**Лабораторная работа №3**

**Задание 1.**

Требовалось рассчитать значение нанесенного урона по DPS – метод был реализован в классе GameLogic:

public static int getDPS(Creatures creature, Damager damager) {  
 int atk = creature.getPowerAttack();  
 int weaponDamage = damager.damage();  
 int aps = damager.getAps();  
  
 return (atk + weaponDamage) \* aps;  
}

Так же в этом классе были созданы статистические методы для введения боя:

public static void battle (Mob mob) {  
 while (!*player*.isDead() && !mob.isDead()) {  
 GameLogic.*fight*(*player*, mob);  
 if (!mob.isDead()) {  
 GameLogic.*fight*(mob, *player*);  
 }  
 else {  
 *player*.addAll(mob.dropLoot());  
 }  
 }  
  
 System.*out*.println(MessageFormat.*format*("{0} {1} is ",  
 *player*.getClass().getSimpleName(), *player*.getName()) +  
 (*player*.isDead() ? "DEAD" : "WIN"));  
  
}

Методы по созданию различных существ и различной экипировки были вынесены в отдельный класс Creation, с которым Вы можете ознакомится в разделе **Приложение**.

Игроку и мобам было добавлено поле EquipmentCell<Armor> armor , которое предусматривает возможность снабжения броней существо. Для брони был создан отдельный класс под названием Armor (см. Приложение). Отличительной чертой брони от оружия являются поле defence – уровень защиты и отсутствие методов нанесения урона.   
В класс Weapon было добавлено поле APS – количество раз нанесенного урона за одну атаку. А в интерфейс Damager был внесен метод getAPS(), отвечающий за то, сколько раз урона может нанести оружие за одну атаку.

Был создан класс Cell (см. Приложение), который представляет из себя клетку для поля. Класс отвечает всем условиям задания и содержит в себе следующие обязательные методы:

* Поле объекта класса Position, которое описывает пространственные координаты клетки на поле;
* Динамический массив объектов GameObject - объекты на клетке;
* Метод для получения текущего объекта на клетке;
* Метод размещения объекта на клетке;
* Метод для удаления объекта с клетки (таким образом, чтобы он не дублировался);

Так же был создан класс Position (см. Приложение), который своим видом представляет координаты для клеток, то есть для объектов класса Cell, и имеет всего два приватных поля x и y и методы работы с ними.

Игровое поле решили реализовать через класс Map (см. Приложение). Этот класс содержит в себе динамический массив клеток.

Для передвижения мобов был создан контроллер CreatureController:

* мобы перемещаются по игровому порядку автоматические, то есть в рандомном направление на одну клетку;
* если моб пришел на клетку, где есть другие мобы, то включается режим ExtraPower, который поднимает силу каждого моба на 1 ед. Когда моб покидает эту клетку, то данный режим отключается и возвращает силу существа к прежнему значению;
* Если моб попадает на клетку, где есть игрок, то включается режим ожидания – моб ждет своей очереди сразиться с игроком;
* Когда потоку отдали сигнал для возобновления выполнения, то проверяется жив ли моб после встречи с игроком – если нет, то с клетки, на которой он находится, удаляется данное существо, а потом происходит выход из потока, если да, то продолжает привычное существование до дальнейших указаний.
* Контроллер убивается либо после поражения моба либо по окончанию игры.

Для передвижения игрока был создан отдельный контроллер PlayerController, так как все-таки характер поведения и роль в игре отличают нашего главного героя (игрока) от мобов. Хоть контроллеры похожи, но есть отличительная черта:

* Когда игрок попадает на клетку с мобами, происходит выбор самого сильного из противников (если моб один, то выбор очевиден);
* Происходит битва – если игрок победил, то вызывается контроллер пораженного моба для возобновления действий потока, а затем выбирается следующий по силе моб. Если игрок проиграл, то он выходит из потока, в классе GameLoop очищаются потоки мобов и игра заканчивается.

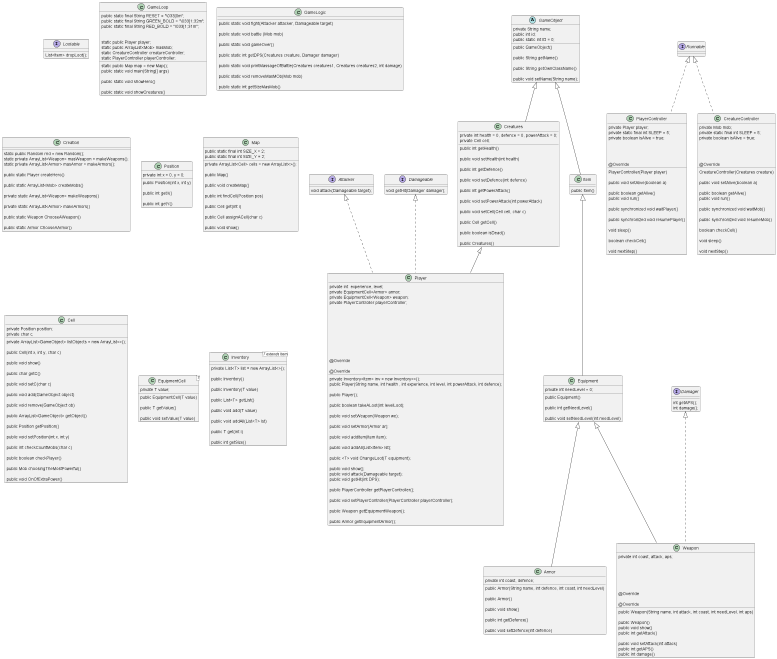
Был улучшен метод по сбору лута с мертвого моба – если экипировка пораженного противника луче, то мы одеваем его снаряжение на себя, а свою экипировку кладем в мешочек с остальным инвентарем. Методы написаны в классе Player:

