МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Игра на основе красно-чёрное дерево

Студентка гр. 9304 _____ Селезнёва А.В.

Студент гр. 9304 _____ Тиняков С.А.

Студент гр. 9304 _____ Цаплин И.В.

Руководитель _____ Фиалковский М.С.

Санкт-Петербург
2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Селезнёва А.В. группы 9304
Студент Тиняков С.А. группы 9304
Студент Цаплин И.В. группы 9304

Тема практики: Игра на основе красно-чёрное дерево

Задание на практику:

Командная итеративная разработка игры на основе алгоритма(ов) на Kotlin с графическим интерфейсом.

Алгоритм: Красно-чёрное дерево, вставка/удаление/поиск.

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 14.07.2021

Дата защиты отчета: 14.07.2021

Студентка	_____	Селезнёва А.В.
Студент	_____	Тиняков С.А.
Студент	_____	Цаплин И.В.
Руководитель	_____	Фиалковский М.С.

АННОТАЦИЯ

Основной целью учебной практики является получения навыков работы в команде и разработки первого проекта. Также целью является изучение новых технологий, например, языка программирования Kotlin или фреймворка для визуализации. Проект является создание игры с графическим интерфейсом на Kotlin. В основе игры лежит алгоритм красно-чёрного дерева — самобалансирующееся двоичного дерева поиска. Для проверки корректности алгоритма использовалось юнит-тестирование при помощи фреймворка Junit.

SUMMARY

The main goal of the practice work is to learn how to work in a team and develop your first project. The goal is also to learn new technologies, such as the Kotlin programming language or a visualization framework. The project is to create a game with a graphical interface on Kotlin. The game is based on the red-black tree algorithm, a self-balancing binary search tree. To test the correctness of the algorithm we used unit testing with the Junit framework.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача практики — командно разработать игру с графическим интерфейсом на основе красно-чёрного дерева. Красно-чёрное дерево является одним из самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла. Реальное применение данного дерева можно найти в ядре операционной системы Linux, где данную структуру данных использует планировщик.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1 Требования к игровому процессу

Суть игры: игрок проходит по комнатам пока не достигнет финальной. Комнаты организованны в виде красно-чёрного дерева. Игрок начинает в корне и должен дойти до листа. Прохождение комнаты происходит по следующему порядку:

1. Игрок заходит в новую комнату
2. Происходит встреча с существом: бой, торговля, избежание боя. Также возможны ещё какие-нибудь взаимодействия. После боя игроку может выпасть предметы, которые можно взять с собой
3. Использование активности в комнате, если она доступна игроку
4. После активности на основе действий игрока в комнате создаётся набор из действий вставки и удаления, которые производятся с красно-чёрным деревом, т.е добавляются и/или удаляются новые комнаты, а уже существовавшие могут быть перемещены и изменены.
5. Выбор следующий комнаты: налево и направо. Если выбора нет, т.е у узла дерева только один сын, то переход в следующую комнату

Игрок имеет следующие характеристики:

- Запас жизней. Максимум равен 100, однако это может быть изменено бонусами предметов
- Запас магии. Максимум равен 100, однако это может быть изменено бонусами предметов
- Количество монет. Минимум равен 0, в долг брать нельзя

Игрок имеет следующие атрибуты, которые вначале случайным образом устанавливаются и могут быть изменены предметами:

- Сила — отвечает за бонус к урону оружием: Сила N = + 10N% к урону оружием
- Ловкость — отвечает за шанс уклонения: Ловкость N = 10N% уклонения

- Интеллект — отвечает за бонус к урону магией: Интеллект N = + 10N% к урону магией
- Внимательность — отвечает за возможность воспользоваться активностью в комнате: Внимательность N = если N > значение активности K, то данной активностью можно воспользоваться

Инвентарь игрока представлен на рис. 1.

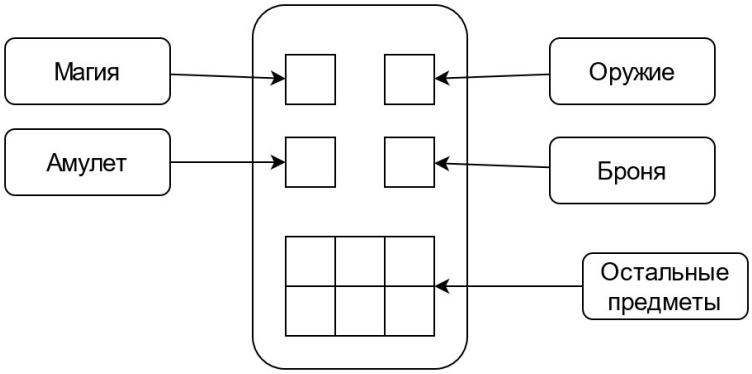

Рисунок 1 — Инвентарь игрока

Игрок может взаимодействовать в любой момент времени, однако в момент боя взаимодействие с инвентарём влечёт получение урона. Верхние четыре слота отвечают за активные предметы, которые дают бонусы игроку и устанавливают базовый урон. Предметы в остальной части инвентаря являются пассивными. Предметы имеют случайно генерируемые бонусы и штрафы. Также предметы могут различаться степенью редкости, чем дальше, тем лучше предметы. Это не обязательно должно специально отображаться, это может быть скрытым параметром.

Бой происходит автоматически. Игрок лишь выбирает способ атаки: оружием или магией. Если за один удар не удалось одолеть врага, то враг атакует игрока, и так это продолжается пока не погибнет враг или игрок. Если у игрока во время боя закончились жизни, но при этом он имеет предмет, восстанавливающий здоровье, то данный предмет будет задействован автоматически без получения урона. Аналогично с запасом магии, если он закончился.

Каждая комната, существо и предмет обладают одной из следующих стихий: Адская, Морозная, Морская, Песчаная, Подземная, Небесная, Обычная(отсутствие). Взаимодействие стихий показано в табл. 1. Строка отвечает за атакующую стихию, столбец — за атакуемую(защищаемую). Легенда для таблицы:

- Зелёный цвет — Бонус к урону
- Красный цвет — Отсутствует бонус к урону
- Фиолетовый цвет — Бонус к урону + урон здоровью атакующего
- Серый цвет — не взаимодействуют

Таблица 1 — Взаимодействие стихий

Атак/Защит	Адская	Морская	Морозная	Песчаная	Подземная	Небесная	Обычная
Адская							
Морская							
Морозная							
Песчаная							
Подземная							
Небесная							
Обычная							

Список текущих активностей в комнате(может быть дополнен/изменён позже):

- Клад — даёт прибавку к монетам
- Осмотр следов — информация(возможно, частичная) о существах в следующих комнатах
- Портал — переход на какую-то другую комнату на той же высоте дерева
- Таверна(Гостиница) — возможность отдохнуть, т.е восстановить частично или полностью запас жизни и магии

Список текущих существ (может быть дополнен/изменён позже):

- Торговец: возможность торговли, можно купить/продать предметы. Если поторговать, то в дерево будут добавлены новые комнаты
- Разбойник: возможность избежав боя, заплатив монетами.

