Министерство образования и науки

Севастопольский университет

Кафедра

информационных систем

КУРСОВАЯ РАБОТА ПО ДИСЦИПЛИНЕ

«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ »

ПРОГРАММА «Tower Defense»

Пояснительная записка

Листов 90

ПРОВЕРИЛ

Ассистент кафедры ИС

( ) А.К. Забаштанский

« » \_\_\_\_\_\_\_\_\_2014

РАЗРАБОТАЛ

Студент группы И-22д

( ) В.И. Петровский

« » \_\_\_\_\_\_\_\_\_2014

Севастополь

2014

АННОТАЦИЯ

В данной пояснительной записке рассматривается описание игрового продукта жанра Tower Defense – проектное решение и программная реализация, а также описание интерфейса пользователя и критерии качества продукта.

СОДЕРЖАНИЕ

*Изм.*

*Лист*

*№ Документа*

*Подпись*

*Дата*

*Лист*

3

КУРСОВАЯ РАБОТА

*Разработал*

*Петровский В.И.*

*Руковод.*

*Забаштанский А.К.*

*Т.контр.*

*Н.контр.*

*Утв*

Tower Defense

*Лит.*

*Листов*

90

гр. И-22д

Аннотация….………………………………………..……………………….…… 2

Введение…………………………………….…………………………………..… 4

1 Постановка задачи

* 1. Цель разработки………………………………………………………...5
  2. Описательная постановка задачи……………………………………...5

2 Проектное решение

2.1 Абстрагирование и выделение объектов...….…….…………………. 7

2.2 Построение иерархии классов………….………………..…………… 8

2.3 Построение информационной модели……………………………….12

2.4 Описание жизненного цикла программы и объектов……………….14

2.5 Диаграмма потоков данных и действий……………………………..16

2.6 Ограничения проектного решения…………………………………...17

3 Программная реализация

3.1 Обоснование выбора языка программирования……………………18

3.2 Описание реализации основных классов и их методов……………19

3.2.1 Класс «Существо»…………………………………………...19

3.2.2 Класс «Башня»……………………………………………….22

3.2.3 Класс «Карта»………………………………………………..24

3.2.4 Класс «Статистика игрока»…………………………………25

3.2.5 Класс «Пункт меню»………………………………………...26

3.2.6 Класс «Игровое меню»……………………………………...27

3.2.7 Класс «Снаряд»……………………………………………...28

3.2.8 Класс «Маг»…………………………………………………30

3.3 Интерфейс пользователя…………………………………………….32

3.4 Критерии качества разработанной программы..…………………...40

3.5 Условия выполнения программы……………..…………………….41

Заключение.……………………………………………………………………..42

Список литературы……….…………………………………………………….43

Приложение А………………………………………………....………………..44

ВВЕДЕНИЕ

В рамках данного курсового проектирования ведется разработка программного продукта под названием «Tower Defense» в соответствии с техническим заданием, утверждённым Севастопольским национальным техническим университетом.

Разрабатываемый программный продукт является продуктом игрового характера. Разработка продукта велась с использованием объектно-ориентированной методологии, актуальность которой заключается как в том, что в настоящее время количество прикладных языков программирования, реализующих объектно-ориентированную парадигму, является наибольшим по отношению к другим парадигмам, так и в том, что на сегодняшний момент данный подход является одним из передовых в IT-индустрии.

Актуальность разработки игрового программного обеспечения связана с тем, что в течение последнего года, показатель покупок игровых приложений вырос в разы. Эти данные постоянно увеличиваются, и в настоящее время статистика не меняется.

Целью данного курсового проектирования является закрепление навыков разработки объектно-ориентированных программ, получение опыта работы с большим проектом, содержащим множество классов и сложную объектную иерархию, а также получение навыков объектного анализа.

В разделе «Проектное решение» описывается ход проектирования объектной иерархии, информационной модели и жизненного цикла программы, и объектов. В разделе «Программная реализация» содержится описание особенностей программной реализации разрабатываемого продукта, описание интерфейса и критерии качества данного программного продукта.

1. ПОСТАНОВКА ЗАДАЧИ
   1. Цель разработки

Разработать игровую систему на базе объектного подхода, получить навыки разработки объектно-ориентированных проектов.

* 1. Описательная постановка задачи

В системе моделируется оборона боевых позиций от сил врага. Действия происходят в атмосфере фентези.

Силы врага представляют собой юниты(живые существа), идущие по дороге. Их цель – добраться до точки назначения. Юнит обладает набором характеристик: скорость, здоровье и цена.

Игрок управляет застройкой боевых позиций. Он должен строить башни. Башни ведут огонь по юнитам. Юниты появляются в специальной точке, которая называется «Спавн», волнами. Каждая волна содержит 10-50 существ одного типа. Финальная волна содержит разнородных существ, и, выстояв её, игрок одерживает победу.

Должно быть предусмотрено три вида обычных башен: стрелковая, ядовитая, парализующая; а также три вида магических башен – ледяной маг, огненный маг, и маг молний.

Стрелковая башня стреляет по идущим юнитам обычными стрелами не используя стрельбу на упреждение. Обладает высокой скоростью атаки. Ядовитая башня стреляет ядовитыми стрелами, которые имеют более высокий показатель урона и замедляют противника на некоторое время. Парализующая башня не наносит урон, но парализует существо противника, заставляя его стоять на месте некоторое время.

Магические башни представляют собой магов трёх стихий – огня, воздуха, воды. Водяной маг стреляет ледяными болтами, которые при столкновении навсегда изменяют скорость противника на минимальную, кроме того наносят значительный урон. Воздушный маг стреляет молниями по всем существам на карте (имеет неограниченный радиус атаки), имеет самый сильный урон в игре, но скорость стрельбы медленная. Огненный маг стреляет огненными шарами, наносящими не очень сильный урон противнику, но это компенсируется его высокой скоростью стрельбы.

Юниты подразделяются на несколько типов, каждый из которых обладает уникальными способностями.

За убийство юнитов игрок получает золото, которое может тратить на постройку новых башен.

Игрок имеет 10 жизней и за каждое существо, дошедшее до пункта назначения игрок теряет одну жизнь.

Также должно быть реализовано главное меню и возможность поставить игру на паузу. Игровое меню должно содержать краткую статистику состояния игрового процесса (количество золота, количество убитых юнитов, число оставшихся жизней), а также кнопки постройки новых башен и кнопку готовности к новой волне сил врага.

Входными данными является ввод с клавиатуры и мыши – размещение башен на карте и выбор пунктов меню, нажатие кнопок. Выходные данные выводятся на экран в игровое окно.

1. ПРОЕКТНОЕ РЕШЕНИЕ
   1. Абстрагирование и выделение объектов

Для выполнения поставленной задачи, необходимо выполнить задачу абстрагирования и выделения объектов.

Очевидно, что для реализации главного меню потребуются следующие классы: меню, и пункт меню, причем меню должно содержать пункты меню в виде полей. Пункт меню в свою очередь содержит в качестве полей изображение и переменные, показывающие активен ли пункт меню.

Для реализации игрового меню понадобится схожий набор классов – класс «игровое меню» и класс «кнопка». Игровое меню должно выполнять функцию вывода информации о пользователе на экран, кроме того оно должно быть независимой от игрового поля частью программы – на нём нельзя рисовать объекты, относящиеся к игровой карте. Также в игровом меню должны находиться кнопки, обеспечивающие влияние на игровой процесс – кнопки постройки башен и кнопка подтверждения готовности к новой волне противников. Кнопки, так же, как и пункты меню, содержат спрайт, а также они взаимодействуют с устройствами ввода-вывода что бы обеспечить подсветку в случае наведения на них мыши, а так же реакцию на нажатие мыши.

Для реализации игрового поля было решено выделить базовые классы «башня», «существо», «снаряд», и «тайл». Каждый из этих классов содержит потомков в виде конкретных типов объектов, например существом являются одновременно и рыцарь смерти, и вор. Для хранения перечисленных объектов был выделен класс «карта», который отвечает за их прорисовку, хранение, а также взаимодействие.

* 1. Построение иерархии классов

Принятая иерархия классов содержит следующие базовые классы: башня, существо, снаряд, спаун, пункт назначения, кнопка, карта игры, пункт меню, а также тайл.

Башня представляет собой сооружение, ведущее непрерывный огонь по юнитам, которые находятся в области её действия. Башни различается по типу поражаемых юнитов и типу атаки. Башня может быть магической. Магические башни атакуют противников заклинаниями. Некоторые типы башен порождают снаряды, которые накладывают различные негативные эффекты на некоторые типы существ. Иерархия башен представлена на рисунке 2.1.



Рисунок 2.1 – Иерархия башен

Снаряды в разрабатываемой игре делятся на три базовых типа: стрела, призрак, магия. Стрела может быть простой и отравленной. Стрела летит в то место, где было существо во время её создания . Ядовитая стрела помимо наносимого урона замедляет существо на 30%.

Призрак – особый тип снаряда, который порождается башней «Монумент». Призрак преследует цель до тех пор пока не настигнет её, после этого он парализует цель на некоторое время и уничтожается.

Магия реализована в трёх видах. Огненный шар помимо наносимого урона вызывает у существ эффект горения. Горящие существа получают урон с течением времени. Ледяной шар навсегда замедляет существо придавая ему минимальную скорость – 1 по каждой из проекций, а также наносит значительный урон. Молния наносит самый высокий урон в игре. Все типы магии гарантировано попадают по противнику, в отличии от стрел. Иерархия снарядов представлена на рисунке 2.2.



Рисунок 2.2 – Иерархия снарядов

Существа делятся на семь типов – пеон, орк-воин, рыцарь хаоса, гоблин, вор, скелет, и демон. Каждое из существ обладает набором уникальных характеристик. Иерархия существ представлена на рисунке 2.3.



Рисунок 2.3 – Иерархия существ

Для обеспечения пользовательского интерфейса, были также реализованы иерархии кнопок и меню, которые обеспечивают взаимодействие с пользователем. Как видно из рисунка 2.4, обособленными остались классы «Карта игры», «Пункт меню», «Спавн», а также «Пункт назначения» т.к. эти классы являются уникальными.



Рисунок 2.4 – Иерархия вспомогательных объектов

* 1. Построение информационной модели

На рисунке ниже представлена информационная модель программы.

Рисунок 2.5 – Графическое представление информационной модели программы

Между картой и существом, картой и снарядом, картой и башней существуют связи «один ко многим». Эта связь говорит о том, что множество объектов классов «Башня», «Снаряд» и «Существо» могут находиться лишь на одной карте, в то время как карта может содержать множество объектов названных классов.

Связь между башней и снарядом «Один ко многим» говорит о том, что башня может создавать большое число снарядов в единицу времени, в то время как у одного конкретного снаряда может быть лишь одна «башня-родитель».

Связь между башней и существом определена как «Многие ко многим». Это говорит о том, что множество объектов типа «Башня» соотносится с множеством объектов типа «Существо». Заранее неизвестно, какая именно башня выберет целью какое именно существо, ровно, как и неизвестно заранее какое существо окажется целью какой башни.

Связь между существом и снарядом «многие ко многим» значит, что многие существа могут взаимодействовать с многими типами снарядов. Неизвестно, с каким существом столкнётся снаряд на своём пути т.к. снаряд может лететь к одной цели, а столкнуться с совершенно другим существом, вставшем у снаряда на пути.

* 1. Описание жизненных циклов программы и объекта

На рисунке ниже изображена диаграмма жизненного цикла программы.



