**СОДЕРЖАНИЕ**

[Введение 5](#_Toc40208255)

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству………………………………………….7](#_Toc40208256)

[1.1 Анализ существующих аналогов 7](#_Toc40208257)

[1.2 Постановка задачи 10](#_Toc40208258)

[2 Анализ требований к программному средству и разработка функциональных требований 11](#_Toc40208259)

[2.1 Описание функциональности программного средства 11](#_Toc40208260)

[2.2 Спецификации функциональных требований 11](#_Toc40208261)

[3 Проектирование программного средства 13](#_Toc40208262)

[3.1 Алгоритм генерации уровня 13](#_Toc40208263)

[3.2 Алгоритм передвижения демонов 15](#_Toc40208264)

[4 Создание программного средства 19](#_Toc40208265)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 23](#_Toc40208266)

[5.1 Тестирование, проверка работоспособности и анализ полученных результатов 23](#_Toc40208267)

[5.2 Движение демонов 26](#_Toc40208268)

[5.3 Анимация 30](#_Toc40208269)

[5.4 Открытие окна на весь экран 32](#_Toc40208270)

[5.5 Вывод из прохождения тестирования 33](#_Toc40208271)

[6 Руководство по установке и использованию 34](#_Toc40208272)

[6.1 Перед игрой 35](#_Toc40208273)

[6.2 Начало игры 37](#_Toc40208274)

[Заключение 41](#_Toc40208275)

[Список литературы 42](#_Toc40208276)

[Приложение А. Исходный код программы 43](#_Toc40208277)

[Приложение Б 64](#_Toc40208278)

ВВЕДЕНИЕ

Головоломка – непростая задача, для решения которой, как правило, требуется сообразительность, а не специальные знания высокого уровня.

Некоторые головоломки известны с глубокой древности. Оригинальные логические задачи находят на стенах египетских пирамид, в древнегреческих манускриптах и в других исторических памятниках. Эпохой расцвета в средневековой истории головоломок можно считать конец IX века. Рост уровня образования и снижение религиозной нетерпимости к наукам привели к расширению круга любителей логических задач. В это время появилась и первая книга головоломок в Европе – сборник ирландского просветителя Алкуина «Задачи для развития молодого ума».

Наиболее широкое распространение головоломки получили на рубеже XIX и XX веков. Благодаря деятельности американца Сэма Лойда и англичанина Генри Дьюдени головоломки проникли во многие периодические издания, стали популярны среди широких слоев населения. Лойд долгое время считался автором популярнейшей во всем мире головоломки «Пятнашки» (в действительности изобретённой Ноем Палмером Чепмэном из Канастоты). Головоломка была настолько популярной, что некоторые работодатели вынуждены были издать приказ о запрете приносить её на работу.

Предтечей жанра компьютерных игр-головоломок являлись настольные, графические и механические головоломки – от кроссвордов до кубика Рубика. Эти головоломки требовали от игрока логики и ловкости в решении, которые также стали играть важную роль в прототипах жанра, таких, как «Q\*bert» и «Boulder Dash». Эталоном же жанра стала игра Тетрис, появившаяся в 1985 году и сочетавшая в себе простой и захватывающий игровой процесс.

В современном мире встретить игру-головоломку в первозданном ее виде практически невозможно, время от времени появляются проекты, однако многие из них так и остаются невостребованными, а некоторые даже и не выходят в свет. Именно поэтому многие игры создают «оболочку» для головоломок, такие как: открытый мир, где игроку необходимо изучать мир и постепенно решать головоломки, чтобы пройти дальше по сюжету (как это сделала вышедшая в 2016 году игра «The Witness», удачно интегрировав одну головоломку, с множеством вариаций, в мир со своей предысторией) или объединяют множество различных головоломок в одну игру, например, как «The Rusty Lake».

Главными правилами игр-головоломок являются:

1. Простота правил. Даже самые сложные для решения игры в преобладающем большинстве случаев имеют короткий и простой набор правил. Именно краткость правил чаще всего делают игру-головоломку –головоломкой, как, например, игра Сапер от Microsoft – правила столь просты, что даже ребенок может в них разобраться, однако без должного логического мышления и проектирования своих ходов пройти ее невозможно.
2. Строгость и четкость. Каждое действие игрока должно нести за собой один и только один результат, фактор случайности должен быть полностью исключен из игры. Игрок должен понимать, какие последствия стоит ожидать от его действий и они должны быть одинаковы в 100% случаях, ведь как только в игре появляется какая-либо случайная зависимость, игроку не удается продумать шаги наперед, следовательно и не удастся пройти ее.

Данная работа является вариантом шахмат: существует поле 8 на 8 клеток, одна фишка «человек» (той, который в последствии будет управлять игрок) и несколько фишек «демон» (те, что будут «охотиться» на фишку «человек»). Игрок будет иметь возможность поставить фишку «щит» на любое свободное место поля, по которой не может передвигаться «демон», а также передвинуться на одно из 8 свободных соседних полей. Главной задачей игрока является защита фишки «человек» от фишек «демон», путем выставления вокруг него (и/или использования местности) непроходимых фишек.

Главными задачами этой работы является создание оптимального набора правил, по которым должны передвигаться фишки «демон», такие что игрок должен иметь полное представление о следующем шаге и расстановке фишек по полю, но при это представлять сложность в прохождении каждого уровня, а также графическая визуализация уровней и фишек, обработка нажатий игрока.

Целью данного курсового проекта является разработка игрового приложения «Игра Дьявола».

1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И   
 ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ   
 ПРОГРАММНОМУ СРЕДСТВУ

1.1 Анализ существующих аналогов

Найти аналоги к данной работе является сложной задачей, так как правила игры были сформулированы не на основе какой-либо существующей игры. Однако на различных сайтах и сервисе App Store можно найти игры со схожими принципами: существует поле, похожее на шахматное и фишки, которыми можно управлять для достижения цели.

**1.1.1** Шахматные задачи

Шахматные задачи как таковые не являются отдельной игрой, однако они наиболее похожи на данную работу.

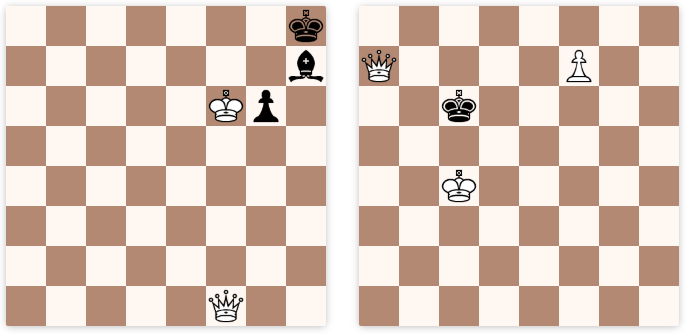


Рисунок 1.1 – Шахматные задачи

Плюсы шахматных игр:

1. сложные и интересные уровни;
2. дружественный интерфейс;
3. большое разнообразие.

Минусы шахматных игр:

1. нельзя отменить выбор фигуры;
2. уровень не перезапускается автоматически при проигрыше.

**1.1.2** Игра Akari

Игра от японского разработчика, в которой надо осветить всю комнату лампочками так, чтобы ни одна лампочка не светила на другую.



Рисунок 1.2 – Akari

Плюсы Akari:

1. cложные и интересные уровни;
2. понятный интерфейс.

Минусы Akari:

1. нельзя перезапустить уровень, пока не сделаешь ошибку;
2. нет правил (их надо знать до того, как запускаешь игру).

**1.1.3** Игра Morphblade

Игра от Suspicious Developments в которой передвигаясь по гексагональному полю нужно отбиваться от жуков.

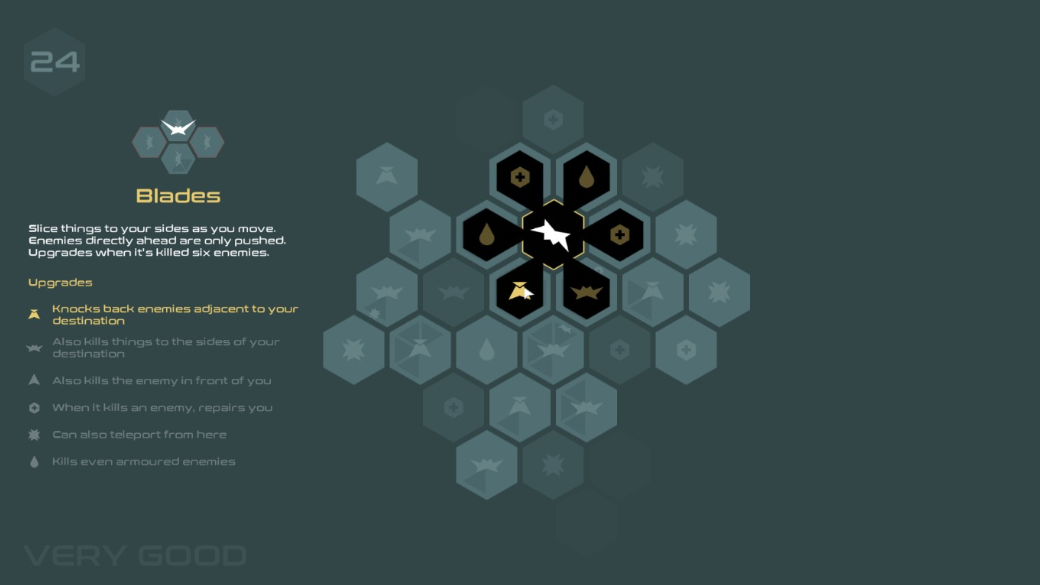


Рисунок 1.3 – Morphblade

Плюсы Morphblade:

1. Простые правила, но сложные задания
2. Интересная механика

Минусы Morphblade:

1. Нельзя перезапустить уровень
2. Не удобный интерфейс/управление

1.2 Постановка задачи

Основной задачей этой работы является написание алгоритма передвижения фишек «демон», такого что при одинаковых входных данных всегда будут одинаковые выходные данные, а также создание уровней, которые будут составлять сложность для прохождения игроком.

Данная программа должна обеспечивать следующие функции:

1. Графическая визуализация поля и меню.
2. Нахождение кратчайшего пути с учетом препятствий (для фишек «демон»).
3. Обработка нажатий мыши пользователем:
   1. передвижение игроком;
   2. установка фишек «щит».
4. Перезапуск и смена уровней:
   1. перезапуск по нажатию на кнопку «Перезапуск»;
   2. перезапуск при проигрыше игрока;
   3. смена уровня при выигрыше игрока.
5. Анимация.

2 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ СРЕДСТВУ И   
 РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Описание функциональности программного средства

Данная программа запускается в оконном режиме фиксированного размера. При первом запуске в левой части сразу отображается первый уровень, в правой – меню с правилами, кнопкой перезапуска уровня и индикаторы уровней.

Поле состоит из ста клеток и является матрицей 10 на 10 клеток. Краевые клетки (те, что находятся на первой и последней строке/столбце) являются непроходимыми (то есть фишками «стена»). Каждый уровень является уникальным и состоит из матрицы 8 на 8 клеток, внутри которой в определенном порядке расставлены фишки «демон» и «стена», а так же одна фишка игрока. При нажатии на фишку «игрок» в соседних 8 клетках отображается стрелки, при нажатии на которой фишка передвигается (на одну клетку). При нажатии на любую свободную клетку (то есть клетку, которая не является «стеной», «демоном», «игроком» или «щитом») ставится клетка «щит». После каждого действия игрока фишки «демон» передвигаются на одну клетку ближе к игроку. Если фишка «демон» становится на одну клетку с игроком, то уровень перезапускается (игра проиграна). Для выигрыша игроку необходимо окружить свою фишку «щитами» и/или «стенами», так что «демоны» не смогут встать с игроком на одну клетку. При победе уровень сменяется на следующий.

При нажатии на кнопку перезапуска уровня текущий уровень перезапускается, то есть переходит в первоначальное положение. При прохождении каждого уровня соответствующий индикатор загорается красным цветом.

2.2 Спецификации функциональных требований

Во время разработки данного программного средства должны быть реализованы следующие функции:

1. Генерация уровня.
2. Передвижение фишки игрока.
3. Установка фишек «щит» игроком.
4. Передвижение фишек «демон».
5. Перезапуск уровня при нажатии на соответствующую кнопку.
6. Индикация прохождения уровней.
7. Распознавание проигрыша игрока и соответствующий перезапуск уровня.
8. Распознавание выигрыша игрока и соответствующая смена уровня.
9. Прорисовка поля с соответствующей анимацией.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Алгоритм генерации уровня

В качестве входных данных алгоритм получает объект класса Human (фишка «игрок»), массив объектов класса Devil (фишки «демон»), массив чисел с номерами полей, на которых установлены непроходимые клетки «стена». За первый проход цикла по всем клеткам поля устанавливаются краевые фишки «стен», все остальные клетки устанавливаются как обычные. Далее устанавливаются «демоны», «игрок» и «стены».

