Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Севастопольский государственный университет»

Кафедра информационных систем

КУРСОВАЯ РАБОТА

ПО ДИСЦИПЛИНЕ «ОБЪЕКТНО-ОРИЕНТИРОВАНИЕ ПРОГРАММИРОВАНИЕ»

ПРОГРАММА «EssenceDefence»

Пояснительная записка

Листов 46

Студентов 2 курса группы ИС/б-21-о

направление подготовки 09.03.02

*(подпись)\_\_\_\_\_\_\_* Повх А.А.

*(подпись)\_\_\_\_\_\_\_* Куркчи А.Э.

*(подпись)\_\_\_\_\_\_\_* Мжачев И.А.

« » \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2016г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(должность, ученое звание, фамилия и инициалы)

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (фамилия и инициалы)

г. Севастополь – 2016 г.

# Аннотация

В данном курсовом проекте рассматривается разработка игрового приложения жанра ‘*Tower Defence*’. Объясняется спроектированная архитектура приложения, представленная в виде UML диаграмм, выбранные решения для разработки приложения. Также подробно рассматривается интерфейс пользователя.

Содержание

[Аннотация 1](#_Toc452129990)

[Введение 3](#_Toc452129991)

[Список исполнителей 4](#_Toc452129992)

[Постановка задачи 5](#_Toc452129993)

[Проектное решение 6](#_Toc452129994)

[Пользовательский интерфейс 14](#_Toc452129995)

[Программная реализация 16](#_Toc452129996)

[Заключение 17](#_Toc452129997)

[Приложение А 18](#_Toc452129998)

Куркчи А.Э

Мжачев И.А.

Дата

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

КУРСОВАЯ РАБОТА

Выполнили

Повх А.А.

Провер.

Пелипас В.О..

Утверд.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Лит.

Листов

Кафедра ИС

Группа ИС/б-21-о

# Введение

Целью написания курсового проекта является закреплением, углублением и обобщение знаний и навыков по пройдённому курсу «Объектно-Ориентированное Программирование». Задача была такова, разработать игровое приложение по жанру ‘*Tower Defence*’, которая была выполнена, название получившегося приложение – ‘*EssenceDefence*’.

В ходе разработки были проведены операции проектирования архитектуры приложения, применены практически все принципы ООП, а также применялись паттерны проектирования, такие как *Singleton, Builder* и другие.

Так как велась командная разработка, мы воспользовались системой контроля версий ‘*GitHub*’, для удобного параллельного программирования и хранения кода.

# Список исполнителей

Команда разработавшая данное приложение состоит из трёх человек, а именно: Повх А.А., Куркчи А.Э., Мжачев И.А.

В проектировании архитектуры принимали участие все из перечисленных людей, также совместно решались основные проблемы и риски проекта, происходил выбор языка программирования, игрового движка, вопросы организации разработки. Перечислим личный вклад каждого участника команды:

* Повх А.А. – создание обработчиков событий, игрового поля, графического интерфейса.
* Куркчи А.Э. – отвечал за все вопросы, связанные с монстром, построения путей для них, сохранения и загрузки объектов.
* Мжачев И.А. – создание классов строений, эссенции, балансировка игры.

# Постановка задачи

Моделируется противостояние множественных врагов и игрока, управляющего защитой крепости.

Игровое поле является ячеистым и состоит из набора путей, крепости игрока и порталов. На игровом поле изначально могут находиться некоторые башни и стены.

Крепость игрока является основным строением на поле. Цель игры – убить всех монстров и отбить волны, защищая крепость. Цель монстров – разрушить крепость. Крепость имеет свой показатель целостности, при достижении нуля, на котором она разрушается.

Пути соединяют все порталы с крепостью в любом порядке, гарантируется, что от любого портала есть хотя бы один путь.

Стены – особый объект, который располагается на любой ячейке поля, но не может быть установлена так, чтобы блокировать полностью путь от портала к крепости.

Портал – единственный способ появления монстров, могут быть несколько на одном поле и могут выпускать различных монстров.

Башни являются оболочкой, в которую заключаются атакующие сущности. Их строительство разрешено в любом месте поля, за исключением случаев блокирования пути. Башни можно размещать на поле, но нельзя разрушить.

Сущности представляют собой ядра для башни, то что её питает и является её атакующим элементом. Они различаются по видам и уровням, их можно улучшать, продавать, комбинировать и перемещать между башнями в процессе игры.

Монстры – единственный, а потому злейший враг. Появляется из порталов и стремиться дойти до крепости по кратчайшему возможному пути. Имеют здоровье, при падении которого до нуля умирают. Могут быть разных видов, для каждого из которых характерно своё значение скорости и начального количества здоровья, а также урона, наносимого крепости при её достижении.

Игрок на момент начала имеет некое количество ресурсов, для создания сущностей, строительства башен и стен. Ресурсы пополняются в процессе игры автоматически и при уничтожении монстров.

# Проектное решение

Были выделены основные классы: игрок, монстр, здание (абстрактный класс), игровое поле, эссенция. Класс наследующие абстрактный класс здание – это крепость, стена, башня, портал. Итого мы имеем девять основных классов, более подробно это представлено на рисунке 1.

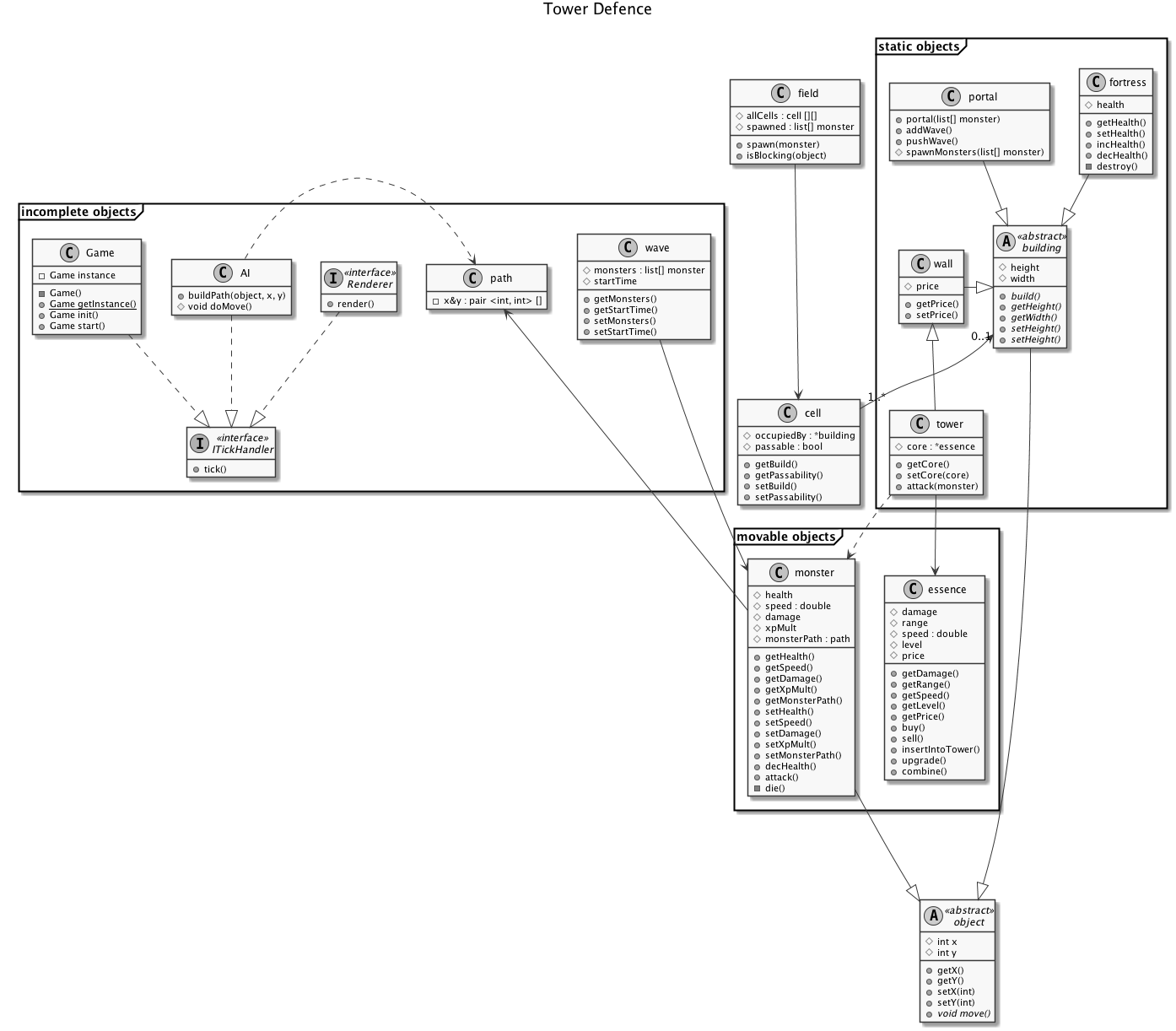


Рисунок 1 – Диаграмма классов

Жизненный цикл программы начинается с главного меню, где имеются кнопки «Новая игра», «Продолжить», «Выход». Нажимая кнопку начала новой игры, мы запускаем приложение, видим перед собой игровое поле, с путями, крепостью и определенным количеством порталов, откуда вскоре будут появляться монстры.

Монстры при появлении начинают свой поход к крепости для уничтожения его, игрок препятствует этому. Конец жизненного цикла приложения достигается двумя способами или монстры уничтожаю крепость, или игрок отбивает все волны нападающих монстров.

На рисунке 2 представлена диаграмма последовательностей.