public boolean takeALoot(int levelLoot) {  
 if (level >= levelLoot) {  
 System.*out*.println("Персонаж может взять оружие!\n");  
 return true;  
 } else {  
 System.*out*.println("Персонаж не может взять оружие!\n");  
 return false;  
 }  
}

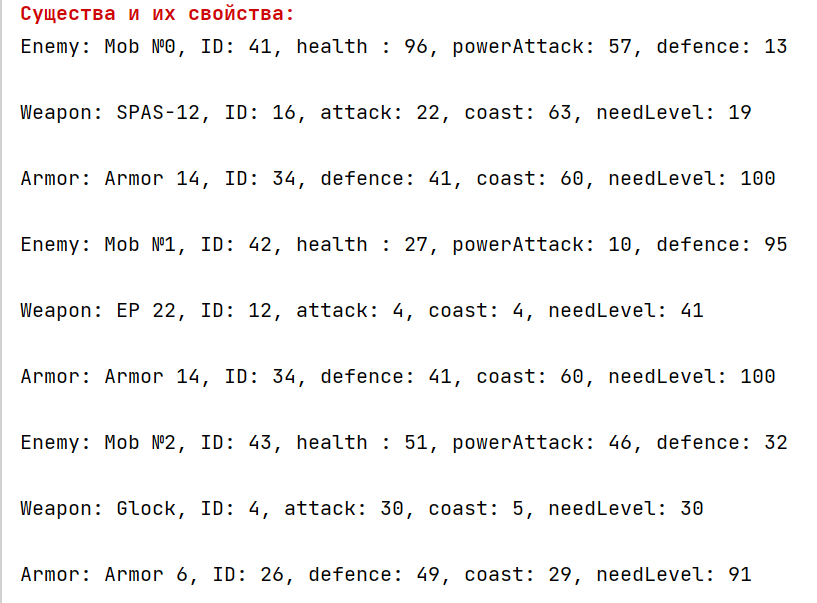
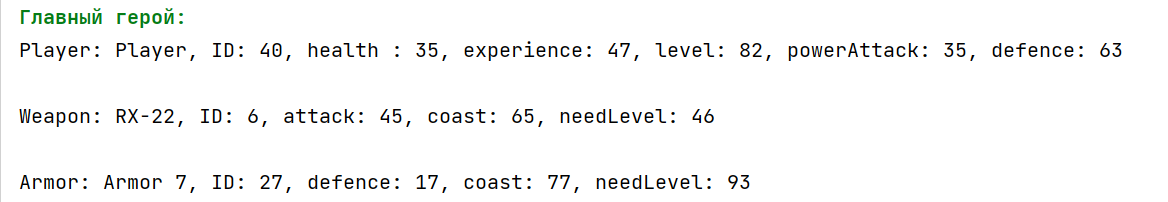
public void addAll(List<Item> lst) {  
 inv.addAll(lst);  
  
 int size = inv.getSize();  
 ChangeLoot(inv.get(size - 2));  
 ChangeLoot(inv.get(size - 1));  
}

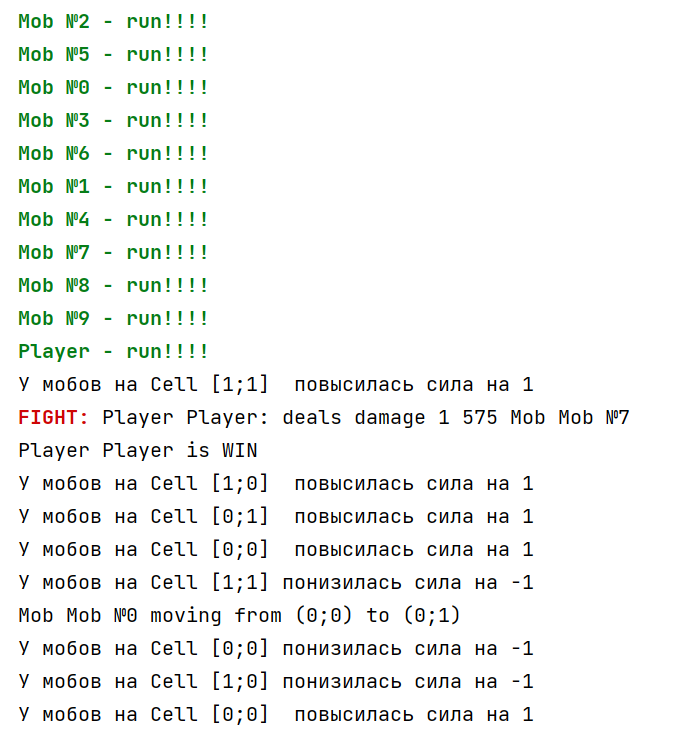
public <T> void ChangeLoot(T equipment) {  
 Class clazz = equipment.getClass();  
 if(clazz == Weapon.class.getClass()) {  
 Weapon weapon = (Weapon) equipment;  
 if (weapon.getAPS() \* weapon.getAPS()  
 > this.weapon.getValue().getAPS() \* this.weapon.getValue().getAttack()  
 && takeALoot(weapon.getNeedLevel())) {  
 this.weapon.setValue(weapon);  
 }  
 }  
 else if(clazz == Armor.class.getClass()) {  
 Armor armor = (Armor) equipment;  
 if (armor.getDefence() > this.armor.getValue().getDefence()  
 && takeALoot(armor.getNeedLevel())) {  
 this.armor.setValue(armor);  
 }  
 }  
}

