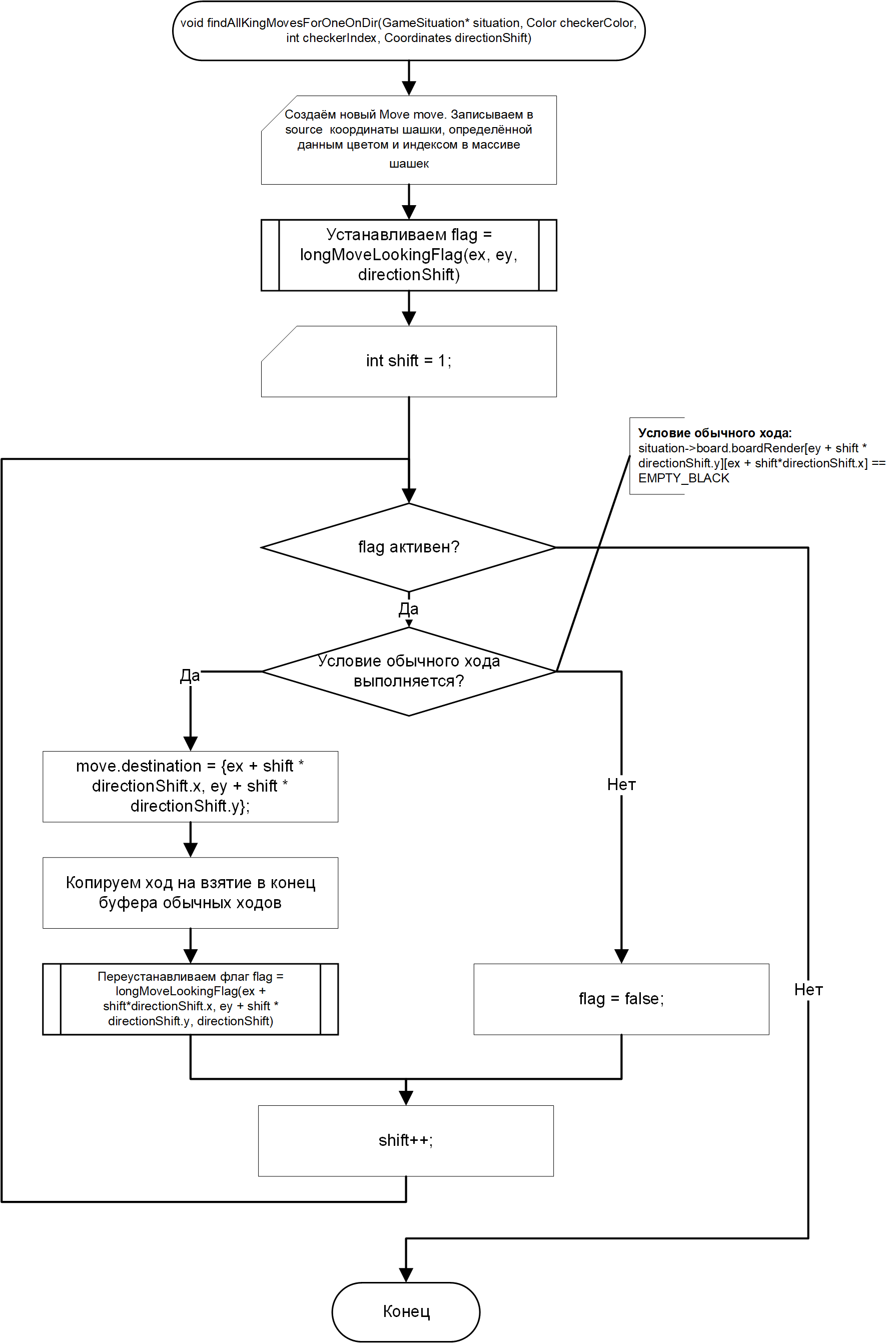


Рисунок 5.7.1. Блок-схема функции findAllKingMovesForOneOnDir

## Функция findAllRegularMovesForOneOnDir

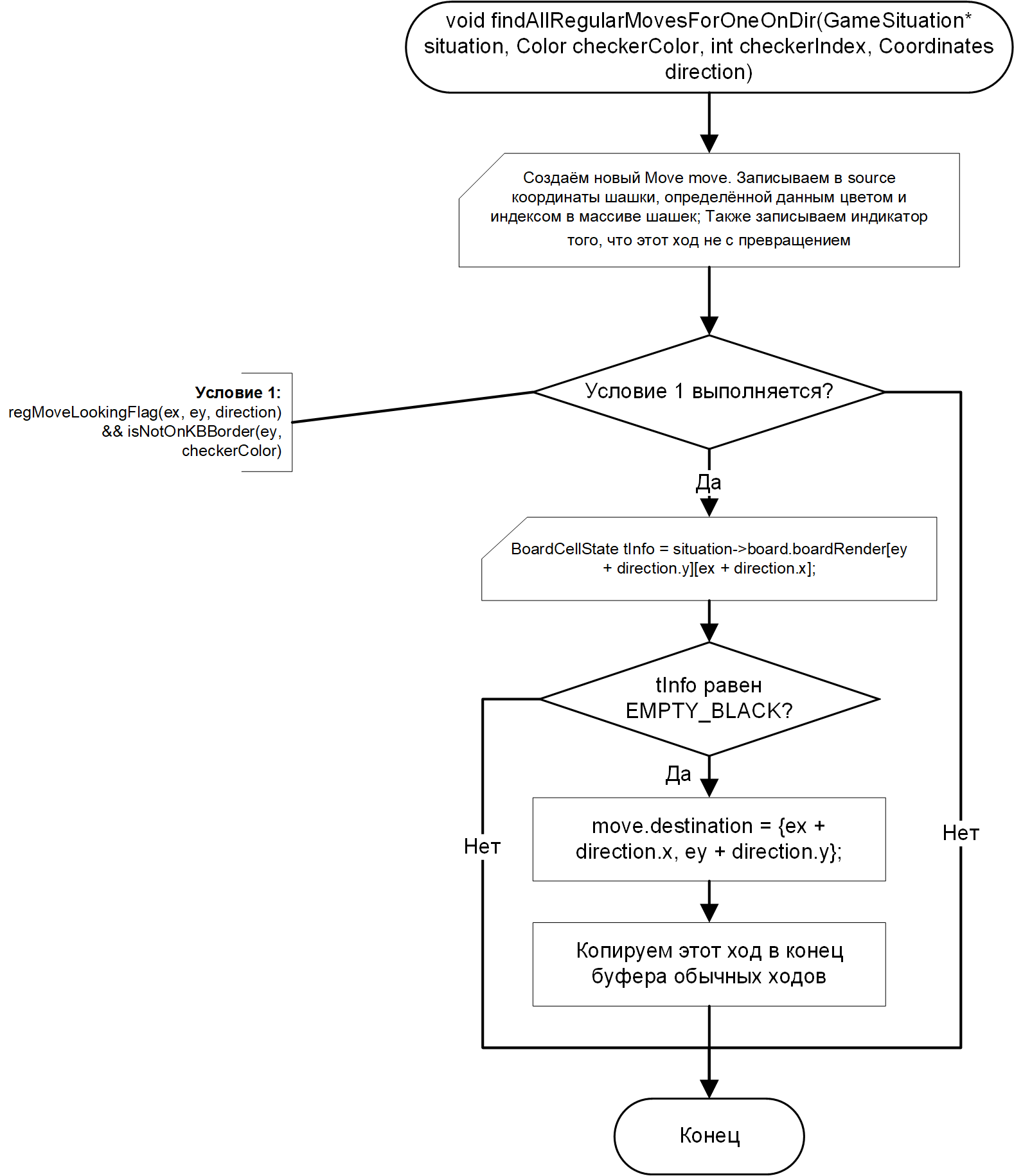


Рисунок 5.8.1. Блок-схема функции findAllRegularMovesForOneOnDir

## Функция findAllTakingMovesForOne

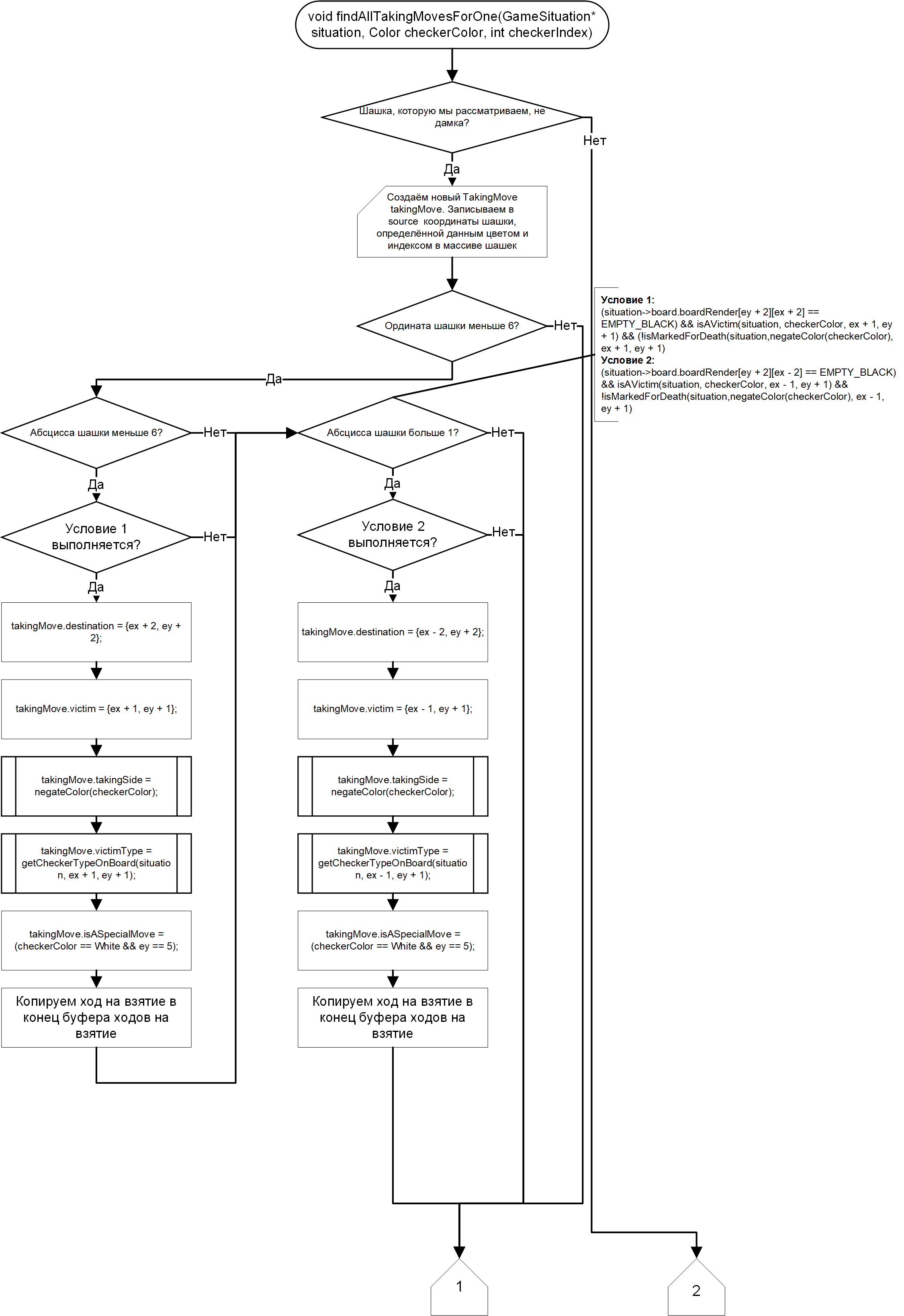


Рисунок 5.9.1. Блок-схема функции findAllTakingMovesForOne (страница 1)

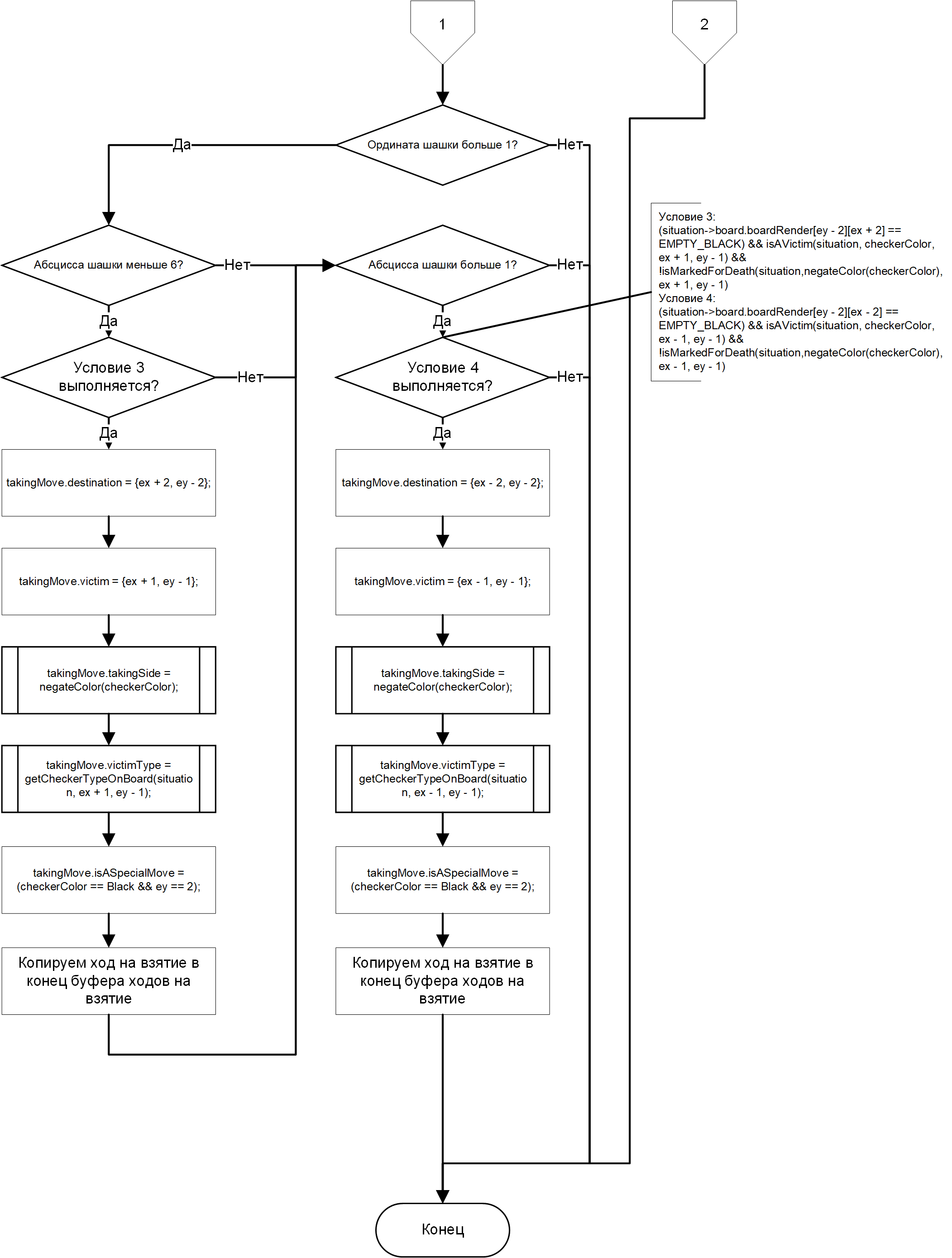


Рисунок 5.9.2. Блок-схема функции findAllTakingMovesForOne (страница 2)