- Дракон: повышенная устойчивость к магическим атакам
- Голем: повышенная устойчивость к физическим атакам

1.1.2 Требования к графическому интерфейсу

Эскиз графического интерфейса представлен на рис. 2.

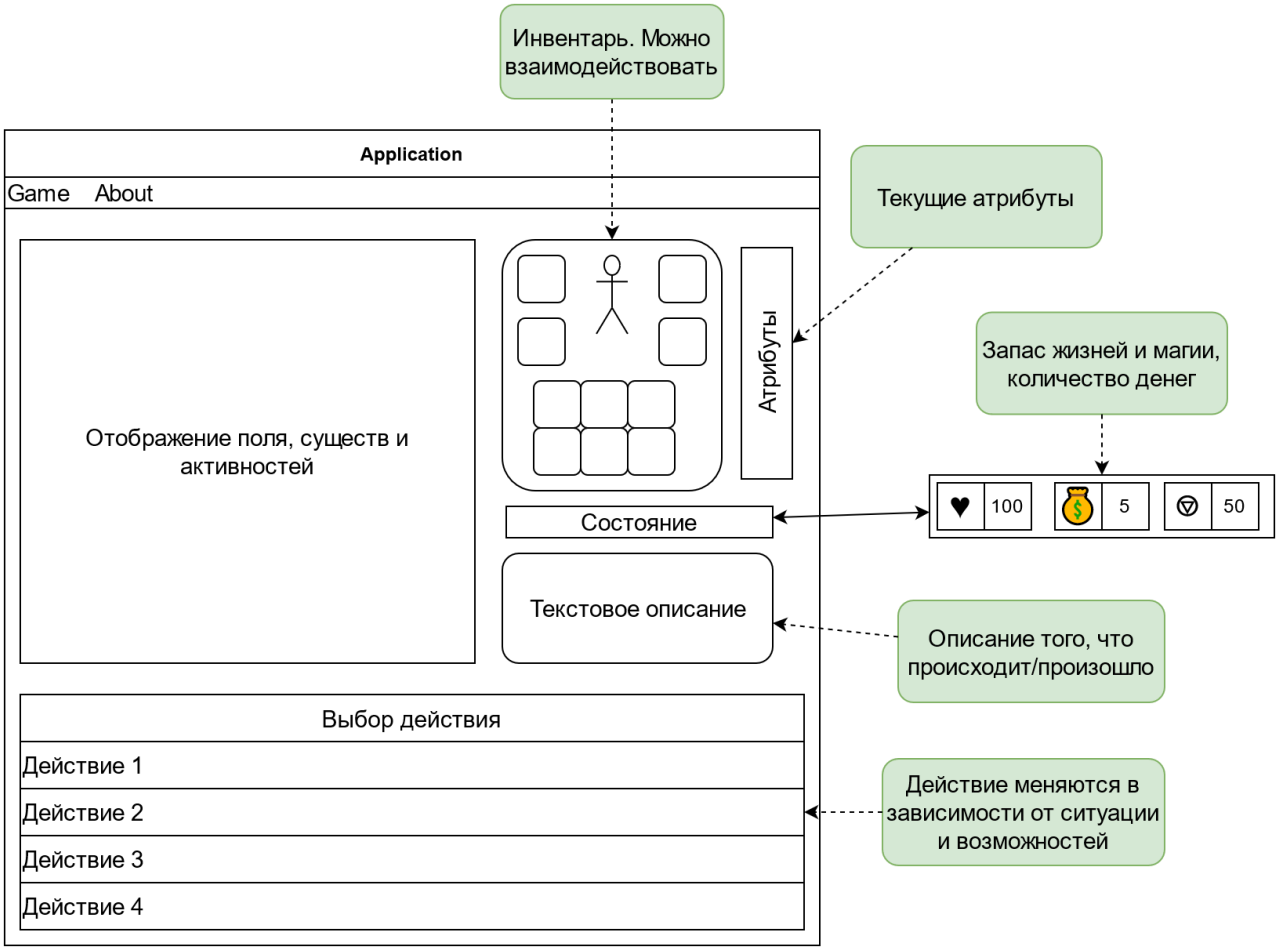

Рисунок 2 — Эскиз графического интерфейса

Слева находится отображение игрового поля. В нём отображается дерево, текущая комната, существа и активности. Камеру по полю можно перемещать. Камеру можно перемещать только если нет встречи с существом или не выбрана активность в комнате. Если будет неудобен совместный режим камеры, то перемещение камеры вынести в отдельное действие.

Снизу находится меню выбора действий в виде списка. Действия меняются в зависимости от ситуации и возможностей игрока. При выборе

некоторых действий возможно появление всплывающего окна для ввода дополнительной информации. Через кнопки действия будет происходить взаимодействие пользователя с игрой.

Также взаимодействие будет происходить через инвентарь игрока. Взаимодействие будет осуществляться при помощи мыши, например, при нажатии правой кнопкой мыши по предмету будет происходить его использование. При наведении на предмет должны показываться его бонусы и штрафы.

В полях состояние и атрибуты будут отображаться текущее состояние игрока. В поле текстового описания будет выводится сообщение о том, что происходит сейчас или что произошло.

1.1.3 Требования к архитектуре приложения

Архитектура приложения представлена на рис. 3.

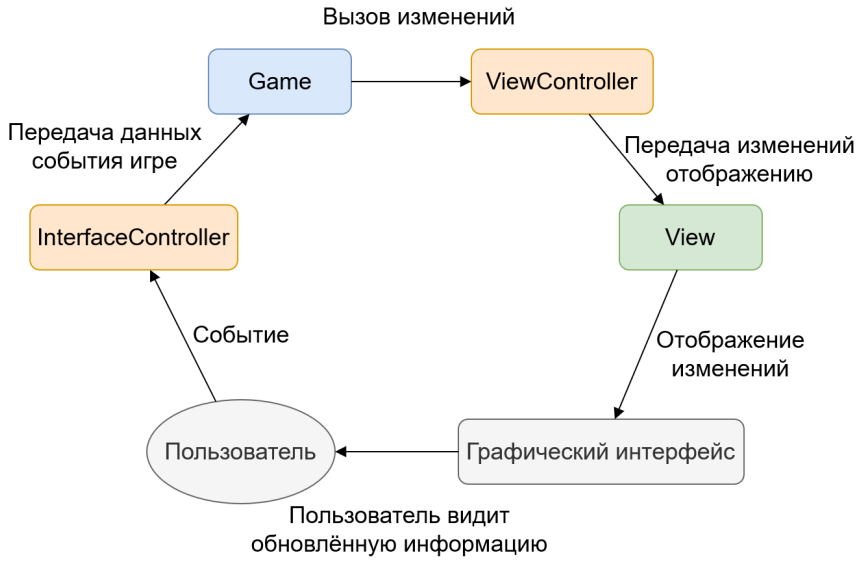

Рисунок 3 — Архитектура приложения

В проекте используется архитектура MVC(Model-View-Controller). Взаимодействие игры с интерфейсом происходит через контроллер. В архитектуре их два: первый — InterfaceController, второй — ViewController. Первый отвечает за передачу данных событий игре, второй отвечает за передачу изменений интерфейсу. Таким образом получается цикл, в котором