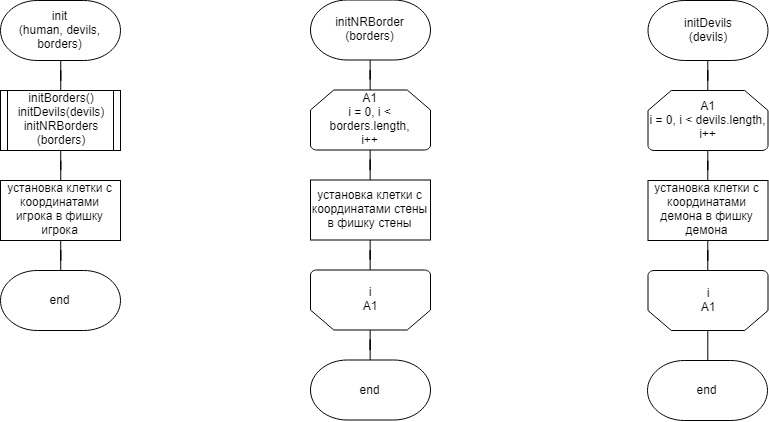


Рисунок 3.1 – Блок-схема генерации уровня

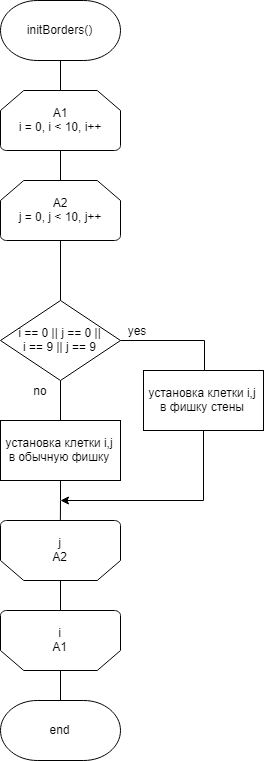


Рисунок 3.2 – Блок-схема генерации уровня

3.2 Алгоритм передвижения демонов

Передвижение фишек «демон» разделено на два этапа: подсчет расстояний (количества ходов, необходимых чтобы добраться до игрока) для каждой клетки и выбор наиболее оптимальной клетки (оптимальная клетка – та клетка, которая будет иметь наименьшее расстояние до игрока, при этом являться логичным ходом, с точки зрения игрока). Первый этап выполняется непосредственно после каждого действия игрока, будь то передвижение или установка фишки «щит», в самом объекте поля, второй этап же выполняется для каждого объекта демона по отдельности. Данный выбор порядка действий обуславливается тем, что расстояние до игрока не зависит от демонов (т.к. они могут становиться на любое свободное место поля, в том числе и на место другого демона), но при этом оптимальная клетка будет зависеть от конкретного расположения демона. Оба алгоритма не имеют входных данных, однако подразумевается, что:

1. выбор наиболее оптимальной клетки идет после подсчета расстояний;
2. подсчет расстояний идет после действий игрока.

**3.2.1** Подсчет расстояний

Алгоритм подсчета расстояний использует на своей основе алгоритм поиска в ширину, который находит расстояния от всех клеток поля, до клетки с игроком, с использованием очередей. Первым действием расстояние в клетке игроком устанавливается на ноль (т.к. это и есть наименьшее расстояние), далее для каждой из 8 соседних клеток устанавливается расстояние на одно больше, от рассматриваемого (т.е. после первого прохода расстояния на клетках, соседних игроку, будут равны 1). После этого алгоритм рассматривает все соседние клетки до тех пор, пока на всем поле не будут подсчитаны расстояния. Блок-схему алгоритма можно посмотреть в приложении Б.

*//обновление дистанций с использованием алгоритма кратчайшего*

*//пути breadth first search alghorithm*

**private** **void** updateDistances() {

*//очищаем все предыдущие дистанции*

    clearDistances();

*//инициализируем очередь*

    Queue<Cell> q = **new** LinkedList<>();

*//наименьшую дистанцию ставим на фишке с человеком*

  gameBoard[human.getRow()][human.getCol()].setDistance(0);

*//добавляем в очередь фишку с игроком*

    q.add(gameBoard[human.getRow()][human.getCol()]);

**while**(!q.isEmpty()) {

*//достаем из очереди клетку*

        Cell curr = q.peek();

        q.remove();

*//все клетки вокруг этой фишки имеют дистанцию на одну больше*

**for**(**int** i = -1; i <= 1; i++) {

**for**(**int** j = -1; j <= 1; j++) {

*//если в эту клетку можно зайти то*

**if**(isValidCell(curr, i, j)) {

*//обновляем дистанцию*

                    gameBoard[curr.getX() + i][curr.getY() + j].setDistance

(curr.getDistance() + 1);

*//добавляем в очередь клетку на рассмотрении*

*//в следующем прохождении цикла*

                    q.add(gameBoard[curr.getX() + i][curr.getY() + j]);

                }

            }

        }

    }

}

*//очистка расстояний*

**private** **void** clearDistances() {

**for**(**int** i = 0; i < BOARD\_SIZE; i++)

**for**(**int** j = 0; j < BOARD\_SIZE; j++)

            gameBoard[i][j].clearDistance();

}

*//проверка на валидность рассмотрения клетки*

*//как места для хождения демона*

**private** **boolean** isValidCell(Cell curr, **int** dx, **int** dy) {

*// проверяем:*

*// 1) клетка в пределах поля*

*// 2) на нее можно ходить (в том числе и на самого игрока, т.к. демон может "ходить"*

*// на него*

*// 3) дистанция не установлена*

**boolean** isValid = (curr.getX() + dx > 0 && curr.getX() + dx < BOARD\_SIZE-1

            && curr.getY() + dy > 0 && curr.getY() + dy < BOARD\_SIZE-1

            && gameBoard[curr.getX() + dx][curr.getY() + dy].isMoveable()

            && gameBoard[curr.getX() + dx][curr.getY() + dy].getDistance() == -1);

**return** isValid;

}

**3.2.2** Выбор оптимальной клетки

После того, как для каждой клетки установлено соответствующее расстояние, для каждого демона проходит выбор самой оптимальной клетки. Из восьми соседних клеток находится клетка с наименьшим расстоянием. Также, после выполнения этого алгоритма, сохраняется значение расстояния от этого демона до игрока, что позволяет определить как поражение, так и победу.

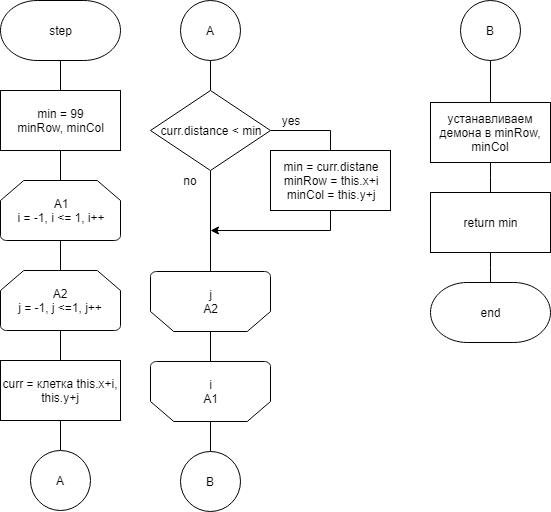


Рисунок 3.3 – Блок-схема оптимальной клетки

4 СОЗДАНИЕ ПРОГРАММНОГО СРЕДСТВА

В ходе разработки программного средства исходный код был поделен на 12 классов:

1. Класс объектов меню (кнопки, лампочки), с обработкой нажатия – GameObject.
2. Основной класс содержащий в себе игровое поле и меню – Menu.
3. Класс обработки нажатий мыши – MouseInput.
4. Класс игрового поля с логикой движения – Board.
5. Класс клетки – Cell.
6. Класс фишки – Player:
   1. наследующий класс фишки «демон» – Devil;
   2. наследующий класс фишки «игрок» – Human.
7. Класс обработки событий – EventListener.
8. Класс графической визуализации – Graphics.
9. Класс обработки изображений и текстур – ImageResource.
10. Основной класс с окном игры – Renderer.

Также было реализовано множество методов для корректной работы программного средства. Список всех функций по классам:

1. GameObject – класс объектов меню (кнопки и лампочки):
   1. public GameObject(int x, int y, int w, int h, ImageResource img) – конструктор класса;
   2. public void setTexture(ImageResource img) – устанавливает текстуру объекту;
   3. public boolean isPressed(int \_x, int \_y) – проверяет находится ли указатель мыши на данном объекте;
   4. public void display() – отображает объекты в окне.
2. Menu – класс с меню и игровым полем:
   1. public Menu(GLAutoDrawable drawable) – конструктор;
   2. public void click(int x, int y) – обрабатывает нажатие, вызывая соответствующие функции;
   3. public void display() – отображает меню и игровое поле;
   4. private Board getLevel(int level) – возвращает объект класса Board в зависимости от номера текущего уровня;
   5. public void restartLevel() – перезапускает текущий уровень;
   6. private void restartGame() – перезапускает игру с первого уровня;
   7. public void nextLevel() – изменяет текущий уровень на следующий.
3. MouseInput – класс обработки нажатий мыши:
   1. public void mouseClicked(MouseEvent event) – обрабатывает нажатие мыши, запуская функцию класса Menu с передачей соответствующих координат.
4. Board – класс игрового поля:
   1. public Board(Devil[] devils, Human human, int[][] borders, float sizeOfWindow) – конструктор;
   2. public Board(int[][] devils, int[] human, int[][] borders, float sizeOfWindow) – конструктор (с другими параметрами);
   3. private void init(Devil[] devils, Human human, int[][] borders, float sizeOfWindow) – инициализирует игровое поле;
   4. private void initBorders() – инициализирует поля;
   5. private void initDevils(Devil[] devils) – инициализирует демонов;
   6. private void initNRBorders(int[][] borders) – инициализирует стены;
   7. public void step(int x, int y) – обрабатывает каждый шаг игры;
   8. private void waitAction(int ms, int action, int i, int j) – создает задержку для плавной анимации;
   9. private void devilsStep() – обрабатывает передвижение демонов;
   10. private void clearBoard() – очищает поле, устанавливая все клетки в обычное состояние;
   11. private void humanStep(int i, int j) – обрабатывает передвижение игрока;
   12. private void directionalStep(int i, int j) – обрабатывает передвижение на стрелки движения;
   13. private void updateDistances() – обновляет значения расстояний до игрока;
   14. private void clearDistances() – очищает значения расстояний;
   15. private boolean isValidCell(Cell curr, int dx, int dy) – проверяет можно ли встать на клетку демону;
   16. public void display() – отображает игровое поле.
5. Cell – класс клеток:
   1. public int getType() – возвращает тип клетки;
   2. public void setType(int type) – устанавливает тип клетки;
   3. public void animateType(int type, float time) – устанавливает тип клетки с соответствующей анимацией;
   4. private void updateTexture() – обновляет текстуру клетки;
   5. public int getDistance() – возвращает расстояние до игрока;
   6. public void setDistance(int dist) – устанавливает дистанцию до игрока;
   7. public void clearDistance() – устанавливает дистанцию до игрока в -1;
   8. public int getX() – возвращает номер строки клетки;
   9. public int getY() – возвращает номер столбца клетки;
   10. Cell(int type, int \_x, int \_y) – конструктор;
   11. Cell(int \_x, int \_y) – конструктор;
   12. public boolean isInitialized() – проверяет, была ли клетка инициализирована (установлен ли ее тип);
   13. public boolean isRegular() – проверяет, является ли клетка обычной;
   14. public boolean isBorder() – проверяет, является ли клетка стеной;
   15. public boolean isShield() – проверяет, является ли клетка щитом;
   16. public boolean isDevil() – проверяет, является ли клетка демоном;
   17. public boolean isHuman() – проверяет, является ли клетка человеком;
   18. public boolean isMoveable() – проверяет, можно ли ходить на эту клетку;
   19. public void setDirectionType(int \_i, int \_j, float time) ­– устанавливает клетку в соответствующую стрелку движения с анимацией;
   20. public boolean isDirectional() – проверяет, является ли клетка стрелкой движения;
   21. public void display(float cellSize) – отображает клетку.
6. Player – суперкласс для демонов и игрока:
   1. public int getRow() – возвращает строку фишки;
   2. public void setRow(int row) – устанавливает строку фишки;
   3. public int getCol() – возвращает столбец фишки;
   4. public void setCol(int col) – устанавливает столбец фишки;
   5. public Player(int i, int j) – конструктор.
7. Devil – класс демонов:
   1. public Devil(int i, int j) – конструктор;
   2. public int step(Cell[][] gameBoard, Human human) – выбирает одну из соседних клеток, на которую должен переместиться демон.
8. Human – класс игрока:
   1. public Human(int i, int j) – конструктор;
   2. public boolean isDirected() – проверяет, отображены ли стрелки движения;
   3. public void setDirected(boolean isDirected) – устанавливает значение отображения стрелок движения.
9. EventListener – класс-обработчик основных событий библиотеки OpenGL:
   1. public void display(GLAutoDrawable drawable) – вызывается 60 раз в секунду;
   2. public void init(GLAutoDrawable drawable) – вызывается при инициализации игрового окна;
   3. public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height) – вызывается при инициализации игрового окна и после каждого его масшатабирования;
   4. public static GL2 getGL();
   5. public static Menu getMenu().
10. Graphics – класс, отображающий фигуры в игровом окне:
    1. public static void drawImage(ImageResource image, float x, float y, float width, float height) ­– рисует прямоугольник с текстурой;
    2. public static void drawImageRot(ImageResource image, float x, float y, float width, float height, float rotation, int vx, int vy, int vz) – рисует прямоугольник с текстурой с определенным поворотом относительно осей координат.
11. ImageResource – класс текстур:
    1. public ImageResource(String path) – конструктор;
    2. public Texture getTexture() – преобразует картинку в текстуру.
12. Renderer – класс игрового окна:
    1. public static GLProfile getProfile();
    2. public static void init() – инициализирует игровое окно.