**Диаграмма:**

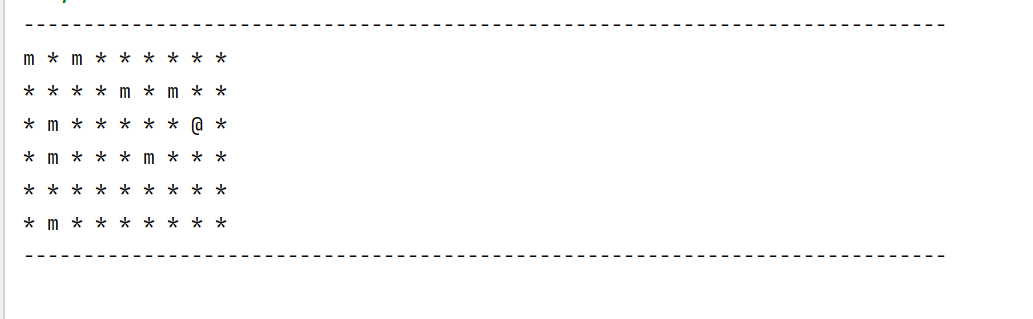


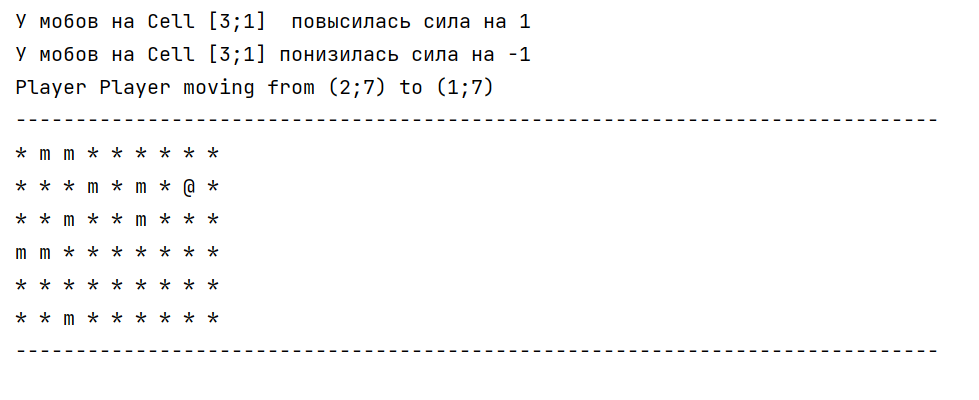
**Вывод программы:**

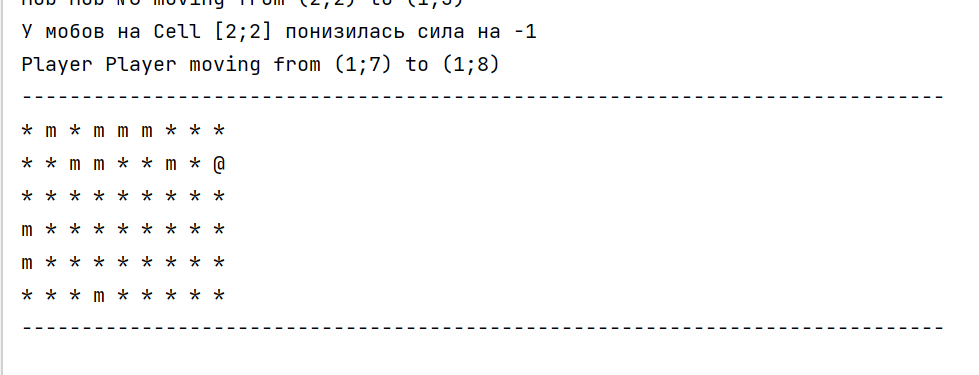
  
…………..Вывод 10 мобов…………..



…………..Работа приложения: сражения, движения по клеткам…………..