пользователь является точкой старта и окончания, а также генератором событий. Благодаря наличию двух контроллеров интерфейс и игра могут быть полностью независимы друг от друга, что упрощает разработку.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

План разработки с датами:

Дата	Описание
5-6 июня	Создание красно-чёрного дерева. Изучение возможностей создания интерфейса и отображения поля.
7 июня	Создание графического интерфейса по спецификации. Отображение дерева(комнат) на поле. Расширение класса красно-чёрного дерева для игры, а также создание API для возможности отображения дерева. **Дедлайн**: Защита вводного задания и сдача первого этапа.
8-9 июня	Создание начальных классов игры с минимальной логикой взаимодействия. Создание контроллеров. Добавление анимации в отображение
10 июня	Частичное создание логики игры и логики контроллеров. Возможность запуска и взаимодействия с игрой, т. е. в приложении уже можно играть, но без полного функционала. **Дедлайн**: Сдача второго этапа.
11-12 июня	Создание тестирования. Завершение создания логики игры и работоспособного(полностью или почти полностью) интерфейса.
13 июня	Исправление ошибок, недочётов. Создание финального отчёта. Проверка, что приложение собирается и полностью работоспособно.
14 июня	Выполнение оставшихся задач. **Дедлайн**: Сдача третьего этапа.

2.2. Распределение ролей в бригаде

Селезнёва Анастасия:

- Фронтенд — Интерфейс
- Gamemaker – Создание взаимодействия интерфейса и игры(InterfaceController и ViewController)
- Документация — Создание документации по своей части

Тиняков Сергей:

- Лидер
- Алгоритмист

- ▽ Документация — Создание финального отчёта
- ▽ Gamemaker – Создание игры и её логики

Цаплин Илья:

- ▽ Фронтенд — Отображение дерева, поля и прочее
- ▽ Тестировщик
- ▽ Документация — Создание документации по своей части

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. UML-Диаграмма

UML-Диаграмма для всех классов представлена на рис. 4.

Рисунок 4 — UML-Диаграмма для всех классов проекта

3.2. Основные классы и методы для игры

Класс	Метод	Описание
Node	Класс реализует узел красно-черного дерева. Хранит в себе цвет, данные, ссылки на потомков и отца, а также информацию о том, является ли данный узел левым сыном.	
	changeSon	Меняет сына на узел переданный первым

		аргументом. Вторым аргументом передаётся переменная Boolean, отвечающая за то, какого сына менять, левого или правого
	getSon	Возвращает левого или правого сына в зависимости от переданного аргумента
	invertLeft	Инвертирует значение поля is_left, т.е делает левый узел правым, а правый левым
	swapKeyAndData	Делает swap для ключа и данным для текущего узла и переданного
RedBlackTree	Класс является реализацией красно-чёрного дерева. Хранит высоту дерева(обычную, не чёрную) и ссылку на корень	
	findNode	Возвращает узел по ключу
	find	Возвращает данные по ключу. Если такого ключа нет, то возвращает null
	insert	Вставляет данные с ключом в дерево
	getData	Возвращает данные в ЛКП порядке
	insertBalance	Выполняет балансировку дерева после вставки
	recolor	Выполняет перекраску дерева
	rotateInsert	Проверяет в балансировке после вставки необходимость выполнить поворот и выполняет необходимый(е) поворот(ы)
	rotate	Совершает поворот в дереве по переданным узлам
	delete	Удаляет ключ из дерева и возвращает значение, которое по нему лежало. Если такого ключа нет, то возвращает null
	deleteNode	Удаляет узел из дерева
	deleteBalance	Выполняет балансировку дерева после удаления
	clear	Очищает дерево
RedBlackTreeGame	Класс реализует дерево для игры. Имеет в себе переопределённые методы для того, чтобы пересоздавать некоторые комнаты, если они не соответствуют своему цвету	
	insertRooms	Вставляет несколько сгенерированных комнат по ключам
	deleteRooms	Удаляет комнаты по ключам
	getRoomWithSons	Возвращает комнату по ключу и потомков этой комнаты
	getLeftRightKeys	Возвращает ключи потомков для переданного ключа
	getKeyDepth	Возвращает глубину ключа
	getRandomKeyOnDepth	Возвращает ключ случайной комнаты на заданной глубине
RedBlackTreeIterator	Итератор для красно-чёрного дерева. При обходе существует	

	возможность получить все данные из узла, не обращаясь напрямую к узлу

3.3. Основные классы и методы для отображения дерева

Класс	Метод	Описание
DrawableNode	Класс реализует визуализацию узла бинарного дерева, хранит в себе ссылки на потомков, ключ узла и группу фигур. Группа содержит окружность, текст, в котором записан ключ, и может содержать две линии, которые будут вести к другим узлам дерева. Также узел содержит параметр draggable, который отвечает за возможность перетаскивания узлов дерева с помощью мыши.	
	moveNode	Метод принимает два числа типа Double, которые задают смещение по оси X и оси Y. Метод перемещает узел и входящие в него концы линий на заданное смещение.
DrawableTree	Класс реализует визуализацию бинарного дерева, хранит в себе корень, который является объектом класса DrawableNode, и высоту дерева.	
	createNode	Рекурсивный метод, который обходит исходное дерево и создаёт узлы для дерева класса DrawableTree
	createDrawnTree	Метод обходит дерево класса DrawableTree и добавляет его фигуры его узлов в группу. Созданная группа возвращается методом.
	getNode	Метод возвращает узел по его ключу
		Метод возвращает корень дерева
	changeTree	Метод получает другое дерево класса DrawableTree, и сравнивает его с текущим. Затем текущее дерево заменяется переданным, и реализуются анимации произошедших с деревом изменений
partialTree	Класс реализует отображение трёх узлов дерева, которые являются текущей позицией игрока на карте. Класс содержит группу фигур, которая представляет собой три узла дерева	
	getTextForKey	Метод создаёт текст, который содержит переданный методу ключ
	getDrawnTree	Метод возвращает группу фигур

3.4. Основные классы и методы для графического интерфейса

Класс	Метод	Описание
InterfaceController	Класс осуществляет связь между классами MyView и Game. Его методы передают классу игры информацию о том, какие действия совершил пользователь, какие предметы он выбрал, продал и т.д.	