5 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И   
 АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

5.1 Тестирование, проверка работоспособности и анализ полученных результатов

Тестирование игры проходило на персональном компьютере с установленной операционной системой Windows 10.

Таблица 5.1 – Тестирование программы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Тестиру  емая функци  ональ  ность | | Последователь  ность действий | Ожидаемый результат | Полученный результат |
| 1 | | Запуск игры | Двойным щелком на исполняемый файл запускаем игру | Открытие окна, прорисовка поля с первым уровнем, прорисовка меню с кнопкой и лампочками | Тест пройден |
| 2 | | Установка щита | Нажатие на любое свободное место поля | Установка щита в месте нажатия | Тест пройден |
| 3 | | Выбор направления движения | Нажатие на фишку игрока | В соседних 8 клетках появляются стрелки движения | Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тестиру  емая функци  ональ  ность | Последователь  ность действий | Ожидаемый результат | Полученный результат |
| 4 | Движение | Нажатие на одну из стрелок движения | Фишка игрока передвигается на соответствующую клетку | Фишка игрока передвигается во всех 8 направлениях.  Тест пройден |
| 5 | Движение демонов (при установке щита) | Нажатие на свободное место поля |  | Тест пройден |
| 6 | Движение демонов (при движении игрока) | Нажатие на фишку игрока, последующее нажатие на стрелку движения |  | Тест пройден |
| 7 | Движение демонов (с учетом препятствий) | Установка щита, передвижение игрока | Демоны обходят препятствия (щиты и стены), при этом постоянно приближаются к игроку | После серии различных тестов было выявлено, что демоны идут не по самому оптимальному пути.  **Тест не пройден** |
| 8 | Перезапуск уровня (при поражении) | Передвижение игрока к демонам | Демоны выигрывают (становятся на фишку игрока), уровень перезапускается | При поражении, спустя небольшую задержу (с учетом анимации), уровень перезапускается.  Тест пройден |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тестиру  емая функци  ональ  ность | Последователь  ность действий | Ожидаемый результат | Полученный результат |
| 9 | Перезапуск уровня (при нажатии на соответств  ующую кнопку) | Нажатие на кнопку перезапуска | Уровень перезапускается | Уровень перезапускается.  Тест пройден |
| 10 | Смена уровня | Прохождение уровня | Уровень сменяется на следующий | Уровень сменяется на следующий.  Тест пройден |
| 11 | Индикация смены уровня | Прохождение уровня | Загорается соответствующая лампочка в меню | Тест пройден |
| 12 | Вывод сообщения при прохождении игры | Прохождение игры | Выводится сообщение о победе | Тест пройден |
| 13 | Анимация | Установка щита, движение, перезапуск уровня | Проигрывается анимация вращения клетки | После серии тестов была выявлена ошибка: при нажатии на игрока стрелки движения проигрывают анимацию в неправильную сторону.  **Тест не пройден** |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тестиру  емая функци  ональ  ность | Последователь  ность действий | Ожидаемый результат | Полученный результат |
| 14 | Отсутствие возможности действий во время анимации | Запуск анимации путем установки щита, движения, затем последующая повторная попытка установки щита или движения | Во время анимации игра никак не реагирует на нажатия игрока | После серии тестов было подтверждено, что во время анимации невозможно запустить какие-либо другие действия.  Тест пройден |
| 15 | Отсутствие возможности изменить размер окна | Захват угла окна и последующее масштабиров  ание | Окно не реагирует на масштабирование | Окно не реагирует на масштабирование.  Тест пройден |
| 16 | Отсутствие возможности открыть окно на весь экран | Нажатие на кнопку открытия окна на весь экран | Окно не реагирует на масштабирование | Окно открывается на весь экран.  **Тест не пройден** |
| 17 | Завершение работы | Нажатие на кнопку закрытия окна | Игра закрывается | Игра закрывается.  Тест пройден |

Как видно из тестов некоторые функциональности работают не правильно. Следует рассмотреть каждую из них по отдельности.

5.2 Движение демонов

Следует рассмотреть движение демонов в том случае, когда между ними и игроком стоит преграда:



Рисунок 5.1 – Движение демонов 1



Рисунок 5.2 – Движение демонов 2

Как видно из рисунка 5.2, демоны пошли не по ожидаемому пути. Для вывода информации о расстоянии до игрока после установки щита, в функции updateDistances в классе Board, после подсчета всех расстояний, выводятся дистанции в консоль:



Рисунок 5.3 – Движение демонов 3

Как видно из рисунка 5.3, демоны ходят по оптимальному пути (отмеченным красным), однако не по ожидаемому (отмеченным зеленым), который также является оптимальным. Исходя из алгоритма движения демонов (Рисунок 3.3), видно, что приоритет имеют соседние клетки находящиеся в левом верхнем углу, приоритет падает по движению вниз по столбцу, вправо по строкам. Для того чтобы решить эту проблему необходимо добавить дополнительную проверку: при совпадении расстояния и минимального расстояния нужно выбрать ту клетку, что является диаметрально наиболее близкой к игроку (т.е. та, сумма расстояний ширины и высоты от игрока до которой является минимальной). Таким образом, после этой дополнительной проверки алгоритм нахождения оптимальной клетки всегда будет выдавать результат, содержащий только одно значение.

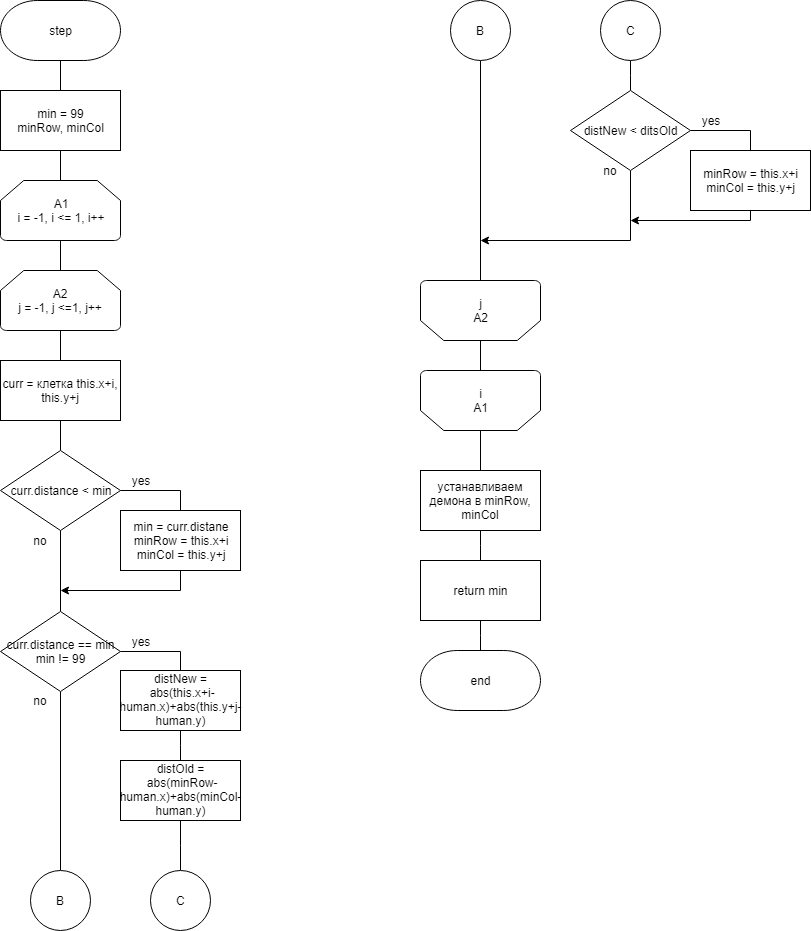


Рисунок 5.4 – Новый алгоритм оптимальной клетки

После данных нововведений демоны передвигаются предсказуемо, но при этом всегда по самому оптимальному пути:



Рисунок 5.5 – Движение демонов 4

5.3 Анимация

Если рассматривать анимацию вращения вокруг оси OY, как проекцию на плоскость OXY, то при вращении на 90 градусов проекция выглядит как сжатие изображения (клетки) по оси OX, а при вращении на 180 градусов – как растяжение:

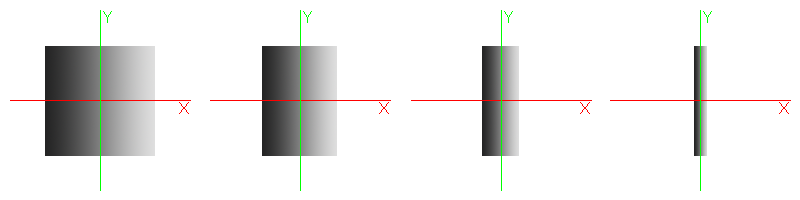


Рисунок 5.6 – Вращение прямоугольника 1

При этом ось OXY отвечает за ширину изображения, а ось OY – за высоту.

Таким образом, легко осуществляется анимация за постепенное сжатие или расширение клетки по ширине, однако стрелки движения поворачиваются не в ту сторону:



Рисунок 5.7 – Анимация 1

Как видно из рисунка 5.7 – стрелки «правая-верхняя», «правая», «левая-нижняя» и «нижняя» сжимаются по высоте, однако все остальные по ширине. Прорисовка в OpenGL работает таким образом, что прорисовка прямоугольника с изображением происходит относительно начала координат и поворот его на определенный угол высчитывается по сложной формуле перехода к другому базису, что не включено в стандартный пакет графической библиотеки. Вместо этого поворот изображения можно осуществить путем поворота плоскости OXY относительно оси OZ, что включено в стандартную библиотеку:

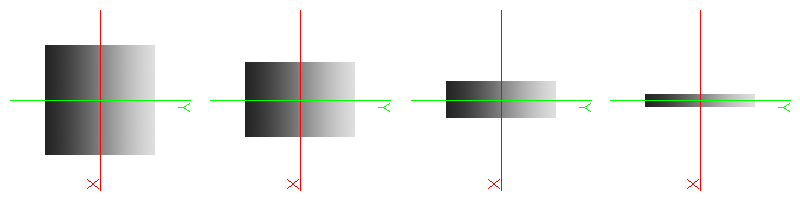


Рисунок 5.8 – Вращение прямоугольника 2

Однако как видно из рисунка 5.8 при повороте изображения на 90 градусов (в любую сторону) ширина и высота меняются местами. Теперь, чтобы сжать изображение по новой ширине, необходимо сжать его по старой высоте, таким образом в новом алгоритме появляется проверка на поворот изображения: при повороте на 90 градусов анимация идет по высоте. Формула все также работает при повороте на -90, 180 градусов, так как в первом случае сжатие происходит к центру, а во втором и третьем ширина и высота не меняются местами. После применения этого нововведения анимация поворота стрелок движения отображается корректно:



Рисунок 5.9 – Анимация 2

5.4 Открытие окна на весь экран

Средствами OpenGl на Java невозможно сделать окно не расширяемым на весь экран. Однако, чтобы даже после открытия окна на весь экран игру все еще можно было играть отрисовка меню была реализована относительно левого верхнего угла, вне зависимости от его размеров.