Рисунок 2.6 – Графическое изображение жизненного цикла программы

В жизненном цикле программы существует пять состояний

S0 – Главное меню

S1 – Собственно игра

S2 – Пауза

S3 – Поражение

S4 – Победа

Система переходов состояний обусловлена следующим: после победы/поражения игрок возвращается в главное меню, где он может начать игру сначала. После паузы, соответственно, может продолжить игру, либо же снова нажать на паузу, победить/проиграть.

На рисунке ниже изображена диаграмма жизненного цикла объекта «Существо»

Рисунок 2.7 – Графическое изображение жизненного цикла объекта «Существо»

W0 – Передвижение

W1 – Проверка направления

W2 – Получение урона

W3 – Смерть

W4 – Отравление

W5 – Парализация

W6 - Замерзание

Наличие ветвления от состояния W2 обусловлено тем, что получить негативный эффект существо может лишь после того как оно получило урон.

* 1. Диаграмма потоков данных и действий

Диаграмма потоков данных и действий для состояния W2 объекта «Существо» изображена на рисунке ниже.



Рисунок 2.8 – Диаграмма потока данных и действий

* 1. Ограничения проектного решения

Для обеспечения удобства игрового процесса, а также в связи с особенностями реализации программы, на игровом поле могут располагаться не более 40 объектов типа «Существо», не более 300 объектов типа «Тайл», а также не более 50 объектов типа «Башня». Размер игрового поля ограничен прямоугольником 600х600 пикселей, размер игрового меню – 600х200 пикселей. Размеры окна программы: 800х600.

Кроме того, окно программы имеет интерфейс, ограниченный операционной системой.

1. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ
   1. Обоснование выбора языка программирования

Выбирая языком программирования Java и используемой интегрированной средой разработки Eclipse, я руководствовался в первую очередь тем, что Java является для меня новым языком программирования и я посчитал целесообразным закрепить свои познания в нём. Кроме того, язык Java является широко распространенным языком программирования высокого уровня, что обеспечивает сравнительную простоту написания программ. Также, выбранный язык является объектно-ориентированным, что позволяет без труда запрограммировать разработанную объектную структуру. Также, т.к. все методы в языке Java являются виртуальными, полиморфизм в разрабатываемой программе реализован на уровне языка.

Кроме того, для языка программирования Java разработано множество библиотек с большим количеством функций, что позволяет ускорить процесс разработки и уменьшить объем текста программы.

Помимо того, среда Eclipse поддерживает работу с графическими приложениями и имеет большое количество дополнений. Поэтому, в условиях поставленной задачи, целесообразным является использование выбранных языка программирования и среды разработки.

* 1. Описание реализации основных классов и их методов

3.2.1 Класс «Существо»

Класс является базовым классом для всех существ, и поэтому содержит реализацию их общего поведения.

Данные поля хранят координаты, абсолютную и текущую скорость передвижения существа по карте.

protected float x = 0;

protected float y = 0;

protected Vector2f speed, nowSpeed;

Направление движения хранится в поле класса Vector2f, который предназначен для хранения векторов.

protected Vector2f direction;

Анимация представлена в виде спрайта. Спрайт хранится в виде совокупности анимаций – анимация движения вверх, вниз, вправо, и влево. Текущая анимация хранится в переменной sprite. Во время смены направления движения полю sprite присваивается одно из значений – up, down, right, left.

protected Animation up;

protected Animation down;

protected Animation right;

protected Animation left;

protected Animation sprite;

В полях, описанных ниже хранится статус объекта – мертв, отравлен, парализован, заморожен.

private boolean isDead = false;

private boolean isParalysed = false;

private boolean isPoisoned = false;

private boolean isColded = false;

Счетчик posionCounter считает время отравления.

protected Integer poisonCounter;

Цветовой фильтр для прорисовки существа. Цвет меняется в случае если существо отравлено или заморожено.

protected Color color;

Карта необходима существу для того что бы обращаться к тайлу, находящемуся под ним и расчета направления движения.

Map gMap;

Очки жизни существа. Когда они заканчиваются, существо погибает.

float hp;

Конструктор. Создаёт и инициализирует поля объекта. В нём происходит загрузка спрайтов и начальная инициализация полей.

public Creature(float \_x, float \_y, Map map);

Методы служат для изменения состояния существа. Существа, невосприимчивые к конкретным негативным эффектам переопределяют эти методы. Таким образом реализована инкапсуляция полей, хранящих статус объекта.

public void setColded(boolean val);

public boolean isColded();

public void setPoisoned(boolean val);

public boolean getPoisonState();

public void setParalyze(boolean val);

public boolean getParalyzeState();

public boolean isDead();

Методы возвращают текущие координаты существа.

public float getX();

public float getY();

Параметр данного метода отнимается от значения hp. Если в результате здоровье станет меньшим либо равным нулю, статус существа меняется на «мёртвое».

public void damage(float dmg);

Вызов метода убивает существо. Метод является абстрактным т.к. каждое существо умирает по-разному.

abstract public void kill();

Метод служит для обновления состояния существа.

public void update(int delta);

Если существо дошло до пункта назначения, вызывается данный метод. Каждое существо выполняет различные действия по достижении пункта назначения. Поэтому метод описан как абстрактный.

abstract void survive();

Данный метод отвечает за прорисовку существа на карте. Карта вызывает данный метод для всех существ, которые на ней находятся.

public void render();

Метод служит для обновления направления движения существа. В нём также реализована корректировка положения существа.

protected void updateDirection();

Данный класс является базовым для всех типов существ. Например, описание класса орк-воин выглядит следующим образом:

public class Grunt extends Creature

{

…

@Override

public void kill()

{

isDead = true;

PlayerStats.addGold(150);

PlayerStats.addKilledUnits(1);

}

}

У каждого существа своя реализация метода kill. Это является примером полиморфизма. Все существа могут умирать, но каждое из них делает это по-разному.

3.2.2 Класс «Башня»

Класс «Башня» является базовым классом для всех видов башен.

Данные поля хранят координаты башни, её высоту, ширину, радиус стрельбы, скорость стрельбы.

float x, y;

int height, width;

int range; //расстояние стрельбы

int fireRate; //частота стрельбы

Счетчик отсчитывает время стрельбы. Декрементируется в цикле (отдельный поток). Когда становится равен нулю, башня создаёт новый снаряд, а счетчик становится равным значению поля fireRate.

int fireCounter;

Спрайт башни. Хранит анимацию башни.

Animation sprite; //спрайт

Хранит карту, на которой расположена башня. Необходимо для поиска цели.

Map gMap; //указатель на карту

Цель. Необходимо для расчета траектории полёта снаряда.

Creature target; //цель

Хранит цену башни

int cost; //цена башни в монетах

Хранит цвет, в котором необходимо прорисовывать башню

//фильтр рендеринга

Color color;

Конструктор класса «Башня». В нём происходит инициализация цвета. Остальные поля инициализируется в конструкторах верхнего уровня.

public Tower();

Данные методы реализуют инкапсуляцию полей класса.

public int getHeight();

public int getWidth();

public float getX();

public float getY();

public int getCost();

public void setColor(Color nc);

public void setMap(Map \_gMap);

Метод служит для поиска цели на карте. В нём происходит обращение к методу getCreature карты, который возвращает указатель на найденное существо в указанном радиусе. Радиусом является поле range.

protected void findTarget();

Метод служит для перемещения башни. Непосредственно во время игрового процесса башни не двигаются. Метод нужен для размещения новых башен.

public void move(int \_x, int \_y);

Метод реализует стрельбу. Т.к. каждая башня стреляет по разному принципу, данный метод описан как абстракттный.

public abstract void fire();

Функции используются для рендеринга и обновления состояния башен. Вызываются картой в бесконечном цикле после создания объекта и добавления его на карту.

public void render();

public void update();

3.2.3 Класс «Карта»

Данный класс хранит предметы классов «тайл», «существо», «снаряд», «башня», «пункт назначения» в отдельных списках. Также он выполняет обработку их состояний и отвечает за их взаимодействие.

Поле хранит палитру тайлов. Используется для выборки изображений для тайлов.

SpriteSheet tileSheet;

Все объекты, расположенные на карте хранятся в соответствующих списках.

private LinkedList<Tile> tiles;

private LinkedList<DirectionTile> directionTiles;

private LinkedList<Tower> towers;

private List<Creature> creatures;

private LinkedList<Missile> missiles;

private Destination dest;

Конструктор. В качестве параметров принимает список тайлов и тайлов направления. Конструирует карту из «блоков».

public Map(LinkedList<Tile> tls, LinkedList<DirectionTile> dts);

Очищает карту от всех предметов. Используется для освобождения ресурсов.

public void clear();

Методы ниже служат для добавления на карту снаряда, существа, а также башни и тайла.

public void setDestination(Destination dst);

public void addMissile(Missile m);

public void addTower(Tower t);

public void addCreature(Creature c);

public void addTile(Tile t);

public void addDirectionTile(DirectionTile dt);

Метод возвращает true если прямоугольник заданной ширины и высоты пересекается с каким-либо объектом на карте. Нужно для размещения новых башен на карте.

public boolean isCollide(int x, int y, int h, int w);

Возвращает существо, находящееся в заданном радиусе окружности с центром в точке x,y.

public Creature getCreature(float x, float y, float r);

Возвращает незамороженное существо по схожему с прошлым методом принципу.

public Creature getNotColdedCreature(float x, float y, float r);

Возвращает true если в окружности заданного радиуса находится существо.

public boolean isCreatureInRadius(Creature c, float x, float y, float r);

Возвращает тайл направления, который находится по координатам, переданным в качестве параметров. Используется существами для определения направления движения.

public DirectionTile getTile(float x, float y);

Данные две функции выполняют обновление рендеринг всех объектов, находящихся на карте. Сначала рендерятся тайлы, потом башни, затем существа, а после – снаряды.

public void update(int delta);

public void render(Graphics g);

3.2.4 Класс «Статистика игрока»

Данный класс хранит общую информацию о достижениях игрока.

Все поля и методы данного класса являются статическими, как, собственно, и сам класс.

Убитые существа

static private int killedUnits = 0;

Золото

static private int gold = 100;

Существ осталось в волне

static private int unitsLeft = 0;

Количество жизней

static private int lives = 10;

Добавляется ли в данный момент башня

public static boolean isTowerPlacing = false;

Добавляемая башня

static private Tower addTower;

Если true, то игрок готов к новой волне

static private boolean readyForNewWave;

Очищает класс

static public void clear()

Инкапсулирует готовность игрока к новой волне

static public void setReady(boolean value)

static public boolean isReady()

Забирает золото у игрока

static public void reduceGold(int value)

Инкапсулирует добавляемую башню

static public void addTower(Tower tower)

static public Tower getAddTower()

Возвращает количество золота

static public int getGold()

Добавляет золото игроку

static public void addGold(int value)

Возвращает и добавляет убитых существ на счет игрока

static public int getKilledUnits()

static public void addKilledUnits(int value)

Забирает жизнь

static public void denyLife(int value)

Возвращает число оставшихся жизней

static public int livesLeft()

3.2.5 Класс «Пункт меню»

Данный класс реализует пункт главного меню.

Данное поле хранит изображение (внешний вид пункта меню).

Image item

Координаты пункта меню на экране

Integer x;

Integer y;

Активен ли пункт меню в данный момент.

Boolean isActive;

Конструктор пункта меню. Выполняет инициализацию всех полей с помощью переданных параметров. Параметр path хранит путь к изображению.

public MenuItem(String path, Integer \_x, Integer \_y, Boolean \_isActive)

Выполняет прорисовку пункта меню в окне программы

public void render(Graphics g)

Активирует либо деактивирует пункт меню (в зависимости от переданного параметра).

public void setActive(Boolean active)

2.3.6 Класс «Игровое меню»

Данный класс отвечает за вывод пользовательской информации на экран и за взаимодействие с пользователем. Также класс размещает на экране кнопки постройки башен и кнопку готовности.