MyView	Класс осуществляет связь между пользователем и InterfaceController. Его методы передают информацию InterfaceController о том, какие кнопки были нажаты пользователем и какое действие нужно выполнить. Также в классе реализована связь с окном «карты».	
ViewController	Класс осуществляет отрисовку и обновление окна игры.	
	win	Метод открывает новое окно с информацией об удачном завершении игры.
	youDied	Метод открывает новое окно с информацией о проигрыше.
	updateTree	Метод получает дерево и отрисовывает его на карте.
	updateLocalTree	Метод получает дерево и ключ. По ключу отрисовывается узел дерева с двумя потомками, если они есть.
	update	Метод обновляет окно игры.
	changeButton	Метод принимает кнопку, которую нужно изменить, изображение, которое нужно поместить в кнопку, тултип и текст, который должен в нем быть. Метод вставляет в кнопку нужное изображение и нужный текст в тултип.
	makeDisable	Метод делает все кнопки невидимыми и неактивными.
	changeButtonAbility	Метод изменяет кнопку: делает ее видимой, активной или нет в зависимости от входящего параметра isDisabled, а также помещает в кнопку и в тултип приходящий текст – параметры text и tip соответственно.
	changeLabel	Метод принимает лейбл и строку, которую нужно вставить в этот лейбл.
MyMap	Класс осуществляет отрисовку всего красно-черного дерева.	

3.5. Основные классы и методы для игры

Класс	Описание
Game	Главный класс игры, в котором находится основная логика. Имеет множество методов для взаимодействия с игрой и предоставления текущей информации
Player	Класс реализует игрока. Хранит в себе информацию об инвентаре и активных предметах, текущее здоровье/монеты/магию. В классе присутствуют методы для взаимодействия с инвентарём, также для полей здоровье/монеты/магия прописаны сеттеры
Room	Интерфейс Комнаты, который используется в игре
Item	Класс реализующий предметы в игре и отвечающий за их логики
RoomEvent	Интерфейс События для игры. В событии определяется, видно оно

	игроку, может ли игрок с ним по взаимодействовать.
Creature	Интерфейс Сущетсва для игры. Имеет в себе стихию, здоровье, урон, а также текстовые пояснения. Также присутствует метод для поучения возможных действий.
Generator	Класс реализует случайную генерацию всего в игре. Является синглтон.
ResourceLoader	Класс реализует загрузку, хранение и предоставление ресурсов необходимых для игры. Является синглтон.

4. ТЕСТИРОВАНИЕ

4.1. Описание тестирования

Тестирование осуществляется двумя способами: ручное и автоматическое. Ручное тестирование применяется для графического интерфейса. Автоматическое тестирование — для всего остального. Автоматическое тестирование реализовано через юнит-тесты. Для тестирования алгоритма применяется два способа. Первый заключается в создании тестов с заранее определёнными значениями. Второй использует случайные значения для проверки правильности работы алгоритма.

4.2. Описание тестовых сценариев

Тестовые сценарии для класса RedBlackTree.

Название теста	Описание
testInsert	Тестирует вставку в дерево. В дерево вставляется 1000 случайных значений, те же значения вставляются в map. Затем сравниваются отсортированные значения из map со значениями в дереве, которые были получены с помощью ЛКП обхода
testDelete	Тестирует удаление из дерева. В дерево вставляется 1000 случайных значений, те же значения вставляются в map. Из дерева и map удаляются 500 случайных значений. Затем сравниваются отсортированные значения из map со значениями в дереве, которые были получены с помощью ЛКП обхода
testFind	Тестирует нахождение элемента в дереве. В дерево вставляется 1000 случайных значений, те же значения вставляются в map. Затем проверяется, что каждое значение из map найдётся в дереве, при этом значение не найдётся в пустом дереве.

Тестовые сценарии для класса RedBlackTreeIterator(итератор для красно-чёрного дерева).

Название теста	Описание
testHasNext	Тестирует метод итератора hasNext. Для пустого дерева метод должен возвращать false, для дерева, которое содержит элемент true. Также тестируется, что для итератора, который прошёл последний элемент дерева, метод возвращает false
testNext()	Тестирует то, что метод next() обходит все элементы дерева. В дерево

	вставляется 1000 значений, значения также вставляются в map. Затем с помощью метода next() составляется массив элементов дерева. Созданный массив после сортировки должен совпасть с массивом значений map

Тестовые сценарии для класса Player.

Название теста	Описание
getItemsCountInInventoryTest	Тестирует метод getItemsCountInInventory. Проверяется, что для пустого инвентаря метод вернёт 0, для инвентаря с 3 предметами 3, а для полного инвентаря вернёт размер инвентаря
addItemTest	Тестирует метод addItem для добавления предмета в инвентарь. Проверяется, что в пустой инвентарь будет добавлен предмет, при чём только один. Также проверяется, что предмет будет добавлен в инвентарь с одним пустым местом, и, что предмет не будет добавлен в полный инвентарь
InventoryItemsTest	Тестирует перестановку предметов в инвентаре местами. Проверяется, что два предмета будут переставлены местами, и, что предмет не будет переставлен местами с предметом, которого нет в инвентаре.

4.3. Тестирование графического интерфейса

При запуске игры открывается стартовое окно, представленное на рис. 5.

