Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВОЙ ПРОЕКТ**

**Дисциплина:** Алгоритмы и структуры данных

**Тема:** Разработка GUI приложения-игры terra incognita и решателя для неё.

Выполнил

студент гр. 3530901/90003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Коротких В.Д.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ахин М.Х.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург   
2020

**Оглавление**

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc58595498)

[**МЕТОД РЕШЕНИЯ** 5](#_Toc58595499)

[**РАБОТА ПРОГРАММЫ** 11](#_Toc58595500)

**ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Проект представляет собой упрощённую реализацию известной игры "Лабиринт" или "Terra incognita". Подробная информация о правилах полноценной реализации имеется в статье Википедии "Лабиринт (игра на бумаге)".

Имеется лабиринт. Обозначения: # стена (недоступна для прохода), T сокровище, S вход, E выход, 0-9 ямы с заданным номером (wormholes). Играет один игрок. Его задача - передвигаясь по лабиринту в темноте (делая шаги в любую из четырёх сторон), найти в нём сокровище, а затем выход из лабиринта. Если игрок достиг выхода, а сокровище ещё не найдено, то игроку необходимо продолжить искать сокровище. При попадании в яму номер N игрок автоматически перескакивает в яму номер N+1, а если эта яма последняя -- то в яму номер 0. Общее количество ям неизвестно игроку.

Игрок выигрывает, достигнув выхода с сокровищем, и проигрывает, израсходовав лимит ходов (ходы, упирающиеся в стену, при этом не считаются).

Ограничения на лабиринты:

* Все лабиринты - прямоугольные, со всех сторон окружённые стенами
* Максимальный размер 40 (ширина) х 25 (высота), минимальный 2 х 2
* Есть один вход, один выход, и минимум одно сокровище
* Лабиринт может содержать от 0 до 10 ям, нумерация их должна быть непрерывной от нуля до N - 1, где N - их число. Не может быть двух ям с одинаковым номером.

Необходимо создать GUI приложение-игру terra incognita с вышеописанными правилами. На главном экране расположены 3 кнопки: «Играть», «Правила» и «Выход». При нажатии кнопки «Правила» выводить экран с описанием правил. При нажатии кнопки «Выход» - диалоговое окно с текстом «Вы уверены, что хотите выйти?» и двумя кнопками «Да» и «Нет».

При нажатии кнопки «Играть» выводится экран с полем ввода имени игрока, размера лабиринта, если игрок хочет сыграть в один из предустановленных лабиринтов, кнопка «Загрузить лабиринт из файла», при нажатии которой выдаётся подсказка по правильному формату файла и открывается диалоговое окно для указания пути к файлу. Лимит ходов предустановлен для каждого дефолтного лабиринта. Для загруженных лабиринтов из файла игрок сам должен задать лимит в соответствующем поле. После этого проводится проверка файла и выдаётся сообщение о корректности. Также на экране присутствует кнопка «Начать», после нажатия которой начинается игра. На экране отрисовывается начальная клетка лабиринта (старт) и остальные клетки лабиринта по мере прохождения их игроком. Также на экране есть кнопка помощь, при нажатии которой игроку предлагается решить лабиринт и показать путь прохождения (либо вывести сообщение о невозможности прохождения данного лабиринта). Это так называемый решатель, особенность которого обязательно состоит в том, что он не видит карту лабиринта, а перед началом игры ему сообщаются его начальные координаты и размеры лабиринта. Затем мастер последовательно запрашивает у него очередной ход и сообщает результат хода. В результате решателю сообщается, удалось ли перемещение, что было в той клетке, куда он пытался пойти и текущее состояние (найдено сокровище / не найдено, выход достигнут / не достигнут). Решателю более НЕ СООБЩАЮТСЯ его текущие координаты.

Ссылка на гитхаб: <https://github.com/vitekkor/terra-incognita>

**МЕТОД РЕШЕНИЯ**

Метод решения лабиринта заключается в следующем:

Мастер запрашивает ходы (лимит – 1000). На каждом запросе хода решатель проверяет, нашёл ли он сокровище и знает ли он где выход. Если нет, то он берёт список направлений, фильтрует его от тех направлений, при использовании которых он из текущего положения попадёт в положение, в котором он уже был (хранятся в карте map), а затем случайным образом берёт одно из них и создаёт WalkMove с этим направлением. Если вдруг после фильтрации у нас пустое множество, то просто берём случайно одно из всех 4х направлений.

При установке мастером результата хода, решатель заполняет свою карту и отмечает, нашел ли он сокровище, нашёл ли он выход. При попадании в червоточину карта очищается и решатель отмечает, что выход он не нашёл (он не знает в какую позицию попал => не может знать и где выход).

Если же решатель нашёл сокровище и знает, где выход, то вызывается функция findPathToExit (from: Location, previousPath: MutableList<Direction> = mutableListOf()): MutableList<Direction>? из текущего положения. Если по карте map from это выход, то просто возвращаем previousPath. Иначе она добавляет положение from в свой список посещённых (visited), затем фильтрует направления и отбрасывает те, по которым мы уже ходили (from + direction in visited) и те, которые ведут нас в стену или в червоточину (таким образом, мы избегаем ненужных ходов в стену и провалов в червоточину, которые влекут за собой стирание карты и забытие, где выход) и возвращаем результат работы рекурсивно вызванной функцию поиска пути findPathToExit, но уже из новых локаций from + direction (те, которые мы отфильтровали) и с previousPath, содержащим весь предыдущий путь (добавляем к нему и новое направление), а также из всех путей выбираем путь с минимальной длинной.