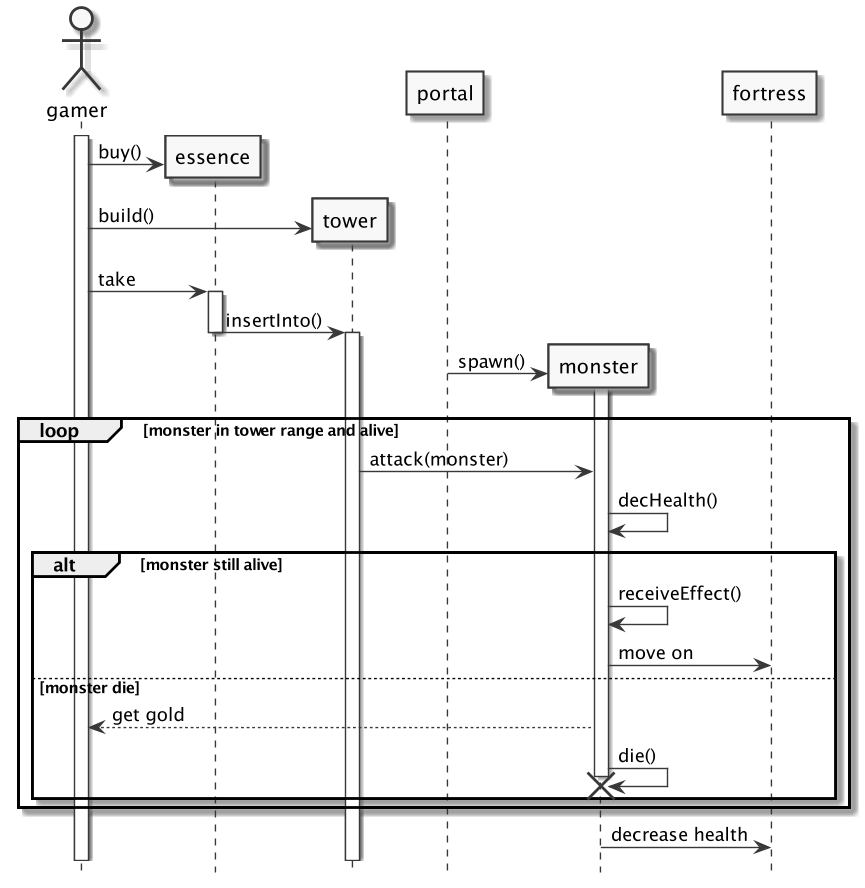


Рисунок 2 – Диаграмма последовательностей

На рисунке 3 диаграмма последовательностей игрока.

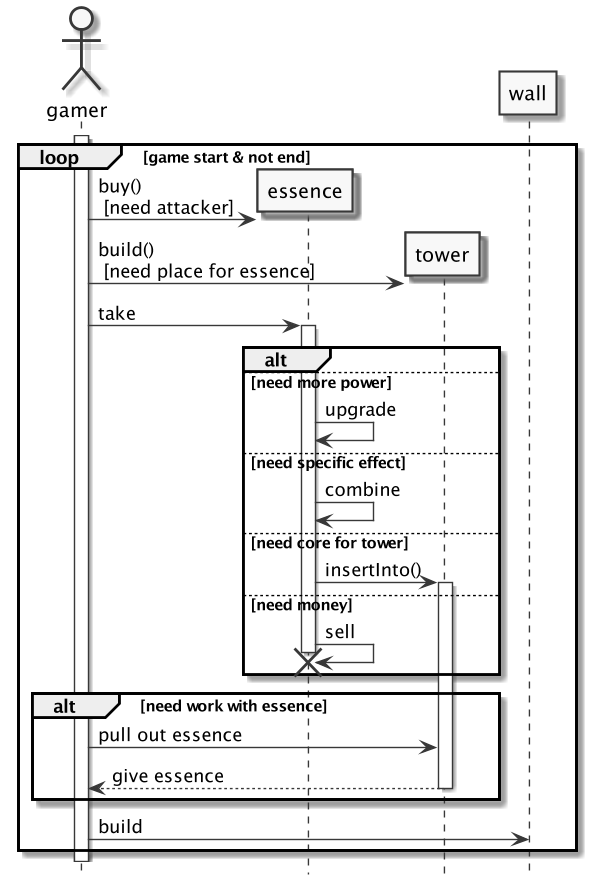


Рисунок 3 – Диаграмма последовательностей игрока

Объект игра. После запуска игры происходит переход в состояние игры. Если крепость будет разрушена, то мы переход в состояние проигрыша. Если все монстры умрут, а волн больше не останется, то мы переходим в состояние победы. Из этих двух состояний мы переходим в состояние завершения игры после нажатия на энтер. Когда все процессы игры будут завершены, объект прекратит свое существование.

Объект игрок. После начала игры, мы переходим в состояние ожидания. Из этого состояния мы может перейти в состояния: строительства стен, если нужно изменить путь монстрам и достаточно денег, деятельность с сущностью, если есть необходимость в атакующих элементах или же в состояние постройки башен, если нужны новые места для сущностей. По завершению любого из этих состояний мы возвращаемся обратно в ожидание. Когда игра завершается объект прекращает свое существование.

Объект волна. После, того как портал начинает призывать волну происходит переход в состояние призыва монстра. Как только призванный монстр начал двигаться происходит переход в состояние задержки. После ее завершения возвращаемся обратно в состояние призыв монстра. Если в данной волне больше не осталось монстров объект прекращает свое существование.

Объект монстр. После того как монстр призван он переходит в состояние движение к крепости. Если во время нахождения в данном состоянии, монстр пройдет возле башни, то при условии, что башня будет атаковать он перейдет в состояние уменьшения здоровья. Если у монстра не останется больше здоровья, то он умирает, объект удаляется, а игрок получает награду. Из состояния уменьшения здоровья можно перейти обратно при условии, что монстр до сих пор жив. При достижения крепости происходит переход в состояние атаки крепости. В данном состоянии происходит уменьшение здоровья и удаления монстра, после чего объект прекращает свое существование

Объект сущность. Выбирая сущность при условии, что она существует или же комбинируя сущности происходит переход в активное состояние. После покупки при условии наличия достаточного количества денег объект переходит в неактивное состояние. Из него можно перейти в активное после того как объект будет вставлен в башню при условии наличия последнего. При переходе в активное состояние происходит задержка. Из него после того как вытащить объект, происходит переход в неактивное состояние. При продаже объект уничтожается. При необходимости большего урона объект переходит в состояние улучшения при условии достаточного количества денег. При необходимости особого эффекта объект переходит в состояние комбинирования при условии достаточного количества денег. Из комбинирования объект уничтожается.

На рисунке 4-7 показана диаграмма состояний объектов игра, игрок, волна, монстр и сущность.

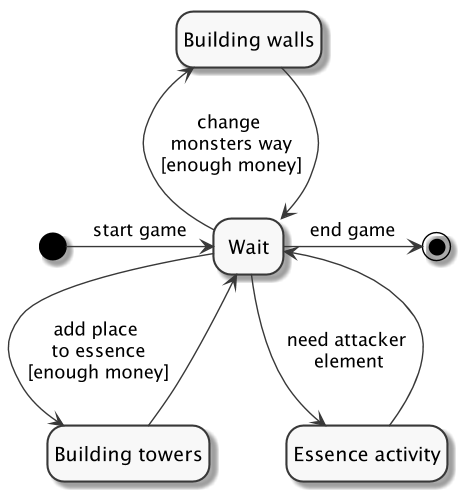
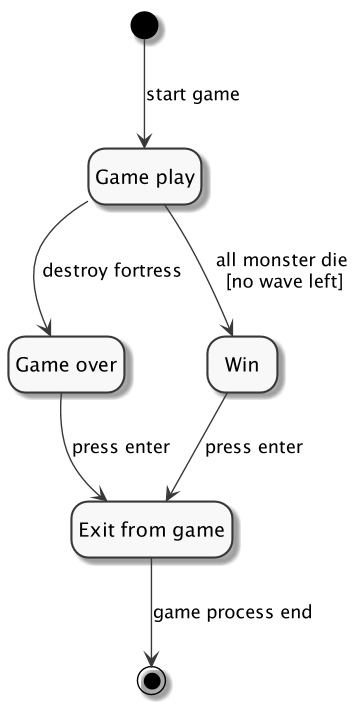