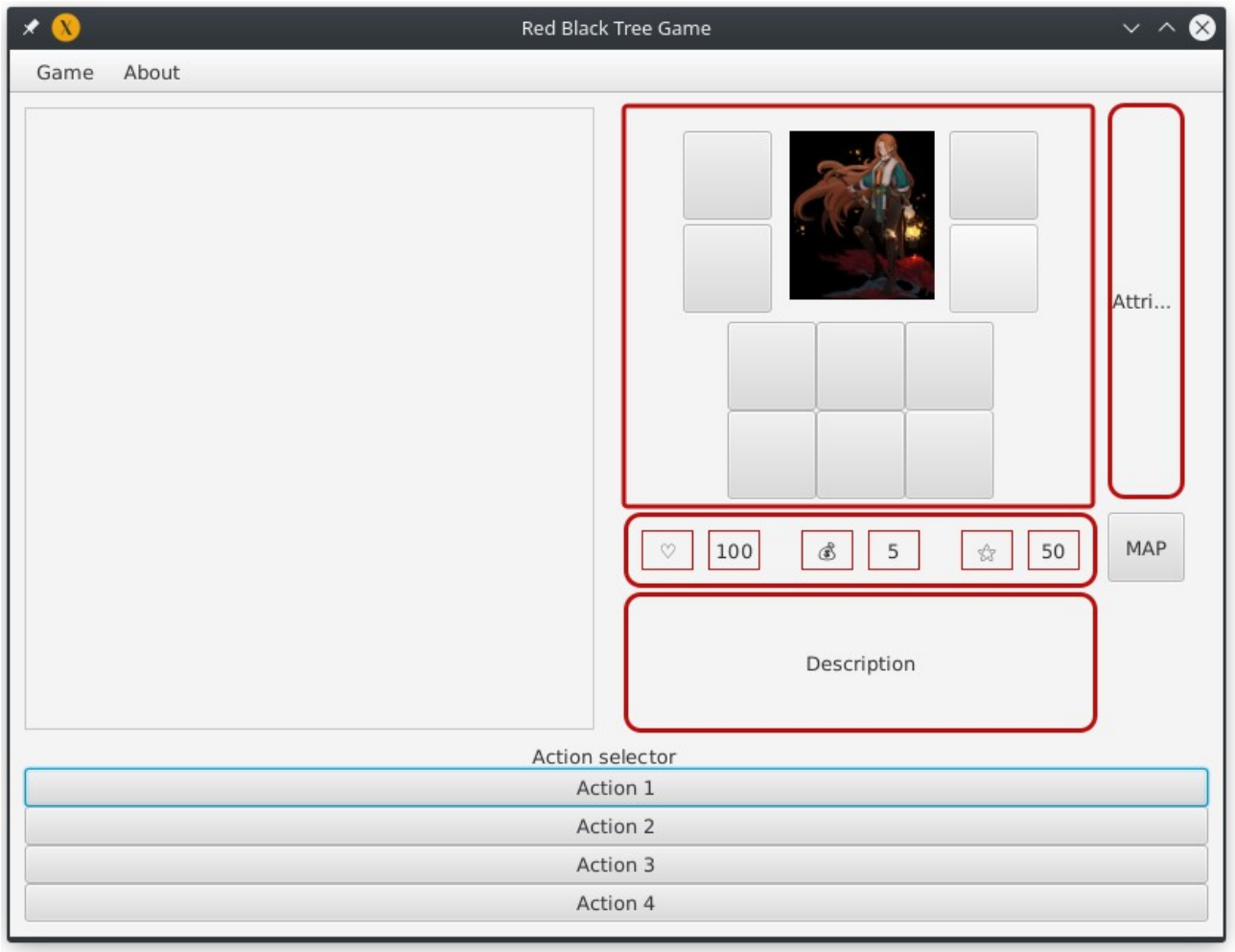

Рисунок 5 — Стартовое окно

При выборе пункта Rules в меню About появится окно с описанием правил игры. Окошко с правилами представлено на рис. 6.

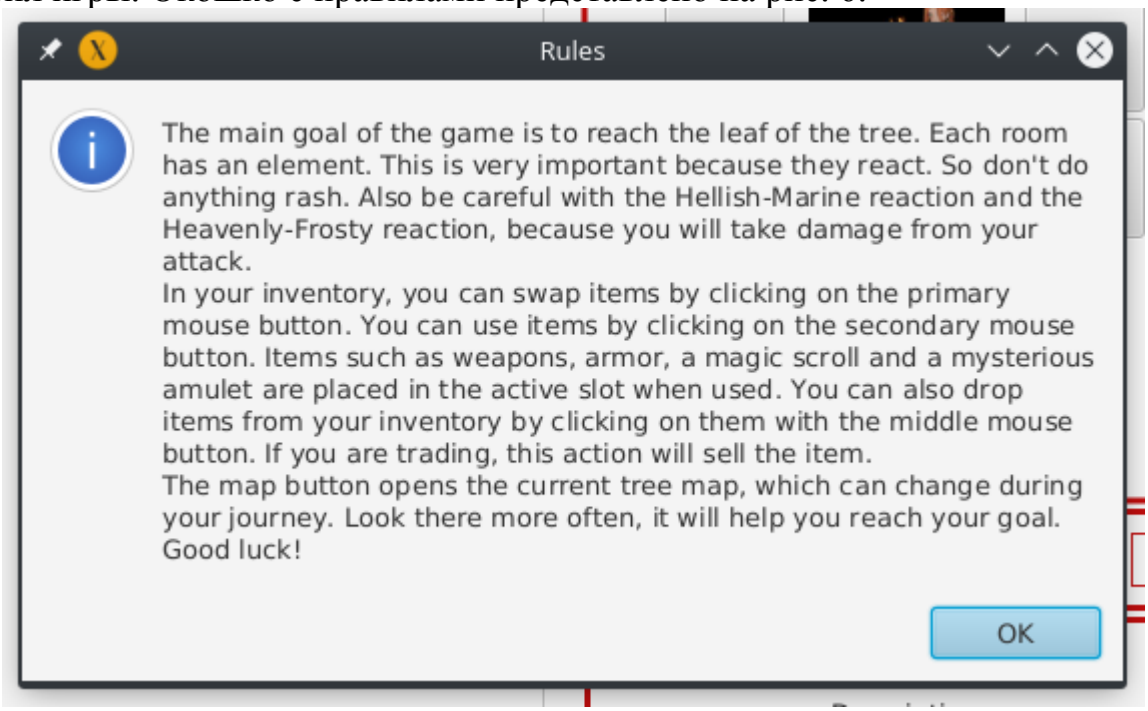

Рисунок 6 — Окошко с описанием правил

При нажатии на кнопку MAP открывается пустое окно. Нажатия на кнопки «действие1-4» не приводят к никаким видимым изменениям. После открытия меню Game и выбора пункта New Game начинается игра. Появляется дерево с текущим положением игрока, у игрока появляются предметы и атрибуты. Начало игры представлено на рис. 7.