## Функция findAllKingTakingMovesForOneOnDir

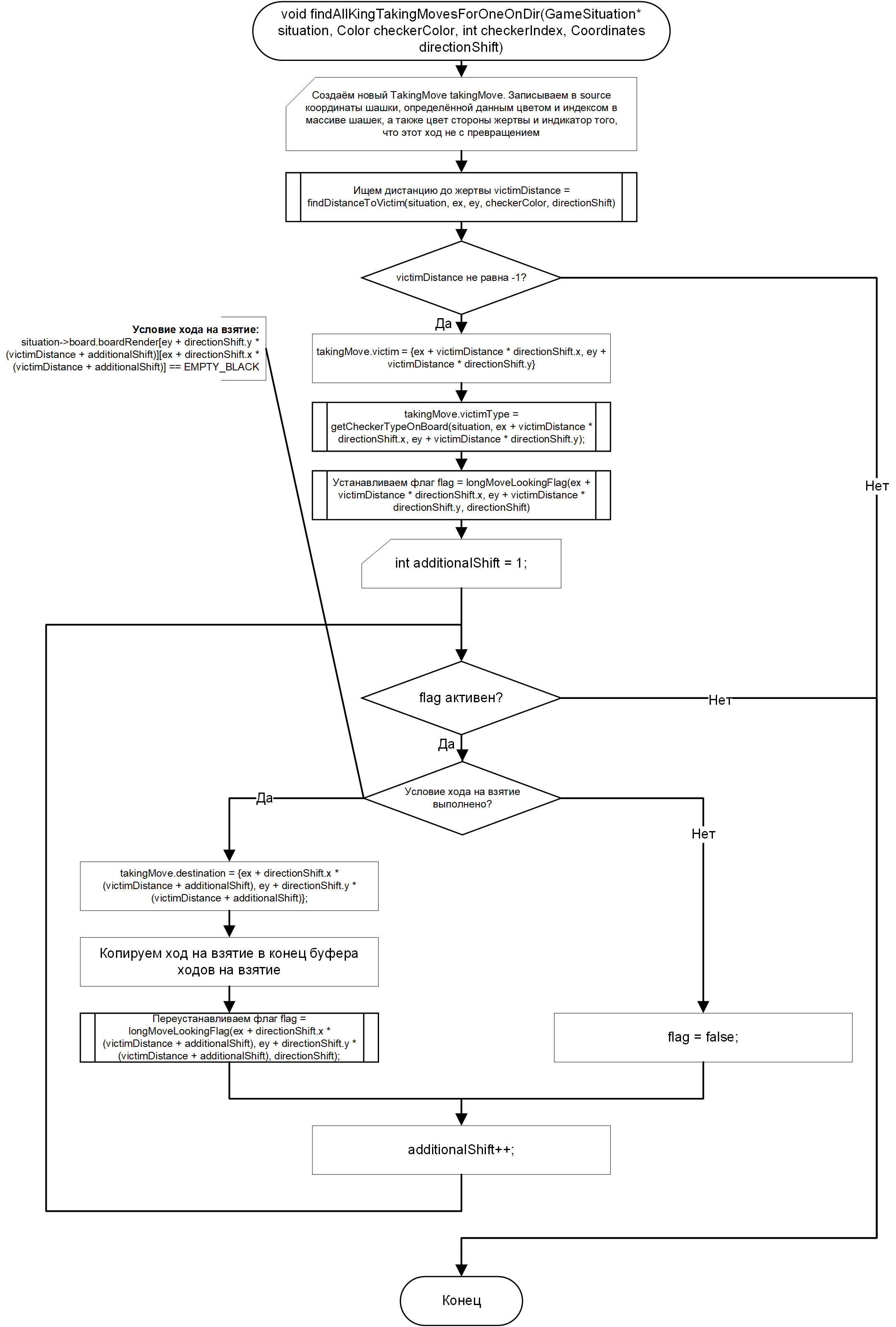


Рисунок 5.10.1. Блок-схема функции findAllKingTakingMovesForOneOnDir

## Функция findAllKBMovesForOne

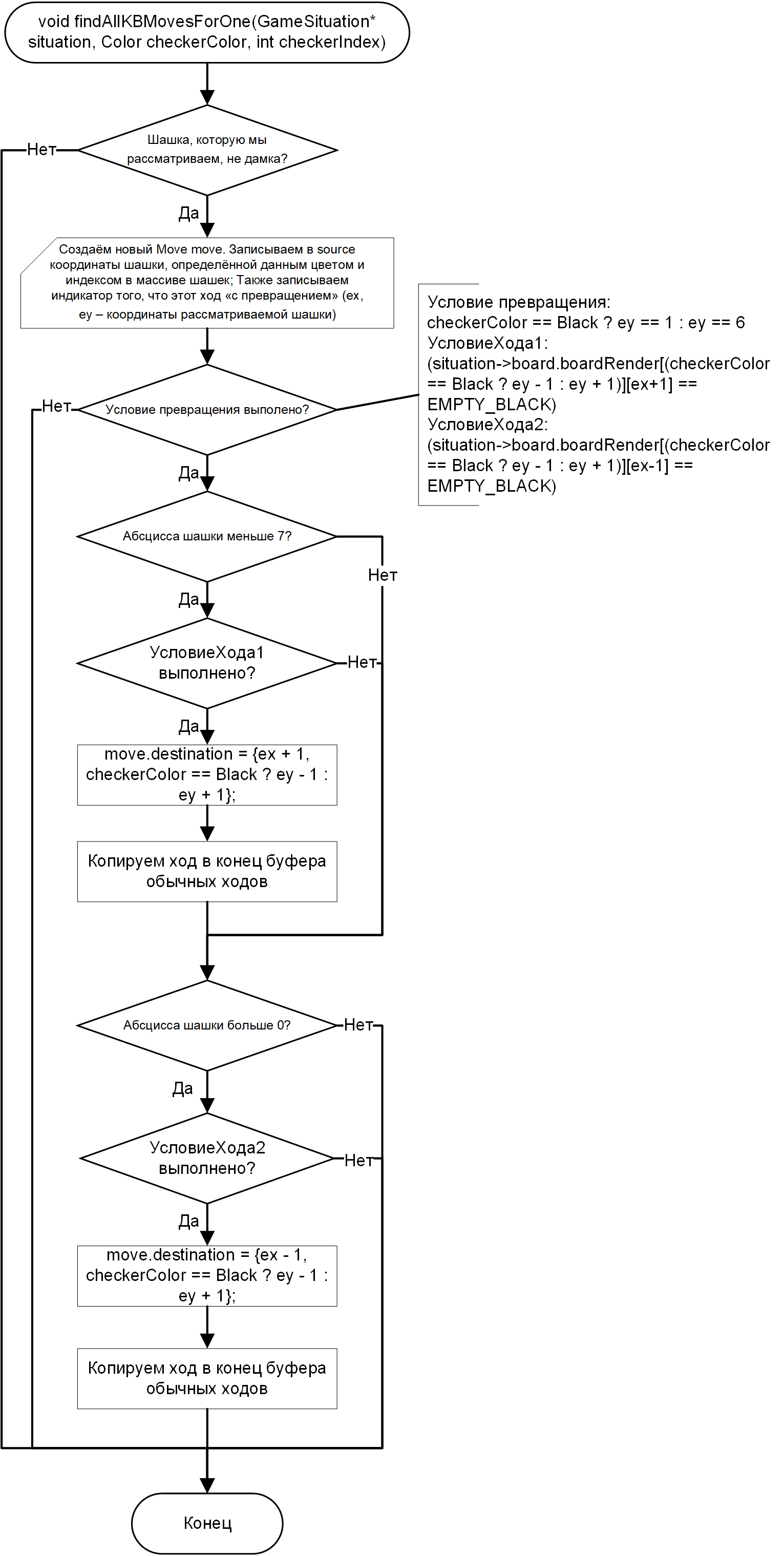


Рисунок 5.11.1. Блок-схема функции findAllKBMovesForOne

## Функция makeAMove

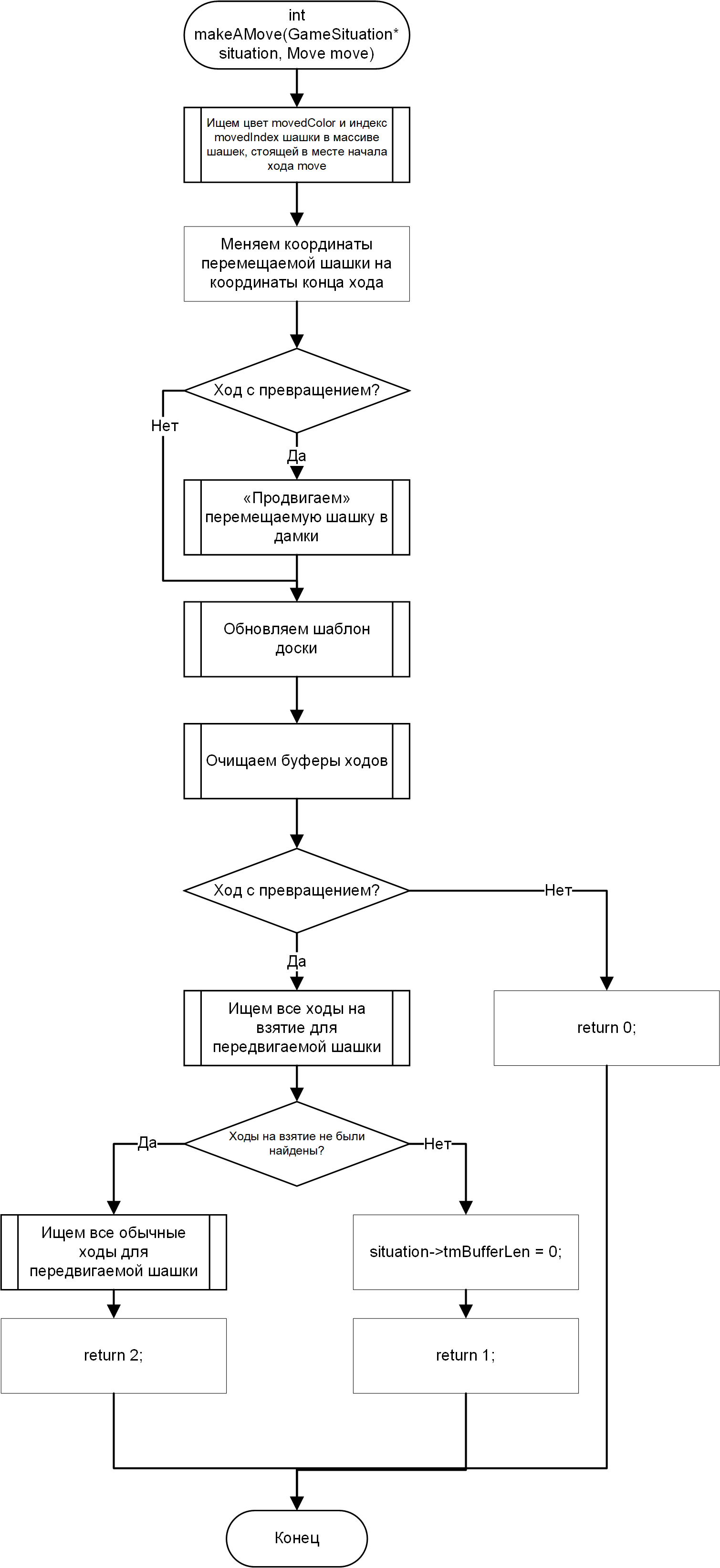


Рисунок 5.12.1. Блок-схема функции makeAMove

## Функция makeATakingMove

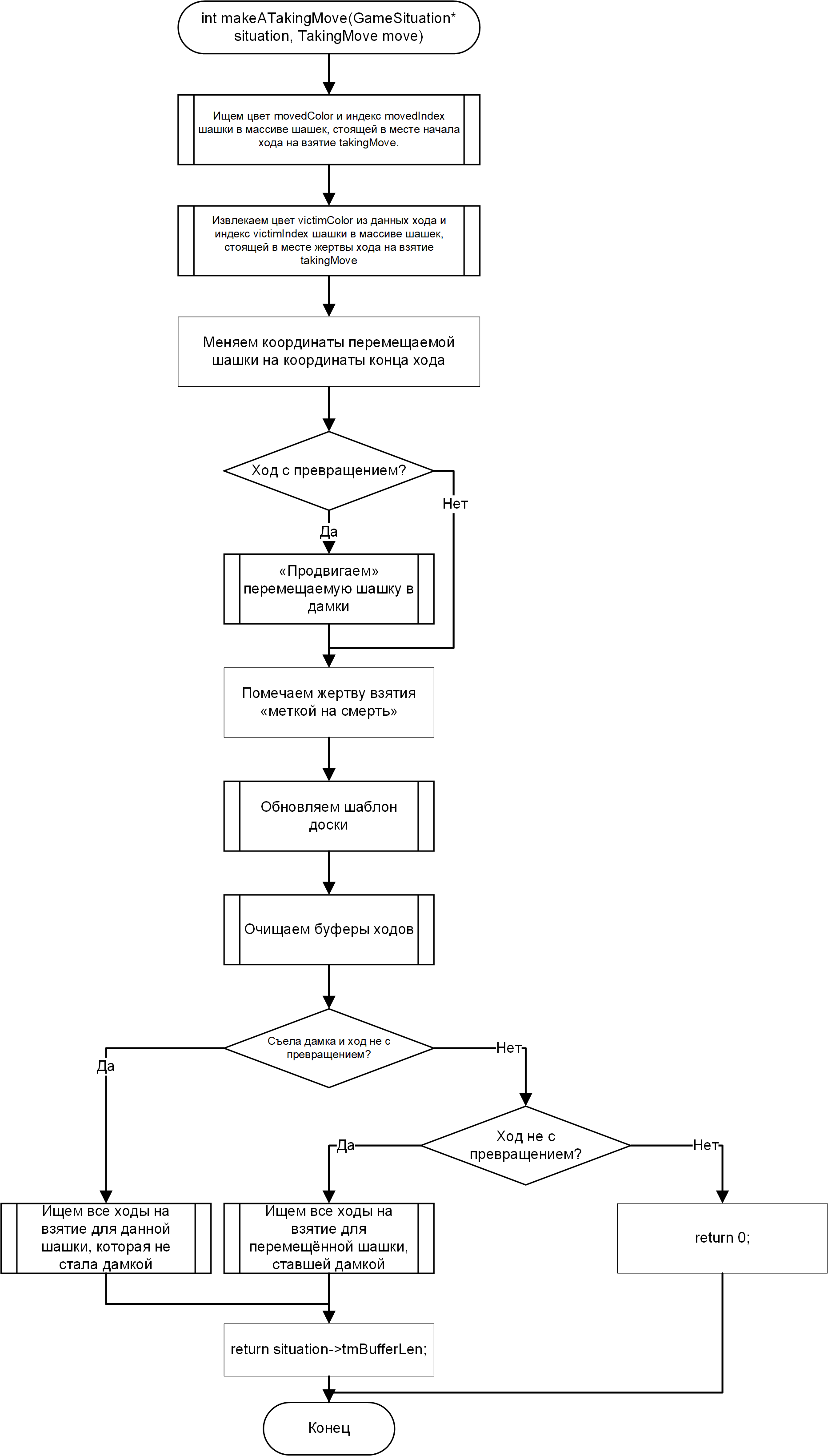


Рисунок 5.13.1. Блок-схема функции makeATakingMove