В итоге функция вернула нам список direction и теперь решатель по одному достаёт из него направления и возвращает мастеру WalkMove(direction) при вызове getNextMove().

Это была одна итерация. Чтобы повысить шансы на нахождение пути, прибегнем к следующему (метод searchPath(labyrinth: Labyrinth, moveLimit: Int): List<Move>):

Создаём Searcher, GameMaster с переданными в него лабиринтом и игроком Searcher. Затем делаем 10000 итераций, на каждой из которых сначала сбрасываем Searcher и GameMaster, затем заставляем мастера запрашивать ходы (gameMaster.makeMoves(1000)), после чего анализируем результат: если достигли выхода и размер пути удовлетворяет заданному лимиту, то возвращаем этот путь. Иначе дальше повторяем итерации. Если под конец всех итераций путь так и не был найден, возвращаем пустой список.

В проекте использована концепция MVC (Model-View-Controller) для отделения бизнес-логики от визуализации. Код разделен на четыре пакета: controller, model.core, view. Также имеются файлы вне этих пакетов:

* main.kt – точка входа в программу
* MyApp.kt – класс-приложение
* Styles.kt – класс, содержащий стили оформления программы

Содержимое пакета view:

* MainMenuView.kt – отвечает за отображение главного меню (три кнопки: Play, Rules, Exit).
* RulesView.kt – отвечает за отображение правил игры.
  + Метод rulesHBox(text: String, nameOfTile: String): HBox – возвращает HBox с текстом text и с картинкой после него, которая используется для отображения клетки лабиринта (пустая клетка, вход, выход, сокровище, стена. nameOfTile – имя необходимой клетки).
* ExitView.kt – отвечает за отображение вопроса, действительно ли пользователь хочет выйти из игры. Содержит 2 кнопки: Yes – закрывает игру, No – возвращает в главное меню.
* GamePreView.kt – отвечает за отображение экрана с вводом имени игрока, лимита ходов, с выбором лабиринта (из предустановленных или из пользовательского файла)
* MainView.kt – контейнер BorderPane. В его центре помещаются MainMenuView, RulesView, ExitView или GamePreView в зависимости от того, где находится пользователь. В вверху отображается название игры «Terra Incognita».
* GameView.kt – отвечает за отображение самой игры (лабиринт, игрок, кнопка, которая запускает поиск и отображение решения лабиринта, имя игрока, оставшиеся ходы).
  + Метод setMovesLeft(movesLeft: Int) – необходим для установки количества оставшихся ходов на экране
  + Метод setPlayersName(name: String) – необходим для установки имени игрока
  + Метод newGame() – начать новую игру. Старый лабиринт с экрана удаляется, загружается новый. Лимит ходов также меняется на новый.
  + Метод movableAndZoomableStackpane(stackPane: StackPane) – делает stackPane подвижным (можно перетаскивать карту лабиринта мышкой) и зумируемым (можно приближать и отдалять).

Cодержимое пакета controller:

* MyController.kt – отвечает за связь игровой логики и её отображения на экране.
  + Метод loadLabyrinth(file: File? = null, size: Pair<Int, Int>? = null): Boolean – необходим для загрузки лабиринта из файла пользователя (file) или из файла одного из предустановленных лабиринтов размера size. Возвращает True, если загрузка произошла успешно и false, если файл неправильно отформатирован.
  + Метод load(file: File): Boolean – приватный метод загрузки лабиринта. Вызывается из метода loadLabyrinth
  + Метод showAlert(message: String) – необходим для отображения всплывающего сообщения об ошибке о неудачной попытке создания лабиринта. Message - строка подробного сообщения об исключении, которое было сгенерировано при попытке создать лабиринт.
  + Метод loadAssets() – загружает картинки лабиринта из ресурсов
  + Метод getTile(type: String): ImageView - Возвращает изображение клетки лабиринта. Type – тип клетки (пустая, стена, сокровище и т.д). По умолчанию возвращает пустую клетку.
  + Метод createMap(): StackPane – Создает карту лабиринта, который был загружен контроллером (путем вызова метода loadLabyrinth). Возвращает StackPane с расположенными на нём изображениями клеток лабиринта и изображением игрока. Используется в GameView
  + Приватный метод getEmptyCell(x: Int, y: Int): ImageView – возвращает изображение пустой клетки с правильным смещением (задаётся параметрами x и y). Необходим для правильного расположения изображения клетки на StackPane при создании карты (метод createMap()).
  + Приватный метод getStartOrExit(entrance: Boolean, x: Int, y: Int): ImageView – аналогичен предыдущему методу, но возвращает изображение не пустой клетки, а входа (или выхода).
  + Метод showMoveResult(result: MoveResult) – необходим для отображения статуса хода (успешный – попали в пустую клетку/вход/выход/клетку с сокровищем/клетку с червоточной, неудачный – уперлись в стену) в всплывающем окошке и для отрисовки анимации перемещения игрока
  + Метод makeMove(direction: Direction) – необходим для совершения хода игроком. Direction – направление, по которому хочет пойти игрок. Передает информацию GameMaster’у.
  + Метод startGame(): Int – необходим для того, чтобы начать игру. Создаёт игрока Human(), если он ещё не был создан, GameMaster’а (аналогично), восстанавливает собранные сокровища в лабиринте, ставит игрока в начальную позицию. Возвращает лимит ходов.
  + Приватный метод endGame(result: GameMaster.GameResult) – показывает диалоговое окно об окончании игры и её результатом – выиграл ли игрок и за сколько ходов, если выиграл. В окне могут быть 2 кнопки в случае проигрыша – Try again и Menu. В случае выигрыша – 3 кнопки: Play Again, Play another labyrinth и Menu
  + Приватный метод playAgain() – необходим, для того, чтобы сыграть заново в тот же лабиринт
  + Метод exitFromGameView() – Показывает сообщение с вопросом, действительно ли игрок хочет прекратить играть. 2 кнопки – Yes и No.
  + Приватный метод createDialog(title: String, headerText: String, contentText: String, alertType: Alert.AlertType): Alert – Создаёт диалоговое окно с определёнными заголовком (title), верхним текстом (headerText), главным текстом (contentText) и типом окна (Alert.AlertType)
  + Метод passLabyrinth() – необходим для решения лабиринта (вызывает решателя из логики игры и передаёт ему лимит ходов и лабиринт, который необходимо решить). Если есть решение, то показывает сообщение об этом и начинает отображать путь, если нет, то показывает соответствующее сообщение.
  + Приватный метод displayPassageCompleted() – необходим для вывода окна с сообщением об успешном показе пути прохождения лабиринта
  + DX\_X, DX\_Y, DY\_X, DY\_Y – константы смещения

Содержимое пакета model.core:

* Пакет labyrinth – содержит GameMaster.kt и Labyrinth.kt
* GameMaster.kt – Класс GameMaster, управляющий всей игрой. Он обеспечивает связь между игроком и лабиринтом.
  + Метод reset() – Вызывается, когда нужно снова начать играть. Сбрасывает состояние GameMaster. GameMaster возвращается в исходное состояние (состояние, когда экземпляр класса только что был создан).
  + Метод setNewPlayer(newPlayer: Player): Boolean – устанавливает нового игрока и сбрасывает состояние GameMaster. Возвращает True, если новый игрок не является старым игроком и false в обратном случае.
  + Метод setNewLabyrinth(newLabyrinth: Labyrinth): Boolean – устанавливает новый лабиринт и сбрасывает состояние GameMaster. Возвращает True, если новый лабиринт не является старым лабиринтом и false в обратном случае.
  + Метод makeMoves(moveLimit: Int): GameResult – мастер постоянно запрашивает следующий ход у игрока, покак тот не выиграет либо не закончится лимит ходов. Возвращает результат игры (количество сделанных ходов и Boolean значение, достиг ли игрок выхода).
  + Метод makeMove(): GameResult – мастер запрашивает у игрока только 1 ход и возвращает его результат. Используется в методе makeMoves(moveLimit: Int): GameResult
* Labyrinth.kt – лабиринт
  + Метод recover() – восстанавливает сокровища. Необходим перед тем, как проходить один и тот же лабиринт заново
  + Статический метод createFromFile(fileName: String) = createFromFile(File(fileName)) – создаёт лабиринт из файл с именем filename
  + Статический метод createFromFile(file: File): Labyrinth – создаёт лабиринт из файла. Выбрасывает IllegalArgumentException, если файл лабиринта отформатирован неправильно, и IllegalStateException, если червоточины размещены неправильно
  + Приватный статический метод checkWormholes(map: HashMap<Location, Room>, wormholeMap: Map<Location, Location>): Boolean – проверяет корректность расставления червоточин. True – если всё верно, false – если нет.
* Пакет player – содержит интерфейс Player, абстрактный класс AbstractPlayer и реализации этого класса Searcher, Human, HumanLike(наследуется от Human).
* Player.kt – интерфейс игрока.
  + Метод setStartLocationAndSize(location: Location, width: Int, height: Int) – Вызывается мастером для установки начального местоположения игрока и размера лабиринта.
  + Метод getNextMove(): Move – Вызывается мастером, чтобы получить следующий ход игрока
  + Метод setMoveResult(result: MoveResult) – Вызывается мастером для установки результата последнего хода игрока
* AbstractPlayer.kt – абстрактный игрок. Переопределяет метод setStartLocationAndSize. От него наследуются все реализации игроков.
* Human.kt – класс игрок-человек. Необходим для того, чтобы пользователь мог совершать ходы (у Human() GameMaster и запрашивает следующий ход)
  + Метод setNextMove(newMove: Move) – необходим для того, чтобы установить следующий ход игрока при нажатии соответствующей кнопки пользователем
* HumanLike.kt – класс человекоподобный игрок. Имитирует ходы человека. В него загружается список ходов (результат работы решателя), из которого он последовательно выдаёт один за другим (при вызове мастером getNextMove()).
* Searcher.kt – класс решатель лабиринта. Необходим для поиска пути выхода из лабиринта.
  + Приватный метод findPathToExit(from: Location, previousPath: MutableList<Direction> = mutableListOf()): MutableList<Direction>? – необходим для поиска пути к выходу. Вызывается только когда решатель уже нашёл сокровище и выход.
  + Приватный метод reset() – сбрасывает состояние решателя
  + Статический метод searchPath(labyrinth: Labyrinth, moveLimit: Int): List<Move> – ищет путь в лабиринте и возвращает список ходов, которые необходимо совершить. Если путь невозможно найти (с заданным количеством moveLimit или просто пути нет), то возвращает пустой список