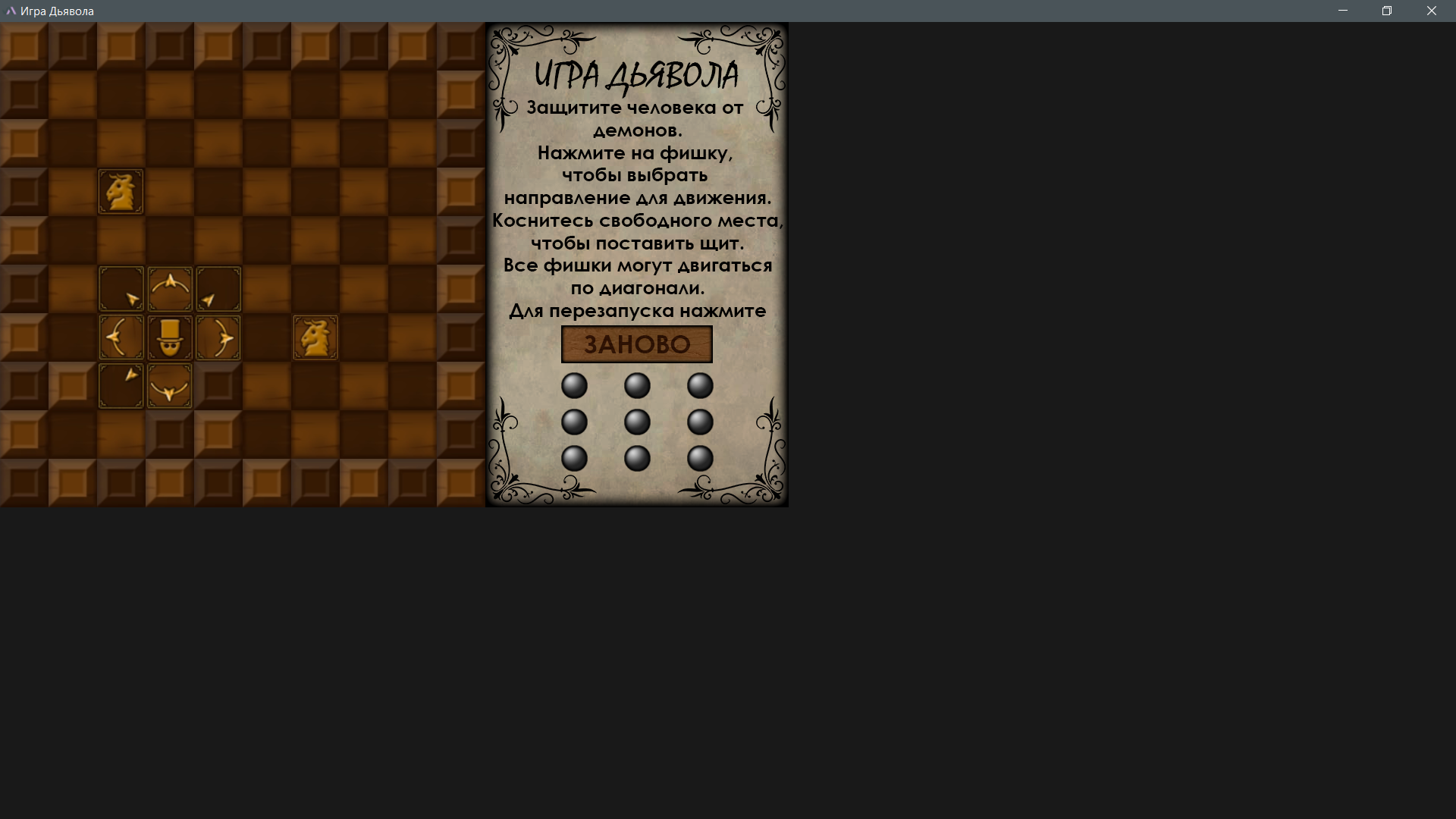


Рисунок 5.10– Полноэкранная игра

5.5 Вывод из прохождения тестирования

После прохождения тестирования в игре были найдены и устранены все критические ошибки. Хоть исправить открытие окна на весь экран не удалось, однако были приняты все меры для устранения некорректной работы программы в полноэкранном режиме. Теперь все функции реализованные в данном программном средстве работают корректно и без ошибок.

6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

Скачать исполняемый файл можно из любого доступного источника. Программа готова к использованию.

Сразу после запуска откроется главное меню программы (Рисунок 6.1).

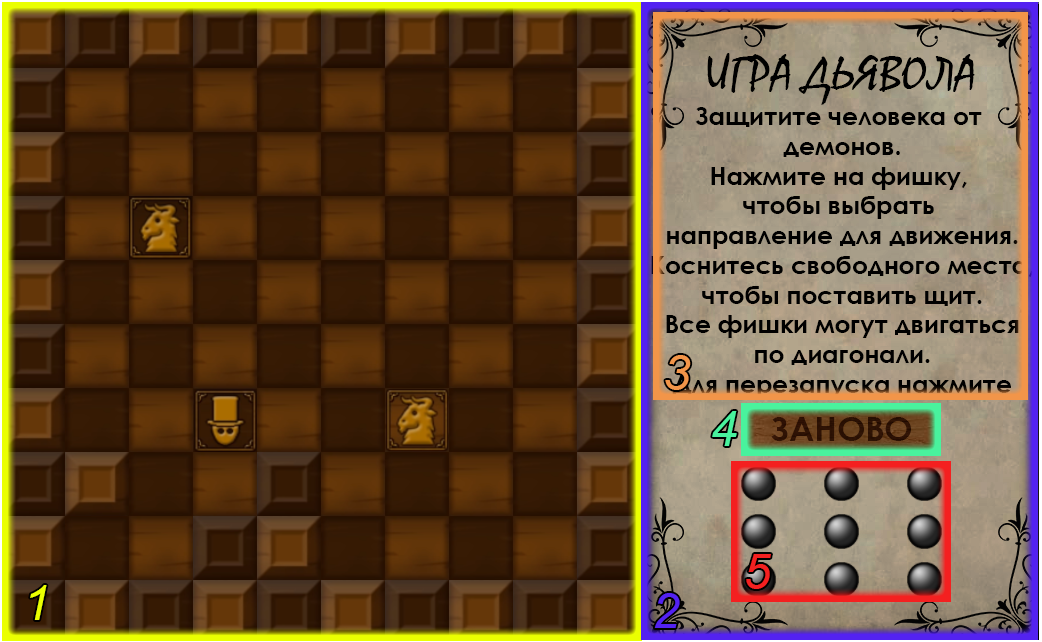


Рисунок 6.1 – Главное меню

1. Игровое поле
2. Меню
3. Правила
4. Кнопка перезапуска уровня
5. Лампочки индикации уровней

В левой части окна можно видеть игровое поле (1), в правой части – меню (2), состоящее из трех частей: правила (3), кнопка перезапуска уровня (4) и лампочки индикации уровней (5).

6.1 Перед игрой

Перед тем как начинать игру следует внимательно прочитать правила (2).

Игровое поле (1) состоит из 100 клеток: 10 по горизонтали и 10 по вертикали. В игре имеются 6 различных фишек:

1. Человек – фишка игрока, которым управляет игрок (Рисунок 6.2).



Рисунок 6.2 – Фишка человека

1. Демон – фишка дьявола, от которых игроку надо спастись (Рисунок 6.3).



Рисунок 6.3 – Фишка дьявола

1. Стена – фишка, на которую нельзя становиться (Рисунок 6.4).



Рисунок 6.4 – Фишка стены

1. Щит – фишка, на которую нельзя становиться и которую можно поставить (Рисунок 6.5).



Рисунок 6.5 – Фишка щита

1. Обычная – фишка на которую можно становиться и ставить щит (Рисунок 6.6).



Рисунок 6.6 – Обычная фишка

1. Стрелка движения – фишка, которая указывает направление движения (клетку, на которую можно встать, рисунок 6.7).



Рисунок 6.7 – Стрелки движения

Цель игры состоит в том, чтобы огородиться от демонов с помощью щитов и/или стен таким образом, чтобы они не могли к вам подойти.

6.2 Начало игры

В качестве руководства будет приведено прохождение первого уровня по пунктам:

1. Для того чтобы сделать первый шаг следует нажать на фишку игрока, таким образом вокруг него отобразятся стрелки движения (Рисунок 6.8).



Рисунок 6.8 – Стрелки движения

1. Далее следует выбрать левую нижнюю стрелку.
2. Фишка игрока переместится на одну клетку. Следовательно за ней и переместятся демоны (Рисунок 6.9).



Рисунок 6.9 – перемещение демонов

1. Далее следует повторить шаги 1 и 2. После этого поле будет выглядеть таким образом (Рисунок 6.10).



Рисунок 6.10 – Состояние после двух шагов

1. Затем, следует нажать на свободную соседнюю клетку справа от игрока, таким образом установив щит (Рисунок 6.11).



Рисунок 6.11 – Щит

1. Следующим шагом следует установить щит над предыдущим. После установки щита у демонов не останется ни одного пути к игрока и первый уровень завершится. Автоматически запустится второй уровень и загорится лампочка первого уровня (Рисунок 6.12).



Рисунок 6.12 – Второй уровень

1. С помощью передвижения и установок щитов можно пройти все оставшиеся уровни. Если игрок оказывается в проигрышной ситуации, можно перезапустить уровень с помощью кнопки перезапуска (Рисунок 6.13).



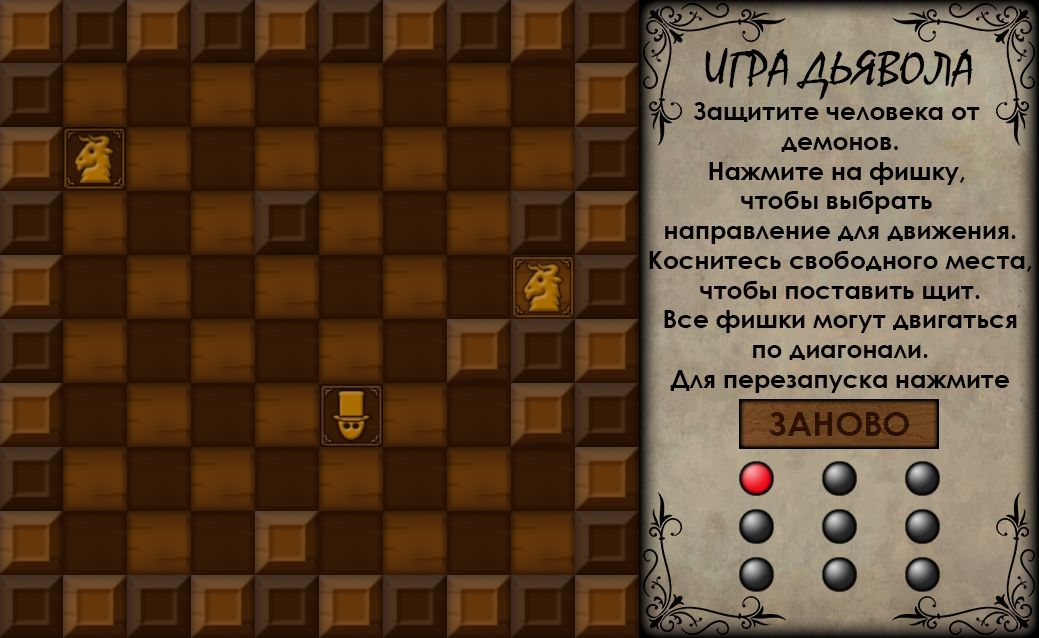


Рисунок 6.13 – Кнопка перезапуска

1. Если демоны выиграют – текущий уровень перезапустится автоматически.

Приятной игры!

ЗАКЛЮЧЕНИЕ

В настоящее время популярность компьютерных игр с каждым днем становится все больше. Немалая часть из них не несет никакой смысловой нагрузки и является, по сути дела, «убийцами времени», однако логический жанр таким не является. Такие игры как «Игра Дьявола» и «The Rasty Lake» c необычной механикой способствует развитию мышления.

В рамках данного курсового проекта было реализовано игровое программное средство «Игра Дьявола», которое обеспечит увлекательное и полезное времяпрепровождение. Согласно постановленным задачам, в данном приложении были реализованы следующие функции:

1. Генерация уровня.
2. Передвижение фишки игрока.
3. Установка фишек «щит» игроком.
4. Передвижение фишек «демон».
5. Перезапуск уровня при нажатии на соответствующую кнопку.
6. Индикация прохождения уровней.
7. Распознавание проигрыша игрока и соответствующий перезапуск уровня.
8. Распознавание выигрыша игрока и соответствующая смена уровня.
9. Прорисовка поля с соответствующей анимацией.