## Функция cancelAMove

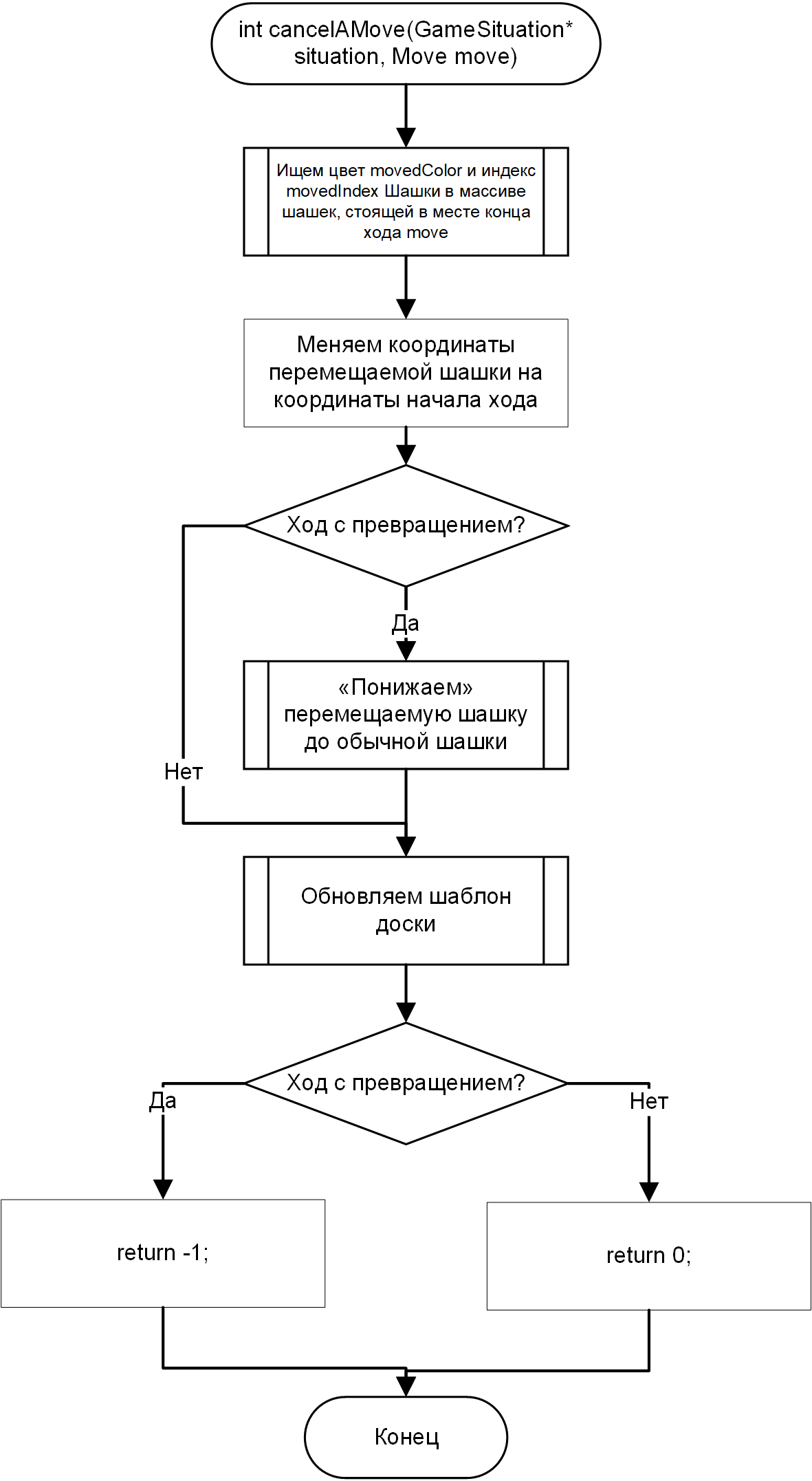


Рисунок 5.14.1. Блок-схема функции cancelAMove

## Функция cancelATakingMove

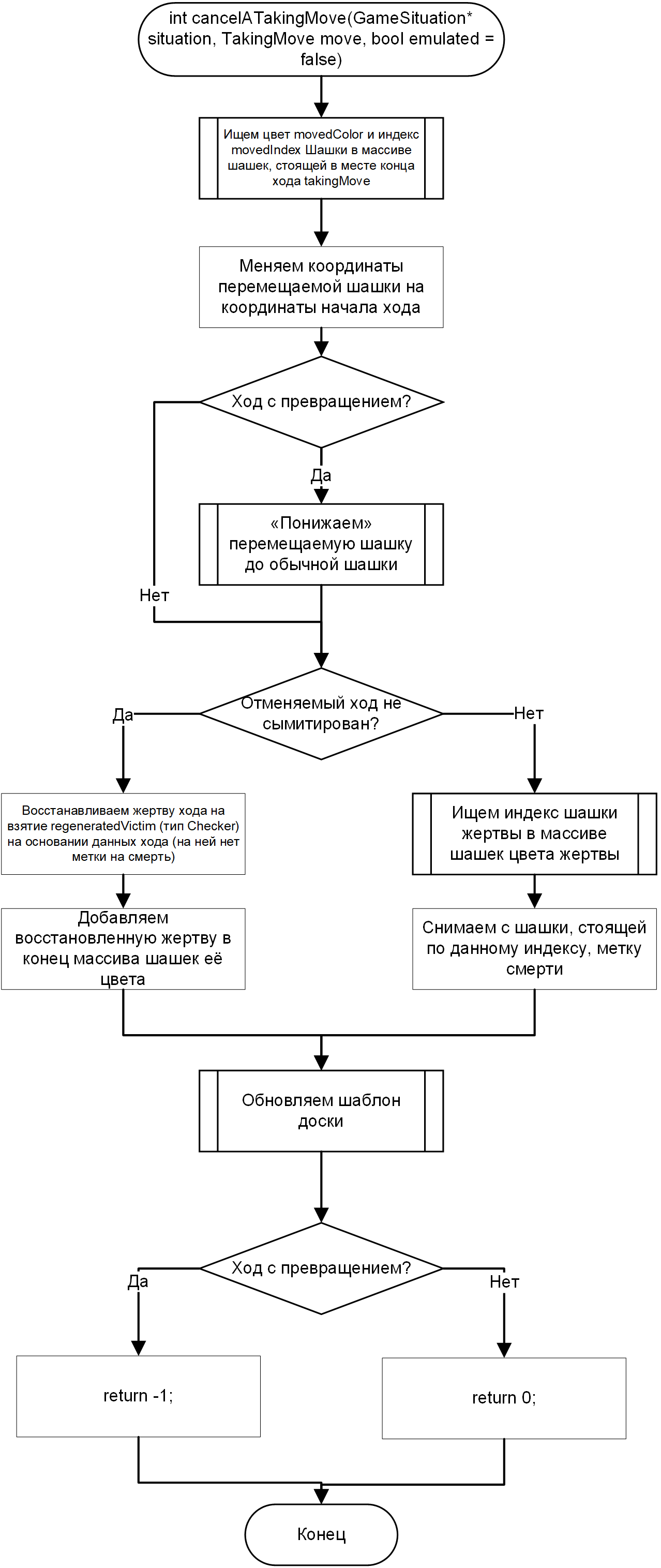


Рисунок 5.15.1. Блок-схема функции cancelATakingMove

## Функция findRegularMoveSequenceForOne

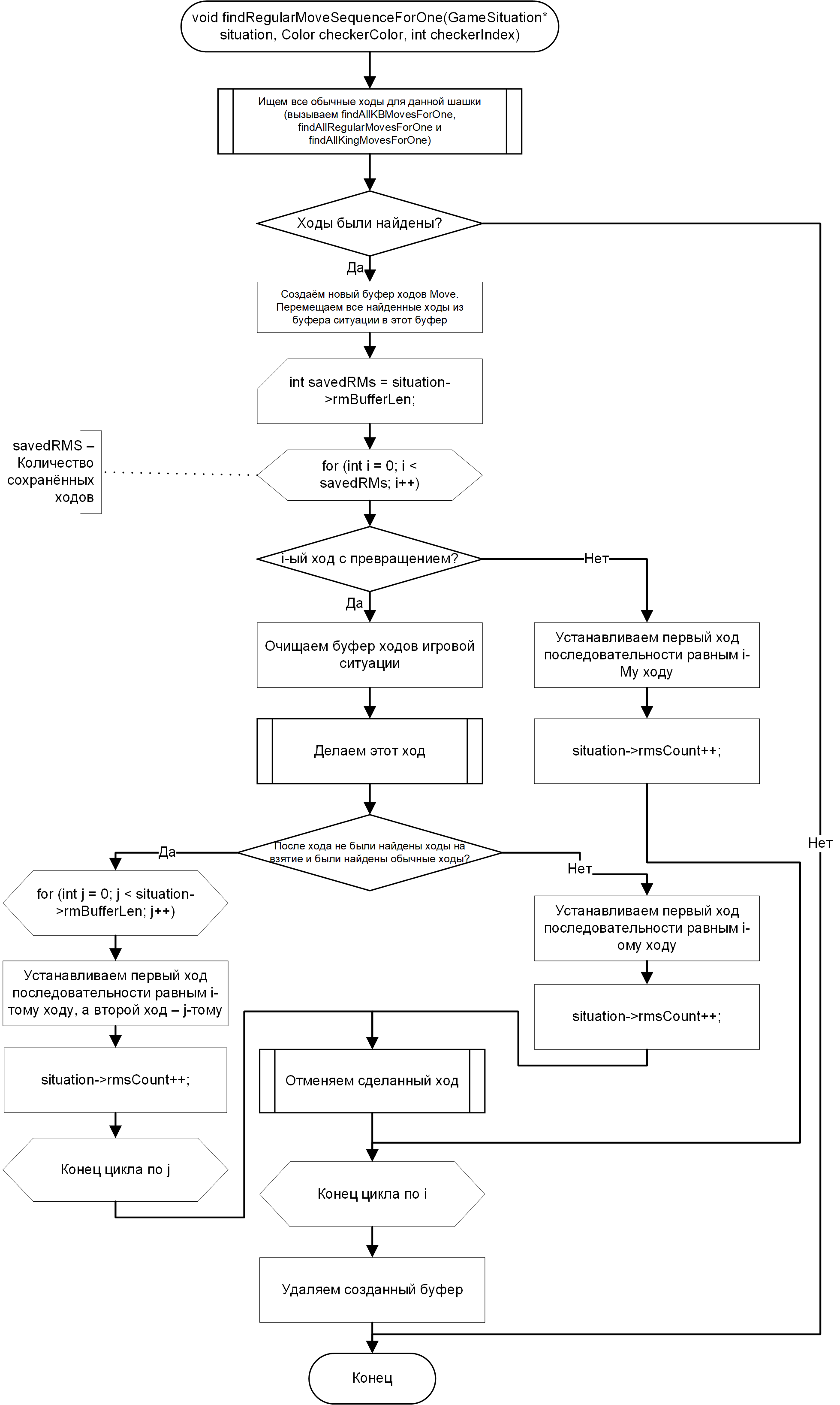


Рисунок 5.16.1. Блок-схема функции findRegularMoveSequencesForOne

## Функция findAllTakingSequencesForOne

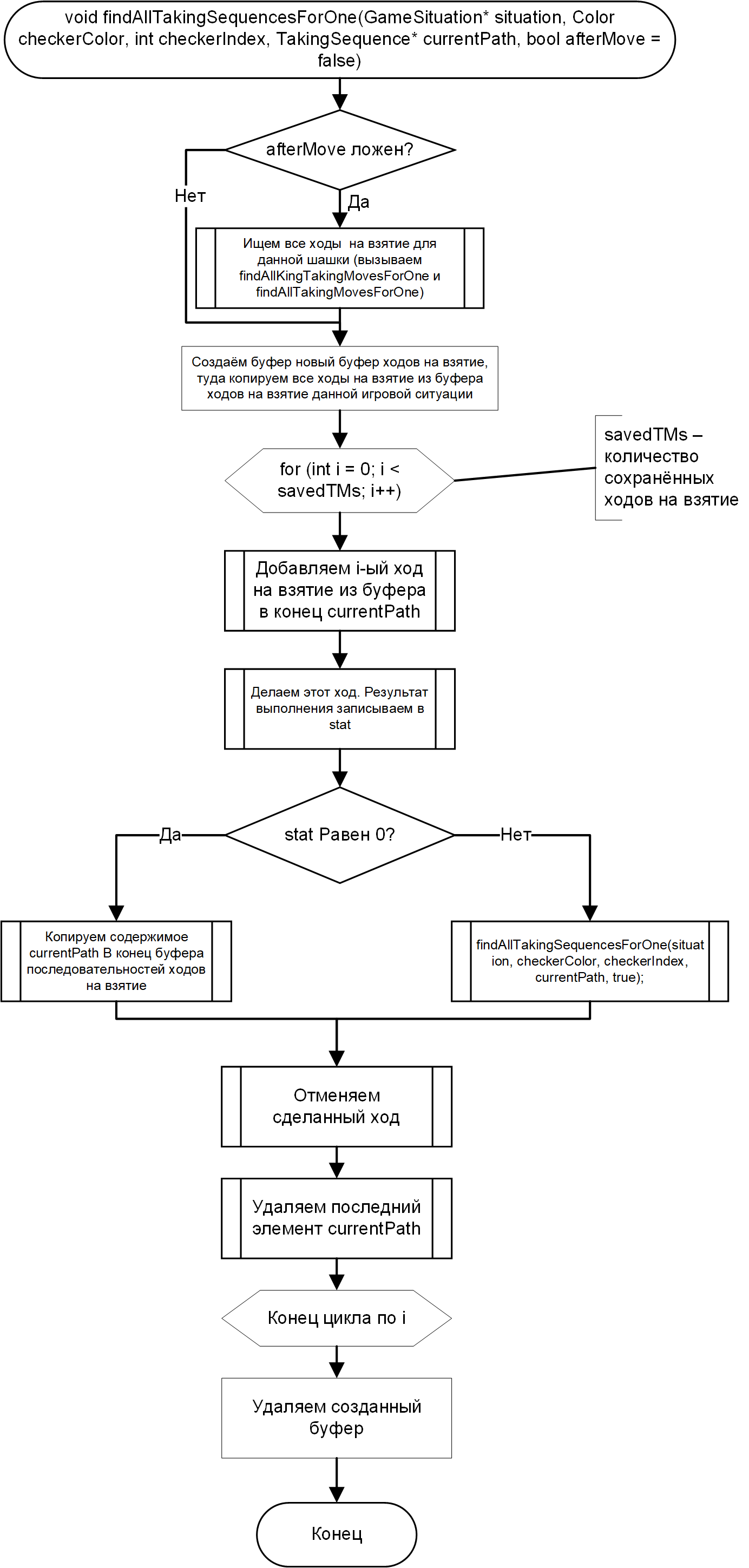


Рисунок 5.17.1. Блок-схема функции findAllTakingSequencesForOne