Корень пакета model.core

* Condition.kt – класс хранящий состояние игрока после хода (список сокровищ, достиг ли он выхода или нет).
* Direction.kt – класс-перечисление направлений движения. 4 направления – NORTH (вверх), EAST (вправо), SOUTH (вниз), WEST (влево).
  + Метод plus(location: Location) – Возвращает новую локацию, полученную перемещением в текущем направлении из текущей локации location
  + Метод turnRight() – Возвращает следующее направление при повороте от текущего вправо.
  + Метод turnLeft() – Возвращает следующее направление при повороте от текущего влево.
  + Метод turnBack() – Возвращает противоположное направление.
* Location.kt – положение игрока в декартовой системе координат. (0, 0) находится в верхнем левом углу (не включая внешние стены).
  + Метод plus(other: Location): Location – Возвращает новую локацию, полученную сложением координат текущей и other локации.
  + Метод plus(direction: Direction): Location – Возвращает новую локацию, полученную перемещением в direction направлении из текущей локации.
* Move.kt – содержит класс Move (движение) и наследуемые от него объект WaitMove (игрок не ход, просто ожидает) и класс WalkMove(val direction: Direction) (движение хождения в направлении direction)
* MoveResult.kt – класс MoveResult, хранящий результат хода: room – Какая комната была целью, condition – Состояние после перемещения, successful – успешный был ход или нет, status – строка, описывающая статус (уперся в стену, попал в пустую клетку и т.д.)
* Room.kt – содержит класс Room (комната. Может содержать в себе какие-то предметы (по умолчанию ничего нет) – в данном случае сокровища) и наследующиеся от него объекты Empty (пустая комната), Wall (стена), Entrance (вход), Exit (выход) и классы WithContent (комната с сокровищем) и Wormhole (червоточина)
* RoomContent.kt – содержит класс RoomContent (предмет в комнате), класс Item (то, что игрок может взять с собой), наследующийся от RoomContent и объект Treasure(сокровище), наследующийся от Item.

# **РАБОТА ПРОГРАММЫ**

В главном меню пользователь может начать играть (кнопка Play – переход к GamePreView), прочитать правила игры (кнопка Rules – переход к RulesView) и выйти из игры (кнопка Exit – переход к ExitView). При нажатии на кнопку escape переходим к ExitView.

В RulesView описаны правила игры. При нажатии на кнопку escape возвращаемся в главное меню. В ExitView спрашивается, действительно ли пользователь хочет выйти из игры. При нажатии на кнопку escape возвращаемся в главное меню.

В GamePreView он задает своё имя, лимит ходов и выбирает лабиринт: либо из уже предложенных, выбирая размер в выпадающих списках, либо в проводнике из txt файла (кнопка Choose labyrinth from file). Затем для начала игры нажимает кнопку Start (начинается загрузка лабиринта и в случае успеха переход к GameView). При нажатии на кнопку escape возвращаемся в главное меню.

Если в файле есть ошибки форматирования, то игра не начнётся, а пользователю будет сообщено об ошибке. При наведении курсора на иконку крестика, будет выведено всплывающее сообщение с сведениями о правильном форматировании файла.

Если всё в порядке, то начинается игра. Пользователь может управлять игроком на экране с помощью кнопок W A S D. При совершении хода на экране становятся видимыми те клетки, в которые хочет переместиться игрок. Если это не стены (проход через них запрещён), то игрок перемещается в эту клетку. Если это червоточина, то ещё перемещается в следующую по порядковому номеру червоточину (либо в 0ю, если это была последняя червоточина).

Также на экране отображается лимит ходов и имя игрока.

Помимо этого, на экране есть кнопка Help, при нажатии на которую запускается решатель лабиринта. Затем выводится сообщение, удалось ли решить лабиринт с заданным лимитом ходов, и если удалось, то показывается путь. Нажатия пользователя на кнопки W A S D при этом игнорируются.