Фоновое изображение игрового меню

private Image bg;

Начальная позиция меню на экране

private final int beginXPos = 0;

private final int beginYPos = 480;

Кнопки добавления башен на карту

GameMenuButton wtButton, rtButton, otButton, lmButton, fmButton, frmButton;

Кнопка подтверждения готовности к новой волне существ. Устанавливает значение true поля isReady класса «Игровая статистика»

NewWaveButton nwButton;

Конструктор. Выполняет загрузку ресурсов.

public GameMenu()

Метод отвечает за прорисовку всех элементов меню.

public void render(Graphics g, GameContainer cont)

Возвращают значения координат игрового меню.

public int getX()

public int getY()

Метод обновляет состояния кнопок.

public void update(int delta, GameContainer container)

2.3.7 Класс «Снаряд»

Данный класс является базовым классом для всех типов снарядов.

Текущие координаты снаряда

float x, y;

Скорость движения снаряда по карте

Vector2f speed;

Направление движения по каждой из проекций.

Vector2f direction;

Урон, наносимый снарядом

float damage;

Анимация снаряда

Animation sprite;

Цель снаряда, в сторону которой он движется.

Creature target;

Карта, по которой движется снаряд. Нужна для проверки столкновения с другими существами.

Map gMap;

True если снаряд может быть удалён с поля

boolean canBeDeleted;

Угол поворота снаряда в радианах относительно положительного направления оси абсцисс.

double angle;

Возвращает значение поля canBeDeleted

public boolean canBeDeleted()

Вычисляет коэффициент скорости движения по каждой из проекций с помощью скалярного произведения вектора, проведенного к координатам цели и орта вдоль оси абсцисс. Центром локальных координат являются текущие координаты снаряда.

public void computeDirection()

Конструктор. Выполняет инициализацию полей объекта данного типа.

public Missile(Creature \_target, Map map, float \_x, float \_y)

Отвечает за обработку столкновения снаряда с существом. Выполняет различные действия в зависимости от типа снаряда. Является случаем полиморфизма.

public abstract void collide();

Методы обновления и прорисовки. В методе обновления выполняется приращение координат, проверка столкновения, расчет направления, декремент скорости в зависимости от пройденного расстояния.

public abstract void update(int delta);

public void render()

3.2.8. Класс «Маг»

Данный класс наследуется от класса «Башня». Содержит ряд уникальных полей и уникальную реализацию методов.

Т.к. маг является живым существом, он должен поворачиваться в сторону объектов, по которым стреляет, и кроме того он должен быть анимированным. Данные поля хранят анимацию.

Animation left, right, top, bot;

Блок инициализации. Такой блок предусмотрен для каждого типа башен. В нём производится инициализация полей объекта.

{

range = 200;

fireRate = 30;

height = 53;

width = 43;

fireCounter = 0;

cost = 1000;

}

Конструктор. В качестве параметров принимает координаты размещения а также строку пути к спрайту.

public Mage(int \_x, int \_y, String spr)

Метод служит для смены анимации в зависимости от того, в какой стороне от мага находится его цель. Расчет ведётся по тому же принципу что и расчет направления движения снаряда. Анимация выбирается следующим образом: если угол находится в промежутке 7П/4 .. П/4, то в качестве спрайта выбирается анимация «право»; если в интервале П/4..3П/4, в качестве спрайта выбирается анимация «вверх» и т.д.

public void computeDirection()

Переопределенные методы обновления и прорисовки объекта. Реализованы особенности обработки анимации.

@Override

public void update()

@Override

public void render()

* 1. Описание интерфейса пользователя

Работа программы начинается с главного меню. Скриншот главного меню представлен на рисунке ниже.



Рисунок 3.1 – Главное меню программы

Главное меню программы содержит три пункта – «Новая игра», «О разработчике» и «Выйти». Переключение между пунктами меню происходит с помощью клавиш «вверх» и «вниз». Выбор пункта меню осуществляется клавишей «Enter». При выборе пункта меню «Новая игра» начинается новая игра. При выборе пункта меню «Выход» происходит выход из программы. Если выбрать пункт меню «О разработчике», на экран будет выведена информация о авторе данной ПЗ и человеке, разрабатывавшем игру (рисунок 3.8).

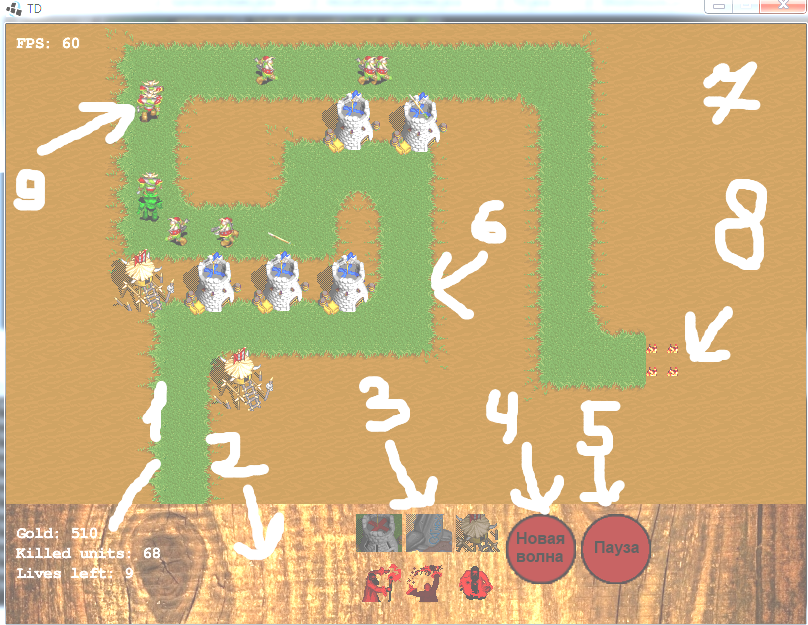


Рисунок 3.2 – Внешний вид окна игрового процесса

Окно игрового процесса изображено на рисунке 3.2. На этом рисунке цифрами изображено:

1. Игровая статистика. При начале игры игроку даётся 200 золота и 10 жизней.
2. Игровое меню. На нём расположены кнопки добавления башен и кнопка «Новая волна», а также игровая статистика.
3. Кнопки добавления башен. Кнопки становятся активными в том случае если на башню хватает золота. Количество золота, необходимое на покупку башни не высвечивается пользователю для усложнения игрового процесса.
4. Кнопка «Новая волна». При нажатии на эту кнопку, игрок подтверждает готовность к новой волне существ
5. Кнопка «Пауза». Приостанавливает игровой процесс до повторного нажатия на эту кнопку
6. Дорога. По дороге существа движутся в сторону пункта назначения.
7. Игровая карта. На ней расположены существа и все игровые объекты.
8. Пункт назначения. Когда существа добираются до него, у игрока отнимаются жизни. Всего игрок имеет десять жизней.
9. Существо. Оно движется к пункту назначения.

Построим башню и нажмём кнопку «Новая волна»



Рисунок 3.3 – Стреляющая башня и идущие существа

По нажатию этой кнопки, начинается шествие существ. За убитых существ даётся золото, которое можно тратить на новые башни. Разумный подход к построению башен гарантирует победу. Построим еще одну башню и продолжим игру.



Рисунок 3.4 – Существа, преодолевающие препятствия

Как видно из рисунка 3.4, неудачное расположение башен привело к катастрофе – существа дошли до пункта назначения. На рисунке 3.4 изображены существа класса «Вор». Если эти существа добираются до пункта назначения, они не только отнимают жизнь, они уносят 100 золотых за каждого пропущенного юнита.

Если у игрока закончатся жизни, то наступит конец игры. Экран конца игры представлен на рисунке 3.5

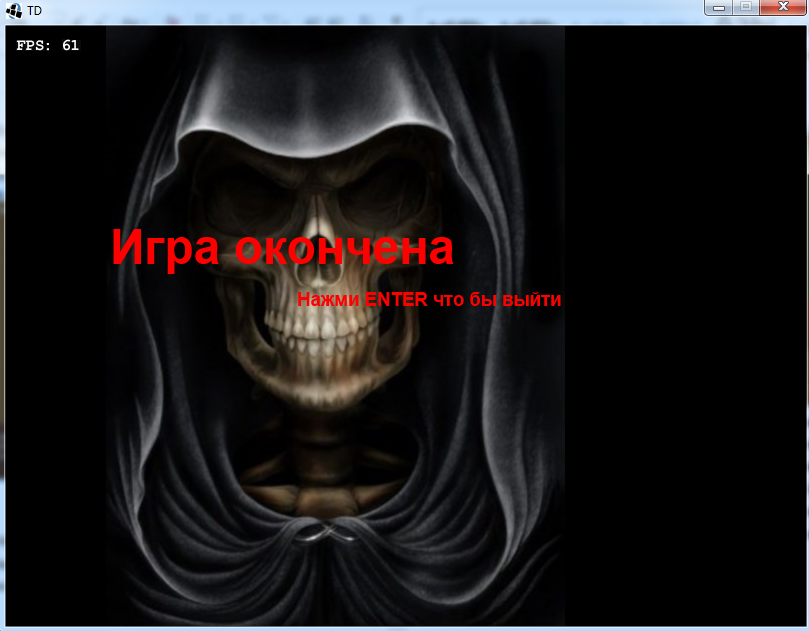


Рисунок 3.5 – Конец игры

После нажатия клавиши Enter, программа возвращается в главное меню.

Игрок одержит победу когда уничтожит все волны существ. При достижении игроком победы, программа переходит в состояние «Победа». Экран победы выглядит соответствующе с рисунком 3.6.



Рисунок 3.6 – Экран победы

При нажатии клавиши Enter, будет совершен переход в главное меню.

Игровой процесс всегда можно приостановить вызвав меню паузы. Для этого в игровое время необходимо нажать клавишу TAB. Меню паузы изображено на рисунке ниже.



Рисунок 3.7 – Меню паузы

В меню паузы игроку доступно два пункта меню: продолжить и вернуться в главное меню. При возвращении в главное меню, вся игровая статистика будет сброшено и игру придётся начинать сначала.

На рисунке ниже изображен экран пункта меню «О разработчике».



Рисунок 3.8 – Пункт главного меню «О разработчике»

3.4 Критерии качества программной системы

Качество программного продукта было оценено по следующим критериям:

* Структурированность. Исходный код имеет хорошую структурированность т.к. при разработке использовалась объектная модель, которая предполагает разбиение программы на классы и объекты.
* Соответствие объектной модели. Разработанная программа соответствует объектной модели т.к. в ней реализованы все принципы ООП – полиморфизм, наследование, инкапсуляция.
* Читаемость кода. Исходный код имеет достаточно хорошую читаемость т.к. все классы проекта содержатся в отдельных файлах и снабжены комментариями.
* Корректная обработка исключительных ситуаций. По данному критерию программу можно считать эффективной, т.к. все участки кода, где могут возникнуть исключительные ситуации, обработаны корректно.
* Эффективность использования ресурсов: памяти и процессорного времени. Т.к. программа разработана на языке Java, которая использует виртуальную машину и сборщик мусора, управление памятью программными средствами не выполнялось. Также использование игрового движка повлекло за собой дополнительные накладные расходы ресурсов. Следовательно, системные ресурсы использовались не слишком эффективно.
* Портируемость. Данное ПО хорошо портируемо к другому окружению т.к. в процессе написания использовался кроссплатформенный язык программирования Java.
* Надёжность. В ходе разработки и тестирования, не было замечено сбоев в работе движка, а все недоработки программной реализации были исправлены в ходе тестирования. Следовательно, разработанная программа достаточно надёжна.
* Тестируемость. Программа имеет встроенную проверку производительности, обеспеченную игровым движком. Следовательно, она обладает свойством тестируемости.
* Удобство использования. Разработанный программный продукт обладает дружественным интерфейсом пользователя, следовательно программа достаточно удобна в использовании.
* Полнота. Т.к. все необходимые части программы полностью реализованы, программа является полной.