**Приложения.**

**package Field.**

**Class Cell**

public class Cell {  
  
 private Position position;  
 private char symbol;  
 private ArrayList<GameObject> listObjects = new ArrayList<>();

private int countMobsOnCell = 0;  
 private int countPlayerOnCell = 0;

public Cell(int x, int y, char symbol) {  
 position = new Position(x, y);  
 this.symbol = symbol;  
 }  
  
 public void show() {  
 System.*out*.print(symbol);  
 }  
  
 public char getSymbol() {  
 return symbol;  
 }  
  
 public void setSymbol(char symbol) {  
 this.symbol = symbol;  
 }  
  
 public void add(GameObject object) {  
 listObjects.add(object);  
 if(object.getClass().equals(Mob.class))  
 countMobsOnCell += 1;  
 if(object.getClass().equals(Player.class))  
 countPlayerOnCell += 1;  
}  
  
public void remove(GameObject object) {  
 if(object.getClass().equals(Player.class))  
 countPlayerOnCell -= 1;  
 else if(object.getClass().equals(Mob.class))  
 countMobsOnCell -= 1;  
 listObjects.remove(object);  
  
 if(countPlayerOnCell > 0) symbol = '@';  
 else if(countMobsOnCell > 0) symbol = 'm';  
 else symbol = '\*';  
  
 if(object.getClass().equals(Mob.class)) {  
 if(((Mob) object).getExtraPower()) {  
 OnOfExtraPower();  
 }  
 }  
}

public ArrayList<GameObject> getObject() {  
 return listObjects;  
 }  
  
 public Position getPosition() {  
 return position;  
 }  
  
 public void setPosition(int x, int y) {  
 this.position = new Position(x, y);  
 }  
  
 public int checkCountMobs() {  
 return countMobsOnCell;  
}  
  
public int checkPlayer() {  
 return countPlayerOnCell;  
}

public Mob choosingTheMostPowerful() {  
 Mob mob = null;  
 for(GameObject ob: listObjects) {  
 if(ob.getClass().equals(Mob.class)) {  
 if(mob == null) {  
 mob = (Mob) ob;  
 continue;  
 }  
 if(GameLogic.*getDPS*((Mob) ob, ((Mob) ob).getEquipmentWeapon()) >  
 GameLogic.*getDPS*(mob, mob.getEquipmentWeapon()) ) {  
 mob = (Mob) ob;  
 }  
 }  
 }  
 return mob;  
 }  
  
 public void OnOfExtraPower() {  
 boolean flag = false;  
 for(GameObject ob: listObjects) {  
 if(ob.getClass().equals(Mob.class)) {  
 ((Mob) ob).OnOffExtraPower();flag = ((Mob) ob).getExtraPower();  
 }  
 }  
 System.*out*.println(MessageFormat.*format*("У мобов на {0} [{1};{2}] ",  
 this.getClass().getSimpleName(), getPosition().getX(), getPosition().getY()) +  
 (flag ? " повысилась сила на 1" : "понизилась сила на -1"));  
 }  
}

**class Map**

public class Map {  
  
 public static int *SIZE\_X* = 0;  
 public static int *SIZE\_Y* = 0;  
  
 private ArrayList<Cell> cells = new ArrayList<>();  
  
 public Map() {  
 *SIZE\_Y* = *rnd*.nextInt(10) + 1;  
 *SIZE\_X* = *rnd*.nextInt(10) + 1;  
 createMap();  
 }  
  
 public void createMap() {  
 for(int i = 0; i < *SIZE\_X*; i++) {  
 for(int j = 0; j < *SIZE\_Y*; j++) {  
 cells.add(new Cell(i, j, '\*'));  
 }  
 }  
 }  
  
 public int findCell(Position pos) {  
 for(int i = 0; i < cells.size(); ++i) {  
 Position position = cells.get(i).getPosition();  
 if(position.getX() == pos.getX() && position.getY() == pos.getY()) {  
 return i;  
 }  
 }  
 return -1;  
 }  
  
 public Cell get(int i) {  
 return cells.get(i);  
 }  
  
 public Cell assignACell(char symbol) {  
 int i = *rnd*.nextInt(cells.size());  
 cells.get(i).setSymbol(symbol);  
 return cells.get(i);  
 }  
  
 public void show() {  
 System.*out*.println("-----------------------------------------------------------------------------");  
 int i = 0;  
 while(i < cells.size()) {  
 for (int j = 0; j < *SIZE\_Y*; j++, i++) {  
 System.*out*.print(cells.get(i).getSymbol() + " ");  
 }  
 System.*out*.println();  
 }  
 System.*out*.print("-----------------------------------------------------------------------------");  
 System.*out*.println("\n\n");  
 }  
}

**class Position**

public class Position {  
  
 private int x = 0, y = 0;  
  
 public Position(int x, int y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public int getX() {  
 return x;  
 }  
  
 public int getY() {  
 return y;  
 }  
}

**package Logic**

**class Creation**

public class Creation {  
  
 static public Random *rnd* = new Random();  
 static private ArrayList<Weapon> *masWeapon* = *makeWeapons*();  
 static private ArrayList<Armor> *masArmor* = *makeArmors*();  
  
 public static Player createHero() {  
 String name = "Player";  
 Weapon weapon = *ChooseAWeapon*();  
 Armor armor = *ChooseAArmor*();  
 int hp = *rnd*.nextInt((100) + 1);  
 int pAttack = *rnd*.nextInt((100) + 1);  
 int def = *rnd*.nextInt((100) + 1);  
 int exp = *rnd*.nextInt((100) + 1);  
 int level = *rnd*.nextInt((100) + 1);  
  