Иллюстрации работы программы:



Рис.1. Главное меню



Рис.2.1. Правила игры



Рис.2.2. Правила игры



Рис.3. Экран ввода имени, лимита ходов и выбора лабиринта



Рис.4.1. Выбор размера предустановленного лабиринта



Рис.4.2. Выбор размера предустановленного лабиринта



Рис.4.3. Выбор размера предустановленного лабиринта



Рис.4.5. Выбор размера предустановленного лабиринта



Рис.5. Ошибка – имя игрока не введено



Рис.6.1. Ошибка – лимит ходов не введён



Рис.6.2. Ошибка – лимит ходов задан неверно



Рис.6.3. Ошибка – лимит ходов задан неверно

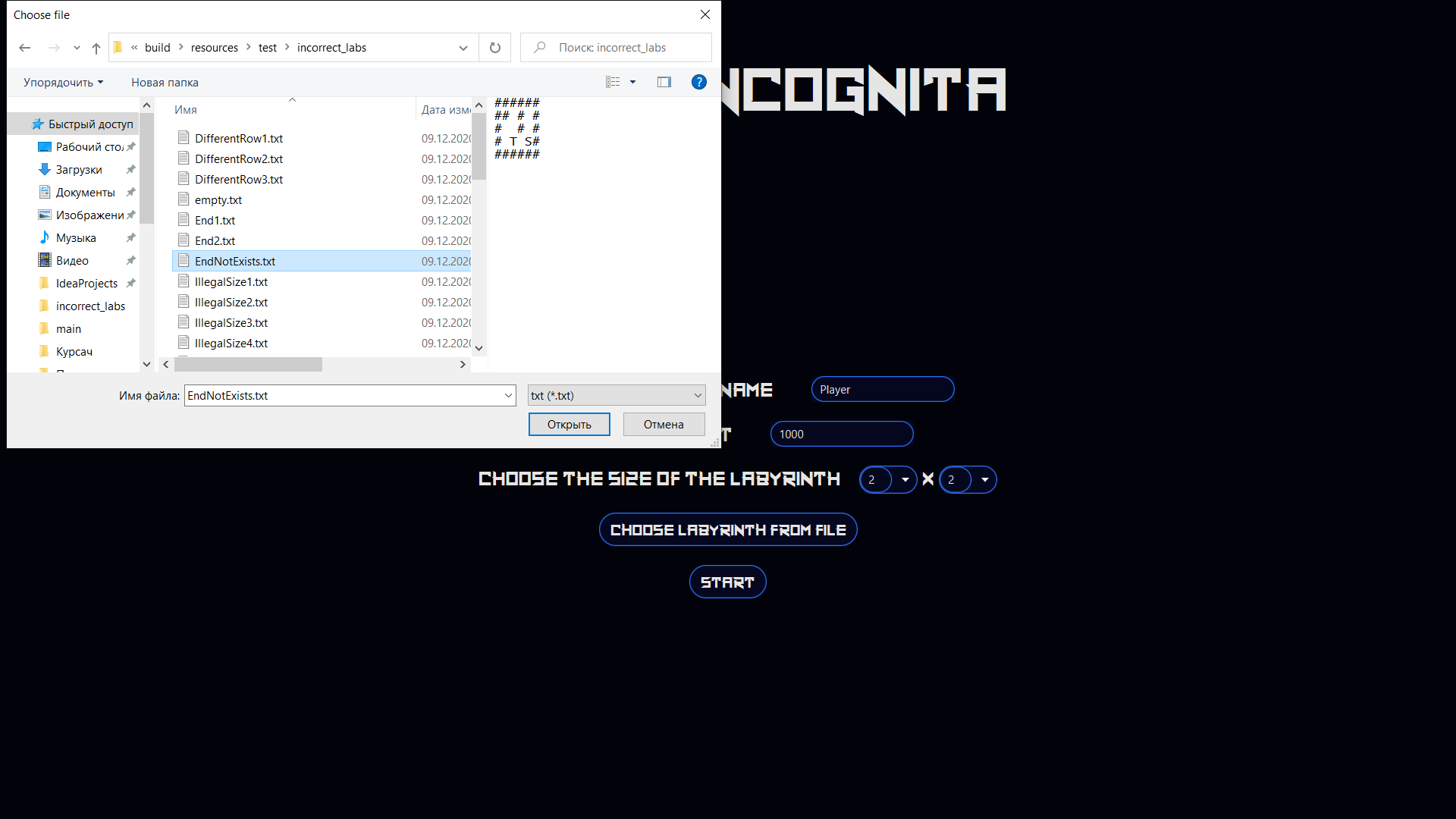


Рис.7.1. Выбор лабиринта из файла



Рис.7.2. Лабиринт выбран из файла

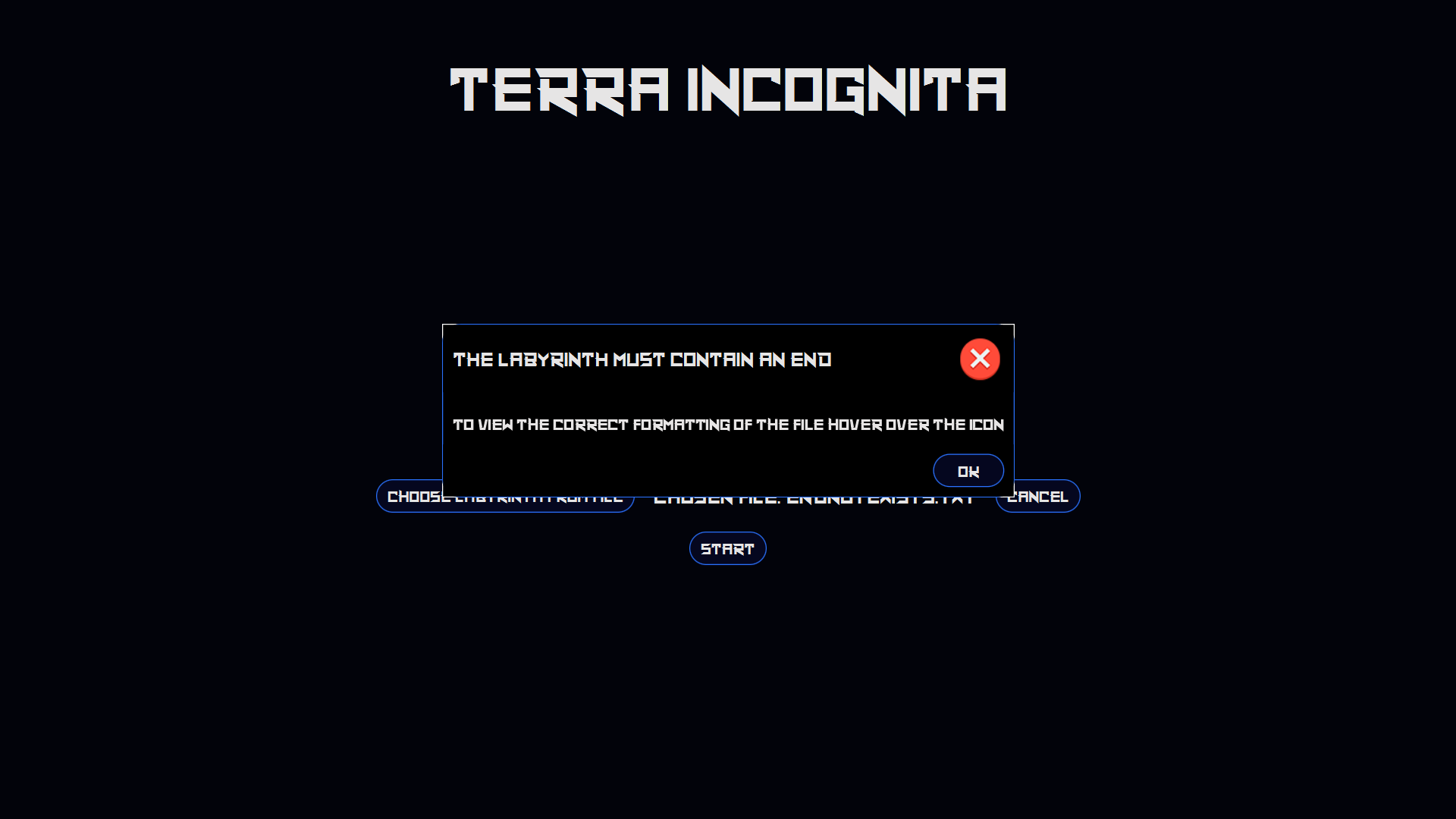


Рис.8.1. Ошибка в форматировании файла – лабиринт должен содержать выход

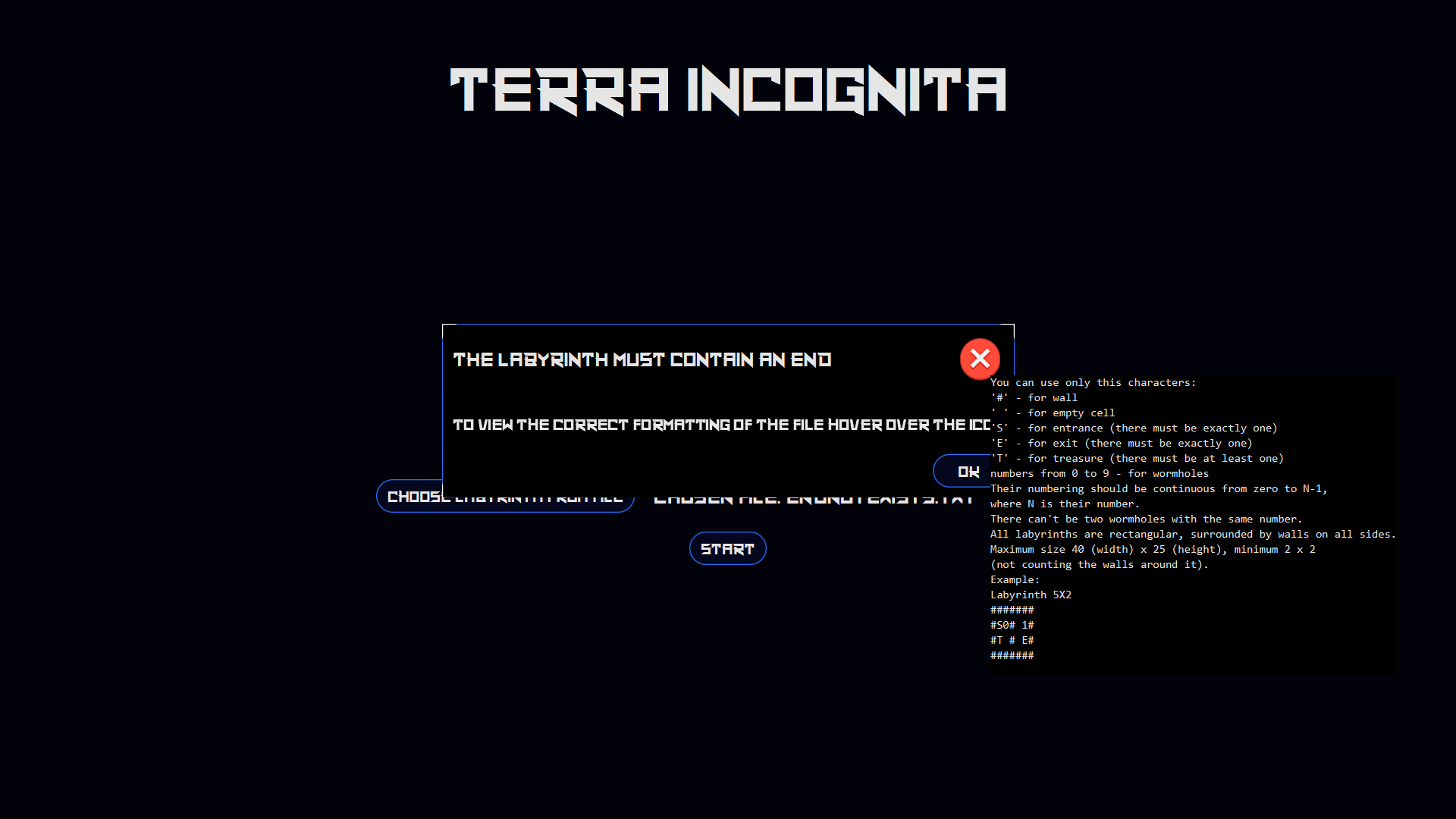


Рис.8.2. Ошибка в форматировании файла – вывод справочной информации при наведении на крестик



Рис.9. Начало игры



Рис.10. Процесс игры – появление пройденных клеток и сообщения о том, в какую клетку игрок попал