3.5 Условия выполнения программы

Для выполнения программы ЭВМ должна удовлетворять следующим системным требованиям:

Тактовая частота процессора не менее 700 ГГц

Видеокарта с объемом видеопамяти не менее 32 МБ

Не менее 512 МБ оперативной памяти

Операционная система: Windows 7/Windows XP/Linux/Mac OS

Для выполнения программы, на ЭВМ должна быть установлена виртуальная Java-машина.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проектирования были получены навыки проектирования и написания объектно-ориентированных информационных систем как отдельных программ. Также были получены навыки проектирования интерфейсов полноценных игровых приложений, закреплены навыки работы со стандартной библиотекой шаблонов языка Java, а также основные принципы работы с графическим движком.

Также были получены навыки разработки и отладки программ, использующих интерактивные интерфейсы, отладки объектно-ориентированных программ.

Была выполнена оценка качества программы по ряду критериев. Было определено, что разработанное программное решение является достаточно эффективным.

Таким образом можно сделать вывод относительно использования ООП в целом. Было определено, что ООП имеет смысл использовать, когда нужно работать с какой-то сложной единицей информации. ООП существует для удобства организации данных, и его использование в небольших, линейных программах является неоправданным.

Также можно сделать вывод относительно того, что использование объектного подхода в рамках поставленной задачи является полностью оправданным во-первых, потому что её реализация выглядела бы громоздкой и неэффективной с точки зрения структурного подхода, во-вторых, потому что разработанная система легко поддаётся абстрагированию и выделить в ней классы объектов не составляет труда.

Таким образом цель курсового проекта была достигнута. Разработанная игровая система может применяться в игровой индустрии при соответствующей доработке.

СПИСОК ЛИТЕРАТУРЫ

1. *Бабий А.А.* О некоторых характеристиках личной производительности труда программистов. // УСИМ, 1985, № 6, с. 17–19.

2. *Жоголев Е.А.* Технология программирования / Е.А. Жоголев. — М.: Научный мир, 2004. — 216 с.

3. *Д. Кнут.* Искусство программирования / Кнут Дональд. — М: Вильямс. 3-е издание. 2000.

4. Модели и структуры данных. Учеб. пособие / В.Д. Далека, А.С. Деревянко, О.Г. Кравец, Л.Е. Тимановская. — Харьков: ХГПУ, 2000.

4. *Буч Г.* Объектно-ориентированное проектирование с примерами применения / Г. Буч; Пер. с англ. — М.: Конкорд, 1992, — 519 с.

ПРИЛОЖЕНИЕ А

Текст программы

/\*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Севастопольский национальный технический университет

Кафедра информационных систем

Программа учета рабочих по номеру участка

Текст программы

РАЗРАБОТАЛ

Студент гр. И-22д

Петровский В.И.

2013

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Данная программа представляет собой игровую систему жанра Tower Defense.

Язык программирования: Java

Среда программирования Eclipse Kepler.

Дата последней коррекции: 19.05.2014 .

Версия 1.0

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++\*/

//Точка входа в программу

public class TD extends StateBasedGame {

public static void main(String args[])

throws SlickException

{

//инициализация и запуск приложения

AppGameContainer app = new AppGameContainer(new TD("TD"));

app.setDisplayMode(800, 600, false);

app.setAlwaysRender(false);

app.setTargetFrameRate(60);

app.start();

}

public TD(String gamename)

{

super(gamename);

}

@Override

public void initStatesList(GameContainer container) throws SlickException

{

//добавление состояний игры

this.addState(new MainMenuState());

this.addState(new Level1());

this.addState(new GameOverState());

this.addState(new VictoryState());

}

}

//Класс реализует игровое состояние "Победа"

public class VictoryState extends BasicGameState

{

//задний план и текст вывода на экран

Image background = null;

Image item;

@Override

public void init(GameContainer cont, StateBasedGame game)

throws SlickException

{

background = new Image("//data//VictorySplash.jpg");

item = new Image("//data//VictoryMenuItem.png");

}

@Override

public void render(GameContainer cont, StateBasedGame game, Graphics g)

throws SlickException

{

//прорисовка изображений и вывод статистики

background.draw(0,0);

item.draw(100,200);

g.setColor(Color.orange);

g.drawString("Killed Units: " + PlayerStats.getKilledUnits(), 100, 400);

g.drawString("Lives Left: " + PlayerStats.livesLeft(), 100, 430);

}

@Override

public void update(GameContainer container, StateBasedGame game, int delta)

throws SlickException

{

//ожидание нажатия клавиши Enter

if (container.getInput().isKeyPressed(Input.KEY\_ENTER))

{

game.getState(1).init(container, game);

game.enterState(0, new FadeOutTransition(), new FadeInTransition());

PlayerStats.clear();

}

}

@Override

public int getID()

{

// TODO Auto-generated method stub

return 3;

}

}

//класс обеспечивает хранение статистической информации

//о игроке

public class PlayerStats

{

static private int killedUnits = 0;

static private int gold = 200;

static private int unitsLeft = 0;

static private int lives = 10;

public static boolean isTowerPlacing = false;

static private Tower addTower;

static private boolean readyForNewWave;

static public void clear()

{

gold = 200;

unitsLeft = 0;

lives = 10;

isTowerPlacing = false;

killedUnits = 0;

}

static public void setReady(boolean value)

{

readyForNewWave = value;

}

static public boolean isReady()

{

return readyForNewWave;

}

static public void reduceGold(int value)

{

gold -= value;

}

static public void addTower(Tower tower)

{

addTower = tower;

}

static public Tower getAddTower()

{

return addTower;

}

static public void setAddTower(Tower tower)

{

addTower = tower;

}

public PlayerStats()

{

}

//инкапсуляция полей

//золото

static public int getGold()

{

return gold;

}

static public void addGold(int value)

{

gold += value;

}

//убитые существа

static public int getKilledUnits()

{

return killedUnits;

}

static public void addKilledUnits(int value)

{

killedUnits += value;

}

static public void denyLife(int value)

{

lives -= value;

}

static public int livesLeft()

{

return lives;

}

}

//Базовый класс для всех типов снарядов

public abstract class Missile

{

float x, y; //текущие координаты

Vector2f speed;

Vector2f direction;

float damage;

Animation sprite;

Creature target; //цель

Map gMap;

boolean canBeDeleted;

double angle;

public boolean canBeDeleted()

{

return canBeDeleted;

}

public void computeDirection()

{

if (target == null)

return;

double a = target.getX() - x; //Координата x вектора, указывающего на атакуемое существо

double b = target.getY() - y; //координата y этого вектора

//скалярное произведение указанного вектора и орта будет равно координате x

angle = Math.acos(a/(Math.sqrt(a\*a+b\*b))); //угол между ортом вдоль оси x и вектором существа

direction = new Vector2f((float)Math.cos(angle),(float)(Math.signum(b)\*Math.sin(angle)));

sprite.getCurrentFrame().setRotation((float)(Math.signum(b)\*Math.toDegrees(angle)));

}

//здесь определяются все поля кроме скорости. она индивидуальна

//для каждого снаряда

public Missile(Creature \_target, Map map, float \_x, float \_y)

{

gMap = map;

target = \_target;

sprite = new Animation();

x = \_x;

y = \_y;

speed = new Vector2f(1,1);

}

//абстрактный метод

//вызывается картой при столкновении с существом

public abstract void collide();

public abstract void update(int delta);

public void render()

{

sprite.draw(x,y);

}

}

//класс реализует стрелу

public class Arrow extends Missile

{

{

speed.x = 8;

speed.y = 8;

}

public Arrow(Creature \_target, Map map, float \_x, float \_y)

{

super(\_target, map, \_x, \_y);

try

{

sprite.addFrame(new Image("data//Arrow.png"), 1000);

} catch (SlickException e)

{

// TODO Auto-generated catch block

System.out.println("Выброшено исключение в конструкторе стрелы");

}

//определение скорости по каждой из проекций

computeDirection();

speed.x \*= direction.x;

speed.y \*= direction.y;

}

@Override

public void collide()

{

Creature c = gMap.getCreature(x, y, 16);

if (c != null)

{

c.damage((float)(3\*Math.sqrt(speed.x\*speed.x+speed.y\*speed.y)));

canBeDeleted = true;

}

}

public void update(int delta)

{

//приращение текущей координаты на значение скорости

x += speed.x;

y += speed.y;

//замедление стрелы

//т.к. вектор скорости может быть как положительным так и отрицательным

//к отрицательному нужно прибавлять скорость

//от положительной координаты убавлять

if (Math.abs(speed.x) != 0)

{

speed.x = (float)(speed.x - (Math.signum(target.getX() - speed.x)\*Math.cos(angle)\*0.2));

}

if (Math.abs(speed.y) != 0)

{

speed.y = (float)(speed.y - (Math.signum(speed.y)\*Math.sin(angle)\*0.2));

}

//если стрела слишком медленна, пометить её для удаления

if (0 <= Math.abs(speed.x) && Math.abs(speed.x) <= 0.5)

speed.x = 0;

if (0 <= Math.abs(speed.y) && Math.abs(speed.y) <= 0.5)

speed.y = 0;

if (speed.x == 0 && speed.y == 0)

canBeDeleted = true;

collide();

}

}

//класс реализует существо орка-метателя топоров

public class AxeThrower extends Creature

{

{

hp = 250;

speed = new Vector2f(4,4);

}

int regenCounter = 20;

public AxeThrower(Map map, float \_x, float \_y)

{

super(\_x, \_y, map);

//буферные массивы для хранения

//изображений анимации

Image [] movementUp;

Image [] movementDown;

Image [] movementRight;

Image [] movementLeft;

int [] duration; //длительность каждой анимации

//инициализация анимации

SpriteSheet ss = null;

try

{

ss = new SpriteSheet("//data/AxeThrowerSprite.png", 32, 32, 1);

movementUp = new Image[5];

movementDown = new Image[5];

movementLeft = new Image[5];

movementRight = new Image[5];

duration = new int[5];

Image [] actionDead = new Image[3];

for (int i = 0; i < 3; i++)

actionDead[i] = ss.getSubImage(4, i);

for(int i = 0; i < 5; i++)

{

movementUp[i] = ss.getSubImage(0, i);

movementDown[i] = ss.getSubImage(3, i);

movementLeft[i] = ss.getSubImage(2, i);

movementRight[i] = ss.getSubImage(1, i);

duration[i] = (int)(speed.x+speed.y)/2\*10;

}

int [] dur = new int[3];

for(int i = 0; i < 3; i++)

{

dur[i] = (int)(speed.x+speed.y)/2\*10;

}

up = new Animation(movementUp, duration, false);

down = new Animation(movementDown, duration, false);

left = new Animation(movementLeft, duration, false);

right = new Animation(movementRight, duration, false);

sprite = down;

}

catch (SlickException e)

{

System.out.println("Ошибка при загрузке спрайтов воришки");

}

catch (NullPointerException e)

{

System.out.println("Ошибка нулевого указателя в конструкторе метателя топоров");

}

}

@Override

public void update(int delta)

{

//обеспечивает троллю регенерацию здоровья

if (hp < 250 && regenCounter <= 0)

hp++;

if (regenCounter >= 0)

regenCounter --;

else

regenCounter = 20;

if (!isParalysed)