Для успешного выполнения всех поставленных целей потребовалось изучить объектно-ориентированные возможности языка Java, изучить основные начальные принципы данной парадигмы, а также изучить возможности и основные функции и методы аппаратно-ускоряемой графической библиотеки OpenGL.

В дальнейшем возможны улучшения и доработки, такие как пользовательское создание уровней, генерация уровней при помощи алгоритма, а также звуковые эффекты. Помимо этого существует множество возможностей для улучшения производительности программного средства, благодаря аппаратной части этой графической библиотеки.

Использование данного приложения позволит развить внимание, логическое мышление и способность просчитывать действия на несколько шагов вперед, что является незаменимыми качествами как и в профессиональной, так и в повседневной жизни.

СПИСОК ЛИТЕРАТУРЫ

[1] Wikipedia [Электронный ресурс]. — Свободная энциклопедия. — Режим доступа: https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%D0%BB%D0%BE%D0%BC%D0%BA%D0%B0

[2] Wikipedia [Электронный ресурс]. — Свободная энциклопедия. — Режим доступа: https://ru.wikipedia.org/wiki/%D0%93%D0%BE%D0%BB%D0%BE%D0%B2%D0%BE%DBC%D0%BA%D0%B0\_(%D0%B6%D0%B0%D0%BD%D1%80\_%D0%BA%D0%BE%D0%BC%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D1%85\_%D0%B8%D0%B3%D1%80)

[3] Habr [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://habr.com/ru/post/160547/>

[4] Habr [Электронный ресурс]. — Электронные данные. — Режим доступа: <https://habr.com/ru/post/456932/>

[5] Репозиторий БГУИР [Учебное пособие]. — Электронные данные. — Режим доступа: <http://libeldoc.bsuir.by/bitstream/123456789/33241/1/RAk.pdf>

[6] Глухова, Л. А. Основы алгоритмизации и программирования: Лаб. Практикум для студ. Спец. I-40 01 01 «Программное обеспечение информационных технологий» дневной формы обуч. В 4 ч. / Л. А. Глухова, Е. Е. Фадеева, Е. П. Фадеева. – Минск: БГУИР, 2007. – Ч. З. – 51 с.

ПРИЛОЖЕНИЕ А

**Исходный код программы**

**Класс GameObject.java**

**package** kz.engine;

**import** kz.graphics.Graphics;

**import** kz.graphics.ImageResource;

*//игровой объект с текстурой и обработкой нажатия*

**public** **class** GameObject {

*//переменные для хранения координат левого верхнего угла объекта*

**private** **int** x;

**private** **int** y;

*//переменные для хранения ширины и высоты объекта*

**private** **int** w;

**private** **int** h;

*//текстура объекта*

**private** ImageResource img;

*//инициализация объекта*

**public** GameObject(**int** x, **int** y, **int** w, **int** h, ImageResource img) {

**this**.x = x;

**this**.y = y;

**this**.w = w;

**this**.h = h;

**this**.img = img;

    }

*//сеттер на текстуру*

**public** **void** setTexture(ImageResource img) {

**this**.img = img;

    }

*//возвращает true если указатель мыши нажал на этот объект*

**public** **boolean** isPressed(**int** \_x, **int** \_y) {

*//если координата указателя входит в прямоугольник*

*//где левый верхний угол - координата объекта*

*//правый нижний - координата + высота/ширина*

*//вернуть true*

**boolean** inRange = (\_x >= x && \_x <= x+w && \_y >= y && \_y <= y+h);

**return** inRange;

    }

*//отрисовывает объект*

**public** **void** display() {

*//отрисовать текстуру img в координатах x, y с размерами w h*

        Graphics.drawImage(img, x, y, w, h);

    }

}

**Класс Menu.java**

**package** kz.engine;

**import** javax.swing.JDialog;

**import** javax.swing.JOptionPane;

**import** com.jogamp.opengl.GLAutoDrawable;

**import** kz.gamelogic.Board;

**import** kz.graphics.Graphics;

**import** kz.graphics.ImageResource;

**import** kz.graphics.Renderer;

*//главное меню: поле и правила*

**public** **class** Menu {

*//размер окна*

**private** **float** sizeOfWindow;

*//данная переменная хранит текущий уровень*

**private** Board board;

*//текущий номер уровная*

**private** **int** currentLevel = 0;

*//кнопка "начать заново"*

**private** GameObject reset;

*//лампочки индикации номера уровня*

**private** GameObject[] lights;

*//максимальное количество уровней*

**private** **static** **final** **int** MAX\_LEVELS = 9;

*//все используемые текстуры (подгружаются один раз)*

**public** **static** **final** ImageResource REG0 = **new** ImageResource("/res/regular0.png");

**public** **static** **final** ImageResource REG1 = **new** ImageResource("/res/regular1.png");

**public** **static** **final** ImageResource BOR0 = **new** ImageResource("/res/border0.png");

**public** **static** **final** ImageResource BOR1 = **new** ImageResource("/res/border1.png");

**public** **static** **final** ImageResource SHI0 = **new** ImageResource("/res/shield0.png");

**public** **static** **final** ImageResource SHI1 = **new** ImageResource("/res/shield1.png");

**public** **static** **final** ImageResource DEV0 = **new** ImageResource("/res/devil0.png");

**public** **static** **final** ImageResource DEV1 = **new** ImageResource("/res/devil1.png");

**public** **static** **final** ImageResource HUM0 = **new** ImageResource("/res/human0.png");

**public** **static** **final** ImageResource HUM1 = **new** ImageResource("/res/human1.png");

**public** **static** **final** ImageResource COR0 = **new** ImageResource("/res/corner0.png");

**public** **static** **final** ImageResource COR1 = **new** ImageResource("/res/corner1.png");

**public** **static** **final** ImageResource FOR0 = **new** ImageResource("/res/forward0.png");

**public** **static** **final** ImageResource FOR1 = **new** ImageResource("/res/forward1.png");

**private** **static** **final** ImageResource RESB = **new** ImageResource("/res/reset.png");

**private** **static** **final** ImageResource MENU = **new** ImageResource("/res/menu.png");

**private** **static** **final** ImageResource LIG0 = **new** ImageResource("/res/light0.png");

**private** **static** **final** ImageResource LIG1 = **new** ImageResource("/res/light1.png");

**private** **static** **final** ImageResource BACK = **new** ImageResource("/res/backTile.png");

*//константы координат, размеров*

**private** **static** **final** **int** RESET\_X = 740;

**private** **static** **final** **int** RESET\_Y = 400;

**private** **static** **final** **int** RESET\_W = 200;

**private** **static** **final** **int** RESET\_H = 50;

**private** **static** **final** **int** FIRST\_LIGHT\_X = 740;

**private** **static** **final** **int** FIRST\_LIGHT\_Y = 462;

**private** **static** **final** **int** LIGHT\_SIZE = 35;

**private** **static** **final** **int** X\_DIST\_LIGHTS = 83;

**private** **static** **final** **int** Y\_DIST\_LIGHTS = 48;

**private** **static** **final** **int** LIGHTS\_ROW\_AMOUNT = 3;

**private** **static** **final** **int** MENU\_WIDTH = 400;

*//строковые константы*

**private** **static** **final** String CONGRATS\_TEXT\_STR = "Ты прошел все уровни! Игра будет перезапущена.";

**private** **static** **final** String CONGRATS\_STR = "Поздравляем!";

**public** Menu(GLAutoDrawable drawable) {

*//в размер окна сохраняем высоту окна (т.к. поле имеет*

*//размер высота\*высота)*

**this**.sizeOfWindow = drawable.getSurfaceHeight();

*//подгружаем первый уровень*

        board = getLevel(currentLevel);

*//инициализируем кнопку и лампочки*

        reset = **new** GameObject(RESET\_X, RESET\_Y, RESET\_W, RESET\_H, RESB);

        lights = **new** GameObject[MAX\_LEVELS];

**for**(**int** i = 0; i < MAX\_LEVELS; i++) {

*//получение координат лампочек в двухмерном пространстве*

**int** x = FIRST\_LIGHT\_X + i % LIGHTS\_ROW\_AMOUNT \* X\_DIST\_LIGHTS;

**int** y = FIRST\_LIGHT\_Y + i / LIGHTS\_ROW\_AMOUNT \* Y\_DIST\_LIGHTS;

            lights[i] = **new** GameObject(x, y, LIGHT\_SIZE, LIGHT\_SIZE, LIG0);

        }

    }

*//обработка нажатия*

**public** **void** click(**int** x, **int** y) {

*//если нажато в области поля, обрабатываем нажатие в поле*

*//иначе если нажата кнопка перезапуска, перезапускаем уровень*

**if**(x <= sizeOfWindow) {

            board.step(x, y);

        } **else** **if** (reset.isPressed(x, y)){

            restartLevel();

        }

    }

**public** **void** display() {

*//отрисовываем в порядке: что рисуется первее, то находится сзади*

*//сначала задний фон поля*

        Graphics.drawImage(BACK, 0, 0, Renderer.HEIGHT, Renderer.HEIGHT);

*//потом само поле*

        board.display();

*//меню*

        Graphics.drawImage(MENU, Renderer.HEIGHT, 0, MENU\_WIDTH, Renderer.HEIGHT);

*//кнопку перезапуска*

        reset.display();

*//все лампочки*

**for**(**int** i = 0; i < MAX\_LEVELS; i++)

            lights[i].display();

    }

*//возвращает уровень в зависимости от его номера*

**private** Board getLevel(**int** level) {

**switch**(level) {

**default**:

**case** 0:

*//последовательно:*

*// 1) массив хранящий в себе номера ячеек с демонами*

*// 2) номер ячейки с игроком*

*// 3) номера ячеек со стенами*

*// 4) размер поля*

**return** **new** Board(**new** **int**[][] { {3, 2}, {6, 6} },

**new** **int**[] {6, 3},

**new** **int**[][] { {7, 1}, {7, 4}, {8,4 }, {8, 3} },

                    sizeOfWindow);

**case** 1:

**return** **new** Board(**new** **int**[][] {{2,1}, {4,8}},

**new** **int**[] {6,5},

**new** **int**[][] {{3,4},{3,8},{5,7},{5,8},{6,8},{8,4}},

                    sizeOfWindow);

**case** 2:

**return** **new** Board(**new** **int**[][] {{3,4},{8,1}},

**new** **int**[] {6,4},

**new** **int**[][] {{5,3},{5,6},{6,6}},

                    sizeOfWindow);

**case** 3:

**return** **new** Board(**new** **int**[][] {{3,7},{5,2},{7,2}},

**new** **int**[] {5,6},

**new** **int**[][] {{3,6},{4,5},{4,7},{5,8},{6,5},{6,7},{7,6}},

                    sizeOfWindow);

**case** 4:

**return** **new** Board(**new** **int**[][] {{2,7},{8,1}},

**new** **int**[] {5,5},

**new** **int**[][] {{3,5},{5,2},{5,8},{7,5}, {3,2}, {3,1}},

                    sizeOfWindow);

**case** 5:

**return** **new** Board(**new** **int**[][] {{1,1},{8,1},{8,7}},

**new** **int**[] {4,4},

**new** **int**[][] {{3,6},{4,1},{6,2},{7,5},{3,3}},

                    sizeOfWindow);

**case** 6:

**return** **new** Board(**new** **int**[][] {{2,1},{2,8},{7,2}},

**new** **int**[] {5,5},

**new** **int**[][] {{2,2},{2,7},{3,2},{3,7},{5,8},{7,7},{8,7}},

                    sizeOfWindow);

**case** 7:

**return** **new** Board(**new** **int**[][] { {2, 8}, {4, 1} },

**new** **int**[] {4, 5},

**new** **int**[][] { {4, 6}, {6, 6}},

                    sizeOfWindow);

**case** 8:

**return** **new** Board(**new** **int**[][] {{2,8}, {7,8}, {7,1}},

**new** **int**[] {5, 4},

**new** **int**[][] {{4,3},{4,4},{5,1},{6,1},{6,5},{7,2},{8,6}},

                    sizeOfWindow);

        }

    }

*//при перезапуске уровня перезаписываем текущий уровень*

**public** **void** restartLevel() {

        board = getLevel(currentLevel);

    }

**private** **void** restartGame() {

*//выключаем все лампочки*

**for**(GameObject l : lights)

            l.setTexture(LIG0);

*//выводим сообщение с победой*

        JOptionPane optionPane = **new** JOptionPane();

        optionPane.setMessage(CONGRATS\_TEXT\_STR);