 Player player = new Player(name, hp, exp, level, pAttack, def);  
 player.setArmor(armor);  
 player.setWeapon(weapon);  
  
 return player;  
 }  
  
 public static ArrayList<Mob> createMobs() {  
 ArrayList<Mob> mobs = new ArrayList<>();  
 for (int i = 0; i < 10; i++) {  
 Weapon weapon = *masWeapon*.get(*rnd*.nextInt(20));  
 Armor armor = *masArmor*.get(*rnd*.nextInt(20));  
 String name = "Mob №" + i;  
 int hp = *rnd*.nextInt((100) + 1);  
 int attack = *rnd*.nextInt((100) + 1);  
 int defence = *rnd*.nextInt((100) + 1);  
  
 mobs.add(new Mob(name, hp, attack, defence));  
 mobs.get(i).setWeapon(weapon);  
 mobs.get(i).setArmor(armor);  
 }  
 return mobs;  
 }  
  
 private static ArrayList<Weapon> makeWeapons() {  
 String[] names = {"Glock", "AK-47", "RX-22", "EP 22", "SPAS-12", "Stinger", "AT4", "RPG-7", "M9", "TMP"};  
 ArrayList<Weapon> weapons = new ArrayList<>();  
 for (int i = 0; i < 20; i++) {  
 String name = names[*rnd*.nextInt(10)];  
 int attack = *rnd*.nextInt((100) + 1);  
 int coast = *rnd*.nextInt((100) + 1);  
 int needLevel = *rnd*.nextInt((100) + 1);  
 weapons.add(new Weapon(name, attack, coast, needLevel, 1));  
 }  
 return weapons;  
 }  
  
 private static ArrayList<Armor> makeArmors() {  
 ArrayList<Armor> armors = new ArrayList<>();  
 for (int i = 0; i < 20; i++) {  
 String name = "Armor " + i;  
 int def = *rnd*.nextInt((100) + 1);  
 int coast = *rnd*.nextInt((100) + 1);  
 int needLevel = *rnd*.nextInt((100) + 1);  
 armors.add(new Armor(name, def, coast, needLevel));  
 }  
 return armors;  
 }  
  
 public static Weapon ChooseAWeapon() {  
 return *masWeapon*.get(*rnd*.nextInt(20));  
 }  
  
 public static Armor ChooseAArmor() {  
 return *masArmor*.get(*rnd*.nextInt(20));  
 }}

**class GameLogic**

public class GameLogic {  
  
 public static void fight(Attacker attacker, Damageable target) {  
 attacker.attack(target);  
 }  
  
 public static void battle (Mob mob) {  
 while (!*player*.isDead() && !mob.isDead()) {  
 GameLogic.*fight*(*player*, mob);  
 if (!mob.isDead()) {  
 GameLogic.*fight*(mob, *player*);  
 }  
 else {  
 *player*.addAll(mob.dropLoot());  
 }  
 }  
  
 System.*out*.println(MessageFormat.*format*("{0} {1} is ",  
 *player*.getClass().getSimpleName(), *player*.getName()) +  
 (*player*.isDead() ? "DEAD" : "WIN"));  
  
 }  
  
 public static void gameOver(){  
 System.*out*.println(*GREEN\_BOLD* + "END GAME!" + *RESET*);  
 }  
  
 public static int getDPS(Creatures creature, Damager damager) {  
 int atk = creature.getPowerAttack();  
 int weaponDamage = damager.damage();  
 int aps = damager.getAPS();  
  
 return (atk + weaponDamage) \* aps;  
 }  
  
 public static void printMassageOfBattle(Creatures creatures1, Creatures creatures2, int damage) {  
 String ownName1 = creatures1.getOwnClassName();  
 String ownName2 = creatures2.getOwnClassName();  
 String name1 = creatures1.getName();  
 String name2 = creatures2.getName();  
  
 String string = MessageFormat.*format*("{0} {1}: deals damage {2} {3} {4}",  
 ownName1, name1, damage, ownName2, name2);  
 System.*out*.print(*RED\_BOLD* + "FIGHT: " + *RESET*);  
 System.*out*.println(string);  
 }  
  
 public static void removeMasMOb(Mob mob) {  
 *masMob*.remove(mob);  
 }  
  
 public static int getSizeMasMob() {  
 return *masMob*.size();  
 }  
}

**class GameLoop**

public class GameLoop {  
  
 public static final String *RESET* = "\033[0m";  
 public static final String *GREEN\_BOLD* = "\033[1;32m";  
 public static final String *RED\_BOLD* = "\033[1;31m";  
  
  
 static public Player *player*;  
 static public ArrayList<Mob> *masMob*;  
 static public Map *map* = new Map();  
 static CreatureController *creatureController*;  
 static PlayerController *playerController*;  
  
 public static void main(String[] args) {  
  
 Thread thread;  
  
 *player* = Creation.*createHero*();  
 *masMob* = Creation.*createMobs*();  
  