Рис.11. Собрано сокровище



Рис.12. Упёрлись в стену



Рис.13.1. Провалились в червоточину



Рис.13.2. Провалились в червоточину



Рис.14. Выигрыш



Рис.15. Проигрыш



Рис.16. Нашли выход, но не нашли сокровище

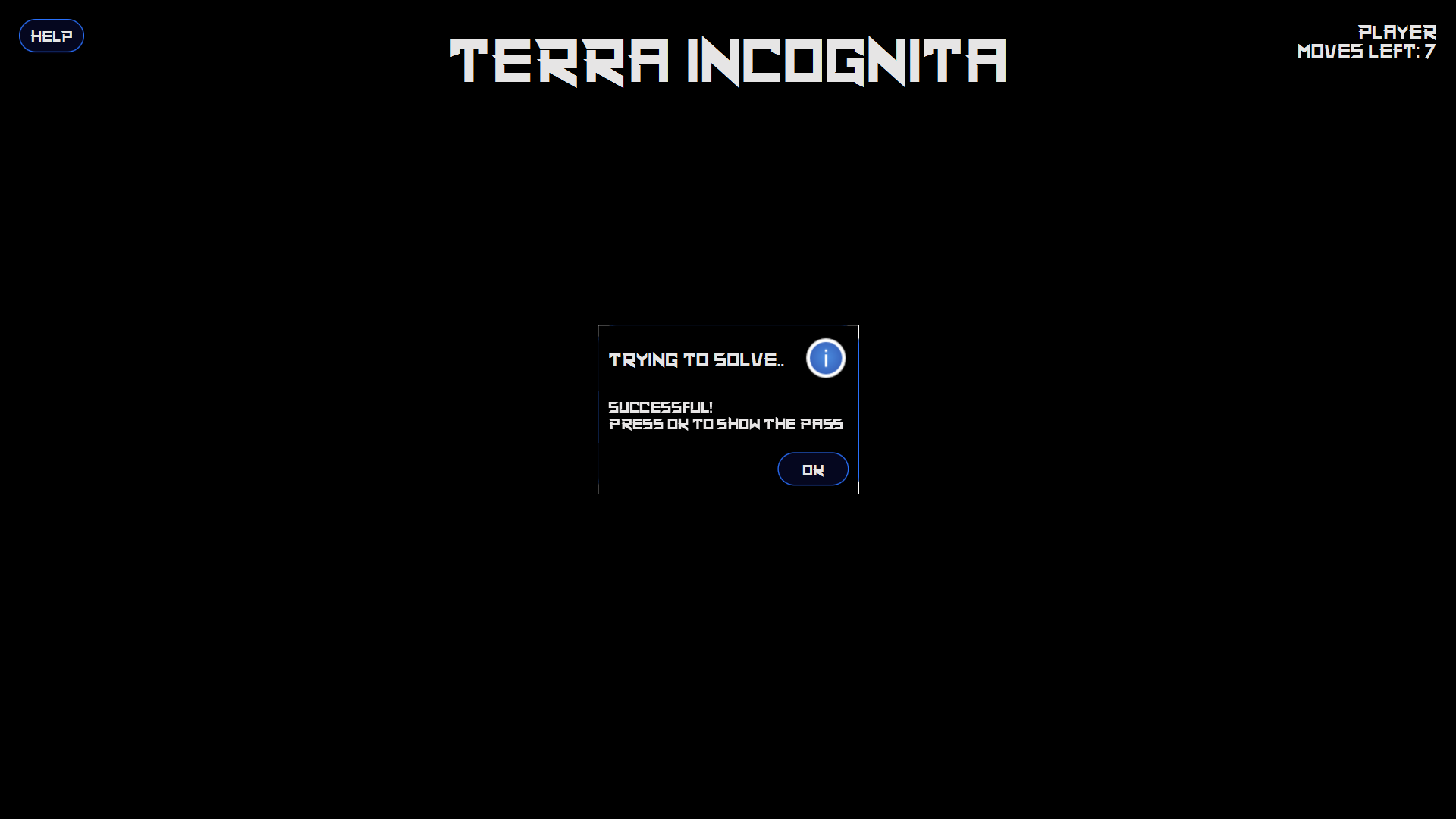