        JDialog dialog = optionPane.createDialog(CONGRATS\_STR);

        dialog.setAlwaysOnTop(**true**);

        dialog.setVisible(**true**);

        dialog.setResizable(**false**);

*//устанавливаем текущий уровень в начало*

        currentLevel = 0;

    }

*//следующий уровень*

**public** **void** nextLevel() {

*//зажигаем лампочку с текущим уровнем*

        lights[currentLevel].setTexture(LIG1);

*//если текущий уровень последний начинаем заново*

**if**(++currentLevel == MAX\_LEVELS) {

            restartGame();

        }

*//перезапускаем новый уровень*

        restartLevel();

    }

}

**Класс MouseInput.java**

**package** kz.engine;

**import** com.jogamp.newt.event.MouseEvent;

**import** com.jogamp.newt.event.MouseListener;

**import** kz.graphics.EventListener;

*//обработка нажатий мыши*

**public** **class** MouseInput **implements** MouseListener{

*//все координаты нажатия отправляем в меню*

**public** **void** mouseClicked(MouseEvent event) {

        EventListener.getMenu().click(event.getX(), event.getY());

    }

**public** **void** mouseDragged(MouseEvent event) {

    }

**public** **void** mouseEntered(MouseEvent event) {

    }

**public** **void** mouseExited(MouseEvent event) {

    }

**public** **void** mouseMoved(MouseEvent event) {

    }

**public** **void** mousePressed(MouseEvent event) {

    }

**public** **void** mouseReleased(MouseEvent event) {

    }

**public** **void** mouseWheelMoved(MouseEvent event) {

    }

}

**Класс Board.java**

**package** kz.gamelogic;

**import** java.util.LinkedList;

**import** java.util.Queue;

**import** kz.graphics.EventListener;

**public** **class** Board {

*//поле как массив клеток*

**private** Cell[][] gameBoard;

*//размер поля (10\*10 т.к. краевые клетки всегда стены)*

**public** **static** **final** **int** BOARD\_SIZE = 10;

*//массив с демонами*

**private** Devil[] devils;

*//игрок*

**private** Human human;

*//переменная хранящая режим поля*

*//если true, то нажатия мыши не обрабатываются*

*//т.к. играет анимация*

**private** **boolean** isSleeping = **false**;

*//размер клетки*

**private** **float** cellSize;

*//константы на анимации*

**private** **static** **final** **float** ANIM\_TIME = 250;

**private** **static** **final** **int** ACT\_DEVIL\_STEP = 0;

**private** **static** **final** **int** ACT\_DEVIL\_CELL = 1;

**private** **static** **final** **int** ACT\_HUMAN = 2;

**private** **static** **final** **int** ACT\_NEXT = 3;

**private** **static** **final** **int** ACT\_RESET = 4;

**private** **static** **final** **int** ACT\_CHANGE\_STATE = 5;

*//инициализируем поле*

**public** Board(Devil[] devils, Human human, **int**[][] borders, **float** sizeOfWindow) {

        init(devils, human, borders, sizeOfWindow);

    }

*//переводим массивы чисел в соответствующий объекты и инициализируем поле*

**public** Board(**int**[][] devils, **int**[] human, **int**[][] borders, **float** sizeOfWindow) {

        Devil[] \_devils = **new** Devil[devils.length];

**for**(**int** i = 0; i < devils.length; i++)

            \_devils[i] = **new** Devil(devils[i][0], devils[i][1]);

        Human \_human = **new** Human(human[0], human[1]);

        init(\_devils, \_human, borders, sizeOfWindow);

    }

*//инициализация*

**private** **void** init(Devil[] devils, Human human, **int**[][] borders, **float** sizeOfWindow) {

*//размер клетки = высота поля / количество клеток*

**this**.cellSize = sizeOfWindow / BOARD\_SIZE;

*//устанавливаем демонов и игрока*

**this**.devils = devils;

**this**.human = human;

*//инициализируем само поле*

        gameBoard = **new** Cell[BOARD\_SIZE][BOARD\_SIZE];

*//устанавливаем краевые стены*

        initBorders();

*//устанавливаем фишки демонов, игрока и НЕ краевых стен*

*//с анимацией*

        initDevils(devils);

        gameBoard[human.getRow()][human.getCol()].animateType(Cell.OCCUPIED\_HUMAN, ANIM\_TIME);

        initNRBorders(borders);

    }

*//установка краевых стен*

**private** **void** initBorders() {

**for**(**int** i = 0; i < BOARD\_SIZE; i++)

**for**(**int** j = 0; j < BOARD\_SIZE; j++)

*//если клетка - краевая, сразу инициализируем как стену*

*//иначе как обычную клетку*

**if**(i == 0 || i == BOARD\_SIZE-1 || j == 0 || j == BOARD\_SIZE-1)

                    gameBoard[i][j] = **new** Cell(Cell.BORDER, i, j);

**else** gameBoard[i][j] = **new** Cell(Cell.REGULAR, i, j);

    }

*//установка демонов*

**private** **void** initDevils(Devil[] devils) {

**for**(**int** i = 0; i < devils.length; i++)

            gameBoard[devils[i].getRow()][devils[i].getCol()].animateType(Cell.OCCUPIED\_DEVIL, ANIM\_TIME);

    }

*//установка стен (не краевых)*

**private** **void** initNRBorders(**int**[][] borders) {

**for**(**int** i = 0; i < borders.length; i++)

            gameBoard[borders[i][0]][borders[i][1]].animateType(Cell.BORDER, ANIM\_TIME);

    }

*//обработка нажатия мыши и переход к следующему шагу*

**public** **void** step(**int** x, **int** y) {

*//если анимация в процессе нажатие не обрабатывается*

**if**(isSleeping) **return**;

**for**(**int** i = 0; i < BOARD\_SIZE; i++) {

**for**(**int** j = 0; j < BOARD\_SIZE; j++) {

*//находим именно ту клетку, которая была нажата*

*//с помощью координат прямоугольника*

**if**(x > j\*cellSize && x < cellSize\*(j+1)

                && y > i \* cellSize && y < cellSize\*(i+1)) {

*//если была нажата стена или демон никак не обрабатываем*

**if**(gameBoard[i][j].isBorder() || gameBoard[i][j].isDevil() ||

gameBoard[i][j].isShield())

**return**;

*//если была нажатая свободная фишка*

*//и игрок не ходит (т.е. он не выбирает одну из 8*

*//соседних фишек для передвежения) то*

**if**(gameBoard[i][j].isRegular() && !human.isDirected()) {

*//включаем анимацию*

                        isSleeping = **true**;

*//устанавливаем фишку в щит с анимацией*

                        gameBoard[i][j].animateType(Cell.SHIELD, ANIM\_TIME);

*//с задержкой с учетом анимаций ходим демонами*

                        waitAction((**int**)ANIM\_TIME\*2, ACT\_DEVIL\_STEP, 0, 0);

*//после демонов снова выключаем isSleeping*

                        waitAction((**int**)ANIM\_TIME\*4, ACT\_CHANGE\_STATE, 0, 0);

**return**;

                    }

*//обработка нажатия на игрока*

*//переключает стрелки передвижений*

**if**(gameBoard[i][j].isHuman()) {

                        humanStep(i, j);

**return**;

                    }

*//если нажата стрелка передвижений то*

**if**(gameBoard[i][j].isDirectional()) {

*//включаем анимацию*

                        isSleeping = **true**;

*//переходим на фишку*

                        directionalStep(i, j);

*//ходим демонами*

                        waitAction((**int**)ANIM\_TIME \* 3, ACT\_DEVIL\_STEP, 0, 0);

*//выключаем анимацию*

                        waitAction((**int**)ANIM\_TIME\*5, ACT\_CHANGE\_STATE, 0, 0);

**return**;

                    }

                }

            }

        }

    }

*//обработка задержки и следующего шага*

*//по порядку*

*// 1) задержка в миллисекундах*

*// 2) номер следующего шага*

*// 3 и 4) номер клетки*

**private** **void** waitAction(**int** ms, **int** action, **int** i, **int** j) {

*//задержка основана на создании нового потока...*

**new** Thread( **new** Runnable() {

**public** **void** run()  {

*//...с помощью функции sleep*

**try**  { Thread.sleep( ms ); }

**catch** (InterruptedException ie)  {ie.printStackTrace();}

*//после задержки переходим к следующему шагу*

*//в зависимости от его номера*

**switch**(action) {

**case** ACT\_DEVIL\_STEP:

*//ходят демоны*

                    devilsStep();

**break**;

**case** ACT\_DEVIL\_CELL:

*//установка клетки в демона с анимацией*

                    gameBoard[i][j].animateType(Cell.OCCUPIED\_DEVIL, ANIM\_TIME);

**break**;

**case** ACT\_HUMAN:

*//установка клетки в человека с анимацией*

                    gameBoard[i][j].animateType(Cell.OCCUPIED\_HUMAN, ANIM\_TIME);

**break**;

**case** ACT\_NEXT:

*//переход на следующий уровень*

                    EventListener.getMenu().nextLevel();

**break**;

**case** ACT\_RESET:

*//перезагрузка уровня*

                    EventListener.getMenu().restartLevel();

**break**;

**case** ACT\_CHANGE\_STATE:

*//убираем анимацию*

                    isSleeping = **false**;

**break**;

                }

            }

        } ).start();

    }

**private** **void** devilsStep() {

*//обновляем дистанции до человека*

        updateDistances();

*//количество дистанций до человека*

*//после всех демонов если меньше нуля*

*//следовательно игрок выиграл*

**int** sumOfDist = 0;

**for**(Devil d : devils) {

*//устанавливаем место где стоял демон в обычную фишку с анимацией*

            gameBoard[d.getRow()][d.getCol()].animateType(Cell.REGULAR, ANIM\_TIME);

*//переходим в наиболее эффективную клетку*

**int** dist = d.step(gameBoard, human);

*//если дистанция равна нулю то демоны победили*

**if**(dist == 0) {

*//завершаем анимацию*

                gameBoard[d.getRow()][d.getCol()].animateType(Cell.OCCUPIED\_DEVIL, ANIM\_TIME);

*//очищаем поле*

                clearBoard();

*//перезапускаем уровень*

                waitAction((**int**)(ANIM\_TIME\*1.5), ACT\_RESET, 0, 0);

**break**;

            }

*//иначе добавляем в переменную*

            sumOfDist += dist;

*//завершаем анимацию*

            waitAction((**int**) (ANIM\_TIME\*1.5), ACT\_DEVIL\_CELL, d.getRow(), d.getCol());

        }

*//проверка на победу игрока*

**if**(sumOfDist < 0) {

*//если игрок победил ставим следующий уровень*

            clearBoard();

            waitAction((**int**)ANIM\_TIME, ACT\_NEXT, 0, 0);

        }

    }

*//очистка поля (анимация)*

**private** **void** clearBoard() {

*//устанавливаем все НЕ обычные ячейки поля в обычные*

**for**(**int** i = 1; i < BOARD\_SIZE-1; i++) {

**for**(**int** j = 1; j < BOARD\_SIZE-1; j++) {

**if**(!gameBoard[i][j].isRegular()) {

                    gameBoard[i][j].animateType(Cell.REGULAR, ANIM\_TIME);

                }

            }

        }

    }

*//шаг для игрока*

**private** **void** humanStep(**int** i, **int** j) {

**boolean** hasChanged = **false**;

*//устанавливаем все соседние ячейки в:*

**for**(**int** k = -1; k <= 1; k++) {

**for**(**int** l = -1; l <= 1; l++) {

**if**(k == 0 && l == 0) **continue**;

*//если стрелки не отображены и на это поле можно ходить - в стрелку*

**if**(gameBoard[i+k][j+l].isRegular() && !human.isDirected()) {

                    gameBoard[i+k][j+l].setDirectionType(k,l, ANIM\_TIME);

                    hasChanged = **true**;

                }

*//если стрелки отображены и на это поле можно было ходить - в обычную*

**if**(gameBoard[i+k][j+l].isDirectional() && human.isDirected()) {

                    gameBoard[i+k][j+l].animateType(Cell.REGULAR, ANIM\_TIME);

                    hasChanged = **true**;

                }

            }

        }

*//если хоть что то изменилось*

*//а не изменится может если игрока окружили*

*//меняем логическую переменную с отображением стрелок на обратную*

**if**(hasChanged)

            human.setDirected(!human.isDirected());

    }

*//шаг на стрелку*

**private** **void** directionalStep(**int** i, **int** j) {

*//после нажатия ставим все стрелки в обычные*

**for**(**int** k = -1; k <= 1; k++) {

**for**(**int** l = -1; l <= 1; l++) {

**if**(k == 0 && l == 0) **continue**;

**if**(gameBoard[human.getRow()+k][human.getCol()+l].isDirectional())

                    gameBoard[human.getRow()+k][human.getCol()+l].animateType(Cell.REGULAR, ANIM\_TIME);