Рисунок 4 – Диаграммы состояний объектов игра и игрок

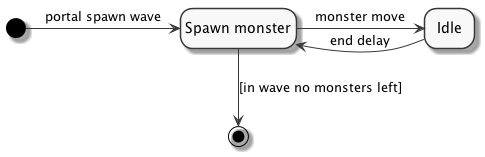


Рисунок 5 – Диаграмма состояний объекта волна

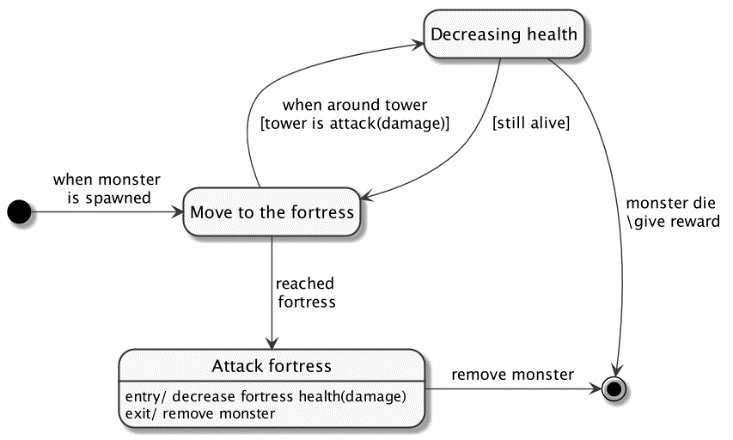


Рисунок 6 – Диаграмма состояний объекта монстр

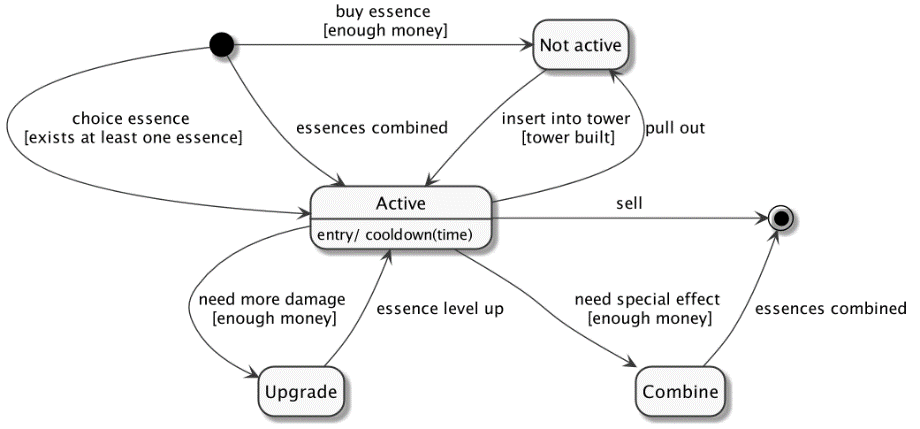


Рисунок 7 – Диаграмма состояний объекта сущность

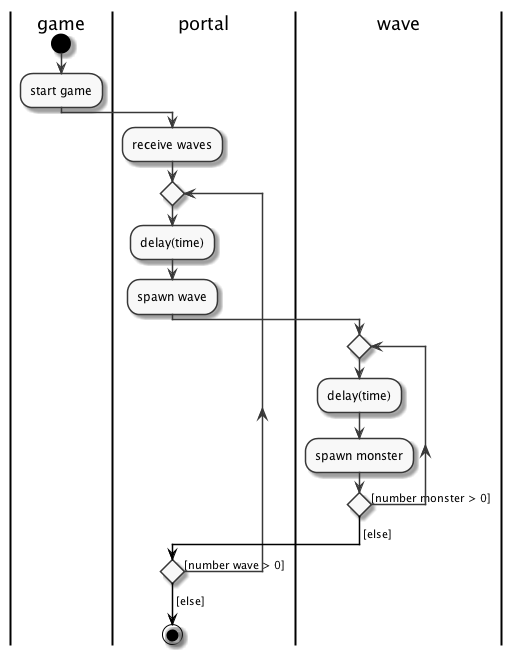
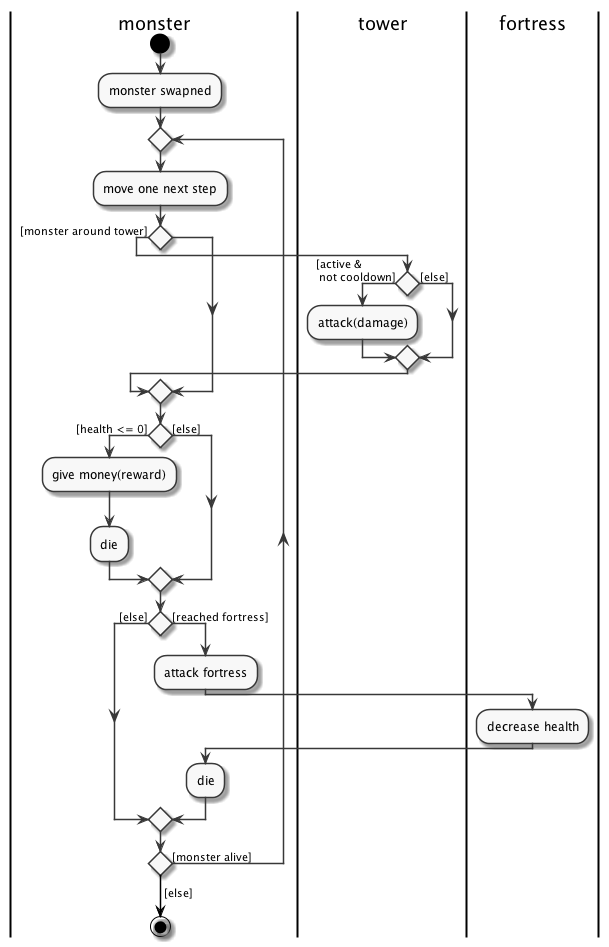
После запуска начала игры игрок получает начальную сумму денег. В зависимости от желания игрока и наличия достаточного количества денег игрок может строить башни, стены, покупать сущности, вставлять их в башни, улучшать или комбинировать сущности. На прямую игрок может получить деньги только продав сущность. Деятельность игрока продолжается до тех пор, пока есть хотя бы один живой монстр и не все волны монстров призваны.

Параллельно деятельности игрока работает портал. После определенной задержки он начинает призывать волны монстров. Призыв волн продолжается до тех пор, пока есть хотя бы еще одна волна в портале.

Волна призывается следующим образом. Спустя задержку, призывается монстр и так до тех пор, пока в волне остался хотя бы один монстр.

Параллельно деятельности портал движутся монстры. Каждый монстр начинает свое движение сразу после призыва. Монстр передвигается по шагу по направлению к крепости. Если он оказывается возле башни, то при условии, что башня активирована и не находится в состоянии перезарядки она атакует монстра. Если здоровье монстра опускается ниже 0, то игрок получает вознаграждении, а монстр умирает. Если же монстр достигает крепости, то он уменьшает ей прочность. Если прочность достигает 0, то игра заканчивается. При любом исходе монстр погибает. После того как монстр погиб, он удаляется.

Если все монстры погибли, в портале не осталось больше волн и крепость все еще не разрушена, то можно поздравить игрока, он победил.

На рисунке 8 показана диаграммы видов деятельности игры, монстра, портала.

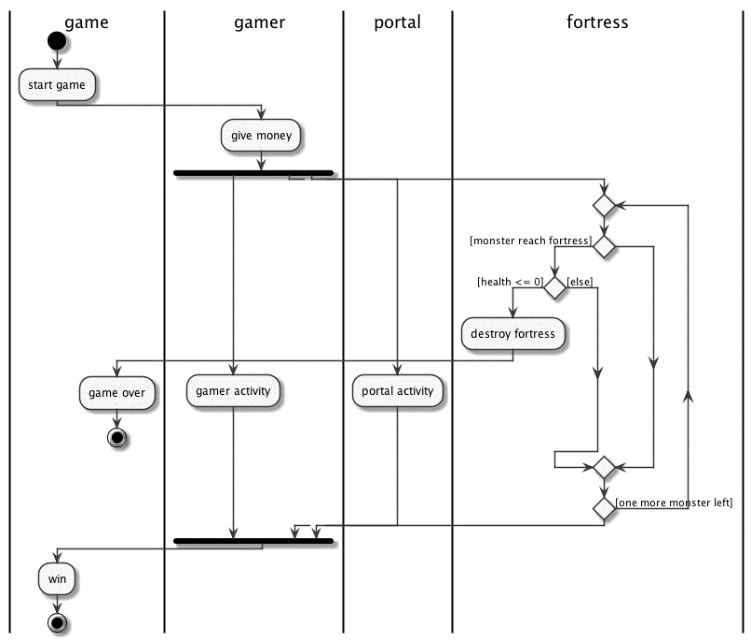


Рисунок 8 – Диаграммы видов деятельности игры, портала, монстра

На рисунке 9 показана диаграмма видов деятельности игрока.

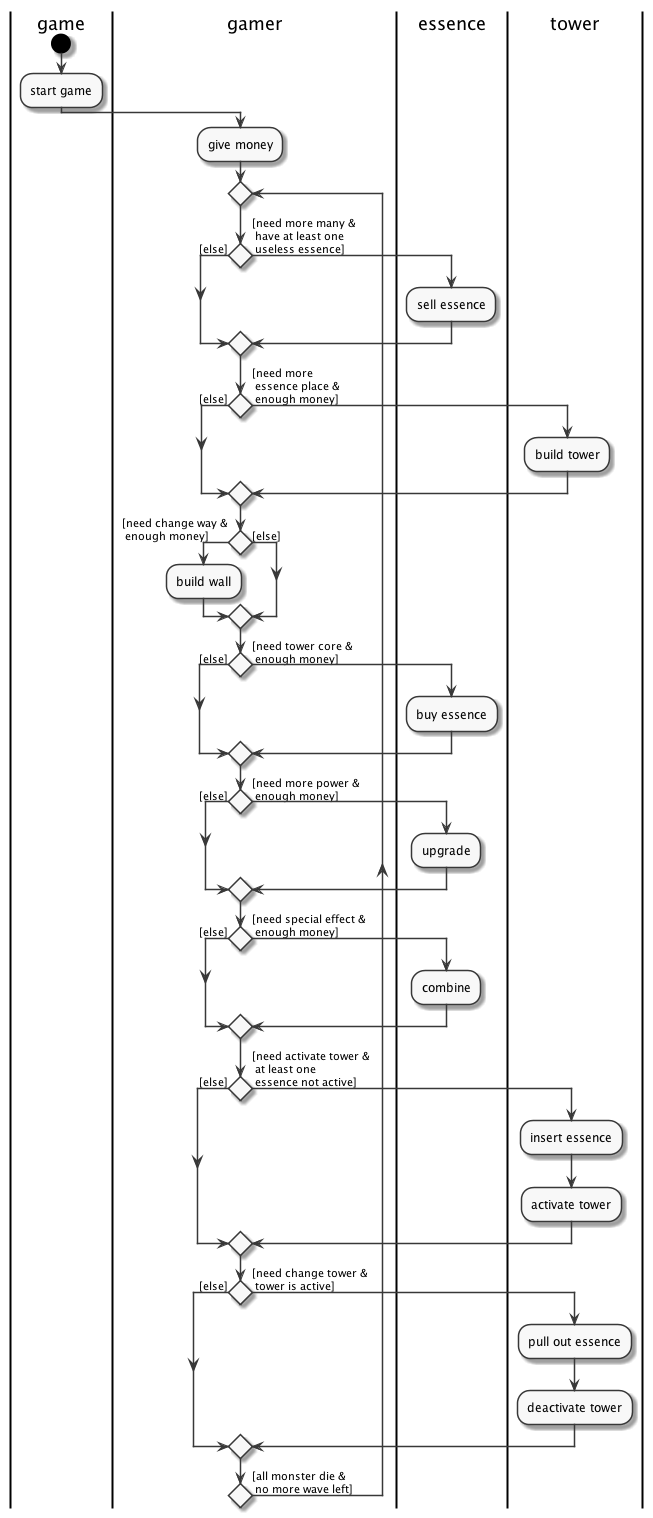


Рисунок 9 – Диаграмма видов деятельности игрок

# Пользовательский интерфейс

После запуска игры, первое что мы видим это меню рисунок 10. В нем есть три пункта: новая игра, продолжить и выход. Кнопка продолжить еще неактивная, так как ни одна игра пока что не была создана.