{

updateDirection();

//смена анимации в зависимости от направления движения

if (direction.x == 0 && direction.y == -1)

sprite = up;

else

if (direction.x == 1 && direction.y == 0)

sprite = right;

else

if (direction.x == -1 && direction.y == 0)

sprite = left;

else

if (direction.x == 0 && direction.y == 1)

sprite = down;

if (isPoisoned)

{

//корректировка скорости и направления движения

nowSpeed.x = (float) Math.round(speed.x\*0.5 \*direction.x);

nowSpeed.y = (float) Math.round(speed.y\*0.5 \*direction.y);

poisonCounter--;

//System.out.println("Я отравлен и мне осталось " + poisonCounter);

if (poisonCounter <= 0)

setPoisoned(false);

}

else

{

//корректировка скорости и направления движения

nowSpeed.x = speed.x\*direction.x;

nowSpeed.y = speed.y\*direction.y;

}

x += nowSpeed.getX()\*0.5;

y += nowSpeed.getY()\*0.5;

sprite.update(delta);

}

else

{

}

}

@Override

public void kill()

{

isDead = true;

PlayerStats.addGold(200);

PlayerStats.addKilledUnits(1);

}

@Override

void survive()

{

isDead = true;

PlayerStats.denyLife(1);

}

}

//класс реализует кнопку меню

public abstract class Button

{

protected Image icon;

protected int x;

protected int y;

protected boolean collided;

protected GameMenu parent;

protected int cost;

public Button(GameMenu par, Image i, int \_x, int \_y)

{

icon = i;

parent = par;

x = par.getX() + \_x;

y = par.getY() + \_y;

collided = false;

}

abstract public void render(GameContainer cont);

abstract public void update(int delta, GameContainer container);

}

//класс реализует существо "рыцарь хаоса"

public class ChaosKnight extends Creature

{

{

hp = 350;

speed = new Vector2f(4,4);

}

public ChaosKnight(Map map, float \_x, float \_y)

{

super(\_x, \_y, map);

//буферные массивы для хранения

//изображений анимации

Image [] movementUp;

Image [] movementDown;

Image [] movementRight;

Image [] movementLeft;

int [] duration; //длительность каждой анимации

//инициализация анимации

SpriteSheet ss = null;

try

{

ss = new SpriteSheet("//data/ChaosKnight.png", 32, 32, 1);

movementUp = new Image[5];

movementDown = new Image[5];

movementLeft = new Image[5];

movementRight = new Image[5];

duration = new int[5];

Image [] actionDead = new Image[3];

for (int i = 0; i < 3; i++)

actionDead[i] = ss.getSubImage(4, i);

for(int i = 0; i < 5; i++)

{

movementUp[i] = ss.getSubImage(0, i);

movementDown[i] = ss.getSubImage(3, i);

movementLeft[i] = ss.getSubImage(2, i);

movementRight[i] = ss.getSubImage(1, i);

duration[i] = (int)(speed.x+speed.y)/2\*10;

}

int [] dur = new int[3];

for(int i = 0; i < 3; i++)

{

dur[i] = (int)(speed.x+speed.y)/2\*10;

}

up = new Animation(movementUp, duration, false);

down = new Animation(movementDown, duration, false);

left = new Animation(movementLeft, duration, false);

right = new Animation(movementRight, duration, false);

sprite = down;

}

catch (SlickException e)

{

System.out.println("Ошибка при загрузке спрайтов гранта");

}

catch (NullPointerException e)

{

System.out.println("Ошибка нулевого указателя в конструкторе гранта");

}

}

@Override

public void kill()

{

isDead = true;

PlayerStats.addGold(300);

PlayerStats.addKilledUnits(1);

}

}

//класс реализует снаряд "ледяной шар"

public class Coldblast extends Missile

{

int animationCounter;

boolean collided = false;

Animation bs; //спрайт взрыва

public Coldblast(Creature \_target, Map map, float \_x, float \_y)

{

super(\_target, map, \_x, \_y);

speed.set(4,4);

sprite = new Animation();

bs = new Animation();

try

{

sprite.addFrame(new Image("//data/Coldblast.png"), 10);

SpriteSheet ss = new SpriteSheet("//data/ColdblastBoom(50x47).png", 50, 47, 0);

for (int i = 0; i < 6; i++)

bs.addFrame(ss.getSubImage(i,0), 150);

}

catch (SlickException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

}

@Override

public void collide()

{

if (!collided)

{

Creature c = gMap.getCreature(x, y, 15);

if (c != null)

{

c.setColded(true);

c.damage(100);

sprite = bs;

collided = true;

speed.x = 0;

speed.y = 0;

}

}

}

@Override

public void update(int delta)

{

if (!collided)

{

if (target.isDead())

canBeDeleted = true;

computeDirection();

x+=(speed.x\*direction.x);

y+=(speed.y\*direction.y);

collide();

}

else

if (sprite.getFrame() == 5)

canBeDeleted = true;

}

}

//класс является базовым для всех существ

public abstract class Creature

{

protected float x = 0;

protected float y = 0;

//абсолютная скорость и текущая скорость

protected Vector2f speed, nowSpeed;

//направление движения

protected Vector2f direction;

//анимации движений и смерти

protected Animation up;

protected Animation down;

protected Animation right;

protected Animation left;

protected Animation dead;

//текущая анимация

protected Animation sprite;

//мертво ли существо

protected boolean isDead = false;

protected boolean isParalysed = false;

protected boolean isPoisoned = false;

protected boolean isColded = false;

protected Integer poisonCounter;

protected Color color;

Map gMap; //карта, на которой расположено существо

float hp; //количество очков здоровья существа

public Creature(float \_x, float \_y, Map map)

{

gMap = map;

direction = new Vector2f(0,0);

nowSpeed = new Vector2f(0,0);

x = \_x;

y = \_y;

color = Color.white;

}

public void setColded(boolean val)

{

isColded = val;

speed.set(1,1);

}

public boolean isColded()

{

return isColded;

}

public void setPoisoned(boolean val)

{

isPoisoned = val;

if (val = true)

poisonCounter = 100;

else

poisonCounter = 0;

}

public boolean getPoisonState()

{

return isPoisoned;

}

public void setParalyze(boolean val)

{

isParalysed = val;

}

public boolean getParalyzeState()

{

return isParalysed;

}

//инкапсуляция координат

public float getX() { return x; }

public float getY() { return y; }

public void damage(float dmg)

{

hp-=dmg;

if (hp <= 0)

kill();

}

//убивает существо

abstract public void kill();

public void update(int delta)

{

if (!isParalysed)

{

updateDirection();

//смена анимации в зависимости от направления движения

if (direction.x == 0 && direction.y == -1)

sprite = up;

else

if (direction.x == 1 && direction.y == 0)

sprite = right;

else

if (direction.x == -1 && direction.y == 0)

sprite = left;

else

if (direction.x == 0 && direction.y == 1)

sprite = down;

if (isPoisoned)

{

//корректировка скорости и направления движения

nowSpeed.x = (float) Math.round(speed.x\*0.5 \*direction.x);

nowSpeed.y = (float) Math.round(speed.y\*0.5 \*direction.y);

poisonCounter--;

//System.out.println("Я отравлен и мне осталось " + poisonCounter);

if (poisonCounter <= 0)

setPoisoned(false);

}

else

{

//корректировка скорости и направления движения

nowSpeed.x = speed.x\*direction.x;

nowSpeed.y = speed.y\*direction.y;

}

x += nowSpeed.getX()\*0.5;

y += nowSpeed.getY()\*0.5;

sprite.update(delta);

}

else

{

}

}

void survive()

{

isDead = true;

PlayerStats.denyLife(1);

}

public void render()

{

if (isPoisoned)

{

color = Color.green;

}

else

if (isColded)

color = Color.blue;

else

color = Color.white;

sprite.draw(x,y, color);

}

//метод служит для смены направления движения

//(ведётся взаимодействие с картой)

//цель - обеспечить смену направления движения только

//в том случае если юнит дошел до конца тайла

protected void updateDirection()

{

try

{

//если юнит стоит, то он стоит посреди тайла,

//значит направление можно брать с центра

//юнита

if (direction.x == 0 && direction.y == 0)

{

//System.out.println("Я болван и стою на месте");

System.out.println("Застрял в координатах " + x + " " + y);

if (x - Math.round(x) != 0 || y - Math.round(y) != 0)

{

x /= 32;

x = (Math.round(x)) \* 32;

y /= 32;

y = Math.round(y) \* 32;

System.out.println("Пофиксили координаты " + x + " " + y);

}

//direction = gMap.getTile(x+15, y+15).getDirection();

System.out.println(direction = gMap.getTile(x+15, y+15).getDirection());

}

//иначе, если текущее направление - направо - нужно менять направление только

//если самая левая точка спрайта вошла на него

//эта точка - текущие координаты спрайта

else if (direction.x == 1 && direction.y == 0)

{

direction = gMap.getTile((int)x, (int)y+15).getDirection();

}

//если же текущее направление - налево, то нменяем направление

//только если самая правая точка перешла в новый тайл

else if (direction.x == -1 && direction.y == 0)

{

direction = gMap.getTile((int)x+32, (int)y).getDirection();

}

//если движемся вверх - то меняем направление только по самой нижней точке

else if (direction.x == 0 && direction.y == -1)

{

direction = gMap.getTile((int)x+1, (int)y+31).getDirection();

}

//иначе движемся вниз. направление меняем если самая верхняя точка

//перешла в новый тайл.

else if (direction.x == 0 && direction.y == 1)

{

direction = gMap.getTile((int)x+1, (int)y+1).getDirection();

}

}

catch (NullPointerException e)

{

direction = new Vector2f(0,0);

System.out.println("Обращение к нулевому тайлу направления. Направления юнита установлено в 0,0");

}

}

public boolean isDead()

{

return isDead;

}

}

//класс реализует пункт назначения

//бегущих существ.