            }

        }

*//меняем направление на false*

        human.setDirected(**false**);

*//прошлое положение игрока ставим на обычную*

        gameBoard[human.getRow()][human.getCol()].animateType(Cell.REGULAR, ANIM\_TIME);

*//ходим игроком с анимацией*

        human.setRow(i);

        human.setCol(j);

        waitAction((**int**)(ANIM\_TIME\*1.5), ACT\_HUMAN, i, j);

    }

*//обновление дистанций с использованием алгоритма кратчайшего*

*//пути breadth first search alghorithm*

**private** **void** updateDistances() {

*//очищаем все предыдущие дистанции*

        clearDistances();

*//инициализируем очередь*

        Queue<Cell> q = **new** LinkedList<>();

*//наименьшую дистанцию ставим на фишке с человеком*

        gameBoard[human.getRow()][human.getCol()].setDistance(0);

*//добавляем в очередь фишку с игроком*

        q.add(gameBoard[human.getRow()][human.getCol()]);

**while**(!q.isEmpty()) {

*//достаем из очереди клетку*

            Cell curr = q.peek();

            q.remove();

*//все клетки вокруг этой фишки имеют дистанцию на одну больше*

**for**(**int** i = -1; i <= 1; i++) {

**for**(**int** j = -1; j <= 1; j++) {

*//если в эту клетку можно зайти то*

**if**(isValidCell(curr, i, j)) {

*//обновляем дистанцию*

                        gameBoard[curr.getX() + i][curr.getY() + j]

.setDistance(curr.getDistance() + 1);

*//добавляем в очередь клетку на рассмотрении*

*//в следующем прохождении цикла*

                        q.add(gameBoard[curr.getX() + i][curr.getY() + j]);

                    }

                }

            }

        }

    }

*//очистка расстояний*

**private** **void** clearDistances() {

**for**(**int** i = 0; i < BOARD\_SIZE; i++)

**for**(**int** j = 0; j < BOARD\_SIZE; j++)

                gameBoard[i][j].clearDistance();

    }

*//проверка на валидность рассмотрения клетки*

*//как места для хождения демона*

**private** **boolean** isValidCell(Cell curr, **int** dx, **int** dy) {

*//проверяем:*

*// 1) клетка в пределах поля*

*// 2) на нее можно ходить (в том числе и на самого игрока, т.к. демон может "ходить" на него*

*// 3) дистанция не установлена*

**boolean** isValid = (curr.getX() + dx > 0 && curr.getX() + dx < BOARD\_SIZE-1

                && curr.getY() + dy > 0 && curr.getY() + dy < BOARD\_SIZE-1

                && gameBoard[curr.getX() + dx][curr.getY() + dy].isMoveable()

                && gameBoard[curr.getX() + dx][curr.getY() + dy].getDistance() == -1);

**return** isValid;

    }

*//отображаем все клетки*

**public** **void** display() {

**for**(**int** i = 0; i < BOARD\_SIZE; i++) {

**for**(**int** j = 0; j < BOARD\_SIZE; j++) {

                gameBoard[i][j].display(cellSize);

            }

        }

    }

}

**Класс Cell.java**

**package** kz.gamelogic;

**import** kz.engine.Menu;

**import** kz.graphics.Graphics;

**import** kz.graphics.ImageResource;

**import** kz.graphics.Renderer;

*//игровая клетка*

**public** **class** Cell {

*//переменная содержащая в себе вид клетки*

**private** **int** type;

*//константы видов клеток*

**public** **static** **final** **int** REGULAR = 1;

**public** **static** **final** **int** BORDER = 2;

**public** **static** **final** **int** SHIELD = 3;

**public** **static** **final** **int** OCCUPIED\_DEVIL = 4;

**public** **static** **final** **int** OCCUPIED\_HUMAN = 5;

**public** **static** **final** **int** DIR\_UL = 7;

**public** **static** **final** **int** DIR\_UU = 8;

**public** **static** **final** **int** DIR\_UR = 9;

**public** **static** **final** **int** DIR\_CL = 10;

**public** **static** **final** **int** DIR\_CR = 11;

**public** **static** **final** **int** DIR\_DL = 12;

**public** **static** **final** **int** DIR\_DD = 13;

**public** **static** **final** **int** DIR\_DR = 14;

*//максимальная дистанция до игрока*

**public** **static** **final** **int** MAX\_DIST = 99;

*//номер клетки (не координаты)*

**private** **int** x;

**private** **int** y;

*//дистанция до игрока*

**private** **int** dist;

*//текстура*

**private** ImageResource image;

*//угол поворота текстуры*

**private** **float** rotation = 0;

*//светлая или темная текстура*

**private** **static** **final** **int** DARK\_INDEX = 1;

*//переменные для анимации смены текстуры*

**private** **boolean** isAnimated = **false**;

**private** **float** deltaSize = 0;

**private** **float** dDeltaSize = 0;

**public** **static** **final** **int** S\_TO\_MS = 1000;

*//геттер для типа*

**public** **int** getType() {

**return** type;

    }

*//смена типа*

**public** **void** setType(**int** type) {

**this**.type = type;

*//если тип существует, тогда меняет текстуру*

**if**(type != -1)

            updateTexture();

    }

*//смена типа с анимацией*

**public** **void** animateType(**int** type, **float** time) {

*//меняем переменную*

**this**.type = type;

*//количество кадров для ПОЛОВИНЫ анимации (т.к*

*//анимация состоит из двух этапов)*

**int** animFrames = (**int**)Math.ceil(time \* Renderer.FPS / (2 \* S\_TO\_MS));

*//на сколько пикселей за один кадр нужно сжать (выглядит как повернуть) клетку*

        dDeltaSize = Renderer.HEIGHT / Board.BOARD\_SIZE / animFrames;

*//включаем анимацию*

        isAnimated = **true**;

    }

*//смена текстуры*

**private** **void** updateTexture() {

*//текстур стрелок только две*

*//остальное зависит от поворота*

        rotation = 0;

**switch**(type) {

**case** DIR\_UR:

**case** DIR\_CR:

            rotation = 90;

**break**;

**case** DIR\_DR:

**case** DIR\_DD:

            rotation = 180;

**break**;

**case** DIR\_DL:

**case** DIR\_CL:

            rotation = -90;

**break**;

        }

*//каждый тип имеет свою текстуру*

**switch**(type) {

**case** REGULAR:

*//каждая клетка чередуется ТЕМНАЯ или СВЕТЛАЯ*

**if**((x + y) % 2 == DARK\_INDEX)

                image = Menu.REG0;

**else**

                image = Menu.REG1;

**break**;

**case** BORDER:

**if**((x + y) % 2 == DARK\_INDEX)

                image = Menu.BOR0;

**else**

                image = Menu.BOR1;

**break**;

**case** SHIELD:

**if**((x + y) % 2 == DARK\_INDEX)

                image = Menu.SHI0;

**else**

                image = Menu.SHI1;

**break**;

**case** OCCUPIED\_DEVIL:

**if**((x + y) % 2 == DARK\_INDEX)

                image = Menu.DEV0;

**else**

                image = Menu.DEV1;

**break**;

**case** OCCUPIED\_HUMAN:

**if**((x + y) % 2 == DARK\_INDEX)

                image = Menu.HUM0;

**else**

                image = Menu.HUM1;

**break**;

**case** DIR\_UL:

**case** DIR\_UR:

**case** DIR\_DL:

**case** DIR\_DR:

**if**((x + y) % 2 == DARK\_INDEX)

                image = Menu.COR0;

**else**

                image = Menu.COR1;

**break**;

**case** DIR\_UU:

**case** DIR\_CL:

**case** DIR\_CR:

**case** DIR\_DD:

**if**((x + y) % 2 == DARK\_INDEX)

                image = Menu.FOR0;

**else**

                image = Menu.FOR1;

**break**;

        }

    }

*//геттер для дистанции*

**public** **int** getDistance() {

**return** dist;

    }

*//сеттер для дистанции*

**public** **void** setDistance(**int** dist) {

**this**.dist = dist;

    }

*//очистка дистанции*

**public** **void** clearDistance() {

**int** dist;

*//если на клетку нельзя вставать устанавливаем максимальное значение*

*//иначе очищаем (0 нельзя так как 0 это игрок)*

**if**(isBorder() || isShield())

            dist = MAX\_DIST;

**else**

            dist = -1;

        setDistance(dist);

    }

*//геттер для строки*

**public** **int** getX() {

**return** x;

    }

*//геттер для столбца*

**public** **int** getY() {

**return** y;

    }

*//конструктор*

    Cell(**int** type, **int** \_x, **int** \_y) {

        x = \_x;

        y = \_y;

        setType(type);

        setDistance(-1);

    }

*//если тип не задан то оставляем -1*

    Cell(**int** \_x, **int** \_y) {

        setType(-1);

        setDistance(-1);

        x = \_x;

        y = \_y;

    }

*//проверка на инициализированность клетки*

**public** **boolean** isInitialized() {

**boolean** isInit = (type != 1);

**return** isInit;

    }

*//проверки на тип клеток...*

**public** **boolean** isRegular() {

**boolean** isReg = (type == REGULAR);

**return** isReg;

    }

**public** **boolean** isBorder() {

**boolean** isBor = (type == BORDER);

**return** isBor;

    }

**public** **boolean** isShield() {

**boolean** isShi = (type == SHIELD);

**return** isShi;

    }

**public** **boolean** isDevil() {

**boolean** isDev = (type == OCCUPIED\_DEVIL);

**return** isDev;

    }

**public** **boolean** isHuman() {

**boolean** isHum = (type == OCCUPIED\_HUMAN);

**return** isHum;

    }

*//можно ли передвинуться на это клетку?*

**public** **boolean** isMoveable() {

*//OCCUPIED\_DEVIL, т.к. демоны могут вставать на одну клетку*

**boolean** isMov = (type == REGULAR || type == OCCUPIED\_DEVIL || type == OCCUPIED\_HUMAN);

**return** isMov;

    }

*//устанавливаем соответствующие направления для стрелки \_i \_j*

**public** **void** setDirectionType(**int** \_i, **int** \_j, **float** time) {

**int** dir = DIR\_UL;

**for**(**int** i = -1; i <=1; i++) {

**for**(**int** j = -1; j <=1; j++) {

**if**(i == 0 && j == 0) **continue**;

**if**(i == \_i && j == \_j) {

                    animateType(dir, time);

**break**;

                }

                dir++;

            }

        }

    }

*//проверка на тип (является ли стрелкой)*

**public** **boolean** isDirectional() {

**boolean** isDir = (type >= DIR\_UL && type <= DIR\_DR);

**return** isDir;

    }

*//отображение с анимацией*

**public** **void** display(**float** cellSize) {

*//если анимация включена*

**if**(isAnimated) {

*//ширина отображаем клеткой*

            deltaSize += dDeltaSize;

*//для стрелок повернутые на 90 градусов*

*//ширина становится высотой*

**if**(rotation == 90 || rotation == -90)

                Graphics.drawImageRot(image, y\*cellSize, x\*cellSize + deltaSize / 2,

cellSize, cellSize- deltaSize, rotation, 0, 0, 1);

**else**

                Graphics.drawImageRot(image, y\*cellSize + deltaSize / 2,

x\*cellSize, cellSize - deltaSize, cellSize, rotation, 0, 0, 1);

*//если ширина больше размера клетки*

*//меняем текстуру и разворачиваем клетку в*

*//противоположенное направление*

**if**(deltaSize >= cellSize) {

                dDeltaSize = -dDeltaSize;

**if**(type != -1) {

                    updateTexture();

                }

            }

*//как только ширина становится меньше нуля (т.е. анимация закончилась)*

*//устанавливаем ее в ноль и выключаем анимацию*

**if**(deltaSize <= 0) {

                deltaSize = 0;

                isAnimated = **false**;

            }

*/\**

*\* по итогу это будет выглядеть так*

*\* 1) клетка уменьшается по ширине*

*\* до тех пор пока ширина не станет равна 0*

*\* 2) меняется текстура клетки*

*\* 3) клетка расширяется обратно до первоначальных размеров*

*\**

*\* для пользователя это выглядит так*

*\* будто клетка развернулась по оси Х*

*\* \*/*

        }

**else**

*//без анимации просто рисуем*

            Graphics.drawImageRot(image, y\*cellSize, x\*cellSize, cellSize,

cellSize, rotation, 0, 0, 1);