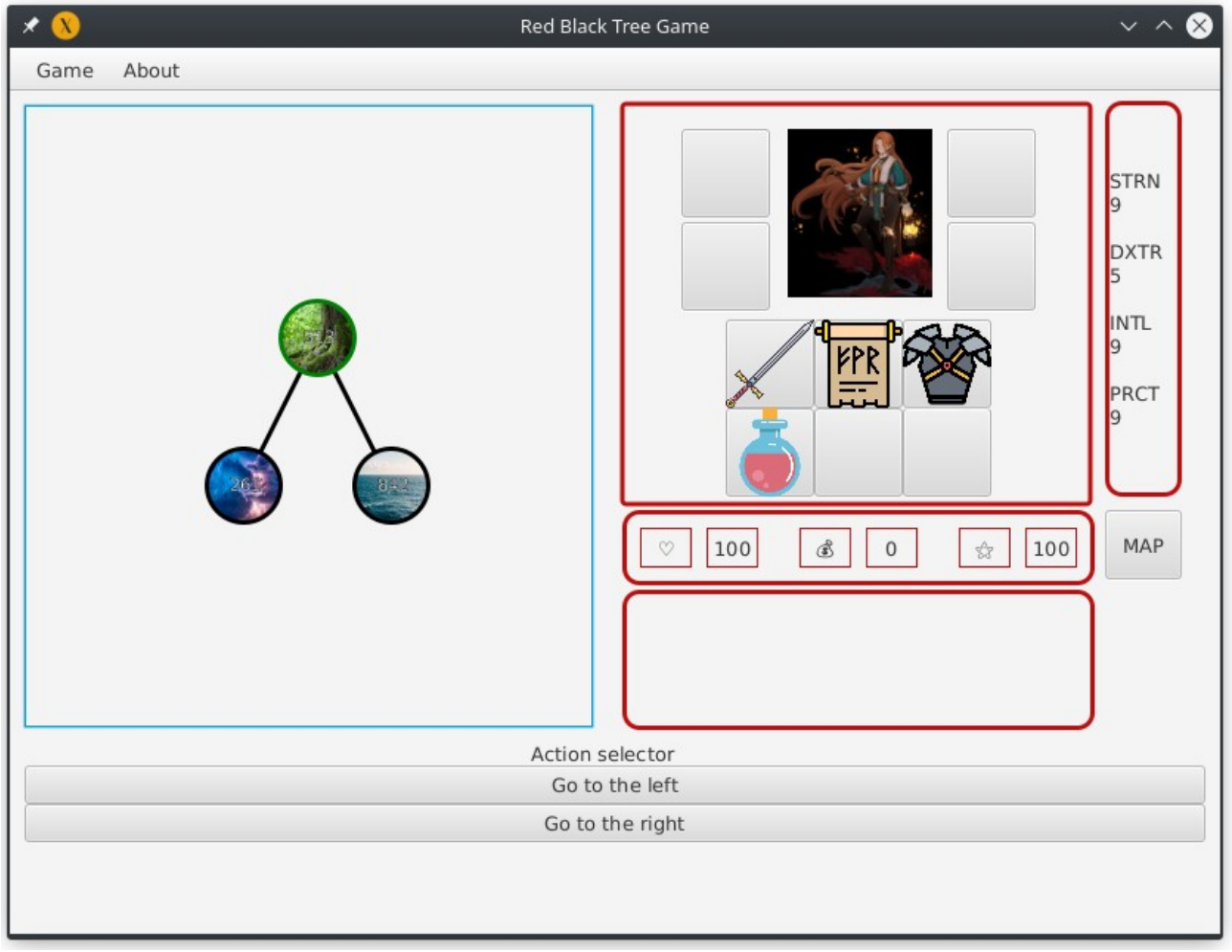

Рисунок 7 — Пример начала игры

При наведении на предметы игроку показвается информация о них. При нажатии правой кнопкой мыши по предмету, предмет используется. Активные предметы (броня, оружие, амулеты, свитки) экипируются в соответствующий слот. Экипированные предметы представлены на рис. 8.

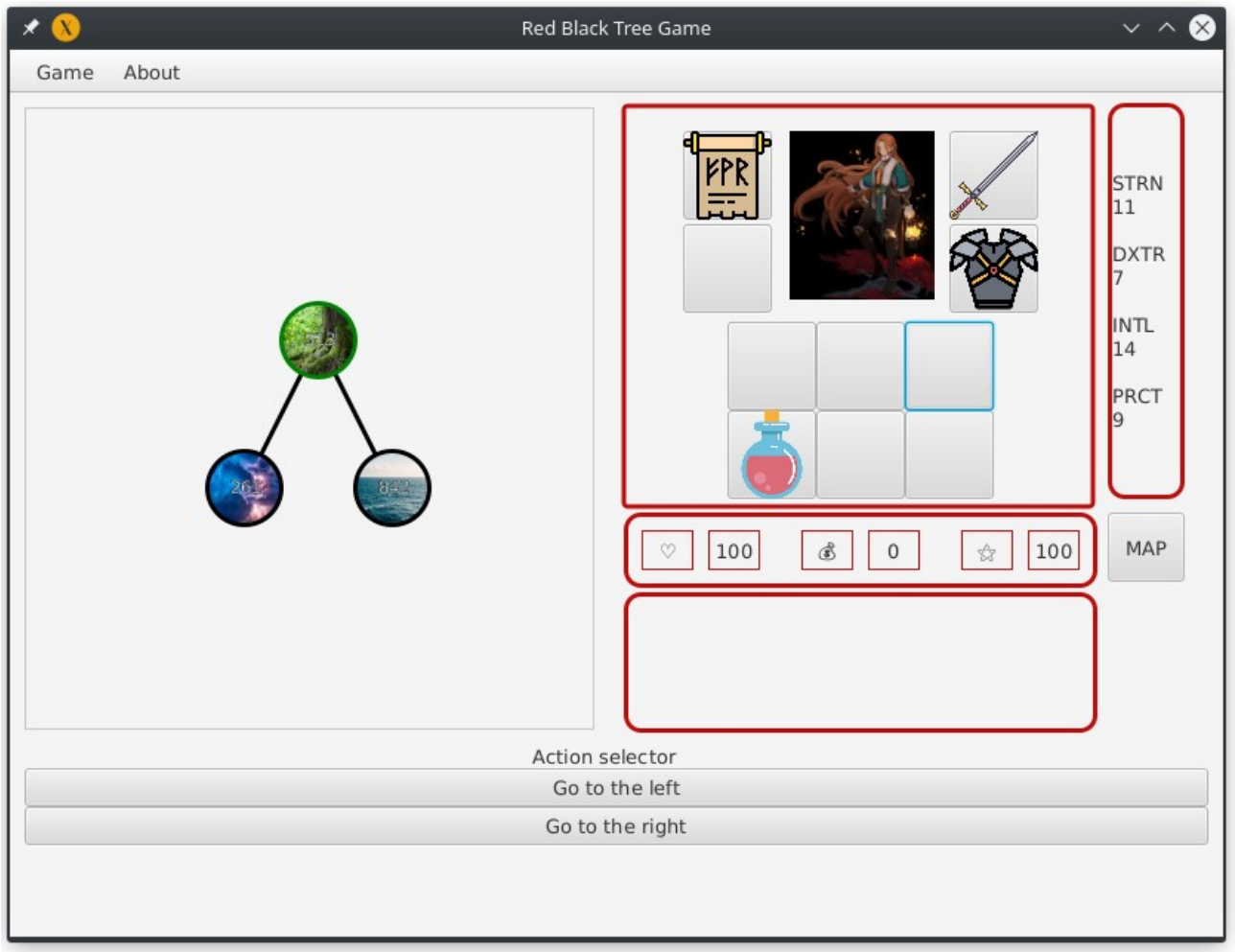

Рисунок 8 — Экипированные по правому нажатию мышки предметы

При нажатии средней кнопкой мыши по предмету, предмет выбрасывается, или продаётся, если сейчас идёт торговля с существом. Выброшенные предметы показаны на рис. 9.