## Функция findAllTakingMoveSequences

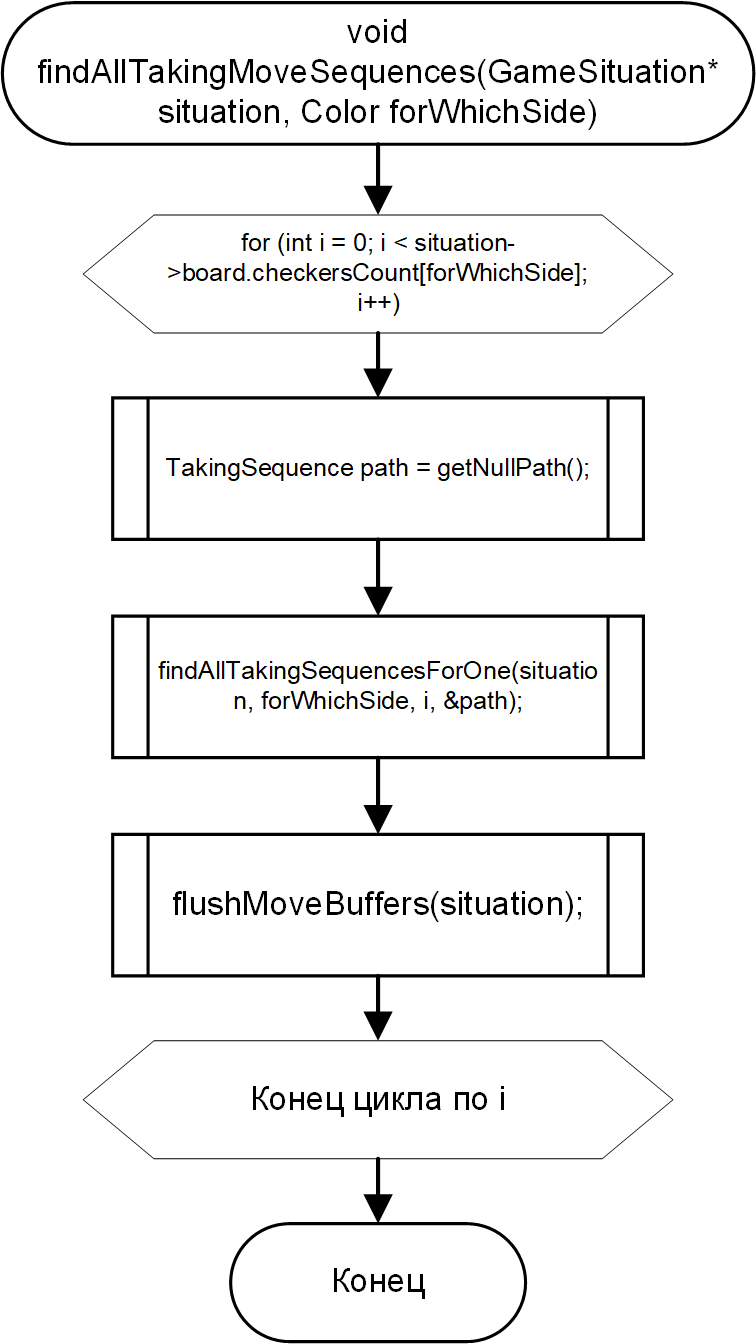


Рисунок 5.18.1. Блок-схема функции findAllTakingMoveSequences

## Функция analyze

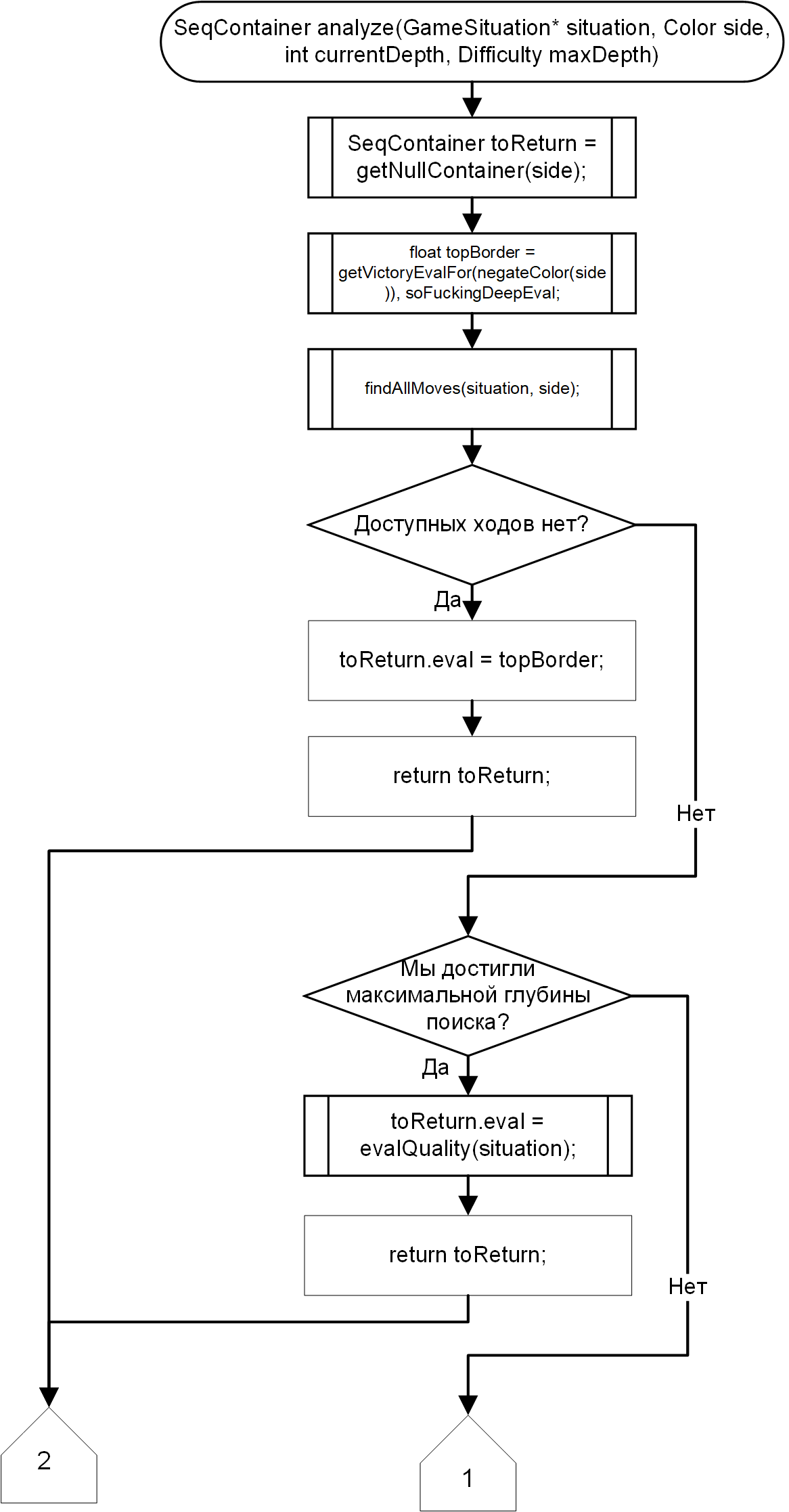


Рисунок 5.19.1. Блок-схема функции analyze (страница 1)

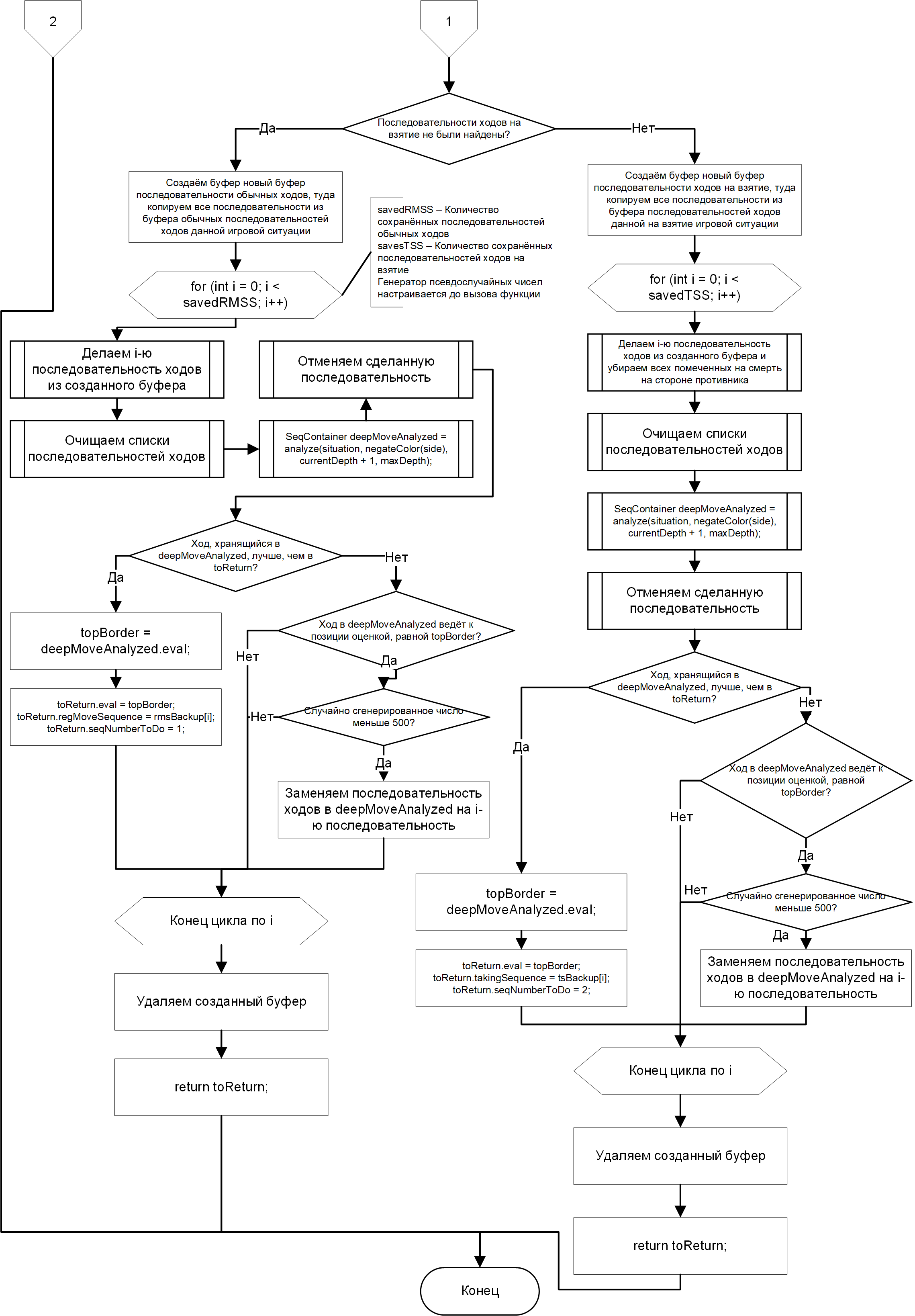


Рисунок 5.19.2. Блок-схема функции analyze (страница 2)

## Интерфейс приложения: обработка нажатия кнопки создания новой игры против компьютера

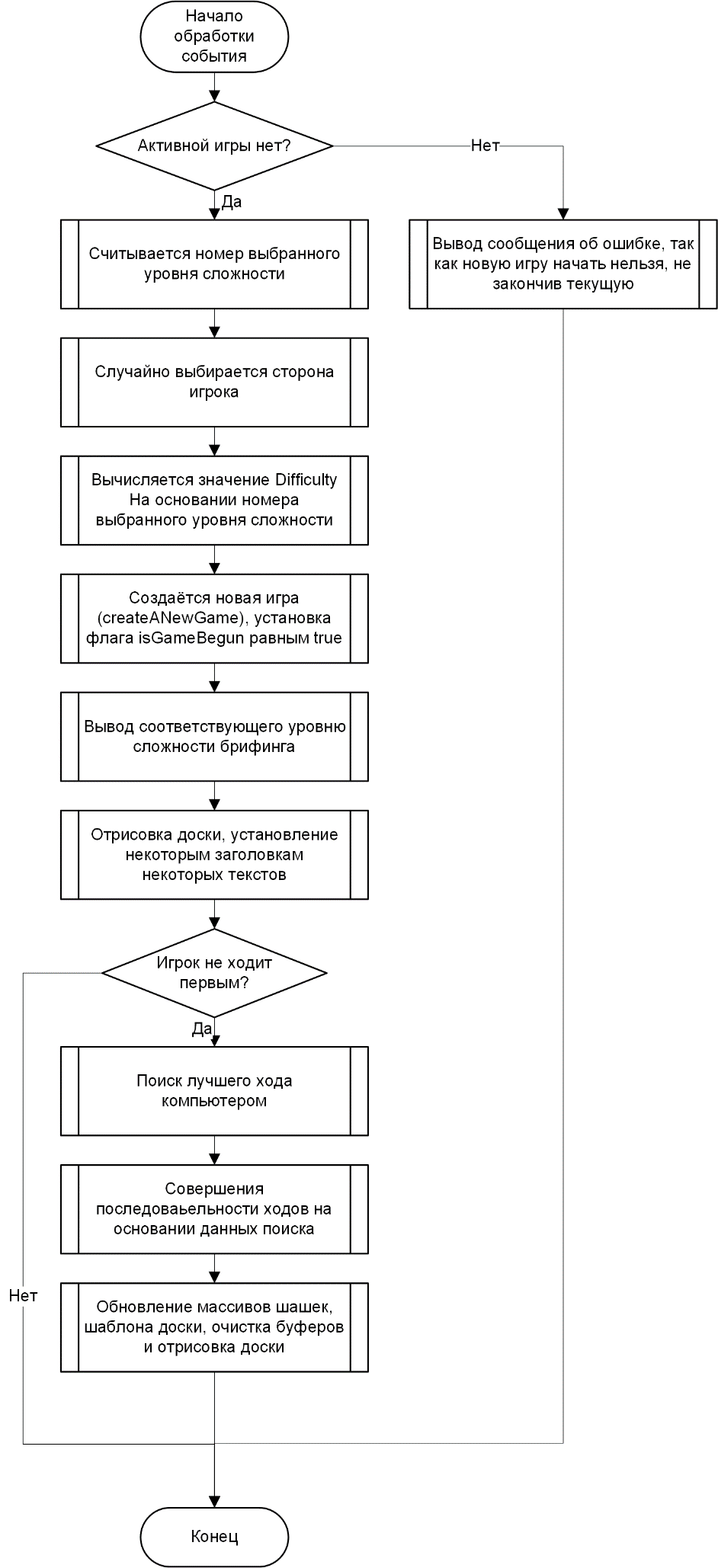


Рисунок 5.20.1. Блок-схема обработки нажатия кнопки создания новой игры против компьютера