//если существа добираются сюда,

//у игрока отнимаются жизни

public class Destination

{

SpriteSheet sheet;

Animation sprite;

int x, y; //координаты

Image[] imgs;

Map gMap;

public Destination(int \_x, int \_y, Map map)

{

gMap = map;

x = \_x;

y = \_y;

imgs = new Image[6];

//загрузка ресурсов

try

{

sheet = new SpriteSheet("data//endTileSprite.png", 32,32, 1);

System.out.println(sheet);

for (int i = 0; i < 6; i++)

{

imgs[i] = sheet.getSubImage(0, i);

}

sprite = new Animation(imgs, 300);

System.out.println(sprite);

}

catch (SlickException e)

{

System.out.println("Выброшено исключение классом Destination");

}

}

public void update(int delta)

{

sprite.update(delta);

Creature c = gMap.getCreature(x, y, 17);

if (c != null)

c.survive();

}

public void render(Graphics g)

{

sprite.draw(x,y);

}

}

//класс реализует тайл, указывающий направление существам

public class DirectionTile extends Tile

{

private Vector2f direction;//Указывает направление. 0 - вправо, 1 - влево, 2 - вверх, 3 - вниз

public DirectionTile(int \_x, int \_y, Image spr, Vector2f dir)

{

super(\_x, \_y, spr);

direction = dir;

}

@Override

public void render(Graphics g)

{

sprite.draw(x,y);

}

public Vector2f getDirection()

{

return direction;

}

}

//класс реализует пустой тайл

public class EmptyTile extends Tile {

public EmptyTile(int \_x, int \_y, Image spr)

{

super(\_x, \_y, spr);

}

@Override

public void render(Graphics g)

{

sprite.draw(x, y);

}

}

//класс реализует огненный шар

public class Fireball extends Missile

{

Animation boomSprite; //спрайт взрыва

int animationCounter;

boolean collided; //было ли столкновение

{

speed.set(4,4);

collided = false;

animationCounter = 50;

}

public Fireball(Creature \_target, Map map, float \_x, float \_y)

{

super(\_target, map, \_x, \_y);

try

{

collided = false;

sprite = new Animation();

boomSprite = new Animation();

SpriteSheet ss = new SpriteSheet("//data/OilMissile.png", 15, 12, 0);

SpriteSheet bss = new SpriteSheet("//data/OilMissileBoom.png", 23, 22, 1);

for (int i = 0; i < 4; i++)

{

sprite.addFrame(ss.getSubImage(i, 0), 10);

boomSprite.addFrame(bss.getSubImage(i, 0), 150);

}

}

catch (SlickException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

}

@Override

public void collide()

{

if (!collided)

{

Creature c = gMap.getCreature(x, y, 15);

if (c != null)

{

c.damage(200);

sprite = boomSprite;

collided = true;

speed.x = 0;

speed.y = 0;

}

}

else

{

if (sprite.getFrame() == 2)

canBeDeleted = true;

}

}

@Override

public void update(int delta)

{

if (!collided)

{

computeDirection();

x+=(speed.x\*direction.x);

y+=(speed.y\*direction.y);

if (target.isDead())

canBeDeleted = true;

}

collide();

if (animationCounter == 0)

{

sprite.update(150);

animationCounter = 150;

}

else

animationCounter--;

}

}

//класс реализует существо "огненный маг"

public class Firemage extends Mage

{

public Firemage(int x, int y)

{

super(x, y, "//data/FireMage.png");

fireRate = 5;

}

@Override

public void fire()

{

if (target != null)

gMap.addMissile(new Fireball(target, gMap, x+20,y+20));

}

}

//класс реализует кнопку создания огненного мага

public class FiremageButton extends GameMenuButton

{

public FiremageButton(GameMenu par, Image i, int \_x, int \_y)

{

super(par, i, \_x, \_y);

cost = 1000;

}

@Override

public void update(int delta, GameContainer cont)

{

if (collided)

if (cont.getInput().isMousePressed(0))

PlayerStats.setAddTower(new Firemage(cont.getInput().getMouseX(), cont.getInput().getMouseY()));

}

}

//класс реализует существо "маг льда"

public class Frostmage extends Mage

{

{

fireRate = 130;

}

public Frostmage(int \_x, int \_y)

{

super(\_x, \_y, "//data/IceMage.png");

// TODO Auto-generated constructor stub

}

@Override

protected void findTarget()

{

if ((target == null) || target.isDead())

{

target = gMap.getNotColdedCreature(x, y, range);

}

if (!gMap.isCreatureInRadius(target,x,y,range))

target = null;

}

@Override

public void fire()

{

if (target != null)

gMap.addMissile(new Coldblast(target, gMap, x+20,y+20));

}

}

//класс реализует кнопку добавления ледяного мага

public class FrostmageButton extends GameMenuButton

{

public FrostmageButton(GameMenu par, Image i, int \_x, int \_y)

{

super(par, i, \_x, \_y);

cost = 1000;

}

@Override

public void update(int delta, GameContainer cont)

{

if (collided)

if (cont.getInput().isMousePressed(0))

PlayerStats.setAddTower(new Frostmage(cont.getInput().getMouseX(), cont.getInput().getMouseY()));

}

}

//класс реализует игровое меню

public class GameMenu

{

private Image bg;

private final int beginXPos = 0;

private final int beginYPos = 480;

//кнопки башен

GameMenuButton wtButton, rtButton, otButton, lmButton, fmButton, frmButton;

NewWaveButton nwButton;

public GameMenu()

{

try

{

bg = new Image("data//GameMenuBackground.png");

wtButton = new WatchTowerButton(this, new Image("data//WatchTowerIcon.png"), 350, 10);

rtButton = new RuneTowerButton(this, new Image("data//RuneTowerIcon.png"), 400, 10);

otButton = new OrcTowerButton(this, new Image("data//OrcTowerIcon.png"), 450, 10);

lmButton = new LightmageButton(this, new Image("data//LightingMageIcon.png"), 350, 60);

nwButton = new NewWaveButton(this, new Image("data//NewWaveButton.png"), 500, 10);

fmButton = new FrostmageButton(this, new Image("data//FrostmageIcon.png"), 400, 60);

frmButton = new FiremageButton(this, new Image("data//FiremageIcon.png"), 450, 60);

}

catch (SlickException e)

{

System.out.println("SlickException выброшено конструктором класса GameMenu");

}

}

public void render(Graphics g, GameContainer cont)

{

//прорисовка фона

bg.draw(beginXPos,beginYPos);

//вывод статистики

g.drawString("Gold: " + PlayerStats.getGold(), beginXPos+10, beginYPos+20);

g.drawString("Killed units: " + PlayerStats.getKilledUnits(), beginXPos+10, beginYPos+40);

g.drawString("Lives left: " + PlayerStats.livesLeft(), beginXPos+10, beginYPos+60);

//прорисовка кнопок

wtButton.render(cont);

rtButton.render(cont);

otButton.render(cont);

nwButton.render(cont);

lmButton.render(cont);

fmButton.render(cont);

frmButton.render(cont);

}

public int getX() { return beginXPos; }

public int getY() { return beginYPos; }

public void update(int delta, GameContainer container)

{

wtButton.update(delta, container);

rtButton.update(delta, container);

otButton.update(delta, container);

nwButton.update(delta, container);

lmButton.update(delta, container);

fmButton.update(delta, container);

frmButton.update(delta, container);

}

}

//класс реализует кнопку игрового меню

public abstract class GameMenuButton extends Button

{

public GameMenuButton(GameMenu par, Image i, int \_x, int \_y)

{

super(par,i,\_x,\_y);

}

public void render(GameContainer cont)

{

int mouseX = cont.getInput().getMouseX();

int mouseY = cont.getInput().getMouseY();

if (cost >= PlayerStats.getGold())

icon.draw(x, y, new Color(255,0,0));

else if ((x < mouseX) && (y < mouseY) && (x+46 > mouseX) && (y+38 > mouseY))

{

icon.draw(x,y);

collided = true;

}

else

{

icon.draw(x, y, new Color(100,100,100));

collided = false;

}

}

}

//класс реализует состояние игрового процесса "игра окончена"

public class GameOverState extends BasicGameState

{

Image background = null;

Image item;

@Override

public void init(GameContainer cont, StateBasedGame game)

throws SlickException

{

background = new Image("//data//GameOverSplash.jpg");

item = new Image("//data//GameOverMenuItem.png");

}

@Override

public void render(GameContainer cont, StateBasedGame game, Graphics g)

throws SlickException

{

background.draw(100,0);

item.draw(100,200);

}

@Override

public void update(GameContainer container, StateBasedGame game, int delta)

throws SlickException

{

if (container.getInput().isKeyPressed(Input.KEY\_ENTER))

{

game.getState(1).init(container, game);

game.enterState(0, new FadeOutTransition(), new FadeInTransition());

}

}

@Override

public int getID()

{

// TODO Auto-generated method stub

return 2;

}

}

//класс реализует снаряд "призрак"

public class Ghostmissile extends Missile

{

Animation stopAnimation;

int lifeCounter = 28;

boolean animationChanged = false;

private Vector2f baseSpeed = new Vector2f();

{

speed.x = 5;

speed.y = 5;

baseSpeed.x = 5;

baseSpeed.y = 5;

}

public Ghostmissile(Creature \_target, Map map, float \_x, float \_y)

{

super(\_target, map, \_x, \_y);

stopAnimation = new Animation();

try

{

SpriteSheet sh = new SpriteSheet("data//StoppetGhostMissile.png", 30, 25);

sprite.addFrame(new Image("data//Ghostmissile.png"), 1000);

for (int i = 0; i < 4; i++)

stopAnimation.addFrame(sh.getSubImage(0, i), 200);

}

catch (SlickException e)

{

System.out.println("Ghostmissile в конструкторе выброшено исключение");

e.printStackTrace();

}

//определение скорости по каждой из проекций

computeDirection();

speed.x \*= direction.x;

speed.y \*= direction.y;

}

@Override

public void collide()

{

if (target == null)

{

canBeDeleted = true;

}

if (!animationChanged)

{

Creature c = gMap.getCreature(x, y, 12);

if (c != null)

{

if (!target.getParalyzeState())

{

sprite = stopAnimation;

target.setParalyze(true);

baseSpeed.x = 0;

baseSpeed.y = 0;

animationChanged = true;

}

}

}

}

@Override

public void update(int delta)

{

//приращение текущей координаты на значение скорости

x += speed.x;

y += speed.y;

if (target.isDead())

canBeDeleted = true;

if (sprite == stopAnimation)

{

lifeCounter--;

}

if (lifeCounter <= 0)

{

target.setParalyze(false);

canBeDeleted = true;

}

computeDirection();

if (target == null)

{

canBeDeleted = true;

return;

}

collide();

speed.x = baseSpeed.x \* direction.x;

speed.y = baseSpeed.y \* direction.y;

if (speed.x == 0 && speed.y == 0 && sprite != stopAnimation)

canBeDeleted = true;

}

}

//класс реализует орка-воина

public class Grunt extends Creature

{

{

hp = 300;

speed = new Vector2f(2,2);

}

int animationCounter = 5;

public Grunt(Map map, float \_x, float \_y)

{

super(\_x, \_y, map);

//буферные массивы для хранения

//изображений анимации

Image [] movementUp;

Image [] movementDown;

Image [] movementRight;

Image [] movementLeft;

int [] duration; //длительность каждой анимации

//инициализация анимации

SpriteSheet ss = null;

try

{

ss = new SpriteSheet("//data/GruntSprite.png", 32, 32, 1);

movementUp = new Image[5];

movementDown = new Image[5];

movementLeft = new Image[5];

movementRight = new Image[5];

duration = new int[5];

Image [] actionDead = new Image[3];

for (int i = 0; i < 3; i++)

actionDead[i] = ss.getSubImage(4, i);

for(int i = 0; i < 5; i++)

{

movementUp[i] = ss.getSubImage(0, i);

movementDown[i] = ss.getSubImage(3, i);

movementLeft[i] = ss.getSubImage(2, i);

movementRight[i] = ss.getSubImage(1, i);

duration[i] = (int)(speed.x+speed.y)/2\*10;

}

int [] dur = new int[3];

for(int i = 0; i < 3; i++)

{

dur[i] = (int)(speed.x+speed.y)/2\*10;

}

up = new Animation(movementUp, duration, false);

down = new Animation(movementDown, duration, false);

left = new Animation(movementLeft, duration, false);

right = new Animation(movementRight, duration, false);

sprite = down;

}

catch (SlickException e)

{

System.out.println("Ошибка при загрузке спрайтов гранта");

}

catch (NullPointerException e)

{

System.out.println("Ошибка нулевого указателя в конструкторе гранта");

}

}

@Override

public void update(int delta)

{

if (!isParalysed)

{

updateDirection();

//смена анимации в зависимости от направления движения

if (direction.x == 0 && direction.y == -1)

sprite = up;

else

if (direction.x == 1 && direction.y == 0)

sprite = right;

else

if (direction.x == -1 && direction.y == 0)

sprite = left;

else

if (direction.x == 0 && direction.y == 1)

sprite = down;

if (isPoisoned)

{

//корректировка скорости и направления движения

nowSpeed.x = (float) Math.round(speed.x\*0.5 \*direction.x);

nowSpeed.y = (float) Math.round(speed.y\*0.5 \*direction.y);

poisonCounter--;