 *showHero*();  
 *showCreatures*();  
  
 for(Mob mob: *masMob*) {  
 mob.setCell(*map*.assignACell('m'), 'm');  
 mob.getCell().add(mob);  
 *creatureController* = new CreatureController(mob);  
 thread = new Thread(*creatureController*);  
 mob.setCreatureController(*creatureController*);  
 thread.start();  
 }  
  
 *player*.setCell(*map*.assignACell('@'), '@');  
 *player*.getCell().add(*player*);  
 *playerController* = new PlayerController(*player*);  
 thread = new Thread(*playerController*);  
 *player*.setPlayerController(*playerController*);  
 thread.start();  
  
 try {  
 thread.join();  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
  
 for(Mob mob: *masMob*) {  
 System.*out*.print(mob.getName() + " ");  
 mob.getCreatureController().setAlive(false);  
 }  
  
 GameLogic.*gameOver*();  
 }  
  
 public static void showHero() {  
 System.*out*.print(*GREEN\_BOLD* + "Главный герой: " + *RESET*);  
 *player*.show();  
 }  
  
 public static void showCreatures() {  
 System.*out*.print(*RED\_BOLD* + "Существа и их свойства: " + *RESET*);  
 for (Mob m : *masMob*) {  
 m.show();  
 }  
 }  
}

**class CreatureController**

public class CreatureController implements Runnable{  
  
 private Mob mob;  
 private static final int *SLEEP* = 5;  
 private boolean isAlive = true;  
  
 CreatureController(Creatures creature) {  
 mob = (Mob) creature;  
 }  
  
 public void setAlive(boolean live) {  
 isAlive = live;  
 return;  
 }  
  
 public boolean getAlive() {  
 return isAlive;  
 }  
  
 @Override  
 public void run() {  
 System.*out*.println(*GREEN\_BOLD* + mob.getName() + " - run!!!!" + *RESET*);  
 while(isAlive) {  
 sleep();  
 if(checkCell()) {  
 waitMob();  
 }  
 if(mob.isDead()) {  
 isAlive = false;  
 mob.getCell().remove(mob);  
 }  
 else {  
 nextStep();  
 }  
 }  
 System.*out*.println(*RED\_BOLD* + mob.getName() + " is DEAD" + *RESET*);  
 GameLogic.*removeMasMOb*(mob);  
 }  
  
 public synchronized void waitMob() {  
 try {  
 wait();  
 }  
 catch (InterruptedException e) {  
 System.*out*.println(mob.getName() + " - Error wait!!!!");  
 }  
 }  
  
 public synchronized void resumeMob() {  
 notify();  
 }  
  
 boolean checkCell() {  
 if(mob.getCell().checkCountMobs() > 1) {  
 mob.getCell().OnOfExtraPower();  
 }  
 return mob.getCell().checkPlayer() > 0;  
 }  
  
 void sleep() {  
 try {  
 TimeUnit.*SECONDS*.sleep(*SLEEP*);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 void nextStep() {  
 Position oldPos = mob.getCell().getPosition();  
 int x = oldPos.getX();  
 int y = oldPos.getY();  
 if (x > 0 && x < *SIZE\_X* - 1) x += *rnd*.nextInt(3) - 1;  
 else if (x == 0) x += *rnd*.nextInt(2);  
 else if (x == *SIZE\_X* - 1) x += *rnd*.nextInt(2) - 1;  
  
 if (y > 0 && y < *SIZE\_Y* - 1) y += *rnd*.nextInt(3) - 1;  
 else if (y == 0) y += *rnd*.nextInt(2);  
 else if (y == *SIZE\_Y* - 1) y += *rnd*.nextInt(2) - 1;  
  
 Position newPos = new Position(x, y);  
 if (newPos.getY() == oldPos.getY() && newPos.getX() == oldPos.getX());  
 else {  
 System.*out*.println(MessageFormat.*format*("{0} {1} moving from ({2};{3}) to ({4};{5})",  
 mob.getClass().getSimpleName(), mob.getName(), oldPos.getX(),  
 oldPos.getY(), newPos.getX(), newPos.getY()));  
 mob.getCell().remove(mob);  
 int i = GameLoop.*map*.findCell(newPos);  
 if(i == -1) return;  
 mob.setCell(GameLoop.*map*.get(i), 'm');  
 mob.getCell().add(mob);  
 }  
 }  
}

**class PlayerController**

public class PlayerController implements Runnable {  
  
 private Player player;  
 private static final int *SLEEP* = 5;  
 private boolean isAlive = true;  
  
 PlayerController(Player player) {  
 this.player = player;  
 }  
  
 public void setAlive(boolean live) {  
 isAlive = live;  
 }  
  
 public boolean getAlive() {  
 return isAlive;  
 }  
  
 @Override  
 public void run() {  
 System.*out*.println(*GREEN\_BOLD* + player.getName() + " - run!!!!" + *RESET*);  
 while(isAlive) {  
 sleep();  
 while(!player.isDead()) {  
 if(!checkCell()) break;  
 Mob mob = player.getCell().choosingTheMostPowerful();  
 GameLogic.*battle*(mob);  
 if(mob.isDead()) {  
 mob.getCreatureController().resumeMob();  
 sleep();  
 }  
 }  
 if(player.isDead() || GameLogic.*getSizeMasMob*() == 0) {  
 isAlive = false;  
 player.getCell().remove(player);  
 }  
 else {  
 nextStep();