## Интерфейс приложения: обработка нажатия кнопки создания новой игры против человека

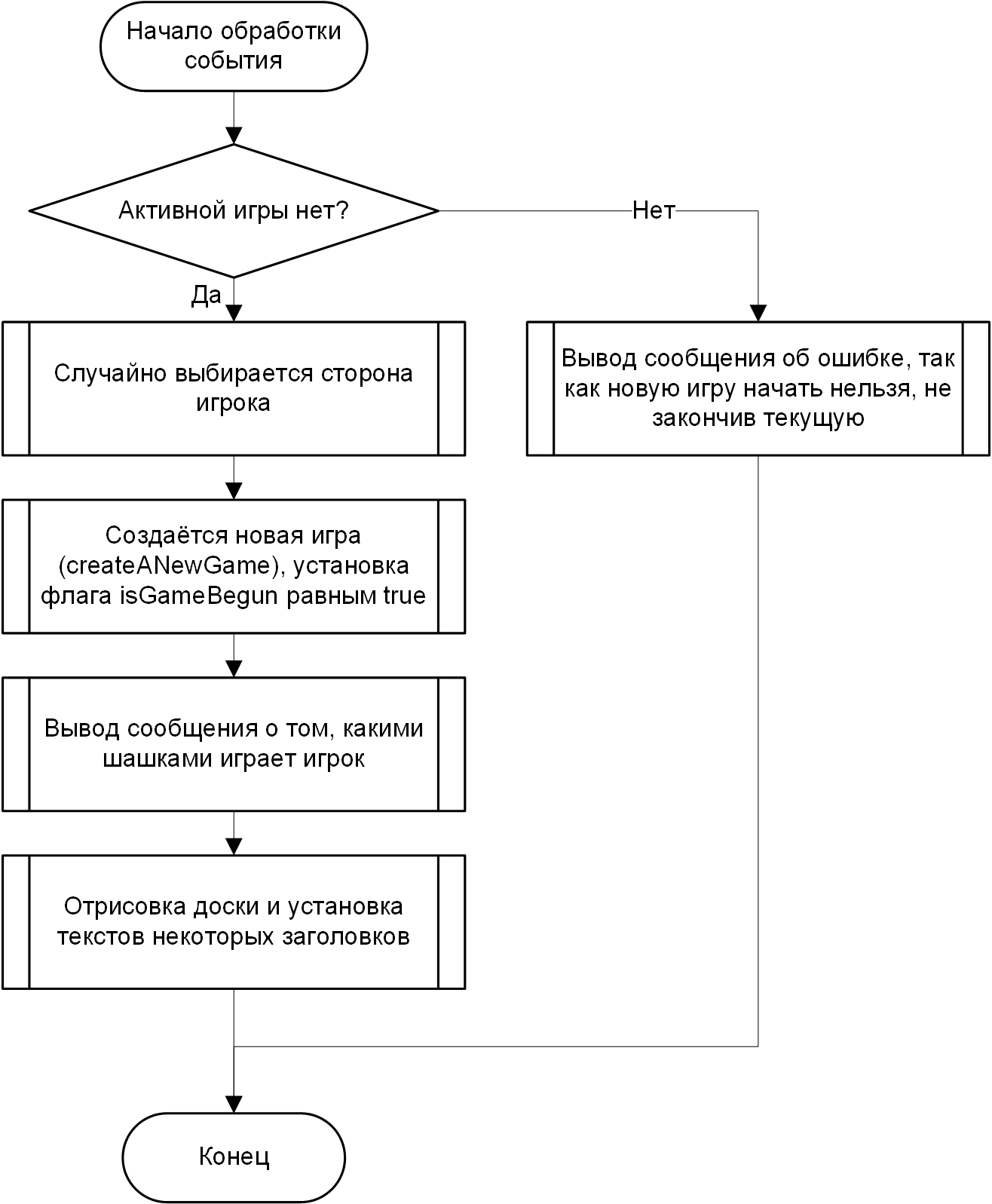


Рисунок 5.21.1. Блок-схема обработки нажатия кнопки создания новой игры против человека

## Интерфейс приложения: обработка нажатия клавиши Enter

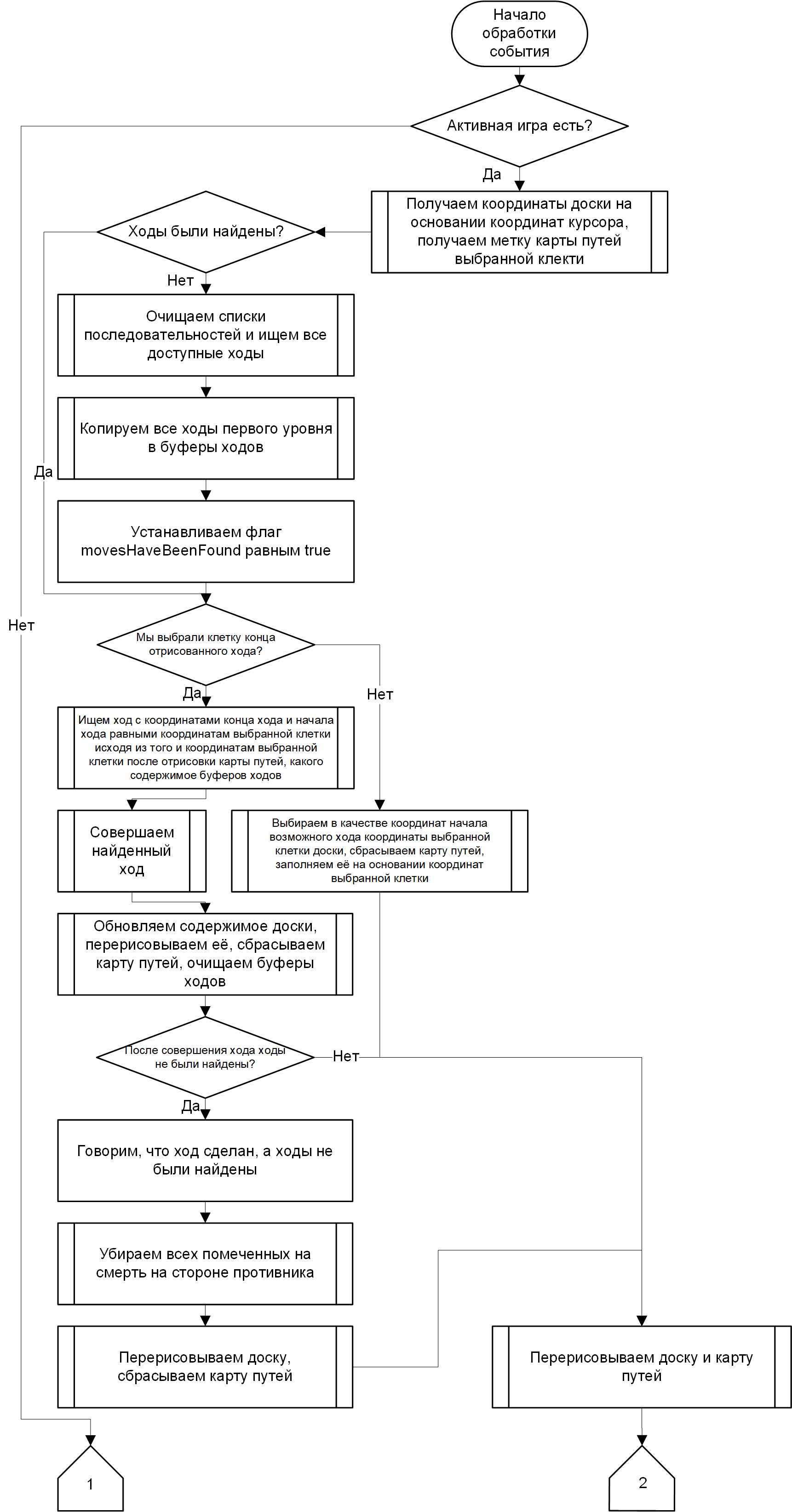


Рисунок 5.22.1. Блок-схема обработки нажатия клавиши Enter (страница 1)

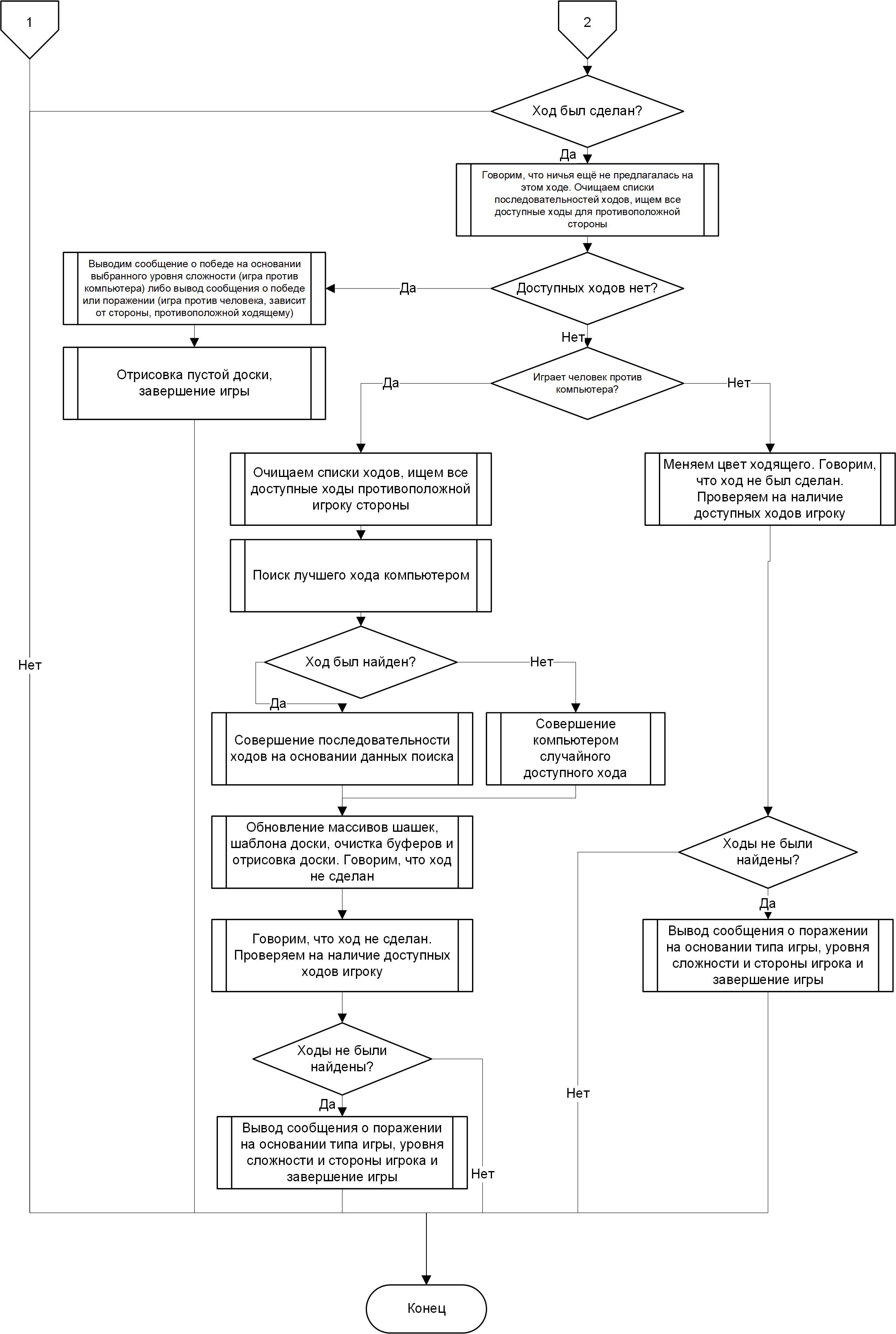


Рисунок 5.22.2Блок-схема обработки нажатия клавиши Enter (страница 2)

# Руководство пользователя

## Руководство по установке приложения

1. Если вы не получили копию данного приложения:
   1. Перейдите на сайт github.io/TheSwagVader/CLCheckers
   2. Выберите релиз приложения 1.0.
   3. Скачайте архив
   4. Распакуйте его
2. Запустите файл CLCheckers.exe

**Примечание:** необходимо, чтобы каталог и всё её содержимое находилось в той же папке, что и исполняемый файл приложения.

## Руководство по пользованию интерфейсом приложения

После запуска приложения, перед вами появится его интерфейс, представленный на рисунке \_\_\_.



Вы можете нажать на одну из следующих кнопок:

* «Играть против человека»: позволяет начать игру против человека, играя при этом за одним компьютером. После нажатия будет выведено сообщение о том, за какие шашки вы будете играть. Сторона игрока определяется случайно.
* «Играть против компьютера»: позволяет начать игру против компьютера на определённом уровне сложности. После нажатия вам также будет выведен краткий брифинг, в котором содержится небольшое описание противника и наставление на игру. Уровень сложности определяется значением выпадающего списка, подписанным соответствующим текстом. На выбор предлагается 6 уровней сложности, которые влияют на глубину поиска лучшего хода компьютерным противником (все имена являются вымышленными, совпадения случайны):
  1. Уровень сложности «Придурок». Глубина поиска – 2 полу хода. Противник – неандерталец Абуган Бунганов Уганович
  2. Уровень сложности «Легко». Глубина поиска – 3 полу хода. Противник – третьеклассник Немагнус Карлсен
  3. Уровень сложности «Нормально». Глубина поиска – 4 полу хода. Противник – лицеист Титов Дмитро Полковников
  4. Уровень сложности «Сложно». Глубина поиска – 6 полу ходов. Противник – Тайлунг
  5. Уровень сложности «Безумно». Глубина поиска – 7 полу ходов. Противник – Джотаро Куджо
  6. Уровень сложности «Экстремально». Глубина поиска – 8 полу ходов. Противник – Доктор Айво «Эггман» Роботник
* «Сдаться»: нажатие имеет эффект только при активной игре. Позволяет сдаться на текущем ходе, но перед этим программа спросит вас о том, действительно ли вы хотите сдаться. Играя против человека, если ходит соперник, то победа засчитывается игроку. Но вне зависимости от типа игры, если игрок сдаётся, ему засчитывается поражение.
* «Предложить ничью» нажатие имеет эффект только при активной игре. Позволяет предложить ничью своему оппоненту. Во время игры против человека будет задан вопрос, не хочет ли противник принять предложение ничьи. Если ответ «Да», то игра заканчивается ничьей, в противном случае – продолжается. В время игры против компьютера противник может принять предложение ничьи с некоторым шансом, и если он принимает предложение, то игра заканчивается:
  1. Уровень сложности «Придурок» - 99,9%
  2. Уровень сложности «Легко» - 80%
  3. Уровень сложности «Нормально» - 50%
  4. Уровень сложности «Сложно» - 25%
  5. Уровень сложности «Безумно» - 8%
  6. Уровень сложности «Экстремально» - 0%
* «Правила игры»: выводит прав ила игры в русские шашки.
* «Загрузить игру»: позволяет загрузить сохранённую ранее игру из файла. Нажатие не имеет эффекта, если на данный момент уже активна какая-нибудь игра.
* «Сохранить игру»: позволяет сохранить активную игру в файл. Нажатие не имеет эффекта, если нет активной игры. После сохранения игра продолжается.
* «Управление»: выводит краткую информацию об управлении приложением во время игры.
* «О программе»: выводит краткую информацию об приложении, такую как её название, автор, версия, лицензия и т.п.

## Правила игры в русские шашки

Правила:

1. Белые всегда ходят первыми
2. Простая шашка ходит по диагонали вперёд на одну клетку.
3. Дамка ходит по диагонали на любое свободное поле как вперёд, так и назад.
4. При выделении шашки ей необходимо сделать ход, за исключением случаев, когда шашка не имеет ходов.
5. Взятие обязательно.
6. Простая шашка, находящаяся рядом с шашкой соперника, за которой имеется свободное поле, переносится через эту шашку на это свободное поле. Если есть возможность продолжить взятие других шашек соперника, то это взятие продолжается, пока бьющая шашка не достигнет положения, из которого бой невозможен. Взятие простой шашкой производится как вперёд, так и назад.
7. Дамка бьёт по диагонали, как вперёд, так и назад, и становится на любое свободное поле после побитой шашки. Аналогично, дамка может бить несколько фигур соперника и должна бить до тех пор, пока это возможно.
8. При бое через дамочное поле простая шашка превращается в дамку и продолжает бой по правилам дамки.
9. При нескольких вариантах взятия, например, одну шашку или две, игрок выбирает вариант взятия по своему усмотрению.
10. Партия считается выигранной в следующих случаях:
    * если у одного из соперников побиты все шашки;
    * если один из участников заявил о том, что сдаётся;
    * если шашки одного из участников заперты и он не может сделать очередной ход.
11. Партия считается закончившейся вничью, если один из участников предлагает ничью, а другой её принимает.