//System.out.println("Я отравлен и мне осталось " + poisonCounter);

if (poisonCounter <= 0)

setPoisoned(false);

}

else

{

//корректировка скорости и направления движения

nowSpeed.x = speed.x\*direction.x;

nowSpeed.y = speed.y\*direction.y;

}

x += nowSpeed.getX()\*0.5;

y += nowSpeed.getY()\*0.5;

if (animationCounter == 0)

{

sprite.update(delta);

animationCounter = 5;

}

else

animationCounter--;

}

else

{

}

}

@Override

public void kill()

{

isDead = true;

PlayerStats.addGold(150);

PlayerStats.addKilledUnits(1);

}

}

/\*\*

\*

\* Класс реализует состояние игры "выполнение"

\*/

public class Level1 extends BasicGameState

{

Map gMap;

LinkedList<Creature> mobs;

GameMenu menu;

int creepCounter;

int waveCounter = 0;

int spawnDelay;

@Override

public void init(GameContainer container, StateBasedGame game)

throws SlickException

{

mobs = new LinkedList<Creature>();

//сюда будем заносить тайлы, позже их передадим в конструктор

LinkedList<Tile> tl = new LinkedList<Tile>();

LinkedList<DirectionTile> dtl = new LinkedList<DirectionTile>();

this.createMap(tl, dtl);

gMap = new Map(tl, dtl);

//gMap.addTower(new Frostmage(5\*32, 7\*32));

//gMap.addCreature(new Peon(gMap, 32\*5f, 32\*14f));

gMap.setDestination(new Destination(32\*20, 32\*10, gMap));

//tl.add(new Destination(32\*20, 32\*10));

//инициализация меню

spawnDelay = 50;

menu = new GameMenu();

}

//логика появления новых существ

private void creepSpawn()

{

if (spawnDelay != 0)

spawnDelay--;

else

{

spawnDelay = 50;

if (creepCounter == 0)

{

creepCounter = 20;

waveCounter++;

PlayerStats.setReady(false);

}

if (PlayerStats.isReady())

{

switch (waveCounter)

{

case 1:

gMap.addCreature(new Peon(gMap, 32\*5f, 32\*14f));

creepCounter--;

break;

case 2:

gMap.addCreature(new Thief(gMap, 32\*5f, 32\*14f));

creepCounter--;

break;

case 3:

gMap.addCreature(new Grunt(gMap, 32\*5f, 32\*14f));

creepCounter--;

break;

case 4:

gMap.addCreature(new AxeThrower(gMap, 32\*5f, 32\*14f));

creepCounter--;

break;

case 5:

gMap.addCreature(new ChaosKnight(gMap, 32\*5f, 32\*14f));

creepCounter--;

break;

}

}

}

}

@Override

public void update(GameContainer container, StateBasedGame game, int delta)

throws SlickException

{

//обработка проигрышной ситуации(кончились жизни)

if (PlayerStats.livesLeft() <= 0)

{

gMap.clear();

waveCounter = 0;

PlayerStats.clear();

game.enterState(2, new FadeOutTransition(), new FadeInTransition());

}

if (waveCounter == 6)

{

game.enterState(3);

}

if (PlayerStats.getAddTower() == null)

{

creepSpawn();

gMap.update(delta);

}

else

{

//обработка добавления башни

if (PlayerStats.getAddTower() != null)

{

//движение башни вслед за курсором

PlayerStats.getAddTower().move(container.getInput().getMouseX(), container.getInput().getMouseY());

//если нажата клавиша Esc, значит нужно удалить добавляемую башню

//и продолжить работу

container.getInput();

if (container.getInput().isKeyPressed(Input.KEY\_ESCAPE))

PlayerStats.setAddTower(null);

//проверка пересечений с другими объектами и меню

if (container.getInput().getMouseY()+64 < 480

&& !gMap.isCollide(container.getInput().getMouseX(), container.getInput().getMouseY(), PlayerStats.getAddTower().getHeight(), PlayerStats.getAddTower().getWidth()))

{

//если пересечений нет, то покрасить башню в белый

PlayerStats.getAddTower().setColor(Color.white);

//если нажата левая кнопка мыши в области карты, то

//добавить башню

if (container.getInput().isMousePressed(0))

{

gMap.addTower(PlayerStats.getAddTower());

PlayerStats.reduceGold(PlayerStats.getAddTower().getCost());

PlayerStats.setAddTower(null);

}

}

else

{

//если есть пересечения, то покрасить башню в красный

PlayerStats.getAddTower().setColor(Color.red);

}

}

}

menu.update(delta, container);

}

@Override

public void render(GameContainer container, StateBasedGame game, Graphics g)

throws SlickException

{

gMap.render(g);

menu.render(g, container);

if (PlayerStats.getAddTower() != null)

PlayerStats.getAddTower().render();

}

@Override

public int getID()

{

return 1;

}

private void createMap(LinkedList<Tile> tl, LinkedList<DirectionTile> dtl)

throws SlickException

{

//коллекция тайлов

SpriteSheet tileSheet = new SpriteSheet("//data//TileMap.png", 32, 32, 1);

tileSheet.startUse();

//создание списка тайлов

for (int i = 0; i < 25; i++)

for (int j = 0; j < 15; j++)

{

tl.add(new EmptyTile(i\*32, j\*32, tileSheet.getSubImage(11, 17)));

}

//создание дорог

for (int i = 0; i < 5; i++)

{

dtl.add(new DirectionTile(32\*5,32\*10+i\*32,tileSheet.getSubImage(14,18), new Vector2f(0,-1)));

}

for (int i = 0; i < 7; i++)

{

dtl.add(new DirectionTile(32\*5+i\*32,32\*9,tileSheet.getSubImage(14,18), new Vector2f(1,0)));

}

//sd

for (int i = 0; i < 5; i++)

{

dtl.add(new DirectionTile(32\*12,32\*9 - 32\*i,tileSheet.getSubImage(14,18), new Vector2f(0,-1)));

}

for (int i = 0; i < 6; i++)

{

tl.add(new EmptyTile(32\*4,32\*9+i\*32,tileSheet.getSubImage(13, 14)));

}

for (int i = 0; i < 3; i++)

{

dtl.add(new DirectionTile(32\*12-32\*i,32\*4,tileSheet.getSubImage(14,18), new Vector2f(-1,0)));

}

for (int i = 0; i < 4; i++)

{

tl.add(new EmptyTile(32\*6,32\*11+i\*32,tileSheet.getSubImage(5, 15)));

}

tl.add(new EmptyTile(32\*6,32\*10,tileSheet.getSubImage(0, 15)));

for (int i = 0; i < 6; i++)

{

tl.add(new EmptyTile(32\*7+32\*i,32\*10,tileSheet.getSubImage(9, 15)));

}

for (int i = 0; i < 6; i++)

{

tl.add(new EmptyTile(32\*13,32\*9 - 32\*i,tileSheet.getSubImage(5, 15)));

}

for (int i = 0; i < 6; i++)

{

tl.add(new EmptyTile(32\*5+i\*32,32\*8,tileSheet.getSubImage(9, 14)));

}

tl.add(new EmptyTile(32\*11,32\*8,tileSheet.getSubImage(4, 14)));

tl.add(new EmptyTile(32\*13,32\*10,tileSheet.getSubImage(13, 15)));

for (int i = 0; i < 2; i++)

{

tl.add(new EmptyTile(32\*11,32\*7-32\*i,tileSheet.getSubImage(15, 14)));

}

tl.add(new EmptyTile(32\*11,32\*5,tileSheet.getSubImage(11, 14)));

tl.add(new EmptyTile(32\*10,32\*5,tileSheet.getSubImage(0, 15)));

for (int i = 0; i < 2; i++)

{

dtl.add(new DirectionTile(32\*9,32\*4+i\*32,tileSheet.getSubImage(14, 18), new Vector2f(0,1)));

}

for (int i = 0; i < 5; i++)

{

dtl.add(new DirectionTile(32\*9-i\*32,32\*6,tileSheet.getSubImage(14, 18), new Vector2f(-1,0)));

}

for (int i = 0; i < 5; i++)

{

dtl.add(new DirectionTile(32\*4,32\*6-32\*i,tileSheet.getSubImage(14, 18), new Vector2f(0,-1)));

}

for (int i = 0; i < 13; i++)

{

dtl.add(new DirectionTile(32\*4+i\*32,32,tileSheet.getSubImage(14, 18), new Vector2f(1,0)));

}

for (int i = 0; i < 9; i++)

{

dtl.add(new DirectionTile(32\*17,32+32\*i,tileSheet.getSubImage(14, 18), new Vector2f(0,1)));

}

for (int i = 0; i < 3; i++)

{

dtl.add(new DirectionTile(32\*17+32\*i,32\*10,tileSheet.getSubImage(14, 18), new Vector2f(1,0)));

}

for (int i = 0; i < 14; i++)

{

tl.add(new EmptyTile(32\*4+32\*i, 0,tileSheet.getSubImage(9, 14)));

}

for (int i = 0; i < 10; i++)

{

tl.add(new EmptyTile(32\*6+32\*i,64,tileSheet.getSubImage(9, 15)));

}

for (int i = 0; i < 6; i++)

{

tl.add(new EmptyTile(32\*3,32+32\*i,tileSheet.getSubImage(14, 14)));

}

for (int i = 0; i < 6; i++)

{

tl.add(new EmptyTile(32\*4+32\*i,32\*7,tileSheet.getSubImage(9, 15)));

}

for (int i = 0; i < 2; i++)

{

tl.add(new EmptyTile(32\*6+32\*i,32\*5,tileSheet.getSubImage(9, 14)));

}

for (int i = 0; i < 4; i++)

{

tl.add(new EmptyTile(32\*9+32\*i,32\*3,tileSheet.getSubImage(9, 14)));

}

tl.add(new EmptyTile(32\*13,32\*3,tileSheet.getSubImage(7, 15)));

tl.add(new EmptyTile(32\*8,32\*5,tileSheet.getSubImage(4, 14)));

tl.add(new EmptyTile(32\*8,32\*4,tileSheet.getSubImage(15, 14)));

tl.add(new EmptyTile(32\*8,32\*3,tileSheet.getSubImage(18, 14)));

tl.add(new EmptyTile(32\*10,32\*6,tileSheet.getSubImage(5, 15)));

tl.add(new EmptyTile(32\*10,32\*7,tileSheet.getSubImage(13, 15)));

tl.add(new EmptyTile(32\*5,32\*5,tileSheet.getSubImage(7, 14)));

tl.add(new EmptyTile(32\*5,32\*2,tileSheet.getSubImage(0, 15)));

tl.add(new EmptyTile(32\*5,32\*3,tileSheet.getSubImage(5, 15)));

tl.add(new EmptyTile(32\*5,32\*4,tileSheet.getSubImage(5, 15)));

tl.add(new EmptyTile(32\*3,0,tileSheet.getSubImage(18, 14)));

tl.add(new EmptyTile(32\*4,32\*8,tileSheet.getSubImage(18, 14)));

tl.add(new EmptyTile(32\*3,32\*7,tileSheet.getSubImage(12, 15)));

tl.add(new EmptyTile(32\*16,64,tileSheet.getSubImage(12, 14)));

for (int i = 0; i < 8; i++)

{

tl.add(new EmptyTile(32\*16,96+32\*i,tileSheet.getSubImage(15, 14)));

}

for (int i = 0; i < 8; i++)

{

tl.add(new EmptyTile(32\*18,32+32\*i,tileSheet.getSubImage(5, 15)));