*map*.show();

}  
 }  
 System.*out*.println(*RED\_BOLD* + player.getName() + " is DEAD" + *RESET*);  
 }  
  
 public synchronized void waitPlayer() {  
 try {  
 wait();  
 }  
 catch (InterruptedException e) {  
 System.*out*.println(player.getName() + " - Error wait!!!!");  
 }  
 }  
  
 public synchronized void resumePlayer() {  
 notify();  
 }  
  
 void sleep() {  
 try {  
 TimeUnit.*SECONDS*.sleep(*SLEEP*);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 boolean checkCell() {  
 return player.getCell().checkCountMobs() > 0;  
 }  
  
 void nextStep() {  
 Position oldPos = player.getCell().getPosition();  
 int x = oldPos.getX();  
 int y = oldPos.getY();  
 if (x > 0 && x < *SIZE\_X* - 1) x += *rnd*.nextInt(3) - 1;  
 else if (x == 0) x += *rnd*.nextInt(2);  
 else if (x == *SIZE\_X* - 1) x -= *rnd*.nextInt(2);  
  
 if (y > 0 && y < *SIZE\_Y* - 1) y += *rnd*.nextInt(3) - 1;  
 else if (y == 0) y += *rnd*.nextInt(2);  
 else if (y == *SIZE\_Y* - 1) y -= *rnd*.nextInt(2);  
  
 Position newPos = new Position(x, y);  
 if (newPos.getY() == oldPos.getY() && newPos.getX() == oldPos.getX());  
 else {  
 System.*out*.println(MessageFormat.*format*("{0} {1} moving from ({2};{3}) to ({4};{5})",  
 player.getClass().getSimpleName(), player.getName(), oldPos.getX(),  
 oldPos.getY(), newPos.getX(), newPos.getY()));  
 player.getCell().remove(player);  
 int i = GameLoop.*map*.findCell(newPos);  
 if(i == -1) return;  
 player.setCell(GameLoop.*map*.get(i), '@');  
 player.getCell().add(player);  
 }  
 }  
}

**package Loot**

**class Item**

public class Item extends GameObject {  
  
 public Item(String name) {  
 super(name);  
 }  
}

**class Inventory**

public class Inventory<T extends Item> {  
  
 private List<T> list = new ArrayList<>();  
  
 public Inventory() {}  
  
 public Inventory(T value) {  
 list.add(value);  
 }  
  
 public List<T> getList() { return list; }  
  
 public void add(T value) {  
 list.add(value);  
 }  
  
 public void addAll(List<T> lst) {  
 list.addAll(lst);  
 }  
  
 public T get(int i) {  
 return list.get(i);  
 }  
  
 public int getSize() {  
 return list.size();  
 }  
}

**class Equipment**

public class Equipment extends Item {  
  
 private int needLevel = 0;  
  
 public Equipment(String name) {  
 super(name);  
 }  
 public int getNeedLevel() {  
 return needLevel;  
 }  
  
 public void setNeedLevel(int needLevel) {  
 this.needLevel = needLevel;  
 }  
}

**class EquipmentCell**

public class EquipmentCell<T> {  
  
 private T value;  
  
 public EquipmentCell(T value) {  
 this.value = value;  
 }  
  
 public T getValue() { return value; }  
  
 public void setValue(T value) {  
 this.value = value;  
 }  
}

**class Weapon**

public class Weapon extends Equipment implements Damager {  
  
 private int coast, attack, aps;  
  
 public Weapon(String name, int attack, int coast, int needLevel, int aps) {  
 super(name);  
 this.attack = attack;  
 this.coast = coast;  
 this.aps = aps;  
 setNeedLevel(needLevel);  
 }  
  
 public Weapon() {  
 super("Weapon");  
 attack = coast = 0;  
 }  
  
  
 public void show() {  
 System.*out*.println(  
 "\nWeapon: " + getName() + ", ID: " + id + ", attack: " + attack +  
 ", coast: " + coast + ", needLevel: " + getNeedLevel()  
 );  
 }  
  
 public int getAttack() {  
 return attack;  
 }  
  
 public void setAttack(int attack) {  
 this.attack = attack;  
 }  
  
 @Override  
 public int getAPS() {  
 return aps;  
 }  
  
 @Override  
 public int damage() {  
 return attack;  
 }  
}

**class Armor**

public class Armor extends Equipment {  
  
 private int coast, defence;  
  
 public Armor(String name, int defence, int coast, int needLevel) {  
 super(name);   
 this.defence = defence;  
 this.coast = coast;  
 setNeedLevel(needLevel);  
 }  
  
 public Armor() {  
 super("Weapon");  
 defence = coast = 0;  
 }  
  
  
 public void show() {  
 System.*out*.println(  
 "\nArmor: " + getName() + ", ID: " + id + ", defence: " + defence +  
 ", coast: " + coast + ", needLevel: " + getNeedLevel()  
 );  
 }  
  
 public int getDefence() {  
 return defence;  
 }  
  
 public void setDefence(int defence) {  
 this.defence = defence;  
 }  
}

**package Structure**

**class GameObject**

abstract public class GameObject {  
 private String name;  
 public int id;  
 public static int *ID* = 0;  
  
 public GameObject(String name) {

this.name = name;  
 id = *ID*;  
 *ID* += 1;  
 }  
  
 public String getName() {  
 return name;  
 }  
 public String getOwnClassName() {  
 return getClass().getSimpleName();  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