# Положение об авторском праве

# Лицензирование проекта

# Приложение 1. Исходный код основных заголовочных файлов

## Заголовочный файл clcengine\_types.h

|  |
| --- |
| //  // Created by EgrZver on 08.04.2023.  //  #ifndef CHECKERS\_CLCENGINE\_STRUCTS\_H  #define CHECKERS\_CLCENGINE\_STRUCTS\_H  enum BoardCellState {EMPTY\_BLACK, EMPTY\_WHITE, REG\_BLACK, REG\_WHITE, KING\_BLACK, KING\_WHITE};  enum PathMapMarkers {NoMove, Source, Destination, VictimRB, VictimRW, VictimKB, VictimKW, MovingLock};  enum Color {Black, White};  enum CheckerType {Regular, King};  enum GameState {PlayerMove, EnemyMoveReal, EnemyMoveComputer, BlackWon, WhiteWon};  enum GameType {RvsR, RvsC}; //RvsR = Real versus Real, RvsC = Real vs Computer  enum Difficulty {Dumbass = 2, Easy, Normal, Hard = 6, Insane, Extreme, Diabolic, Invincible = 11};  // may delete last two  const int regularCheckerCost = 100;  const int kingCheckerCost = 250;  const bool ENABLED\_EMULATION = true;  const float BLACK\_VICTORY = 100000.0;  const float WHITE\_VICTORY = -100000.0;  const float qualityFactor[8][4] = {  {1.2, 1.2, 1.2, 1.2},  {1.15, 1.2, 1.2, 1.15},  {1.15, 1.2, 1.2, 1.13},  {1.0, 1.2, 1.15, 1.0},  {1.0, 1.2, 1.2, 1.0},  {1.0, 1.0, 1.0, 1.0},  {1.0, 1.0, 1.0, 1.0},  {1.0, 1.0, 1.0, 1.0},  };  const int probsOfDrawOfCPU[8] = {999, 800, 500, 250, 80, 0, 0, 0};  struct Coordinates {  int x, y;  };  const Coordinates NORTH\_EAST = {1, 1};  const Coordinates NORTH\_WEST = {-1, 1};  const Coordinates SOUTH\_EAST = {1, -1};  const Coordinates SOUTH\_WEST = {-1, -1};  struct Checker {  Coordinates coordinates;  Color color;  CheckerType type;  bool markedForDeath;  };  struct Move {  Coordinates source, destination;  bool isKingMove, isKingBecomingMove;  // tod remove king/taking types  };  struct TakingMove {  Coordinates source, destination, victim;  Color takingSide;  CheckerType victimType;  bool isASpecialMove;  };  struct RegMoveSequence {  Move regularMoves[2];  int rmsCount;  };  struct TakingSequence {  TakingMove takingMoves[16];  int tmsCount;  };  /\*  struct MixedSequence {  Move kingBecomingMove;  TakingSequence takingSequence;  };  \*/  struct Board {  BoardCellState boardRender[8][8];  PathMapMarkers pathMap[8][8];  Checker checkers[2][12];  int checkersCount[2];  };  struct GameSituation {  Board board;  Color playerSide;  Move regularMovesBuffer[24];  int tmsCount, rmsCount, mmsCount;  TakingMove takingMovesBuffer[24];  int tmBufferLen, rmBufferLen;  TakingSequence takingSequences[16];  //MixedSequence mixedSequences[16];  RegMoveSequence regMoveSequences[32];  };  struct SeqContainer {  RegMoveSequence regMoveSequence;  TakingSequence takingSequence;  //MixedSequence mixedSequence;  int seqNumberToDo;  float eval;  };  struct Game {  GameSituation situation;  GameState state;  GameType type;  };  #endif //CHECKERS\_CLCENGINE\_STRUCTS\_H |

## Заголовочный файл clcengine\_auxiliary.h

|  |
| --- |
| //  // Created by EgrZver on 11.05.2023.  //  #include "clcengine\_types.h"  #ifndef CHECKERS\_CLCENGINE\_AUXILIARY\_H  #define CHECKERS\_CLCENGINE\_AUXILIARY\_H  bool isCoordinatesEqual(Coordinates arg1, Coordinates arg2) {  return arg1.x == arg2.x && arg1.y == arg2.y;  }  void appendToATakingSequence(TakingSequence\* sequence, TakingMove move) {  sequence->takingMoves[sequence->tmsCount++] = move;  }  void deleteLastFromATakingSequence(TakingSequence\* sequence) {  sequence->tmsCount--;  }  void copyToAnotherTakingSequence(TakingSequence\* source, TakingSequence\* destination) {  for (int i = 0; i < source->tmsCount; i++) {  destination->takingMoves[i] = source->takingMoves[i];  }  destination->tmsCount = source->tmsCount;  }  float getVictoryEvalFor(Color side) {  if (side == Black) return BLACK\_VICTORY; else return WHITE\_VICTORY;  }  bool lostByMoves(GameSituation\* situation) {  return (situation->tmsCount + situation->rmsCount + situation->mmsCount) == 0;  }  void copyLevelOneMovesToBuffers(GameSituation\* situation) {  for (int i = 0; i < situation->rmsCount; i++) {  situation->regularMovesBuffer[situation->rmBufferLen++] = situation->regMoveSequences[i].regularMoves[0];  }  /\*for (int i = 0; i < situation->mmsCount; i++) {  situation->regularMovesBuffer[situation->rmBufferLen++] = situation->mixedSequences[i].kingBecomingMove;  }\*/  for (int i = 0; i < situation->tmsCount; i++) {  situation->takingMovesBuffer[situation->tmBufferLen++] = situation->takingSequences[i].takingMoves[0];  }  }  void flushSequenceLists(GameSituation\* situation) {  situation->rmsCount = 0;  situation->tmsCount = 0;  situation->mmsCount = 0;  }  bool replaceConditionFor(float nEval, float oEval, Color side) {  if (side == White) return (nEval < oEval); else return (nEval > oEval);  }  float evalQuality(GameSituation\* situation) {  float eval = 0;  for (int i = 0; i < situation->board.checkersCount[White]; i++) {  if (situation->board.checkers[White][i].type == King) {  eval -= kingCheckerCost;  }  else {  eval -= regularCheckerCost \* qualityFactor[7 - situation->board.checkers[White][i].coordinates.y][7 - situation->board.checkers[White][i].coordinates.x];  }  }  for (int i = 0; i < situation->board.checkersCount[Black]; i++) {  if (situation->board.checkers[Black][i].type == King) {  eval += kingCheckerCost;  }  else {  eval += regularCheckerCost \* qualityFactor[situation->board.checkers[Black][i].coordinates.y][situation->board.checkers[Black][i].coordinates.x];  }  }  return eval;  }  PathMapMarkers getVictimMarker(Color victimColor, CheckerType victimType) {  if (victimColor == Black) {  if (victimType == Regular) {  return VictimRB;  }  else {  return VictimKB;  }  }  else {  if (victimType == Regular) {  return VictimRW;  }  else {  return VictimKW;  }  }  }  void updateBoardRender(Board\* board) {  for (int i = 0; i < 8; i++) {  for (int j = 0; j < 8; j++) {  board->boardRender[j][i] = ((i + j) % 2 == 0) ? EMPTY\_BLACK : EMPTY\_WHITE;  }  }  for (int i = 0; i < board->checkersCount[Black]; i++) {  int rcx = board->checkers[Black][i].coordinates.x, rcy = board->checkers[Black][i].coordinates.y;  board->boardRender[rcy][rcx] = (board->checkers[Black][i].type == King) ? KING\_BLACK : REG\_BLACK;  }  for (int i = 0; i < board->checkersCount[White]; i++) {  int rcx = board->checkers[White][i].coordinates.x, rcy = board->checkers[White][i].coordinates.y;  board->boardRender[rcy][rcx] = (board->checkers[White][i].type == King) ? KING\_WHITE : REG\_WHITE;  }  }  void fillPathMap(GameSituation\* situation, Coordinates source) {  if (situation->tmBufferLen != 0) {  for (int i = 0; i < situation->tmBufferLen; i++) {  if (isCoordinatesEqual(situation->takingMovesBuffer[i].source, source)) {  Coordinates dest = situation->takingMovesBuffer[i].destination,  victim = situation->takingMovesBuffer[i].victim;  situation->board.pathMap[dest.y][dest.x] = Destination;  situation->board.pathMap[victim.y][victim.x] = getVictimMarker(situation->takingMovesBuffer[i].takingSide, situation->takingMovesBuffer[i].victimType);  }  }  }  else {  for (int i = 0; i < situation->rmBufferLen; i++) {  if (isCoordinatesEqual(situation->regularMovesBuffer[i].source, source)) {  Coordinates dest = situation->regularMovesBuffer[i].destination;  situation->board.pathMap[dest.y][dest.x] = Destination;  }  }  }  }  int getCheckerIndexByCoordsAndColor(GameSituation\* situation, int cx, int cy, Color color) {  for (int i = 0; i < situation->board.checkersCount[color]; i++) {  if (situation->board.checkers[color][i].coordinates.x == cx && situation->board.checkers[color][i].coordinates.y == cy) {  return i;  }  }  }  Color sideColor(GameSituation\* situation, int cx, int cy) {  BoardCellState tInfo = situation->board.boardRender[cy][cx];  if (tInfo == REG\_WHITE || tInfo == KING\_WHITE) {  return White;  }  else if (tInfo == REG\_BLACK || tInfo == KING\_BLACK) {  return Black;  }  }  Color negateColor(Color color) {  if (color == White) return Black;  else if (color == Black) return White;  }  CheckerType getCheckerTypeOnBoard(GameSituation\* situation, int cx, int cy) {  BoardCellState tInfo = situation->board.boardRender[cy][cx];  if (tInfo == REG\_WHITE || tInfo == REG\_BLACK) {  return Regular;  }  else if (tInfo == KING\_BLACK || tInfo == KING\_WHITE) {  return King;  }  }  bool isAFriend(GameSituation\* situation, Color forWhichSide, int tx, int ty) {  BoardCellState tInfo = situation->board.boardRender[ty][tx];  if (forWhichSide == White) {  return tInfo == REG\_WHITE || tInfo == KING\_WHITE;  }  else if (forWhichSide == Black) {  return tInfo == REG\_BLACK || tInfo == KING\_BLACK;  }  }  bool isMarkedForDeath(GameSituation\* situation, Color checkerColor, int mx, int my) {  if (situation->board.boardRender[my][mx] == EMPTY\_BLACK) return false;  int index = getCheckerIndexByCoordsAndColor(situation, mx, my, checkerColor);  return situation->board.checkers[checkerColor][index].markedForDeath;  }  bool isAVictim(GameSituation\* situation, Color forWhichSide, int tx, int ty) {  BoardCellState tInfo = situation->board.boardRender[ty][tx];  if (forWhichSide == Black) {  return (tInfo == REG\_WHITE || tInfo == KING\_WHITE);  } else if (forWhichSide == White) {  return (tInfo == REG\_BLACK || tInfo == KING\_BLACK);  } else return false;  }  bool isNotOnKBBorder(int y, Color side) {  return (side == White) ? y < 6 : y > 1;  }  bool longMoveLookingFlag(int x, int y, Coordinates direction) {  if (direction.x == NORTH\_EAST.x && direction.y == NORTH\_EAST.y) {  return !(x >= 7 || y >= 7);  }  else if (direction.x == NORTH\_WEST.x && direction.y == NORTH\_WEST.y) {  return !(x <= 0 || y >= 7);  }  else if (direction.x == SOUTH\_EAST.x && direction.y == SOUTH\_EAST.y) {  return !(x >= 7 || y <= 0);  }  else if (direction.x == SOUTH\_WEST.x && direction.y == SOUTH\_WEST.y) {  return !(x <= 0 || y <= 0);  }  }  bool regMoveLookingFlag(int x, int y, Coordinates direction) {  if (direction.x == NORTH\_EAST.x) {  return x < 7;  }  else if (direction.x == NORTH\_WEST.x) {  return x > 0;  }  }  int findDistanceToVictim(GameSituation\* situation, int x, int y, Color forWhichSide, Coordinates direction) {  int shift = 1;  while (longMoveLookingFlag(x + shift\*direction.x, y + shift \* direction.y, direction)) {  if (isAFriend(situation, forWhichSide, x + shift\*direction.x, y + shift \* direction.y) || isMarkedForDeath(situation, negateColor(forWhichSide), x + shift\*direction.x, y + shift \* direction.y)) return -1;  else if (isAFriend(situation, forWhichSide, x + shift\*direction.x, y + shift \* direction.y)) return -1;  else if (isAVictim(situation, forWhichSide, x + shift \* direction.x, y + shift \* direction.y)) {  return shift;  }  else if (situation->board.boardRender[y + shift \* direction.y][x + shift\*direction.x] == EMPTY\_BLACK) shift++;  }  return -1;  }  inline void flushMoveBuffers(GameSituation\* situation) {  situation->tmBufferLen = 0;  situation->rmBufferLen = 0;  }  void removeChecker(Board\* board, int index, Color color) {  if (index < board->checkersCount[color]) {  for (int i = index + 1; i < board->checkersCount[color];i++) {  board->checkers[color][i - 1] = board->checkers[color][i];  }  board->checkersCount[color]--;  }  }  void descendChecker(Checker\* checker) {  checker->type = Regular;  }  void ascendChecker(Checker\* checker) {  checker->type = King;  }  void removeMarkedForDeath(GameSituation\* situation, Color inWhere) {  for (int i = 0; i < situation->board.checkersCount[inWhere]; i++) {  if (situation->board.checkers[inWhere][i].markedForDeath) {  removeChecker(&(situation->board), i, inWhere);  i--;  }  }  updateBoardRender(&(situation->board));  }  #endif //CHECKERS\_CLCENGINE\_AUXILIARY\_H |