}

tl.add(new EmptyTile(32\*18,0,tileSheet.getSubImage(7, 15)));

tl.add(new EmptyTile(32\*18,96+32\*6,tileSheet.getSubImage(7, 14)));

tl.add(new EmptyTile(32\*19,96+32\*6,tileSheet.getSubImage(9, 14)));

for (int i = 0; i < 3; i++)

{

tl.add(new EmptyTile(32\*17+32\*i,32\*11,tileSheet.getSubImage(9, 15)));

}

tl.add(new EmptyTile(32\*16,32\*11,tileSheet.getSubImage(12, 15)));

tileSheet.endUse();

}

}

//класс реализует снаряд "молния"

public class Lighting extends Missile

{

int animationCounter;

boolean collided = false;

Animation bs;

public Lighting(Creature \_target, Map map, float \_x, float \_y)

{

super(\_target, map, \_x, \_y);

speed.set(8,8);

sprite = new Animation();

bs = new Animation();

try

{

sprite.addFrame(new Image("//data/LightingMissile.png"), 10);

SpriteSheet ss = new SpriteSheet("//data/LightingMissileBoom(31x30).png", 31, 30, 0);

for (int i = 0; i < 4; i++)

bs.addFrame(ss.getSubImage(i,0), 150);

}

catch (SlickException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

}

@Override

public void collide()

{

if (!collided)

{

Creature c = gMap.getCreature(x, y, 15);

if (c != null)

{

c.damage(400);

sprite = bs;

collided = true;

speed.x = 0;

speed.y = 0;

}

}

}

@Override

public void update(int delta)

{

if (!collided)

{

if (target.isDead())

canBeDeleted = true;

computeDirection();

x+=(speed.x\*direction.x);

y+=(speed.y\*direction.y);

collide();

}

else

if (sprite.getFrame() == 3)

canBeDeleted = true;

}

}

//класс реализует воздушного мага

public class Lightmage extends Mage

{

public Lightmage(int \_x, int \_y)

{

super(\_x, \_y, "//data/LightingMage.png");

fireRate = 200;

range = 1000;

}

@Override

public void fire()

{

if (target != null)

gMap.addMissile(new Lighting(target, gMap, x+20,y+20));

}

}

//класс реализует кнопку создания светлого мага

public class LightmageButton extends GameMenuButton

{

public LightmageButton(GameMenu par, Image i, int \_x, int \_y)

{

super(par, i, \_x, \_y);

cost = 1000;

}

@Override

public void update(int delta, GameContainer cont)

{

if (collided)

if (cont.getInput().isMousePressed(0))

PlayerStats.setAddTower(new Lightmage(cont.getInput().getMouseX(), cont.getInput().getMouseY()));

}

}

//класс реализует существо "маг"

public abstract class Mage extends Tower

{

Animation left, right, top, bot;

int fireCounter;

{

range = 200;

fireRate = 30;

height = 53;

width = 43;

fireCounter = 0;

cost = 1000;

}

public Mage(int \_x, int \_y, String spr)

{

x = \_x;

y = \_y;

Image[] lft, rht, tp, bt;

try

{

SpriteSheet ss = new SpriteSheet(spr, 43, 53, 1);

tp = new Image[5]; //инициализация изображений

bt = new Image[5]; //спрайта

rht = new Image[5];

lft = new Image[5];

for (int i = 0; i < 5; i++)

{

tp[i] = ss.getSubImage(0, i);

bt[i] = ss.getSubImage(3, i);

rht[i] = ss.getSubImage(1, i);

lft[i] = ss.getSubImage(2, i);

}

//создание спрайтов мага

left = new Animation(lft, 100, false);

right = new Animation(rht, 100, false);

top = new Animation(tp, 100, false);

bot = new Animation(bt, 100, false);

sprite = bot;

}

catch (SlickException e)

{

// TODO Auto-generated catch block

e.printStackTrace();

}

}

public void computeDirection()

{

if (target == null)

return;

double a = target.getX() - x; //Координата x вектора, указывающего на атакуемое существо

double b = target.getY() - y; //координата y этого вектора

//скалярное произведение указанного вектора и орта будет равно координате x

double angle = Math.acos(a/(Math.sqrt(a\*a+b\*b))); //угол между ортом вдоль оси x и вектором существа

sprite = bot;

//System.out.println(angle);

if (angle >= 0 && angle <= Math.PI/4 || angle >= 7\*Math.PI/4 && angle <= 2\*Math.PI)

{

sprite = right;

return;

}

else if (angle >= Math.PI/4 && angle <= 3\*Math.PI/4)

{

if (b > 0)

sprite = bot;

else

sprite = top;

return;

}

else if (angle >= 3\*Math.PI/4 && angle <= 5\*Math.PI/4)

{

sprite = left;

return;

}

}

@Override

public void update()

{

findTarget();

if (target == null)

sprite.setCurrentFrame(0);

if (fireCounter == 0)

{

if (target != null)

{

if (sprite.getFrame() == 4)

{

fire();

fireCounter = fireRate;

sprite.setCurrentFrame(0);

}

sprite.update(10);

}

}

else

{

fireCounter--;

}

}

@Override

public void render()

{

computeDirection();

sprite.draw(x,y);

}

}

//класс реализует состояние игры "главное меню"

public class MainMenuState extends BasicGameState

{

Image background = null;

private final Integer nItems = 2;

private Integer itemActive = 0;

MenuItem[] items = new MenuItem[nItems];

@Override

public void init(GameContainer container, StateBasedGame game) throws SlickException

{

background = new Image("//data//MainMenuBackground.png");

items[0] = new MenuItem("//data//NewGameItem.png",100,100,true);

items[1] = new MenuItem("//data//ExitItem.png",100,200,false);

}

@Override

public void update(GameContainer container, StateBasedGame game, int delta) throws SlickException

{

if (container.getInput().isKeyPressed(Input.KEY\_DOWN))

{

items[itemActive].setActive(false);

if (itemActive == nItems-1)

itemActive = 0;

else

itemActive++;

items[itemActive].setActive(true);

}

if (container.getInput().isKeyPressed(Input.KEY\_UP))

{

items[itemActive].setActive(false);

if (itemActive == 0)

itemActive = nItems-1;

else

itemActive--;

items[itemActive].setActive(true);

}

if (container.getInput().isKeyPressed(Input.KEY\_ENTER))

{

switch (itemActive)

{

case 0: game.enterState(1, new FadeOutTransition(), new FadeInTransition()); break;

case 1: container.exit(); break;

}

}

}

@Override

public void render(GameContainer container, StateBasedGame game, Graphics g) throws SlickException

{

background.draw();

for (int i = 0; i < nItems; i++)

items[i].render(g);

}

@Override

public int getID()

{

return 0;

}

}

/\*\*

\*

\* класс реализует игровую карту

\*/

public class Map

{

SpriteSheet tileSheet;

private LinkedList<Tile> tiles;

private LinkedList<DirectionTile> directionTiles;

private LinkedList<Tower> towers;

private List<Creature> creatures;

private LinkedList<Missile> missiles;

private Destination dest;

public void clear()

{

//tiles.clear();

//directionTiles.clear();

//towers.clear();

creatures.clear();

missiles.clear();

//dest = null;

}

public void setDestination(Destination dst)

{

dest = dst;

}

public void addMissile(Missile m)

{

missiles.add(m);

}

public void addTower(Tower t)

{

towers.add(t);

t.setMap(this);

}

//возвращает true если прямоугольник заданной ширины и высоты

//пересекается с любым объектом на карте

public boolean isCollide(int x, int y, int h, int w)

{

Rectangle a, b;

a = new Rectangle(x,y,w,h);

b = new Rectangle();

for (DirectionTile t: directionTiles)

{

b.setBounds(t.getX(), t.getY(), 32, 32);

if (a.intersects(b))

{

return true;

}

}

for (Tower t: towers)

{

b.setBounds((int)t.getX(),(int) t.getY(), t.getWidth(), t.getHeight());

if (a.intersects(b))

{

return true;

}

}

return false;

}

//возвращает указатель на первое прпавшееся существо, находящееся

//в кругу заданного радиуса с центром в точке x, y

public Creature getCreature(float x, float y, float r)

{

float dif1;

float dif2;

for (Creature c : creatures)

{

dif1 = (c.getX()-x);

dif2 = (c.getY()-y);

if ((dif1\*dif1 + dif2\*dif2) <= r\*r)

return c;

}

return null;

}

//возвращает незамороженное существо по тому же принципу что и прошлый

//метод

public Creature getNotColdedCreature(float x, float y, float r)

{

float dif1;

float dif2;

for (Creature c : creatures)

{

if (!c.isColded())

{

dif1 = (c.getX()-x);

dif2 = (c.getY()-y);

if ((dif1\*dif1 + dif2\*dif2) <= r\*r)

return c;

}

}

return null;

}

//проверяет, находится ли сущство внутри круга заданного радиуса

public boolean isCreatureInRadius(Creature c, float x, float y, float r)

{

if (c == null)

return false;

float dif1 = (c.getX()-x);

float dif2 = (c.getY()-y);

if ((dif1\*dif1 + dif2\*dif2) <= r\*r)

return true;

else return false;

}

//добавляет существо на карту

public void addCreature(Creature c)

{

creatures.add(c);

}

//добавляет тайл на карту

public void addTile(Tile t)

{

tiles.add(t);

}

//добавляет тайл направления на карту

public void addDirectionTile(DirectionTile dt)

{

directionTiles.add(dt);

}

public Map(LinkedList<Tile> tls, LinkedList<DirectionTile> dts) throws SlickException

{

tiles = tls;

directionTiles = dts;

creatures = Collections.synchronizedList(new LinkedList<Creature>());

towers = new LinkedList<Tower>();

missiles = new LinkedList<Missile>();

//addMissile(new Arrow(1,1,1,1));

}

public DirectionTile getTile(float x, float y)

{

for (DirectionTile t: directionTiles)

{

if ((x >= t.getX()) && (x <= t.getX()+32) && (y >= t.getY()) && (y <= t.getY()+32))

return t;

}

return null;

}

public void update(int delta)

{

//обработка состояния коллекции существ

Creature creatureForDelete = null;

//позже можно засунуть этот цикл во внутрь следующего

for (Creature c : creatures)

{

c.update(delta);

//пометка удаляемого существа

if (c.isDead())

creatureForDelete = c;

}

//и удаление его из коллекции

creatures.remove(creatureForDelete);

//обработка состояния коллекции снарядов

//проверка столкновений осуществляется объектом

//класса Missile в абстрактном методе

//damageTarget

Missile missileForDelete = null;

for (Missile m : missiles)

{

//обновление состояния

m.update(delta);

if (m.canBeDeleted())

missileForDelete = m;

}

missiles.remove(missileForDelete);

//обработка состояния коллекции башен

for (Tower t : towers)

t.update();

dest.update(delta);

}

public void render(Graphics g)

{

for (Tile c : tiles)

{

c.render(g);

}

for (DirectionTile c : directionTiles)

{

c.render(g);

}

for (Tower c : towers)

{

c.render();

}

for (Creature c : creatures)

{

c.render();

}

for (Missile m : missiles)

{

m.render();

}

if (dest != null)

{

dest.render(g);

}

}

}

//класс реализует кнопку добавления смотровой башни

public class WatchTowerButton extends GameMenuButton {

public WatchTowerButton(GameMenu par, Image i, int \_x, int \_y)

{

super(par, i, \_x, \_y);

cost = 110;

}

@Override

public void update(int delta, GameContainer cont)

{

if (collided)

if (cont.getInput().isMousePressed(0))

PlayerStats.setAddTower(new Watchtower(cont.getInput().getMouseX(), cont.getInput().getMouseY()));

}

}