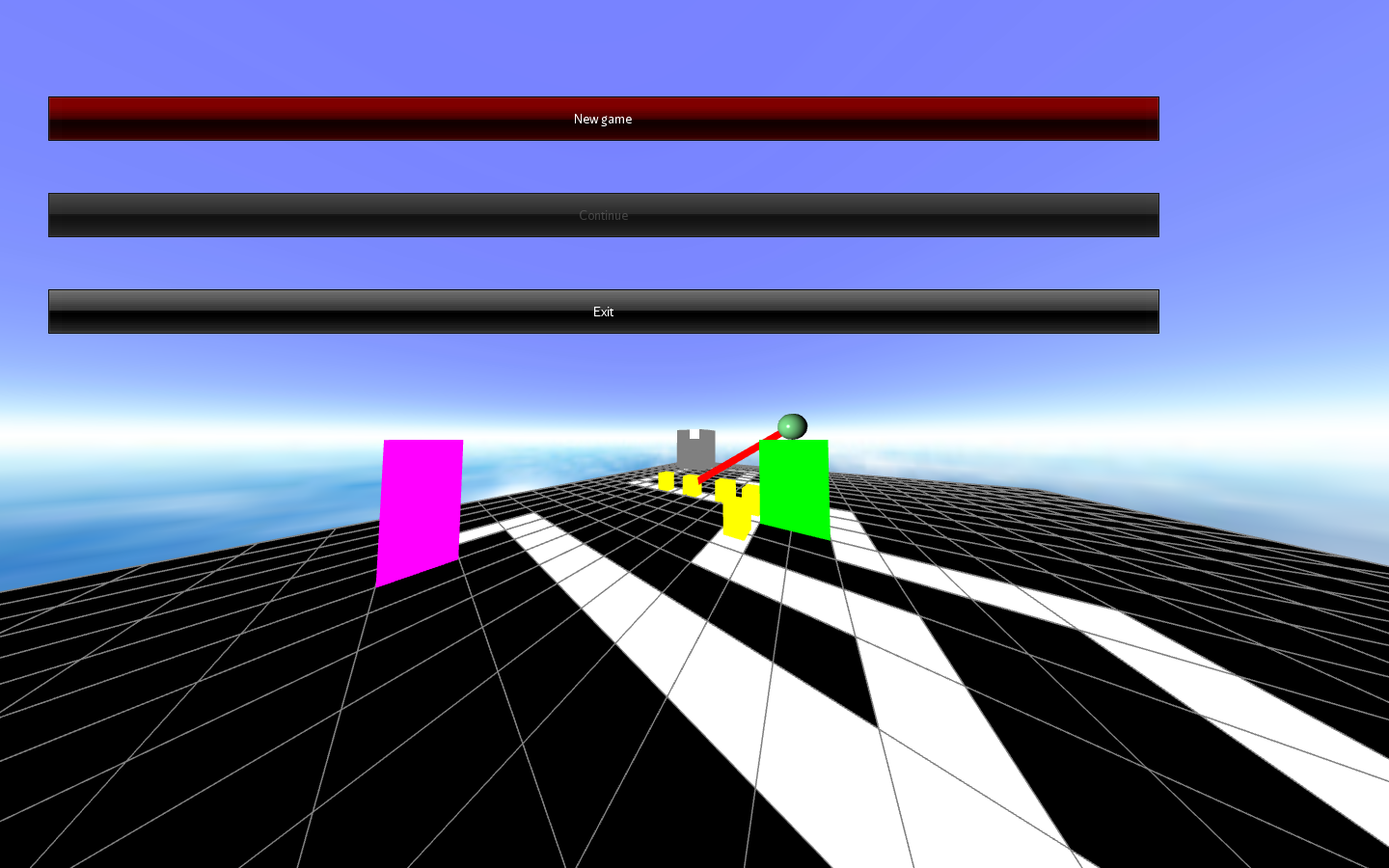


Рисунок 10 – Главное меню игры

После выбора новой игры появляется игровое поле (рисунок 11). На нем присутствуют следующие элементы: полоска денег игрока, поле на котором происходит основные действия игры, инвентарь, в котором мы можем хранить сущности и магазин сущностей.

В магазине мы можем купить 1 из 9 сущностей, которые были сгенерированы случайным образом. Покупка осуществляется простым перетаскиванием сущности из магазина. Мы можем продать сущности просто кинув ее в магазин.

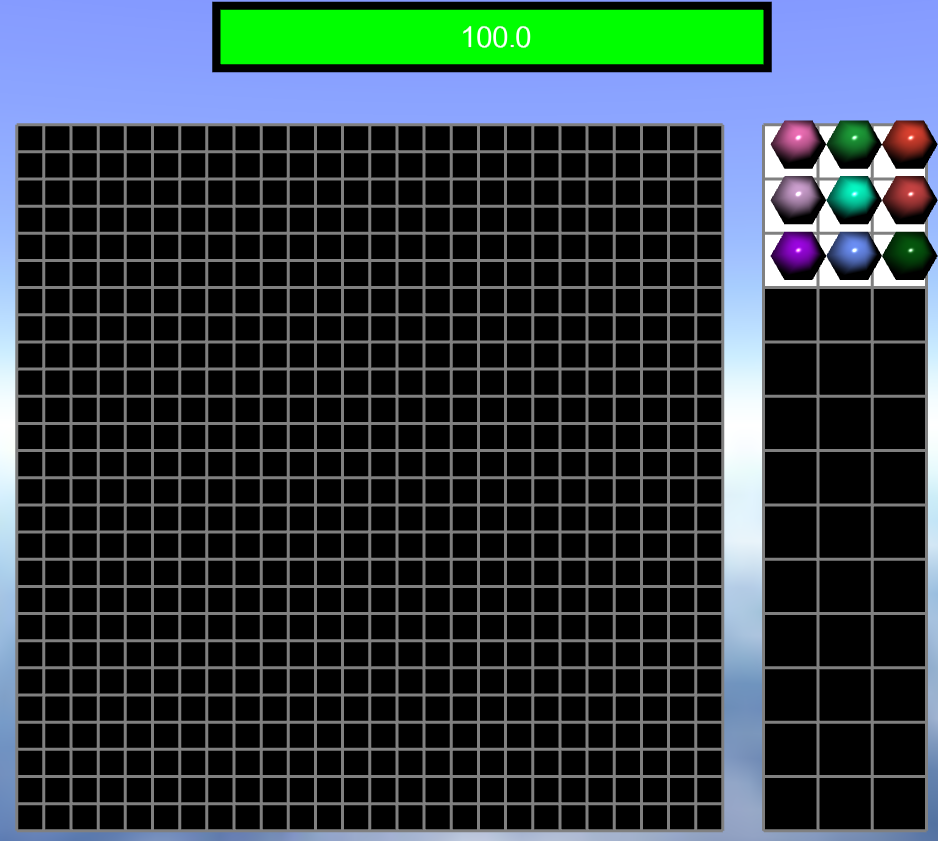


Рисунок 11 – Игровое поле

Для того чтобы следить за прочностью своей крепости и здоровью монстров, для каждого из них существует полоска жизни вместе с численным эквивалентом указанной величины (рисунок 12). Цвет полоски медленно меняется от зеленого к желтому, а затем к красному.

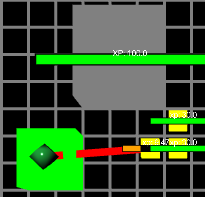


Рисунок 12 – Полоски жизни

Чтобы узнать параметры сущности (уровень, урон, скорость и так далее) достаточно просто навести курсор мыши на сущность и нажать правую кнопку мыши. В результате этих действий появиться информация о сущности (рисунок 13). Данная информация сама исчезнет через некоторое время.

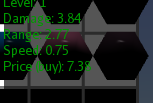


Рисунок 13 – Информация о сущности

# Программная реализация

Одним из важнейших вопросов разработки приложения, это был вопрос выбора языка программирования, выбор стоял между языками *C++*, *C#*, *Java*. В итоге был выбран язык Java, так как он кроссплатформенен, удобен и прост в использовании. Еще одни фактором выбора данного языка было то, что был найден бесплатный игровой *3D* движок *JMonkeyEngine*, функционал которого наc вполне устроил.

# Заключение

В ходе выполнения курсового проекта было закреплены, углублены и усовершенствованы знания и навыки по дисциплине «Объектно-Ориентированное Программирование».

Было проведено архитектурное проектирование приложения с помощью принципов ООП, в ходе которого были построены UML-диаграммы.

Также в ходе разработки игрового приложения были сформированы навыки работы в команде, работы с системой контроля версий ‘*GitHub*’.

Были получены практические навыки применения принципов объектно-ориентированного программирования, наследования, полиморфизма, инкапсуляции.

А также был получен опыт в применение паттернов проектирования, из ярко-выраженных это *Singleton* и *Builder*.