## Заголовочный файл clcengine\_inits.h

|  |
| --- |
| //  // Created by EgrZver on 11.05.2023.  //  #include "clcengine\_types.h"  #include "clcengine\_auxiliary.h"  #ifndef CHECKERS\_CLCENGINE\_INITS\_H  #define CHECKERS\_CLCENGINE\_INITS\_H  SeqContainer getNullContainer(Color side) {  SeqContainer container;  container.eval = getVictoryEvalFor(negateColor(side));  container.seqNumberToDo = -1;  container.takingSequence.tmsCount = 0;  container.regMoveSequence.rmsCount = 0;  //container.mixedSequence.takingSequence.tmsCount = 0;  return container;  }  TakingSequence getNullPath() {  TakingSequence path;  path.tmsCount = 0;  return path;  }  void initiateChecker(Checker\* checker, int x, int y, Color color) {  checker->coordinates.x = x;  checker->coordinates.y = y;  checker->color = color;  checker->type = Regular;  checker->markedForDeath = false;  }  void resetPathMap(Board\* board) {  for (int i = 0; i < 8; i++) {  for (int j = 0; j < 8; j++) {  board->pathMap[j][i] = NoMove;  }  }  }  Board createANewBoard() {  Board newBoard;  newBoard.checkersCount[Black] = 12; newBoard.checkersCount[White] = 12;  int valueOfInitedPairs = 0;  for (short j = 0; j < 3; j++) {  for (short k = 0; k < 8; k += 2) {  initiateChecker(&newBoard.checkers[White][valueOfInitedPairs], k + (j % 2), j, White);  initiateChecker(&newBoard.checkers[Black][valueOfInitedPairs++], k + 1 - (j % 2), 7 - j, Black);  }  }  resetPathMap(&newBoard);  updateBoardRender(&newBoard);  return newBoard;  }  GameSituation makeNullGameSituation(Color playerSide) {  GameSituation gameSituation;  gameSituation.board = createANewBoard();  gameSituation.playerSide = playerSide;  gameSituation.rmBufferLen = 0;  gameSituation.tmBufferLen = 0;  gameSituation.tmsCount = 0;  gameSituation.rmsCount = 0;  gameSituation.mmsCount = 0;  for (int i = 0; i < 16; i++) gameSituation.takingSequences[i].tmsCount = 0;  //Color turnOf;  return gameSituation;  }  Game createANewGame(Color playerSide, Color firstMove, GameType type) {  Game newGame;  newGame.situation = makeNullGameSituation(playerSide);  //newGame.state = (playerSide == firstMove) ? PlayerMove : ((type == RvsR) ? EnemyMoveReal : EnemyMoveComputer);  newGame.type = type;  newGame.situation.playerSide = playerSide;  return newGame;  }  #endif //CHECKERS\_CLCENGINE\_INITS\_H |

## Заголовочный файл clcengine\_move\_searching.h

|  |
| --- |
| //  // Created by EgrZver on 11.05.2023.  //  #include "clcengine\_types.h"  #include "clcengine\_auxiliary.h"  #ifndef CHECKERS\_CLCENGINE\_MOVE\_SEARCHING\_H  #define CHECKERS\_CLCENGINE\_MOVE\_SEARCHING\_H  void findAllKingMovesForOneOnDir(GameSituation\* situation, Color checkerColor, int checkerIndex, Coordinates directionShift) {  Move move;  move.isKingBecomingMove = false; move.isKingMove = false;  int ex = situation->board.checkers[checkerColor][checkerIndex].coordinates.x;  int ey = situation->board.checkers[checkerColor][checkerIndex].coordinates.y;  move.source = {ex, ey};  bool flag = longMoveLookingFlag(ex, ey, directionShift);  int shift = 1;  while (flag) {  if (situation->board.boardRender[ey + shift \* directionShift.y][ex + shift\*directionShift.x] == EMPTY\_BLACK) {  move.destination = {ex + shift \* directionShift.x, ey + shift \* directionShift.y};  situation->regularMovesBuffer[situation->rmBufferLen++] = move;  flag = longMoveLookingFlag(ex + shift\*directionShift.x, ey + shift \* directionShift.y, directionShift);  } else {  flag = false;  }  shift++;  }  }  void findAllKingMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex) {  if (situation->board.checkers[checkerColor][checkerIndex].type == King) {  findAllKingMovesForOneOnDir(situation, checkerColor, checkerIndex, NORTH\_EAST);  findAllKingMovesForOneOnDir(situation, checkerColor, checkerIndex, NORTH\_WEST);  findAllKingMovesForOneOnDir(situation, checkerColor, checkerIndex, SOUTH\_EAST);  findAllKingMovesForOneOnDir(situation, checkerColor, checkerIndex, SOUTH\_WEST);  }  }  void findAllRegularMovesForOneOnDir(GameSituation\* situation, Color checkerColor, int checkerIndex, Coordinates direction) {  Move move;  move.isKingBecomingMove = false; move.isKingMove = false;  int ex = situation->board.checkers[checkerColor][checkerIndex].coordinates.x;  int ey = situation->board.checkers[checkerColor][checkerIndex].coordinates.y;  move.source = {ex, ey};  if (regMoveLookingFlag(ex, ey, direction) && isNotOnKBBorder(ey, checkerColor)) {  BoardCellState tInfo = situation->board.boardRender[ey + direction.y][ex + direction.x];  if (tInfo == EMPTY\_BLACK) {  move.destination = {ex + direction.x, ey + direction.y};  situation->regularMovesBuffer[situation->rmBufferLen++] = move;  }  }  }  void findAllRegularMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex) {  if (checkerColor == White) {  findAllRegularMovesForOneOnDir(situation, checkerColor, checkerIndex, NORTH\_EAST);  findAllRegularMovesForOneOnDir(situation, checkerColor, checkerIndex, NORTH\_WEST);  }  else {  findAllRegularMovesForOneOnDir(situation, checkerColor, checkerIndex, SOUTH\_EAST);  findAllRegularMovesForOneOnDir(situation, checkerColor, checkerIndex, SOUTH\_WEST);  }  }  void findAllTakingMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex) {  if (situation->board.checkers[checkerColor][checkerIndex].type != King) {  TakingMove takingMove;  int ex = situation->board.checkers[checkerColor][checkerIndex].coordinates.x;  int ey = situation->board.checkers[checkerColor][checkerIndex].coordinates.y;  takingMove.source = {ex, ey};  if (ey < 6) {  if (ex < 6) {  if ((situation->board.boardRender[ey + 2][ex + 2] == EMPTY\_BLACK) && isAVictim(situation, checkerColor, ex + 1, ey + 1) && (!isMarkedForDeath(situation,negateColor(checkerColor), ex + 1, ey + 1))/\* && ((ex + 1 != vx) || (ey + 1 != vy))\*/) {  takingMove.destination = {ex + 2, ey + 2};  takingMove.victim = {ex + 1, ey + 1};  takingMove.takingSide = negateColor(checkerColor);  takingMove.victimType = getCheckerTypeOnBoard(situation, ex + 1, ey + 1);  takingMove.isASpecialMove = (checkerColor == White && ey == 5);  situation->takingMovesBuffer[situation->tmBufferLen++] = takingMove;  }  }  if (ex > 1) {  if ((situation->board.boardRender[ey + 2][ex - 2] == EMPTY\_BLACK) && isAVictim(situation, checkerColor, ex - 1, ey + 1) && !isMarkedForDeath(situation,negateColor(checkerColor), ex - 1, ey + 1)/\* && ((ex - 1 != vx) || (ey + 1 != vy))\*/) {  takingMove.destination = {ex - 2, ey + 2};  takingMove.victim = {ex - 1, ey + 1};  takingMove.takingSide = negateColor(checkerColor);  takingMove.victimType = getCheckerTypeOnBoard(situation, ex - 1, ey + 1);  takingMove.isASpecialMove = (checkerColor == White && ey == 5);  situation->takingMovesBuffer[situation->tmBufferLen++] = takingMove;  }  }  }  if (ey > 1) {  if (ex < 6) {  if ((situation->board.boardRender[ey - 2][ex + 2] == EMPTY\_BLACK) && isAVictim(situation, checkerColor, ex + 1, ey - 1) && !isMarkedForDeath(situation,negateColor(checkerColor), ex + 1, ey - 1)/\* && ((ex + 1 != vx) || (ey - 1 != vy))\*/) {  takingMove.destination = {ex + 2, ey - 2};  takingMove.victim = {ex + 1, ey - 1};  takingMove.takingSide = negateColor(checkerColor);  takingMove.victimType = getCheckerTypeOnBoard(situation, ex + 1, ey - 1);  takingMove.isASpecialMove = (checkerColor == Black && ey == 2);  situation->takingMovesBuffer[situation->tmBufferLen++] = takingMove;  }  }  if (ex > 1) {  if ((situation->board.boardRender[ey - 2][ex - 2] == EMPTY\_BLACK) && isAVictim(situation, checkerColor, ex - 1, ey - 1) && !isMarkedForDeath(situation,negateColor(checkerColor), ex - 1, ey - 1)/\* && ((ex - 1 != vx) || (ey - 1 != vy))\*/) {  takingMove.destination = {ex - 2, ey - 2};  takingMove.victim = {ex - 1, ey - 1};  takingMove.takingSide = negateColor(checkerColor);  takingMove.victimType = getCheckerTypeOnBoard(situation, ex - 1, ey - 1);  takingMove.isASpecialMove = (checkerColor == Black && ey == 2);  situation->takingMovesBuffer[situation->tmBufferLen++] = takingMove;  }  }  }  }  }  void findAllKingTakingMovesForOneOnDir(GameSituation\* situation, Color checkerColor, int checkerIndex, Coordinates directionShift) {  TakingMove takingMove;  takingMove.isASpecialMove = false;  takingMove.takingSide = negateColor(checkerColor);  int ex = situation->board.checkers[checkerColor][checkerIndex].coordinates.x;  int ey = situation->board.checkers[checkerColor][checkerIndex].coordinates.y;  takingMove.source = {ex, ey};  int victimDistance = findDistanceToVictim(situation, ex, ey, checkerColor, directionShift);  if (victimDistance != -1) {  takingMove.victim = {ex + victimDistance \* directionShift.x, ey + victimDistance \* directionShift.y};  takingMove.victimType = getCheckerTypeOnBoard(situation, ex + victimDistance \* directionShift.x, ey + victimDistance \* directionShift.y);  bool flag = longMoveLookingFlag(ex + victimDistance \* directionShift.x, ey + victimDistance \* directionShift.y, directionShift);  int additionalShift = 1;  while (flag) {  if (situation->board.boardRender[ey + directionShift.y \* (victimDistance + additionalShift)][ex + directionShift.x \* (victimDistance + additionalShift)] == EMPTY\_BLACK) {  takingMove.destination = {ex + directionShift.x \* (victimDistance + additionalShift), ey + directionShift.y \* (victimDistance + additionalShift)};  situation->takingMovesBuffer[situation->tmBufferLen++] = takingMove;  flag = longMoveLookingFlag(ex + directionShift.x \* (victimDistance + additionalShift), ey + directionShift.y \* (victimDistance + additionalShift), directionShift);  } else {  flag = false;  }  additionalShift++;  }  }  }  void findAllKBMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex) {  if (situation->board.checkers[checkerColor][checkerIndex].type != King) {  Move move;  move.isKingMove = false; move.isKingBecomingMove = true;  int ex = situation->board.checkers[checkerColor][checkerIndex].coordinates.x;  int ey = situation->board.checkers[checkerColor][checkerIndex].coordinates.y;  move.source = {ex, ey};  if (checkerColor == Black ? ey == 1 : ey == 6) {  if (ex < 7) {  if ((situation->board.boardRender[(checkerColor == Black ? ey - 1 : ey + 1)][ex+1] == EMPTY\_BLACK)) {  move.destination = {ex + 1, checkerColor == Black ? ey - 1 : ey + 1};  situation->regularMovesBuffer[situation->rmBufferLen++] = move;  }  }  if (ex > 0) {  if ((situation->board.boardRender[(checkerColor == Black ? ey - 1 : ey + 1)][ex-1] == EMPTY\_BLACK)) {  move.destination = {ex - 1, checkerColor == Black ? ey - 1 : ey + 1};  situation->regularMovesBuffer[situation->rmBufferLen++] = move;  }  }  }  }  }  void findAllKingTakingMovesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex) {  if (situation->board.checkers[checkerColor][checkerIndex].type == King) {  findAllKingTakingMovesForOneOnDir(situation, checkerColor, checkerIndex, NORTH\_EAST);  findAllKingTakingMovesForOneOnDir(situation, checkerColor, checkerIndex, NORTH\_WEST);  findAllKingTakingMovesForOneOnDir(situation, checkerColor, checkerIndex, SOUTH\_EAST);  findAllKingTakingMovesForOneOnDir(situation, checkerColor, checkerIndex, SOUTH\_WEST);  }  }  #endif //CHECKERS\_CLCENGINE\_MOVE\_SEARCHING\_H |

## Заголовочный файл clcengine\_moving.h

|  |
| --- |
| //  // Created by EgrZver on 11.05.2023.  //  #include "clcengine\_types.h"  #include "clcengine\_auxiliary.h"  #include "clcengine\_move\_searching.h"  #ifndef CHECKERS\_CLCENGINE\_MOVING\_H  #define CHECKERS\_CLCENGINE\_MOVING\_H  int makeAMove(GameSituation\* situation, Move move) {  Color movedColor = sideColor(situation, move.source.x, move.source.y);  int movedIndex = getCheckerIndexByCoordsAndColor(situation, move.source.x, move.source.y, movedColor);  situation->board.checkers[movedColor][movedIndex].coordinates = {move.destination.x, move.destination.y};  if (move.isKingBecomingMove) ascendChecker(&(situation->board.checkers[movedColor][movedIndex]));  updateBoardRender(&(situation->board));  flushMoveBuffers(situation);  if (move.isKingBecomingMove) {  findAllKingTakingMovesForOne(situation, movedColor, movedIndex);  if (situation->tmBufferLen == 0) {  findAllKingMovesForOne(situation, movedColor, movedIndex);  return 2;  }  else {  situation->tmBufferLen = 0;  return 1;  }  //return 1;  //findAllKingTakingMovesForOne(situation, movedColor, movedIndex);  //findAllKingMovesForOne(situation, movedColor, movedIndex);  //return 2;  } else return 0;  }  int makeATakingMove(GameSituation\* situation, TakingMove move) {  Color movedColor = negateColor(move.takingSide);  int movedIndex = getCheckerIndexByCoordsAndColor(situation, move.source.x, move.source.y, movedColor);  Color victimColor = move.takingSide;  int victimIndex = getCheckerIndexByCoordsAndColor(situation, move.victim.x, move.victim.y, victimColor);  situation->board.checkers[movedColor][movedIndex].coordinates = {move.destination.x, move.destination.y};  if (move.isASpecialMove) ascendChecker(&(situation->board.checkers[movedColor][movedIndex]));  situation->board.checkers[victimColor][victimIndex].markedForDeath = true;  flushMoveBuffers(situation);  updateBoardRender(&(situation->board));  if (situation->board.checkers[movedColor][movedIndex].type == King && !move.isASpecialMove) {  //return 0;  findAllKingTakingMovesForOne(situation, movedColor, movedIndex);  }  else if (!move.isASpecialMove) {  findAllTakingMovesForOne(situation, movedColor, movedIndex);  }  else {  return 0;  }  return situation->tmBufferLen;  }  int cancelAMove(GameSituation\* situation, Move move) {  Color movedColor = sideColor(situation, move.destination.x, move.destination.y);  int movedIndex = getCheckerIndexByCoordsAndColor(situation, move.destination.x, move.destination.y, movedColor);  situation->board.checkers[movedColor][movedIndex].coordinates = {move.source.x, move.source.y};  if (move.isKingBecomingMove) descendChecker(&(situation->board.checkers[movedColor][movedIndex]));  updateBoardRender(&(situation->board));  if (move.isKingBecomingMove) return -1; else return 0;  }  int cancelATakingMove(GameSituation\* situation, TakingMove move, bool emulated = false) {  Color movedColor = negateColor(move.takingSide);  int movedIndex = getCheckerIndexByCoordsAndColor(situation, move.destination.x, move.destination.y, movedColor);  situation->board.checkers[movedColor][movedIndex].coordinates = {move.source.x, move.source.y};  if (move.isASpecialMove) descendChecker(&(situation->board.checkers[movedColor][movedIndex]));  if (!emulated) {  Checker regeneratedVictim;  regeneratedVictim.coordinates = {move.victim.x, move.victim.y};  regeneratedVictim.type = move.victimType;  regeneratedVictim.color = move.takingSide;  regeneratedVictim.markedForDeath = false;  situation->board.checkers[move.takingSide][situation->board.checkersCount[move.takingSide]++] = regeneratedVictim;  }  else {  int falseVictimIndex = getCheckerIndexByCoordsAndColor(situation, move.victim.x, move.victim.y, move.takingSide);  situation->board.checkers[move.takingSide][falseVictimIndex].markedForDeath = false;  }  updateBoardRender(&(situation->board));  if (move.isASpecialMove) return -1; else return 0;  }  #endif //CHECKERS\_CLCENGINE\_MOVING\_H |