    }

}

**Класс Devil.java**

**package** kz.gamelogic;

*//логика демона*

**public** **class** Devil **extends** Player{

*//конструктор вызывает конструктор в Player*

**public** Devil(**int** i, **int** j) {

**super**(i, j);

    }

*//самый оптимальный шаг*

**public** **int** step(Cell[][] gameBoard, Human human) {

*//эта функция вызывается после заполнения*

*//дистанциями до игрока*

*//в ней находится номер клетки*

*//с самой оптимальной дистанцией до игрока*

**int** minRow = getRow();

**int** minCol = getCol();

*//минимальное расстоние для инициализации возьмем максимальным*

**int** min = Cell.MAX\_DIST;

**for**(**int** i = -1; i <= 1; i++)

**for**(**int** j = -1; j <= 1; j++) {

*//достаем клетку для рассмотрения (одну из 8 соседних)*

                Cell current = gameBoard[getRow() + i][getCol() + j];

*//если дистанция в это клетке меньше, чем минимальная то*

**if**(current.getDistance() < min) {

*//переохраняем минимальную*

*//и сохраняем номер это клетки*

                    min = current.getDistance();

                    minRow = getRow() + i;

                    minCol = getCol() + j;

*//если дистанция в рассматриваемой клетке равна минимальной*

*//и при этом минимальное расстояние не равно максимальному*

*//(то есть другие клетки уже были рассмотрены)*

*//соответственно есть несколько клеток с одинаковым расстоянием до игрока*

*//тогда рассматриваем клетку которая ДИАГОНАЛЬНО ближе к игроку*

*//для того чтобы точно знать куда пойдет демон*

                } **else** **if**(current.getDistance() == min && min != Cell.MAX\_DIST) {

**int** distNew = Math.abs(getRow()+i - human.getRow()) + Math.abs(getCol()+j –

human.getCol());

**int** distOld = Math.abs(minRow - human.getRow()) + Math.abs(minCol - human.getCol());

*//если диагонально дистанция меньше чем в минимальном*

*//пересохраняем*

**if**(distNew < distOld) {

                        minRow = getRow() + i;

                        minCol = getCol() + j;

                    }

                }

            }

*//после нахождения клетки переставляем демона*

        setRow(minRow);

        setCol(minCol);

*//и возвращаем расстояние до игрока*

*//для определения поражения или победы*

**return** min;

    }

}

**Класс Human.java**

**package** kz.gamelogic;

*//логика игрока*

**public** **class** Human **extends** Player {

*//переменная хранит в себе состояния стрелок вокруг игрока*

*// если стрелки отображена то переменная равна true*

**private** **boolean** isDirected;

*//конструктор вызывает конструктор из Player*

**public** Human(**int** i, **int** j) {

**super**(i, j);

        setDirected(**false**);

    }

*//геттер и сеттер для переменной*

**public** **boolean** isDirected() {

**return** isDirected;

    }

**public** **void** setDirected(**boolean** isDirected) {

**this**.isDirected = isDirected;

    }

}

**Класс Player.java**

**package** kz.gamelogic;

*//суперкласс для игрока и демона*

**public** **class** Player{

*//номер строки и столбца*

**private** **int** row;

**private** **int** col;

*//геттеры и сеттеры*

**public** **int** getRow() {

**return** row;

    }

**public** **void** setRow(**int** row) {

**this**.row = row;

    }

**public** **int** getCol() {

**return** col;

    }

**public** **void** setCol(**int** col) {

**this**.col = col;

    }

*//конструктор*

**public** Player(**int** i, **int** j) {

        setRow(i);

        setCol(j);

    }

}

**Класс EventListener.java**

**package** kz.graphics;

**import** com.jogamp.opengl.GL2;

**import** com.jogamp.opengl.GLAutoDrawable;

**import** com.jogamp.opengl.GLEventListener;

**import** kz.engine.Menu;

*//opengl функции для отображения и инициализации*

**public** **class** EventListener **implements** GLEventListener {

*//хранит переменную gl2 (саму графическую оболочку)*

**private** **static** GL2 gl;

*//главное меню*

**private** **static** Menu menu;

*//вызывается каждый раз при отображении*

*//зависит от фпс*

*//в данном случае 60 раз в секунду*

**public** **void** display(GLAutoDrawable drawable) {

*//очищаются все пиксели, устанавливается дефолтный цвет*

        gl.glClear(GL2.GL\_COLOR\_BUFFER\_BIT);

*//отображается меню*

        menu.display();

*//очищается буфер*

        gl.glFlush();

    }

**public** **void** dispose(GLAutoDrawable drawable) {

    }

*//вызывается вместе с программой*

**public** **void** init(GLAutoDrawable drawable) {

*//при инициализации устанавливаются настройки*

*//для оболочки*

*//такие как цвет очистки пикселей, режим прорисовки и тд*

        gl = drawable.getGL().getGL2();

        gl.glClearColor(0.1f,0.1f,0.1f, 1);

        gl.glEnable(GL2.GL\_TEXTURE\_2D);

        gl.glEnable(GL2.GL\_BLEND);

        gl.glBlendFunc(GL2.GL\_SRC\_ALPHA, GL2.GL\_ONE\_MINUS\_SRC\_ALPHA);

*//а также инициализируется меню*

        menu = **new** Menu(drawable);

    }

*//вызывается при первом отображении окна*

*//а также каждый раз при изменении его размеров*

**public** **void** reshape(GLAutoDrawable drawable, **int** x, **int** y, **int** width, **int** height) {

*//устанавливается режим отображения и координатная сетка*

*//левый верхний угол имеет координаты 0 и 0*

*//правый нижний ШИРИНА и ВЫСОТА*

        gl.glMatrixMode(GL2.GL\_PROJECTION);

        gl.glLoadIdentity();

        gl.glOrtho(0, width, height, 0, -1, 1);

        gl.glMatrixMode(GL2.GL\_MODELVIEW);

    }

*//геттер для gl*

**public** **static** GL2 getGL() {

**return** gl;

    }

*//геттер для меню*

**public** **static** Menu getMenu() {

**return** menu;

    }

}

**Класс Graphics.java**

**package** kz.graphics;

**import** com.jogamp.opengl.GL2;

**import** com.jogamp.opengl.util.texture.Texture;

*//данный класс рисует в окне разные фигуры*

**public** **class** Graphics {

*//рисует картинку в данных координатах с данными размерами*

*//без поворота*

**public** **static** **void** drawImage(ImageResource image, **float** x, **float** y, **float** width, **float** height) {

*//вызывает эту функцию с нулевым поворотом*

        drawImageRot(image, x, y, width, height, 0, 0, 0, 1);

    }

*//рисует картинку в данных координатах с данными размерами и поворотом*

**public** **static** **void** drawImageRot(ImageResource image, **float** x, **float** y, **float** width, **float** height, **float** rotation, **int** vx, **int** vy, **int** vz) {

*//получает оболочку*

        GL2 gl = EventListener.getGL();

*//достает текстуру из картинки*

        Texture tex = image.getTexture();

*//если текстура существует связывает ее с объектом*

**if**(tex != **null**) {

            gl.glBindTexture(GL2.GL\_TEXTURE\_2D, tex.getTextureObject());

        }

*//переносит начало координат в середину прямоугольника*

*//который надо отобразить*

*//(чтобы поворот был вокруг правильной оси)*

        gl.glTranslatef(x+width/2, y+height/2, 0);

*//поворачивает вокруг центра*

        gl.glRotatef(rotation, vx, vy, vz);

*//устанавливает белый цвет наложения*

        gl.glColor4f(1, 1, 1, 1);

*//режим прорисовки - четырехугольник*

        gl.glBegin(GL2.GL\_QUADS);

*//рисует четырехугольник относительно его центра*

*//а также отображает текстуру в правильном последовательности*

        gl.glTexCoord2f(0, 0);

        gl.glVertex2f(-width/2, -height/2);

        gl.glTexCoord2f(1, 0);

        gl.glVertex2f(width/2, -height/2);

        gl.glTexCoord2f(1, 1);

        gl.glVertex2f(width/2, height/2);

        gl.glTexCoord2f(0, 1);

        gl.glVertex2f(-width/2, height/2);

        gl.glEnd();

        gl.glBindTexture(GL2.GL\_TEXTURE\_2D, 0);

*//обратно поворачивает координатную сетку и очищает буфер*

        gl.glRotatef(-rotation, vx, vy, vz);

        gl.glTranslatef(-x-width/2, -y-height/2, 0);

        gl.glFlush();

    }

}

**Класс ImageResource.java**

**package** kz.graphics;

**import** java.awt.image.BufferedImage;

**import** java.io.IOException;

**import** java.net.URL;

**import** javax.imageio.ImageIO;

**import** com.jogamp.opengl.util.texture.Texture;

**import** com.jogamp.opengl.util.texture.awt.AWTTextureIO;

*//класс с картинкой и преобразованием в текстуру*

**public** **class** ImageResource {

*//изначально и текстура и картинка равны нулю*

**private** Texture texture = **null**;

**private** BufferedImage image = **null**;

**public** ImageResource(String path) {

*//при инициализации достаем картину по ссылке и сохраняем ее*

        URL url = ImageResource.**class**.getResource(path);

**try** {

            image = ImageIO.read(url);

        } **catch** (IOException e) {

            e.printStackTrace();

        }

**if**(image != **null**)

            image.flush();

    }

*//достаем текстуру из картинки*

**public** Texture getTexture() {

*//если картинки нет возвращаем ноль*

**if**(image == **null**)

**return** **null**;

*//если есть создаем текстуру для оболочки*

**if**(texture == **null**)

            texture = AWTTextureIO.newTexture(Renderer.getProfile(), image, **true**);

**return** texture;

    }

}

**Класс Renderer.java**

**package** kz.graphics;

**import** com.jogamp.newt.event.WindowEvent;

**import** com.jogamp.newt.event.WindowListener;

**import** com.jogamp.newt.event.WindowUpdateEvent;

**import** com.jogamp.newt.opengl.GLWindow;

**import** com.jogamp.opengl.GLCapabilities;

**import** com.jogamp.opengl.GLProfile;

**import** com.jogamp.opengl.util.FPSAnimator;

**import** kz.engine.MouseInput;

*//отображение окна*

**public** **class** Renderer {

*//переменные для хранения окна*

**private** **static** GLWindow window = **null**;

**private** **static** GLProfile profile = **null**;

*//название игры*

**private** **static** **final** String GAME\_NAME = "Игра Дьявола";

*//фпс*

**public** **static** **final** **int** FPS = 60;

*//размеры*

**public** **static** **final** **int** WIDTH = 1040;

**public** **static** **final** **int** HEIGHT = 640;

*//геттер*

**public** **static** GLProfile getProfile() {

**return** profile;

    }

*//инициализация*

**public** **static** **void** init() {

*//создается новый профайл opengl версии 2*

        GLProfile.initSingleton();

        profile = GLProfile.get(GLProfile.GL2);

        GLCapabilities caps = **new** GLCapabilities(profile);

*//создается новое окно с обработкой нажатий*

*//и событий*

*//а также названием*

        window = GLWindow.create(caps);

        window.setSize(WIDTH, HEIGHT);

        window.setResizable(**false**);

        window.setSurfaceSize(WIDTH, HEIGHT);

        window.addGLEventListener(**new** EventListener());

        window.addMouseListener(**new** MouseInput());

        window.setTitle(GAME\_NAME);

*//создается объект который вызывает события*

*//такие как init и display*

*//с фпс 60 кадров в секунду*

        FPSAnimator animator = **new** FPSAnimator(window, FPS);

        animator.start();

        window.setVisible(**true**);

        window.addWindowListener(**new** WindowListener() {

*//при закрытии окна выключается объект созданный ранее*

*//и приложение закрывается*

            @Override

**public** **void** windowDestroyed(WindowEvent arg0) {

*// TODO Auto-generated method stub*

                animator.stop();

                System.exit(0);

            }

            @Override

**public** **void** windowGainedFocus(WindowEvent arg0) {

*// TODO Auto-generated method stub*

            }

            @Override

**public** **void** windowLostFocus(WindowEvent arg0) {

*// TODO Auto-generated method stub*

            }

            @Override

**public** **void** windowMoved(WindowEvent arg0) {

*// TODO Auto-generated method stub*

            }

            @Override

**public** **void** windowRepaint(WindowUpdateEvent arg0) {

*// TODO Auto-generated method stub*

            }

            @Override

**public** **void** windowResized(WindowEvent arg0) {

*// TODO Auto-generated method stub*

            }

            @Override

**public** **void** windowDestroyNotify(WindowEvent arg0) {

*// TODO Auto-generated method stub*

            }

        });

    }

*//входная точка программы*

**public** **static** **void** main(String[] args) {

        init();

}

}

ПРИЛОЖЕНИЕ Б