# Приложение А

Класс GamePlayAppState

package io.github.sevjet.essensedefence;

import com.jme3.app.Application;

import com.jme3.app.state.AbstractAppState;

import com.jme3.app.state.AppStateManager;

import com.jme3.font.BitmapText;

import com.jme3.math.ColorRGBA;

import com.jme3.scene.Node;

import io.github.sevjet.essensedefence.control.GameControl;

import io.github.sevjet.essensedefence.field.Field;

import io.github.sevjet.essensedefence.field.MapField;

import io.github.sevjet.essensedefence.util.Configuration;

import io.github.sevjet.essensedefence.util.Creator;

public class GamePlayAppState extends AbstractAppState {

public static Field field;

protected Node localGui = new Node("localGui");

protected Node localRoot = new Node("localRoot");

private static final String text =

"W - Wall\n" +

"T - Tower\n" +

"P - Portal\n" +

"F - Fortress\n" +

"\n" +

"E - make Cell passable\n" +

"R - reset\n" +

"M - Monster\n" +

"G - Start Wave\n" +

"B - Buy Essence\n" +

"S - Sell Essence";

public GamePlayAppState() {

}

public Node getLocalRoot() {

return localRoot;

}

public Node getLocalGui() {

return localGui;

}

protected void initStartData() {

Configuration.getGuiNode().attachChild(localGui);

Configuration.getRootNode().attachChild(localRoot);

Gamer gamer = Configuration.getGamer();

gamer.setGold(100f);

gamer.setGui();

localGui.attachChild(gamer.getGui());

BitmapText btext = Creator.text2D(text, ColorRGBA.Black);

btext.setLocalTranslation(10f, Configuration.getSettings().getHeight(), 0);

btext.scale(0.7f);

localGui.attachChild(btext);

}

private void placeGameFields() {

int n, m;

n = 26;

m = n;

field = new MapField(n, m);

field.setLocalTranslation(0f, 0f, 0f);

field.addControl(new GameControl(10f));

localRoot.attachChild(field);

Configuration.getGamer().resetShop();

Node shop = Configuration.getGamer().getShop();

shop.setLocalTranslation(m + 2, 20f + 0.5f, 0f);

shop.scale(2f);

localRoot.attachChild(shop);

Configuration.getGamer().resetInventory();

Node invent = Configuration.getGamer().getInventory();

invent.setLocalTranslation(m + 2, 0.5f, 0f);

invent.scale(2f);

localRoot.attachChild(invent);

}

@Override

public void initialize(AppStateManager stateManager, Application app) {

super.initialize(stateManager, app);

initStartData();

placeGameFields();

setEnabled(false);

}

@Override

public void update(float tpf) {

super.update(tpf);

}

@Override

public void stateDetached(AppStateManager stateManager) {

super.stateDetached(stateManager);

localGui.removeFromParent();

localRoot.removeFromParent();

}

@Override

public void stateAttached(AppStateManager stateManager) {

super.stateAttached(stateManager);

Configuration.getRootNode().attachChild(localRoot);

Configuration.getGuiNode().attachChild(localGui);

}

}

Класс Field

package io.github.sevjet.essensedefence.field;

import com.jme3.asset.plugins.FileLocator;

import com.jme3.export.InputCapsule;

import com.jme3.export.JmeExporter;

import com.jme3.export.JmeImporter;

import com.jme3.export.OutputCapsule;

import com.jme3.export.binary.BinaryExporter;

import com.jme3.math.ColorRGBA;

import com.jme3.scene.Geometry;

import com.jme3.scene.Node;

import io.github.sevjet.essensedefence.entity.Entity;

import io.github.sevjet.essensedefence.util.Configuration;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import static io.github.sevjet.essensedefence.util.Creator.gridXY;

public abstract class Field<T extends Cell> extends Node {

protected ArrayList<T> cells;

protected Map<Class<? extends Entity>, Node> objects;

protected Node grid;

private int rows;

private int cols;

@SuppressWarnings("unused")

public Field() {

}

public Field(final int cols, final int rows) {

this.rows = rows;

this.cols = cols;

this.setName("field");

this.objects = new HashMap<>();

final int len = cols \* rows;

cells = new ArrayList<>(len);

for (int i = 0; i < len; i++) {

final int x = i % cols;

final int y = i / cols;

final T cell = newCell(x, y);

cells.add(cell);

addObject(cell);

}

gridOn();

}

public static boolean save(Field field, File file) {

if (field != null) {

BinaryExporter exporter = BinaryExporter.getInstance();

try {

return exporter.save(field, file);

} catch (IOException ex) {

ex.printStackTrace(System.err);

}

}

return false;

}

public static Field load(File file) {

if (file != null && file.exists()) {

Configuration.getAssetManager().registerLocator(file.getParent(), FileLocator.class);

Field field = (Field) Configuration.getAssetManager().loadModel(file.getAbsolutePath());

Configuration.getAssetManager().unregisterLocator(file.getParent(), FileLocator.class);

return field;

}

return null;

}

protected abstract T newCell(final int x, final int y);

public T getCell(final int x, final int y) {

if (x < 0 || x >= cols || y < 0 || y >= rows) {

return null;

}

final int index = y \* cols + x;

return cells.get(index);

}

public T getCell(final Geometry geom) {

if (geom != null &&

geom.getParent() != null &&

geom.getParent().getParent() == this) {

final int x = (int) geom.getLocalTranslation().getX();

final int y = (int) geom.getLocalTranslation().getY();

return getCell(x, y);

}

return null;

}

public Node getObjects(final Class<? extends Entity> clazz) {

return objects.get(clazz);

}

public boolean addObject(final Entity object) {

Node node = objects.get(object.getClass());

if (node == null) {

node = new Node();

objects.put(object.getClass(), node);

attachChild(node);

}

if (!node.hasChild(object.getGeometry())) {

node.attachChild(object.getGeometry());

guiFor(object);

return true;

}

return false;

}

public abstract boolean canGet(T cell, Class<? extends Entity> contentClass);

public abstract boolean canSet(T cell, Class<? extends Entity> contentClass);

public abstract Entity getContent(T cell, Class<? extends Entity> contentClass);

public Entity getContent(final int x, final int y, Class<? extends Entity> contentClass) {

return getContent(getCell(x, y), contentClass);

}

public abstract boolean setContent(T cell, Entity content);

public boolean setContent(final int x, final int y, Entity content) {

return setContent(getCell(x, y), content);

}

public boolean removeObject(final Entity object) {

Node node = objects.get(object.getClass());

if (node == null) {

return false;

}

if (node.hasChild(object.getGeometry())) {

node.detachChild(object.getGeometry());

return true;

}

return false;

}

public void removeAll() {

objects.keySet().forEach(this::removeAll);

}

public boolean removeAll(final Class<? extends Entity> clazz) {

final Node node = objects.get(clazz);

if (node != null) {

node.detachAllChildren();

return true;

}

return false;

}

public int getRows() {

return rows;

}

public int getCols() {

return cols;

}

protected boolean gridOn() {

grid = gridXY(getCols() + 1, getRows() + 1, 1, ColorRGBA.Gray, 3f);

grid.setLocalTranslation(-0.5f, -0.5f, 0);

attachChild(grid);

return true;

}

public Node getGrid() {

return grid;

}

@Override

public void write(JmeExporter ex) throws IOException {

super.write(ex);

OutputCapsule capsule = ex.getCapsule(this);

capsule.write(rows, "rows", 1);

capsule.write(cols, "cols", 1);

capsule.writeSavableArrayList(cells, "cells", null);

}

@Override

public void read(JmeImporter im) throws IOException {

super.read(im);

InputCapsule capsule = im.getCapsule(this);

rows = capsule.readInt("rows", 1);

cols = capsule.readInt("cols", 1);

ArrayList data = capsule.readSavableArrayList("cells", null);

final int len = cols \* rows;

cells = new ArrayList<>(len);

for (int i = 0; i < len; i++) {

final Object el = data.get(i);

cells.add((T) el);

}

}

}

Класс Entity

package io.github.sevjet.essensedefence.entity;

import com.jme3.export.\*;

import com.jme3.math.Vector3f;

import com.jme3.scene.Geometry;

import io.github.sevjet.essensedefence.field.Field;

import io.github.sevjet.essensedefence.util.GeometryManager;

import java.io.IOException;

public abstract class Entity implements Savable {

protected int x = 0;

protected int y = 0;

protected int z = 0;

protected Geometry geometry = null;

public Entity() {

this(0, 0);

}

public Entity(int x, int y) {

this.x = x;

this.y = y;

geometry = GeometryManager.getDefault(getClass());

if (geometry == null) {

geometry = GeometryManager.getDefault(Entity.class);

}

if (geometry != null) {

geometry.setUserData("entity", this);

}

update();

}

public int getX() {

return x;

}

@Deprecated

public void setX(int x) {

this.x = x;

update();

}

public int getY() {

return y;

}

@Deprecated

public void setY(int y) {

this.y = y;

update();

}

public int getZ() {

return z;

}

@Deprecated

public void setZ(int z) {

this.z = z;

update();

}

public Geometry getGeometry() {

return geometry;

}

@Deprecated

public void setGeometry(Geometry geometry) {

if (geometry == null) {

throw new IllegalArgumentException("Geometry can not be null");

}

this.geometry = geometry;

update();

}

public void translate(int x, int y) {

translate(x, y, 0);

}

public void translate(int x, int y, int z) {

translate(x, y, z, true);

}

public void translate(int x, int y, int z, boolean doUpdate) {

move(this.x + x, this.y + y, this.z + z, doUpdate);

}

public void move(int x, int y) {

move(x, y, this.z);

}

public void move(int x, int y, int z) {

move(x, y, z, true);

}

public void move(int x, int y, int z, boolean doUpdate) {

this.x = x;

this.y = y;

this.z = z;

if (doUpdate) {

update();

}

}

public Field getField() {

if (geometry != null &&

geometry.getParent() != null &&

geometry.getParent().getParent() != null) {

if (geometry.getParent().getParent() instanceof Field) {

return (Field) geometry.getParent().getParent();

}

}

return null;

}

public Vector3f getCenter() {

return new Vector3f(x, y, z);

}

public Vector3f getPhysicalCenter() {

return geometry == null ?