## Заголовочный файл clcengine\_sequence\_searching.h

|  |
| --- |
| //  // Created by EgrZver on 11.05.2023.  //  #include "clcengine\_types.h"  #include "clcengine\_auxiliary.h"  #include "clcengine\_move\_searching.h"  #include "clcengine\_moving.h"  #include "clcengine\_inits.h"  #ifndef CHECKERS\_CLCENGINE\_SEQUENCE\_SEARCHING\_H  #define CHECKERS\_CLCENGINE\_SEQUENCE\_SEARCHING\_H  void findRegularMoveSequenceForOne(GameSituation\* situation, Color checkerColor, int checkerIndex) {  findAllKBMovesForOne(situation, checkerColor, checkerIndex);  findAllRegularMovesForOne(situation, checkerColor, checkerIndex);  findAllKingMovesForOne(situation, checkerColor, checkerIndex);  if (situation->rmBufferLen != 0) {  Move\* buffer = new Move[situation->rmBufferLen];  for (int i = 0; i < situation->rmBufferLen; i++) buffer[i] = situation->regularMovesBuffer[i];  int savedRMs = situation->rmBufferLen;  for (int i = 0; i < savedRMs; i++) {  Move extracted = buffer[i];  if (extracted.isKingBecomingMove) {  flushMoveBuffers(situation);  int insertIndex;  makeAMove(situation, extracted);  if (situation->tmBufferLen == 0 && situation->rmBufferLen != 0) {  for (int j = 0; j < situation->rmBufferLen; j++) {  insertIndex = situation->rmsCount;  situation->regMoveSequences[insertIndex].regularMoves[0] = extracted;  situation->regMoveSequences[insertIndex].regularMoves[1] = situation->regularMovesBuffer[j];  situation->regMoveSequences[insertIndex].rmsCount = 2;  situation->rmsCount++;  }  }  else {  insertIndex = situation->rmsCount;  situation->regMoveSequences[insertIndex].regularMoves[0] = extracted;  //situation->regMoveSequences[insertIndex].regularMoves[1] = situation->regularMovesBuffer[j];  situation->regMoveSequences[insertIndex].rmsCount = 1;  situation->rmsCount++;  }  cancelAMove(situation, extracted);  }  else {  int insertIndex = situation->rmsCount;  situation->regMoveSequences[insertIndex].regularMoves[0] = extracted; // maybe 0  situation->regMoveSequences[insertIndex].rmsCount = 1;  situation->rmsCount++;  }  }  delete [] buffer;  }  }  void findAllRegularMoveSequences(GameSituation\* situation, Color forWhichSide) {  for (int i = 0; i < situation->board.checkersCount[forWhichSide]; i++) {  findRegularMoveSequenceForOne(situation, forWhichSide, i);  flushMoveBuffers(situation);  }  }  void findAllTakingSequencesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex, TakingSequence\* currentPath, bool afterMove = false) {  if (!afterMove) {  findAllKingTakingMovesForOne(situation, checkerColor, checkerIndex);  findAllTakingMovesForOne(situation, checkerColor, checkerIndex);  }  TakingMove\* buffer = new TakingMove[situation->tmBufferLen];  for (int i = 0; i < situation->tmBufferLen; i++) buffer[i] = situation->takingMovesBuffer[i];  int savedTMs = situation->tmBufferLen;  for (int i = 0; i < savedTMs; i++) {  TakingMove extracted = buffer[i];  appendToATakingSequence(currentPath, extracted);  int stat = makeATakingMove(situation, extracted);  if (stat == 0) {  copyToAnotherTakingSequence(currentPath, &(situation->takingSequences[situation->tmsCount++]));  }  else {  findAllTakingSequencesForOne(situation, checkerColor, checkerIndex, currentPath, true);  }  cancelATakingMove(situation, extracted, ENABLED\_EMULATION);  deleteLastFromATakingSequence(currentPath);  }  delete [] buffer;  }  void findAllTakingMoveSequences(GameSituation\* situation, Color forWhichSide) {  for (int i = 0; i < situation->board.checkersCount[forWhichSide]; i++) {  TakingSequence path = getNullPath();  findAllTakingSequencesForOne(situation, forWhichSide, i, &path);  flushMoveBuffers(situation);  }  }  /\*  void findAllTakingSequencesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex, TakingSequence\* currentPath, Move kingBecomingMoveToInsert, bool afterMove = false) {  if (!afterMove) {  findAllKingTakingMovesForOne(situation, checkerColor, checkerIndex);  findAllTakingMovesForOne(situation, checkerColor, checkerIndex);  }  TakingMove\* buffer = new TakingMove[situation->tmBufferLen];  for (int i = 0; i < situation->tmBufferLen; i++) buffer[i] = situation->takingMovesBuffer[i];  int savedTMs = situation->tmBufferLen;  for (int i = 0; i < savedTMs; i++) {  TakingMove extracted = buffer[i];  appendToATakingSequence(currentPath, extracted);  int stat = makeATakingMove(situation, extracted);  if (stat == 0) {  situation->mixedSequences[situation->mmsCount].kingBecomingMove = kingBecomingMoveToInsert;  copyToAnotherTakingSequence(currentPath, &(situation->mixedSequences[situation->mmsCount++].takingSequence));  }  else {  findAllTakingSequencesForOne(situation, checkerColor, checkerIndex, currentPath, true);  }  cancelATakingMove(situation, extracted, ENABLED\_EMULATION);  deleteLastFromATakingSequence(currentPath);  }  delete [] buffer;  }  \*/  /\*void findAllMixedSequencesForOne(GameSituation\* situation, Color checkerColor, int checkerIndex) {  findAllKBMovesForOne(situation, checkerColor, checkerIndex);  int savedTMs = 0;  if (situation->rmBufferLen != 0) {  Move\* buffer = new Move[situation->rmBufferLen];  for (int i = 0; i < situation->rmBufferLen; i++) buffer[i] = situation->regularMovesBuffer[i];  int savedRMs = situation->rmBufferLen;  for (int i = 0; i < savedRMs; i++) {  Move extracted = buffer[i];  situation->mixedSequences[situation->mmsCount].kingBecomingMove = extracted;  if (extracted.isKingBecomingMove) {  flushMoveBuffers(situation);  makeAMove(situation, extracted);  if (situation->tmBufferLen != 0) {  savedTMs = situation->tmBufferLen;  flushMoveBuffers(situation);  TakingSequence null = getNullPath();  findAllTakingSequencesForOne(situation, checkerColor, checkerIndex, &null, extracted);  }  cancelAMove(situation, extracted);  }  }  delete [] buffer;  }  }  \*/  /\*  void findAllMixedSequences(GameSituation\* situation, Color forWhichSide) {  for (int i = 0; i < situation->board.checkersCount[forWhichSide]; i++) {  findAllMixedSequencesForOne(situation, forWhichSide, i);  flushMoveBuffers(situation);  }  }  \*/  inline void findAllMoves(GameSituation\* situation, Color forWhichSide) {  findAllTakingMoveSequences(situation, forWhichSide);  if (situation->tmsCount == 0) {  //findAllMixedSequences(situation, forWhichSide);  findAllRegularMoveSequences(situation, forWhichSide);  }  }  #endif //CHECKERS\_CLCENGINE\_SEQUENCE\_SEARCHING\_H |

## Заголовочный файл clcengine\_sequence\_moving.h

|  |
| --- |
| //  // Created by EgrZver on 11.05.2023.  //  #include "clcengine\_types.h"  #include "clcengine\_moving.h"  #ifndef CHECKERS\_CLCENGINE\_SEQUENCE\_MOVING\_H  #define CHECKERS\_CLCENGINE\_SEQUENCE\_MOVING\_H  void makeARegMoveSequence(GameSituation\* situation, RegMoveSequence regMoveSequence) {  for (int i = 0; i < regMoveSequence.rmsCount; i++) makeAMove(situation, regMoveSequence.regularMoves[i]);  }  void cancelARegMoveSequence(GameSituation\* situation, RegMoveSequence regMoveSequence) {  for (int i = regMoveSequence.rmsCount - 1; i > -1; i--) cancelAMove(situation, regMoveSequence.regularMoves[i]);  }  void makeATakingMoveSequence(GameSituation\* situation, TakingSequence takingSequence) {  for (int i = 0; i < takingSequence.tmsCount; i++) makeATakingMove(situation, takingSequence.takingMoves[i]);  }  void cancelATakingMoveSequence(GameSituation\* situation, TakingSequence takingSequence) {  for (int i = takingSequence.tmsCount - 1; i > -1; i--) cancelATakingMove(situation, takingSequence.takingMoves[i]);  }  /\*  void makeAMixedSequence(GameSituation\* situation, MixedSequence mixedSequence) {  makeAMove(situation, mixedSequence.kingBecomingMove);  makeATakingMoveSequence(situation, mixedSequence.takingSequence);  }  void cancelAMixedSequence(GameSituation\* situation, MixedSequence mixedSequence) {  cancelATakingMoveSequence(situation, mixedSequence.takingSequence);  cancelAMove(situation, mixedSequence.kingBecomingMove);  }  \*/  #endif //CHECKERS\_CLCENGINE\_SEQUENCE\_MOVING\_H |

## Заголовочный файл clcengine\_analyzer.h

|  |
| --- |
| //  // Created by EgrZver on 11.05.2023.  //  #include <random>  #include "clcengine\_types.h"  #include "clcengine\_inits.h"  #include "clcengine\_sequence\_searching.h"  #include "clcengine\_moving.h"  #ifndef CHECKERS\_CLCENGINE\_ANALYZER\_H  #define CHECKERS\_CLCENGINE\_ANALYZER\_H  std::random\_device analyzerRD;  std::mt19937 agen(analyzerRD());  std::uniform\_int\_distribution<> analyzerDistribution(1, 1000);  SeqContainer analyze(GameSituation\* situation, Color side, int currentDepth, Difficulty maxDepth) {  SeqContainer toReturn = getNullContainer(side);  float topBorder = getVictoryEvalFor(negateColor(side));//container->eval = getVictoryEvalFor(negateColor(side));  findAllMoves(situation, side);  if (lostByMoves(situation)) {  toReturn.eval = topBorder;  return toReturn;  }  if (currentDepth == maxDepth) {  toReturn.eval = evalQuality(situation);  return toReturn;  }  if (situation->tmsCount == 0) {  /\*MixedSequence\* msBackup = new MixedSequence[situation->mmsCount];  for (int i = 0; i < situation->mmsCount; i++) msBackup[i] = situation->mixedSequences[i];  int savedMSS = situation->mmsCount;  RegMoveSequence\* rmsBackup = new RegMoveSequence[situation->rmsCount];  for (int i = 0; i < situation->rmsCount; i++) rmsBackup[i] = situation->regMoveSequences[i];  int savedRMSS = situation->rmsCount;  for (int i = 0; i < savedMSS; i++) {  makeAMixedSequence(situation, msBackup[i]);  removeMarkedForDeath(situation, negateColor(side));  flushSequenceLists(situation);  SeqContainer deepMoveAnalyzed = analyze(situation, negateColor(side), currentDepth + 1, maxDepth);  cancelAMixedSequence(situation, msBackup[i]);  if (replaceConditionFor(deepMoveAnalyzed.eval, topBorder, side)) {  topBorder = deepMoveAnalyzed.eval;  toReturn.eval = topBorder;  toReturn.mixedSequence = msBackup[i];  toReturn.seqNumberToDo = 3;  }  }  delete [] msBackup;\*/  RegMoveSequence\* rmsBackup = new RegMoveSequence[situation->rmsCount];  for (int i = 0; i < situation->rmsCount; i++) rmsBackup[i] = situation->regMoveSequences[i];  int savedRMSS = situation->rmsCount;  for (int i = 0; i < savedRMSS; i++) {  makeARegMoveSequence(situation, rmsBackup[i]);  flushSequenceLists(situation);  SeqContainer deepMoveAnalyzed = analyze(situation, negateColor(side), currentDepth + 1, maxDepth);  cancelARegMoveSequence(situation, rmsBackup[i]);  if (replaceConditionFor(deepMoveAnalyzed.eval, topBorder, side)) {  topBorder = deepMoveAnalyzed.eval;  toReturn.eval = topBorder;  toReturn.regMoveSequence = rmsBackup[i];  toReturn.seqNumberToDo = 1;  }  else if (deepMoveAnalyzed.eval == topBorder) {  if (analyzerDistribution(agen) < 500) {  //topBorder = deepMoveAnalyzed.eval;  //toReturn.eval = topBorder;  toReturn.regMoveSequence = rmsBackup[i];  toReturn.seqNumberToDo = 1;  }  }  }  delete [] rmsBackup;  return toReturn;  }  else {  TakingSequence\* tsBackup = new TakingSequence[situation->tmsCount];  for (int i = 0; i < situation->tmsCount; i++) tsBackup[i] = situation->takingSequences[i];  int savedTSS = situation->tmsCount;  for (int i = 0; i < savedTSS; i++) {  makeATakingMoveSequence(situation, tsBackup[i]);  removeMarkedForDeath(situation, negateColor(side));  flushSequenceLists(situation);  SeqContainer deepMoveAnalyzed = analyze(situation, negateColor(side), currentDepth + 1, maxDepth);  cancelATakingMoveSequence(situation, tsBackup[i]);  if (replaceConditionFor(deepMoveAnalyzed.eval, topBorder, side)) {  topBorder = deepMoveAnalyzed.eval;  toReturn.eval = topBorder;  toReturn.takingSequence = tsBackup[i];  toReturn.seqNumberToDo = 2;  }  else if (deepMoveAnalyzed.eval == topBorder) {  if (analyzerDistribution(agen) < 500) {  //topBorder = deepMoveAnalyzed.eval;  //toReturn.eval = topBorder;  toReturn.takingSequence = tsBackup[i];  toReturn.seqNumberToDo = 2;  }  }  }  delete [] tsBackup;  return toReturn;  }  }  SeqContainer getBestMove(GameSituation situation, Color side, Difficulty depth) {  GameSituation copyOfGS = situation;  return analyze(&copyOfGS, side, 0, depth);  }  #endif //CHECKERS\_CLCENGINE\_ANALYZER\_H |

## Заголовочный файл clcengine.h

|  |
| --- |
| //  // Created by EgrZver on 11.05.2023.  //  #ifndef CHECKERS\_CLCENGINE\_H  #define CHECKERS\_CLCENGINE\_H  #include "clcengine\_types.h"  #include "clcengine\_inits.h"  #include "clcengine\_auxiliary.h"  #include "clcengine\_move\_searching.h"  #include "clcengine\_moving.h"  #include "clcengine\_sequence\_searching.h"  #include "clcengine\_sequence\_moving.h"  #include "clcengine\_analyzer.h"  #endif //CHECKERS\_CLCENGINE\_H |