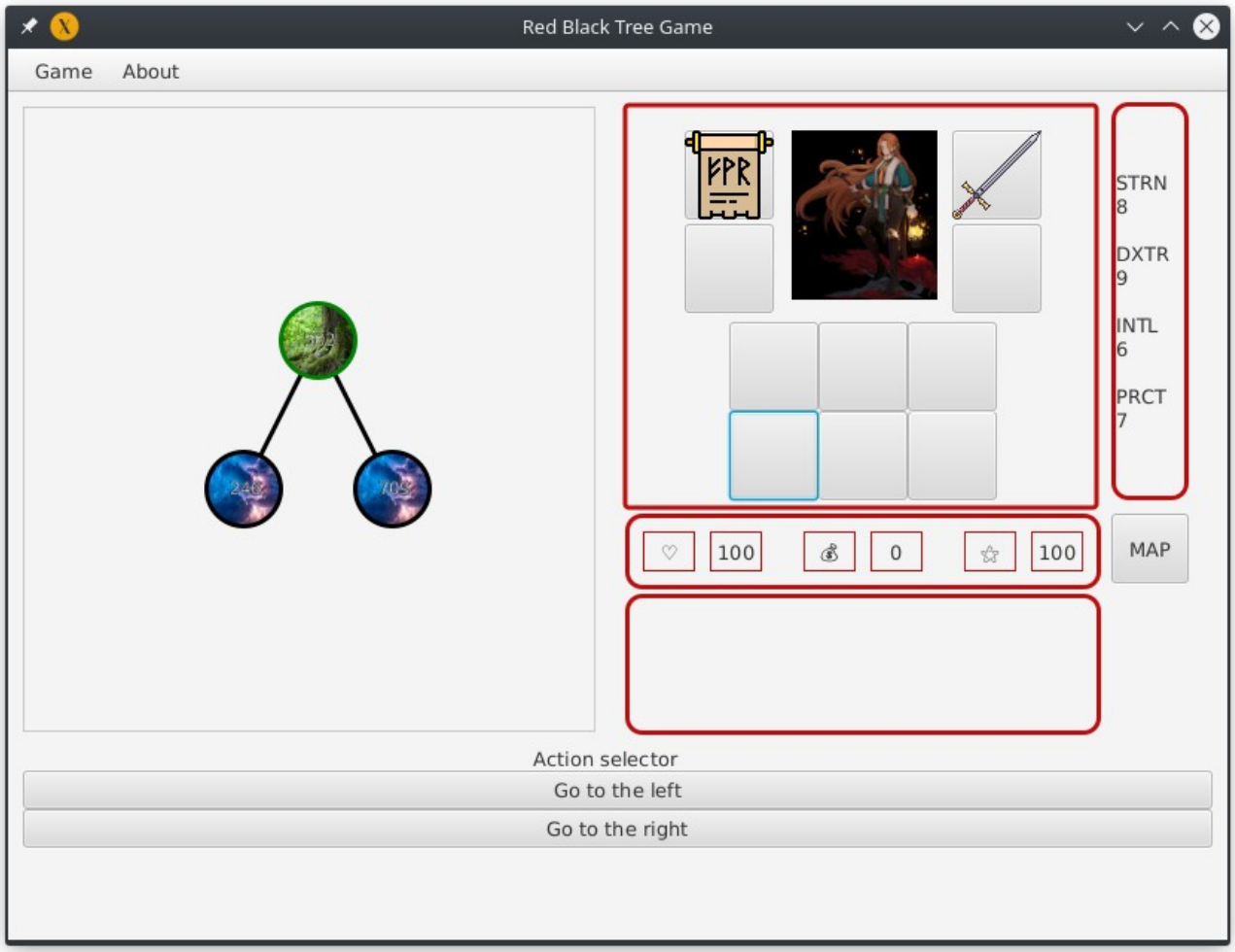

Рисунок 9 — Отсутствие некоторых предметов в инвентаре

При нажатии на кнопку MAP открывается отдельное окно, которое содержит карту всего дерева. Окно с картой представлено на рис. 10.

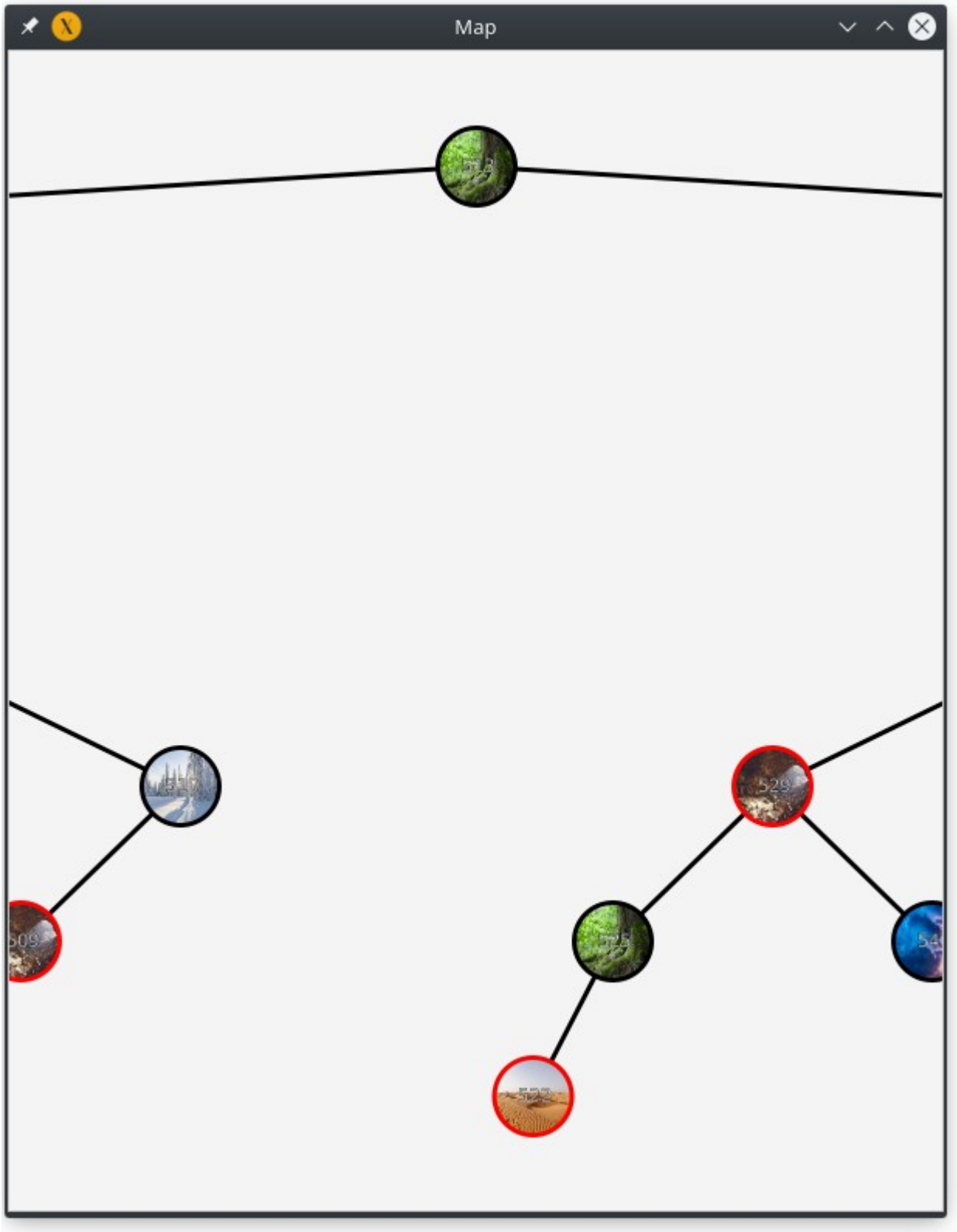

Рисунок 10 — Окно с картой дерева

При нажатии на кнопки Go to the left/right происходит переход в соответствующий узел дерева, дерево с текущим положением игрока обновляется. При переходе в новый узел происходит событие, о котором сообщается в окне описания события. Пример события представлен на рис. 11, в данном случае событие — нападение бандита на игрока.