Vector3f.ZERO :

geometry.getLocalTranslation();

}

protected boolean moveToCenter() {

geometry.setLocalTranslation(getCenter());

return true;

}

protected boolean update() {

return geometry != null && moveToCenter();

}

@Override

public void write(JmeExporter ex) throws IOException {

OutputCapsule capsule = ex.getCapsule(this);

capsule.write(x, "x", 0);

capsule.write(y, "y", 0);

capsule.write(z, "z", 0);

capsule.write(geometry, "geometry", null);

}

@Override

public void read(JmeImporter im) throws IOException {

InputCapsule capsule = im.getCapsule(this);

x = capsule.readInt("x", 0);

y = capsule.readInt("y", 0);

z = capsule.readInt("z", 0);

geometry = (Geometry) capsule.readSavable("geometry", null);

}

}

Класс Entity3D

package io.github.sevjet.essensedefence.entity;

import com.jme3.math.Vector3f;

import io.github.sevjet.essensedefence.util.BoxSize;

public abstract class Entity3D extends Entity {

private final BoxSize size;

public Entity3D() {

this(0, 0);

}

public Entity3D(int x, int y) {

this(x, y, BoxSize.FLAT);

}

public Entity3D(BoxSize size) {

this(0, 0, size);

}

public Entity3D(int x, int y, BoxSize size) {

super(x, y);

this.size = size;

update();

}

public BoxSize getSize() {

return size;

}

@Override

public Vector3f getCenter() {

BoxSize size = this.size == null ? BoxSize.FLAT : this.size;

return new Vector3f(x + (size.getWidth() - 1) / 2.0F,

y + (size.getHeight() - 1) / 2.0F,

z + (size.getDepth()) / 2.0F);

}

}

Класс Building

package io.github.sevjet.essensedefence.entity.building;

import com.jme3.export.InputCapsule;

import com.jme3.export.JmeExporter;

import com.jme3.export.JmeImporter;

import com.jme3.export.OutputCapsule;

import io.github.sevjet.essensedefence.entity.Entity3D;

import io.github.sevjet.essensedefence.field.Field;

import io.github.sevjet.essensedefence.field.MapCell;

import io.github.sevjet.essensedefence.field.MapField;

import io.github.sevjet.essensedefence.util.BoxSize;

import java.io.IOException;

public abstract class Building extends Entity3D {

protected float health = -1f;

private boolean destroyed = false;

public Building() {

super();

}

public Building(BoxSize size, float health) {

super(size);

setHealth(health);

}

public Building(int x, int y, BoxSize size, float health) {

super(x, y, size);

setHealth(health);

}

public float getHealth() {

return health;

}

public void setHealth(float health) {

if (health <= 0f && health != -1f) {

throw new IllegalArgumentException("Health can not be negative. For infinite use -1f");

}

this.health = health;

}

public void build() {

MapField field = getField();

if (field != null) {

for (int i = getX(); i < getX() + getSize().getWidth(); i++) {

for (int j = getY(); j < getY() + getSize().getHeight(); j++) {

final MapCell cell = field.getCell(i, j);

if (cell != null) {

cell.setContent(this);

}

}

}

}

}

public MapField getField() {

final Field field = super.getField();

return field != null ? (MapField) field : null;

}

public void destroy() {

destroyed = true;

}

public boolean isDestroyed() {

return destroyed;

}

@Override

public void write(JmeExporter ex) throws IOException {

super.write(ex);

OutputCapsule capsule = ex.getCapsule(this);

capsule.write(health, "health", -1f);

capsule.write(destroyed, "destroyed", false);

}

@Override

public void read(JmeImporter im) throws IOException {

super.read(im);

InputCapsule capsule = im.getCapsule(this);

health = capsule.readFloat("health", -1f);

destroyed = capsule.readBoolean("destroyed", false);

}

}

Класс Fortress

package io.github.sevjet.essensedefence.entity.building;

import io.github.sevjet.essensedefence.field.Field;

import io.github.sevjet.essensedefence.gui.ITextual;

import io.github.sevjet.essensedefence.util.BoxSize;

public class Fortress extends Building implements ITextual {

public static final BoxSize SIZE = new BoxSize(3, 3, 4);

public Fortress() {

super(SIZE, -1f);

}

public Fortress(float health) {

super(SIZE, health);

}

public Fortress(int x, int y, float health) {

super(x, y, SIZE, health);

}

public void hit(float damage) {

if (health != -1f) {

health -= damage;

if (health <= 0f) {

die();

}

}

update();

}

private void die() {

Field field = getField();

if (field != null) {

field.removeObject(this);

}

destroy();

}

@Override

public boolean update() {

super.update();

return true;

}

@Override

public String outputValue() {

return Float.toString(health);

}

@Override

public boolean isEnded() {

return (health <= 0 || isDestroyed() || geometry.getParent() == null);

}

}

Класс Portal

package io.github.sevjet.essensedefence.entity.building;

import com.jme3.export.InputCapsule;

import com.jme3.export.JmeExporter;

import com.jme3.export.JmeImporter;

import com.jme3.export.OutputCapsule;

import io.github.sevjet.essensedefence.control.WaveControl;

import io.github.sevjet.essensedefence.entity.monster.Monster;

import io.github.sevjet.essensedefence.field.Field;

import io.github.sevjet.essensedefence.util.BoxSize;

import java.io.IOException;

import java.util.ArrayList;

public class Portal extends Building {

public static final BoxSize SIZE = new BoxSize(1, 2, 3);

private ArrayList<WaveControl> waves = new ArrayList<>();

private int pushIndex = 0;

public Portal() {

super(SIZE, -1f);

}

public Portal(float health) {

super(SIZE, health);

}

public Portal(int x, int y, float health) {

super(x, y, SIZE, health);

}

public void addWave(WaveControl wave) {

waves.add(wave);

}

public boolean pushWave() {

if (waves.size() > pushIndex) {

geometry.addControl(waves.get(pushIndex++));

return true;

}

return false;

}

public boolean isWaveActive() {

WaveControl wave = geometry.getControl(WaveControl.class);

return wave != null;

}

public void spawn(Monster monster) {

Field field;

if ((field = getField()) != null) {

monster.move(getX(), getY());

field.addObject(monster);

}

}

@Override

public void write(JmeExporter ex) throws IOException {

super.write(ex);

OutputCapsule capsule = ex.getCapsule(this);

capsule.writeSavableArrayList(waves, "waves", null);

capsule.write(pushIndex, "pushIndex", 0);

}

@Override

public void read(JmeImporter im) throws IOException {

super.read(im);

InputCapsule capsule = im.getCapsule(this);

ArrayList list = capsule.readSavableArrayList("waves", null);

if (!list.isEmpty()) {

waves = new ArrayList<>(list.size());

for (Object el : list) {

if (el instanceof WaveControl) {

waves.add((WaveControl) el);

}

}

}

pushIndex = capsule.readInt("pushIndex", 0);

}

}

Класс Tower

package io.github.sevjet.essensedefence.entity.building;

import com.jme3.export.InputCapsule;

import com.jme3.export.JmeExporter;

import com.jme3.export.JmeImporter;

import com.jme3.export.OutputCapsule;

import com.jme3.math.Vector3f;

import com.jme3.scene.Node;

import com.jme3.scene.Spatial;

import io.github.sevjet.essensedefence.control.BasicControl;

import io.github.sevjet.essensedefence.control.TowerControl;

import io.github.sevjet.essensedefence.entity.Entity;

import io.github.sevjet.essensedefence.entity.Essence;

import io.github.sevjet.essensedefence.entity.monster.Monster;

import io.github.sevjet.essensedefence.field.Field;

import io.github.sevjet.essensedefence.util.BoxSize;

import io.github.sevjet.essensedefence.util.Getter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class Tower extends BuyableBuilding {

public static final BoxSize SIZE = new BoxSize(2, 2, 3);

private BasicControl control = new TowerControl();

private Essence core;

public Tower() {

super(SIZE, -1f, 0f);

geometry.addControl(control);

}

public Tower(float health, float price) {

super(SIZE, health, price);

geometry.addControl(control);

}

public Tower(int x, int y, float health, float price) {

super(x, y, SIZE, health, price);

geometry.addControl(control);

}

public Essence getCore() {

return core;

}

public void putCore(Essence core) {

if (this.core == null) {

this.core = core;

placeCore();

}

}

public float extractCore() {

float result = 0f;

Field field = getField();

if (field != null && core != null) {

result = core.getPrice();

core.setOffsetX(0f);

core.setOffsetY(0f);

field.removeObject(core);

core = null;

}

return result;

}

private void placeCore() {

core.move(getX(), getY(), getSize().getDepth());

core.setOffsetX(0.5f);

core.setOffsetY(0.5f);

Field field = getField();

if (field != null) {

field.addObject(core);

}

}

public List<Monster> getCloseMonsters() {

if (isEmpty()) {

return Collections.emptyList();

}

Field field = getField();

Node monstersNode = field.getObjects(Monster.class);

if (monstersNode == null) {

return Collections.emptyList();

}

List<Monster> list = new ArrayList<>();

List<Spatial> monsterSpatials = monstersNode.getChildren();

Vector3f bottomCenter = geometry.getLocalTranslation();

bottomCenter.setZ(0);

for (Spatial spat : monsterSpatials) {

if (bottomCenter.subtract(spat.getLocalTranslation()).length() <= core.getRange()) {

Entity entity = Getter.getEntity(spat);

if (entity instanceof Monster) {

list.add((Monster) entity);

} else {

throw new IllegalArgumentException(entity.getClass() + " isn't monster!");

}

}

}

return list;

}

@Override

public void destroy() {

super.destroy();

Field field = getField();

if (field != null && core != null) {

field.removeObject(core);

}

}

public float getCooldownTime() {

return (!isEmpty() ? 1f / core.getSpeed() : 0f);

}

public boolean isEmpty() {

return core == null;

}

@Override

public void write(JmeExporter ex) throws IOException {

super.write(ex);

OutputCapsule capsule = ex.getCapsule(this);

capsule.write(core, "core", null);

capsule.write(control, "control", null);

}

@Override

public void read(JmeImporter im) throws IOException {

super.read(im);

InputCapsule capsule = im.getCapsule(this);

core = (Essence) capsule.readSavable("core", null);

control = (BasicControl) capsule.readSavable("control", null);