Рис.17. Решатель нашёл путь к выходу и сокровище за заданный лимит ходов



Рис.18. Решатель закончил показ прохождения лабиринта

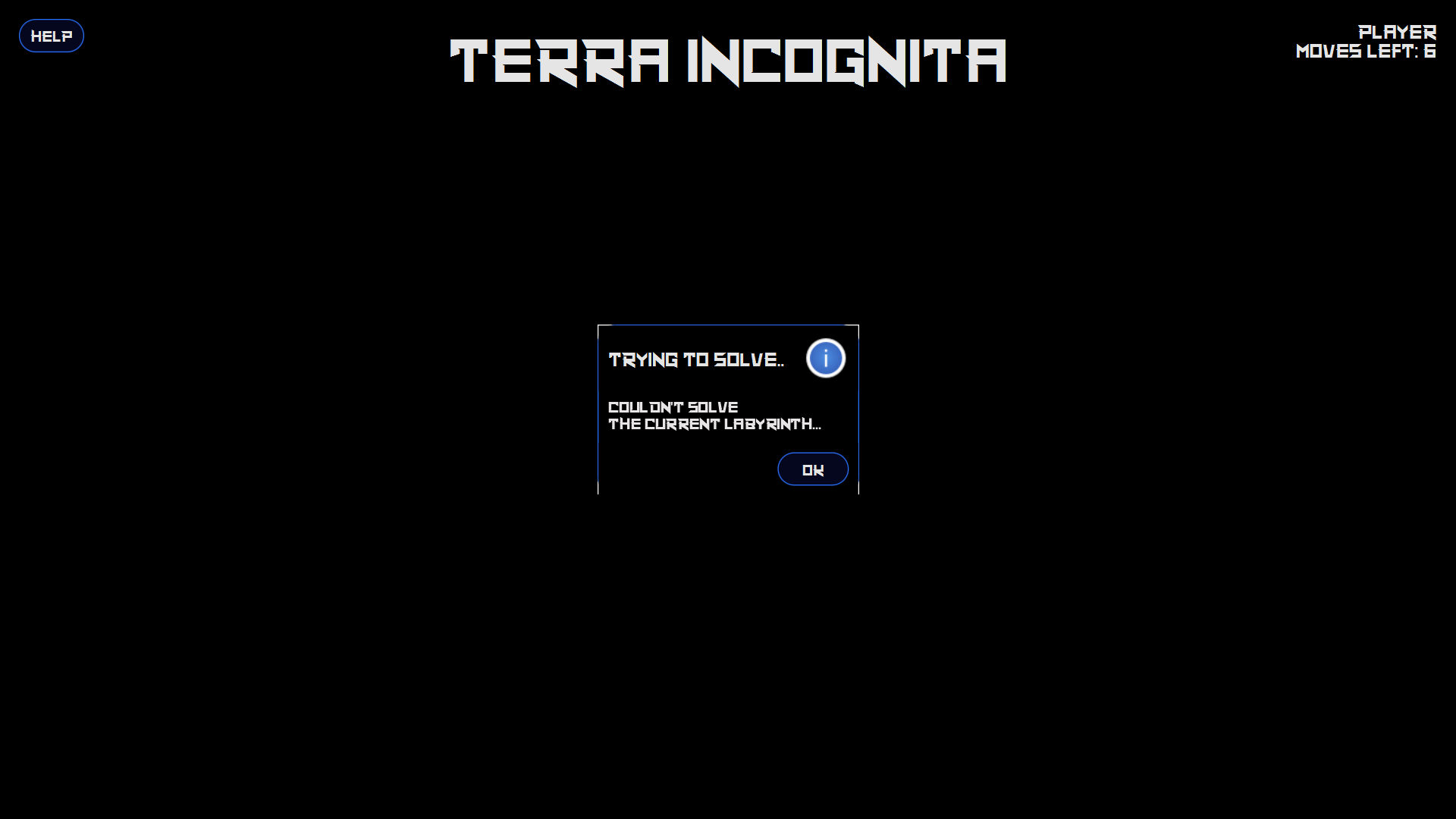


Рис.19. Решатель не может найти путь к выходу и сокровище за заданный лимит ходов



Рис.20. Экран выхода из игры