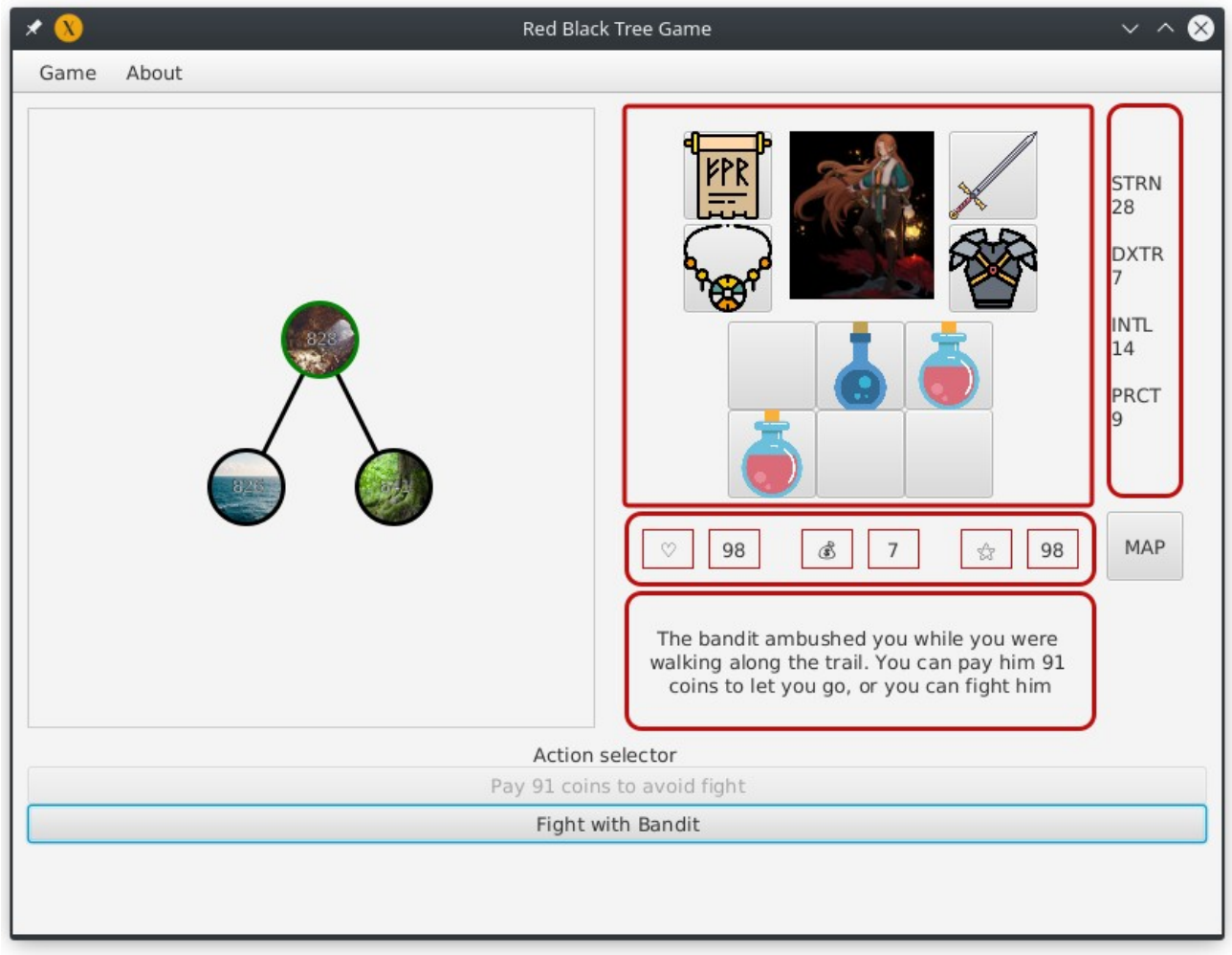

Рисунок 11 — Нападение бандита на игрока

Если в результате событий здоровье игрока опускается ниже нуля, игра заканчивается, открывается окно с сообщением об окончании игры. Окно окончания игры представлено после проигрыша на рис. 12.

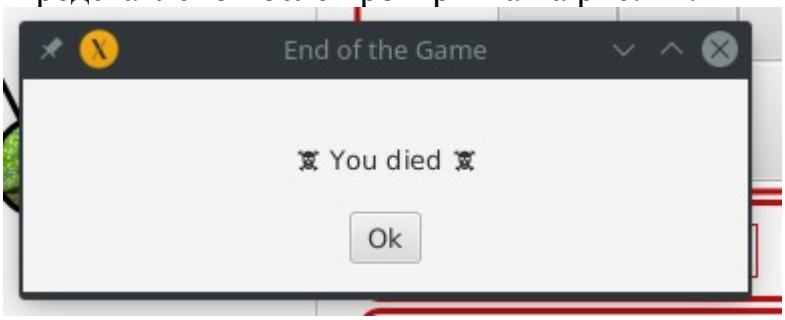

Рисунок 12 — Окно окончания игры

Если игрок доходит до листа дерева, игра заканчивается победой игрока, об этом сообщается в окне с описанием события. Окно окончания игры после выигрыша представлено на рис. 13.

Рисунок 13 — Пример победы в игре

ЗАКЛЮЧЕНИЕ

Были получены навыки работы в команде и разработки проекта. Были изучены новые такие технологии, как язык программирования Kotlin и фреймворк TornadoFX. По результатам учебной практики была создана игра на основе красно-чёрного дерева на языке программирования Kotlin. Также для данной игры был создан графически пользовательский интерфейс при помощи TornadoFX. Полученные результаты полностью соответствуют поставленным целям и задачам.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Kotlin docs. URL: https://kotlinlang.org/docs/home.html (дата обращения: 10.07.2021).

2. Wikipedia. Red-Black Tree. URL: https://en.wikipedia.org/wiki/Red%E2%80%93black_tree (дата обращения: 06.07.2021).

3. Красно-черное дерево: https://neerc.ifmo.ru/wiki/index.php?title=%D0%9A%D1%80%D0%B0%D1%81%D0%BD%D0%BE-%D1%87%D0%B5%D1%80%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE (дата обращения: 06.07.2021).

4. Курс на Stepik: «Введение в Kotlin JVM». URL: https://stepik.org/course/5448/promo#toc(дата обращения: 03.07.2021).

5. TornadoFX Guide. URL: https://docs.tornadofx.io/ (дата обращения: 12.07.2021).

6. Java Docs API Specification. URL: https://docs.oracle.com/en/java/javase/16/docs/api/index.html (дата обращения: 10.07.2021).