}

}

Класс Monster

package io.github.sevjet.essensedefence.entity.monster;

import com.jme3.export.InputCapsule;

import com.jme3.export.JmeExporter;

import com.jme3.export.JmeImporter;

import com.jme3.export.OutputCapsule;

import com.jme3.math.FastMath;

import io.github.sevjet.essensedefence.control.BasicControl;

import io.github.sevjet.essensedefence.control.MonsterControl;

import io.github.sevjet.essensedefence.entity.Entity3D;

import io.github.sevjet.essensedefence.field.Field;

import io.github.sevjet.essensedefence.field.MapField;

import io.github.sevjet.essensedefence.gui.ITextual;

import io.github.sevjet.essensedefence.util.BoxSize;

import io.github.sevjet.essensedefence.util.Configuration;

import java.io.IOException;

public class Monster extends Entity3D implements ITextual {

private static final BoxSize SIZE = new BoxSize(1, 1, 1);

private BasicControl control = new MonsterControl();

private float health = 0f;

private float speed = 0f;

private float damage = 0f;

private float exp = 1f;

private float money = 10f;

public Monster() {

super(SIZE);

geometry.addControl(control);

}

public Monster(float health, float speed, float damage) {

super(SIZE);

this.health = health;

this.speed = speed;

this.damage = damage;

geometry.addControl(control);

}

public Monster(int x, int y, float health, float speed, float damage) {

super(x, y, SIZE);

this.health = health;

this.speed = speed;

this.damage = damage;

geometry.addControl(control);

}

public static Monster getDefaultMonster() {

return new Monster(30, 1, 10);

}

public float getHealth() {

return health;

}

public float getSpeed() {

return speed;

}

public void setSpeed(float speed) {

this.speed = speed;

}

public float getDamage() {

return damage;

}

public void setDamage(float damage) {

this.damage = damage;

}

public float getExp() {

return exp;

}

public void setExp(float exp) {

this.exp = exp;

}

public float getMoney() {

return money;

}

public void setMoney(float money) {

this.money = money;

}

@Override

public MapField getField() {

Field field = super.getField();

if (field instanceof MapField) {

return (MapField) field;

}

return null;

}

public double hit(float damage) {

this.health = FastMath.floor((this.health - damage) \* 100f) / 100f;

if (this.health <= 0f) {

giveReward();

die();

}

return this.health;

}

private void giveReward() {

Configuration.getGamer().incGold(money);

}

// TODO: 12/05/2016 change to protected

@Deprecated

public void die() {

Field field = getField();

if (field != null) {

field.removeObject(this);

}

}

@Override

public void write(JmeExporter ex) throws IOException {

super.write(ex);

OutputCapsule capsule = ex.getCapsule(this);

capsule.write(control, "control", null);

capsule.write(health, "health", 0f);

capsule.write(speed, "speed", 0f);

capsule.write(damage, "damage", 0f);

capsule.write(exp, "exp", 1f);

capsule.write(money, "money", 10f);

}

@Override

public void read(JmeImporter im) throws IOException {

super.read(im);

InputCapsule capsule = im.getCapsule(this);

control = (BasicControl) capsule.readSavable("control", null);

health = capsule.readFloat("health", 0f);

speed = capsule.readFloat("speed", 0f);

damage = capsule.readFloat("damage", 0f);

exp = capsule.readFloat("exp", 1f);

money = capsule.readFloat("money", 10f);

}

@Override

public String outputValue() {

return Float.toString(health);

}

@Override

public boolean isEnded() {

return health <= 0 || geometry == null || geometry.getParent() == null || getField() == null;

}

}

Класс Essence

package io.github.sevjet.essensedefence.entity;

import com.jme3.export.InputCapsule;

import com.jme3.export.JmeExporter;

import com.jme3.export.JmeImporter;

import com.jme3.export.OutputCapsule;

import com.jme3.math.ColorRGBA;

import com.jme3.math.FastMath;

import com.jme3.math.Vector3f;

import io.github.sevjet.essensedefence.entity.building.Tower;

import io.github.sevjet.essensedefence.util.BoxSize;

import io.github.sevjet.essensedefence.util.Configuration;

import java.io.IOException;

public class Essence extends Entity3D implements IBuyable {

private static final BoxSize SIZE = new BoxSize(1, 1, 1);

private final float damageK = 1.25f;

private final float rangeK = 1.03f;

private final float speedK = 1.1f;

private float offsetX = 0f;

private float offsetY = 0f;

private float offsetZ = 0f;

private float damage = 0f;

private float range = 0f;

private float speed = 0f;

private int level = 1;

private float price = 0f;

public Essence() {

super(SIZE);

geometry.getMaterial().setColor("Diffuse", ColorRGBA.randomColor());

}

public Essence(float damage, float range, float speed, int level, float price) {

super(SIZE);

this.damage = trim(damage);

this.range = trim(range);

this.speed = trim(speed);

this.level =

level;

this.price = trim(price);

geometry.getMaterial().setColor("Diffuse", ColorRGBA.randomColor());

}

public static Essence buy() {

Essence essence = getNew();

if (Configuration.getGamer().decGold(essence.getPrice())) {

return essence;

}

return null;

}

public static void sell(Tower tower) {

Configuration.getGamer().incGold(tower.extractCore());

}

public static Essence getNew() {

return new Essence(2f, 3f, 1f, 1, 10f);

}

public float sell() {

Configuration.getGamer().incGold(price);

this.geometry.removeFromParent();

return price;

}

public float getDamage() {

return damage;

}

public float getRange() {

return range;

}

public float getSpeed() {

return speed;

}

public int getLevel() {

return level;

}

@Override

public float getPrice() {

return price;

}

public void setPrice(float price) {

this.price = price;

}

public float getOffsetX() {

return offsetX;

}

public void setOffsetX(float offsetX) {

this.offsetX = offsetX;

update();

}

public float getOffsetY() {

return offsetY;

}

public void setOffsetY(float offsetY) {

this.offsetY = offsetY;

update();

}

public float getOffsetZ() {

return offsetZ;

}

public void setOffsetZ(float offsetZ) {

this.offsetZ = offsetZ;

update();

}

@Override

public Vector3f getCenter() {

return super.getCenter().add(offsetX, offsetY, offsetZ);

}

public boolean upgrade() {

if (Configuration.getGamer().decGold(price)) {

level++;

damage = trim(damage \* damageK);

range = trim(range \* rangeK);

speed = trim(speed \* speedK);

price = trim(price \* FastMath.sqrt(price));

return true;

}

return false;

}

public boolean combine(Essence essence) {

float price = (this.price + essence.price) \* 0.5f;

price = FastMath.floor(price \* 100f) / 100f;

if (Configuration.getGamer().decGold(price)) {

int level = this.level + essence.level;

float rightK = this.level \* 1.0f / level;

float leftK = essence.level \* 1.0f / level;

this.damage = trim((this.damage \* leftK + essence.damage \* rightK) \* damageK);

this.range = trim((this.range \* leftK + essence.range \* rightK) \* rangeK);

this.speed = trim((this.speed \* leftK + essence.speed \* rightK) \* speedK);

this.level = level;

this.price = trim((this.price \* leftK + essence.price \* rightK) \* 0.5f);

return true;

}

return false;

}

private float trim(final float num) {

return FastMath.floor(num \* 100f) / 100f;

}

@Override

public void write(JmeExporter ex) throws IOException {

super.write(ex);

OutputCapsule capsule = ex.getCapsule(this);

capsule.write(offsetX, "offsetX", 0f);

capsule.write(offsetY, "offsetY", 0f);

capsule.write(offsetZ, "offsetZ", 0f);

capsule.write(damage, "damage", 0f);

capsule.write(range, "range", 0f);

capsule.write(speed, "speed", 0f);

capsule.write(level, "level", 1);

capsule.write(price, "price", 0f);

}

@Override

public void read(JmeImporter im) throws IOException {

super.read(im);

InputCapsule capsule = im.getCapsule(this);

offsetX = capsule.readFloat("offsetX", 0f);

offsetY = capsule.readFloat("offsetY", 0f);

offsetZ = capsule.readFloat("offsetZ", 0f);

damage = capsule.readFloat("damage", 0f);

range = capsule.readFloat("range", 0f);

speed = capsule.readFloat("speed", 0f);

level = capsule.readInt("level", 1);

price = capsule.readFloat("price", 0f);

}

public Essence clone() {

Essence essence = new Essence(damage, range, speed, level, price);

essence.setGeometry(geometry.clone());

return essence;

}

public String getInfo() {

return "Level: " + getLevel() + '\n' +

"Damage: " + getDamage() + '\n' +

"Range: " + getRange() + '\n' +

"Speed: " + getSpeed() + '\n' +

"Price (buy): " + getPrice();

}

}

Класс MonsterControl

package io.github.sevjet.essensedefence.control;

import com.jme3.cinematic.MotionPath;

import com.jme3.cinematic.events.MotionEvent;

import com.jme3.export.InputCapsule;

import com.jme3.export.JmeExporter;

import com.jme3.export.JmeImporter;

import com.jme3.export.OutputCapsule;

import com.jme3.math.Spline;

import com.jme3.math.Vector3f;

import com.jme3.scene.Spatial;

import io.github.sevjet.essensedefence.entity.building.Fortress;

import io.github.sevjet.essensedefence.entity.monster.Monster;

import io.github.sevjet.essensedefence.field.MapCell;

import io.github.sevjet.essensedefence.field.MapField;

import io.github.sevjet.essensedefence.util.PathBuilder;

import java.io.IOException;

public class MonsterControl extends BasicControl {

private Monster monster = null;

private Fortress fortress = null;

private MotionPath path = null;

private MotionEvent event = null;

private float delay = 0f;

private long lastUpdated = System.currentTimeMillis();

@Override

public void setSpatial(Spatial spatial) {

super.setSpatial(spatial);

if (entity instanceof Monster) {

monster = (Monster) entity;

} else {

throw new IllegalArgumentException("Not a Monster spatial");

}

}

@Override

protected void controlUpdate(float tpf) {

delay -= tpf;

if (delay > 0f) {

return;

}

delay = 0f;

if (path == null || fortress == null || fortress.isEnded() || monster.getField().wasUpdated(lastUpdated)) {

if (event != null) {

event.stop();

spatial.removeControl(MotionEvent.class);

event = null;

}

initPath();

delay += 2f;

lastUpdated = System.currentTimeMillis();

}

if (path == null) {

delay += 10f;

}

}

private void initPath() {

buildPath();

if (path == null) {

delay += 5f;

return;

}

path.addListener((motionControl, wayPointIndex) -> {

if (motionControl.getPath().getNbWayPoints() == wayPointIndex + 1) {

fortress.hit(monster.getDamage());

monster.die();

spatial.removeControl(this);

}

Vector3f curWayPoint = motionControl.getPath().getWayPoint(motionControl.getCurrentWayPoint());

monster.move(Math.round(curWayPoint.getX()), Math.round(curWayPoint.getY()), monster.getZ(), false);

});

path.setPathSplineType(Spline.SplineType.Linear);

event = new MotionEvent(spatial, path) {

@Override

public void update(float tpf) {

super.update(tpf);

}

};

event.setInitialDuration(path.getNbWayPoints());

event.setSpeed(monster.getSpeed());

event.setDirectionType(MotionEvent.Direction.PathAndRotation);

event.play();

}

private void buildPath() {

MapField field = monster.getField();

path = PathBuilder.atField(field)

.from(monster.getX(), monster.getY())

.floatingAt(monster.getPhysicalCenter())

.to(Fortress.class)

.build();

if (path != null && path.getNbWayPoints() > 0) {

Vector3f point = path.getWayPoint(path.getNbWayPoints() - 1);

MapCell cell = field.getCell(Math.round(point.getX()), Math.round(point.getY()));

if (cell.hasContent() && cell.getContent() instanceof Fortress) {

fortress = (Fortress) cell.getContent();

}

}

}

@Override

public void write(JmeExporter ex) throws IOException {

super.write(ex);

OutputCapsule capsule = ex.getCapsule(this);

capsule.write(monster, "monster", null);

capsule.write(fortress, "fortress", null);

capsule.write(path, "path", null);

capsule.write(event, "event", null);

capsule.write(delay, "delay", 0f);

}

@Override

public void read(JmeImporter im) throws IOException {

super.read(im);

InputCapsule capsule = im.getCapsule(this);

monster = (Monster) capsule.readSavable("monster", null);

fortress = (Fortress) capsule.readSavable("fortress", null);

path = (MotionPath) capsule.readSavable("path", null);

event = (MotionEvent) capsule.readSavable("event", null);

delay = capsule.readFloat("delay", 0f);

}

}

Класс PathBuilder

package io.github.sevjet.essensedefence.util;

import com.jme3.cinematic.MotionPath;

import com.jme3.math.Vector3f;

import com.jme3.scene.Node;

import com.jme3.scene.Spatial;

import io.github.sevjet.essensedefence.entity.Entity;

import io.github.sevjet.essensedefence.entity.building.Building;

import io.github.sevjet.essensedefence.entity.building.Fortress;

import io.github.sevjet.essensedefence.entity.building.Portal;

import io.github.sevjet.essensedefence.field.MapCell;

import io.github.sevjet.essensedefence.field.MapField;

import java.awt.\*;

import java.util.\*;

import java.util.Queue;

public class PathBuilder {

private MapField field = null;

private Point start = null;

private Vector3f base = Vector3f.ZERO;

private Set<Class<? extends Entity>> startClasses = new LinkedHashSet<>();

private Class<? extends Building> finishClass = Fortress.class;

private Class<? extends Building> buildingClass = null;

private Point buildingAt = null;

private BoxSize buildingSize = null;

private char[][] previous;

private PathBuilder(final MapField field) {

if (field == null) {

throw new IllegalArgumentException("Field can not be null");

}

this.field = field;

}

public static PathBuilder atField(final MapField field) {

return new PathBuilder(field);

}

public PathBuilder from(final Set<Class<? extends Entity>> startClasses) {

if (startClasses.size() == 0) {

throw new IllegalArgumentException("Start Entities Classes list can not be empty");

}

this.startClasses.clear();

this.startClasses.addAll(startClasses);

return this;

}

public PathBuilder from(Class<? extends Entity> startClass) {

startClasses.clear();

return andFrom(startClass);

}

public PathBuilder andFrom(Class<? extends Entity> startClass) {

if (startClass == null) {

throw new IllegalArgumentException("Finish Building Class can not be null");

}

startClasses.add(startClass);

return this;

}

public PathBuilder from(final int x, final int y) {

final Point point = new Point(x, y);

if (!isInMap(point)) {

throw new IllegalArgumentException("'From' coordinates is out of the field");

}

this.start = point;

return this;

}

public PathBuilder floatingAt(final Vector3f base) {

this.base = base;

return this;

}

public PathBuilder to(final Class<? extends Building> finishClass) {

if (finishClass == null) {

throw new IllegalArgumentException("Finish Building Class can not be null");

}

this.finishClass = finishClass;

return this;

}

public PathBuilder withBuilding(final Class<? extends Building> buildingClass) {

if (buildingClass == null) {

throw new IllegalArgumentException("Building Class can not be null");

}

this.buildingClass = buildingClass;

return this;

}

public PathBuilder atPoint(final int x, final int y) {

final Point point = new Point(x, y);

if (!isInMap(point)) {

throw new IllegalArgumentException("Building 'at' Point coordinates is out of the field");

}

this.buildingAt = point;

return this;

}

public PathBuilder withSize(BoxSize size) {

if (size == null) {

throw new IllegalArgumentException("Building Size can not be null");

}

this.buildingSize = size;

return this;

}

public boolean isValid() {

if (field == null ||

startClasses.size() == 0 ||

finishClass == null ||

(buildingClass != null &&

(buildingAt == null || buildingSize == null))) {

throw new IllegalStateException("PathBuilder is not initialized for isValid() yet");

}

MapCell cell;

Queue<Point> queue = new LinkedList<>();

previous = new char[field.getRows()][];

for (int i = 0; i < field.getRows(); i++) {

previous[i] = new char[field.getCols()];

for (int j = 0; j < field.getCols(); j++) {

previous[i][j] = ' ';

cell = field.getCell(i, j);

if (cell.hasContent() && finishClass.isInstance(cell.getContent())) {

queue.add(new Point(i, j));

previous[i][j] = '•';

}

}

}

Point current;

while (queue.size() > 0) {

current = queue.poll();

queueNearest(queue, new Point(current.x + 1, current.y), '↑');

queueNearest(queue, new Point(current.x - 1, current.y), '↓');

queueNearest(queue, new Point(current.x, current.y + 1), '←');

queueNearest(queue, new Point(current.x, current.y - 1), '→');

}

Node currentNode;

Entity entity;

for (Class<? extends Entity> clazz : startClasses) {

currentNode = field.getObjects(clazz);

if (currentNode != null) {

for (Spatial spatial : currentNode.getChildren()) {

entity = spatial.getUserData("entity");

if (entity != null) {

char prev = previous[entity.getX()][entity.getY()];

previous[entity.getX()][entity.getY()] = '+';

if (prev == ' ') {

previous[entity.getX()][entity.getY()] = '-';

return false;

}

} else {

throw new IllegalStateException("Spatial in Node is not belongs to Entity " + clazz.getName());

}

}

}

}

return true;

}

public MotionPath build() {

if (field == null ||

start == null ||

finishClass == null) {

throw new IllegalStateException("PathBuilder is not initialized for build() yet");

}

previous = new char[field.getRows()][];

for (int i = 0; i < field.getRows(); i++) {

previous[i] = new char[field.getCols()];

for (int j = 0; j < field.getCols(); j++) {

previous[i][j] = ' ';

}

}

ArrayList<Point> list = new ArrayList<>();

Queue<Point> queue = new LinkedList<>();

queue.add(start);

previous[start.x][start.y] = '•';

Point current;

Point finish = null;

while (queue.size() > 0) {

current = queue.poll();

if (isFinish(current)) {

finish = current;

queue.clear();

} else {

queueNearest(queue, new Point(current.x + 1, current.y), '↑');

queueNearest(queue, new Point(current.x - 1, current.y), '↓');

queueNearest(queue, new Point(current.x, current.y + 1), '←');

queueNearest(queue, new Point(current.x, current.y - 1), '→');

}

}

if (finish != null) {

current = new Point(finish);

while (previous[current.x][current.y] != '•') {

list.add(new Point(current));

switch (previous[current.x][current.y]) {

case '↑':

current.translate(-1, 0);

break;

case '↓':

current.translate(1, 0);

break;

case '←':

current.translate(0, -1);

break;

case '→':

current.translate(0, 1);

break;

}

}

Collections.reverse(list);

MotionPath path = new MotionPath();

path.addWayPoint(base);

for (Point point : list) {

path.addWayPoint(new Vector3f(point.x, point.y, base.getZ()));

}

return path;

}

return null;

}

private void queueNearest(final Queue<Point> queue, final Point next, final char dir) {

if (isInMap(next) && isPassable(next) && previous[next.x][next.y] == ' ') {

queue.add(next);

previous[next.x][next.y] = dir;

}

}

private boolean isInMap(final Point point) {

return 0 <= point.x && point.x < field.getRows() &&

0 <= point.y && point.y < field.getCols();

}

private boolean isPassable(final Point point) {

if (buildingClass != null &&

(!buildingClass.isAssignableFrom(Portal.class) &&

!buildingClass.isAssignableFrom(Fortress.class))) {

Point a = new Point(buildingAt);

Point b = new Point(a.x + buildingSize.getHeight(), a.y + buildingSize.getWidth());

if (a.x <= point.x && point.x < b.x && a.y <= point.y && point.y < b.y) {

return false;

}

}

return field.getCell(point.x, point.y).isPassable();

}

private boolean isFinish(final Point point) {

return field.getCell(point.x, point.y).contains(finishClass);

}

}