**class Creatures**

public class Creatures extends GameObject {  
 private int health = 0, defence = 0, powerAttack = 0;  
 private Cell cell;  
  
 public int getHealth() {  
 return health;  
 }  
  
 public void setHealth(int health) {  
 this.health = health;  
 }  
  
 public int getDefence() {  
 return defence;  
 }  
  
 public void setDefence(int defence) {  
 this.defence = defence;  
 }  
  
 public int getPowerAttack() {  
 return powerAttack;  
 }  
  
 public void setPowerAttack(int powerAttack) {  
 this.powerAttack = powerAttack;  
 }  
  
 public void setCell(Cell cell, char symbol) {  
 this.cell = cell;  
 this.cell.setC(symbol);  
 }  
  
 public Cell getCell() {  
 return cell;  
 }  
  
 public boolean isDead() {  
 return getHealth() <= 0 ;  
 }  
  
 public Creatures(String name) {  
 super(name);  
 }  
  
}

**interface Attacker**

public interface Attacker {  
 void attack(Damageable target);  
}

**interface Damageable**

public interface Damageable {  
 void getHit(int DPS);  
}

**interface Damager**

public interface Damager {  
 int getAPS();  
 int damage();  
}

**interface Lootable**

public interface Lootable {  
 List<Item> dropLoot();  
}

**class Player**

public class Player extends Creatures implements Attacker, Damageable{  
 private int experience, level;  
 EquipmentCell<Armor> armor;  
 EquipmentCell<Weapon> weapon;  
 Inventory<Item> inv = new Inventory<>();  
 private PlayerController playerController;  
  
 public Player(String name, int health , int experience, int level, int powerAttack, int defence) {  
 super(name);   
 setHealth(health);  
 setPowerAttack(powerAttack);  
 setDefence(defence);  
 this.experience = experience;  
 this.level = level;  
 }  
  
 public Player() {  
 super("Hero");  
 }  
  
 public boolean takeALoot(int levelLoot) {  
 if (level >= levelLoot) {  
 System.*out*.println("Персонаж может взять оружие!\n");  
 return true;  
 } else {  
 System.*out*.println("Персонаж не может взять оружие!\n");  
 return false;  
 }  
 }  
  
 public void setWeapon(Weapon we) {  
 weapon = new EquipmentCell<>(we);  
 addItem(we);  
 }  
  
 public void setArmor(Armor ar) {  
 armor = new EquipmentCell<>(ar);  
 addItem(ar);  
 }  
  
 public void addItem(Item item) {  
 inv.add(item);  
 }  
  
 public void addAll(List<Item> lst) {  
 inv.addAll(lst);  
  
 int size = inv.getSize();  
 ChangeLoot(inv.get(size - 2));  
 ChangeLoot(inv.get(size - 1));  
 }  
  
 public <T> void ChangeLoot(T equipment) {  
 Class clazz = equipment.getClass();  
 if(clazz == Weapon.class.getClass()) {  
 Weapon weapon = (Weapon) equipment;  
 if (weapon.getAPS() \* weapon.getAPS()  
 > this.weapon.getValue().getAPS() \* this.weapon.getValue().getAttack()  
 && takeALoot(weapon.getNeedLevel())) {  
 this.weapon.setValue(weapon);  
 }  
 }  
 else if(clazz == Armor.class.getClass()) {  
 Armor armor = (Armor) equipment;  
 if (armor.getDefence() > this.armor.getValue().getDefence()  
 && takeALoot(armor.getNeedLevel())) {  
 this.armor.setValue(armor);  
 }  
 }  
 }  
  
 public void show() {  
 System.*out*.println(  
 "\nPlayer: " + getName() + ", ID: " + id + ", health : " +  
 getHealth() + ", experience: " + experience  
 + ", level: " + level + ", powerAttack: " +  
 getPowerAttack() + ", defence: " + getDefence()  
 );  
 weapon.getValue().show();  
 armor.getValue().show();  
 }  
  
 @Override  
 public void attack(Damageable target) {  
 int DPS = GameLogic.*getDPS*(this, weapon.getValue());  
 GameLogic.*printMassageOfBattle*(this, (Creatures) target, DPS);  
 target.getHit(DPS);  
 }  
  
 @Override  
 public void getHit(int DPS) {  
 int defence = getDefence();  
 if(defence - DPS > 0)  
 setDefence(defence - DPS);  
 else {  
 setHealth(getHealth() - DPS + defence);  
 setDefence(0);  
 }  
 }  
  
 public PlayerController getPlayerController() {  
 return playerController;  
 }  
  
 public void setPlayerController(PlayerController playerController) {  
 this.playerController = playerController;  
 }  
  
 public Weapon getEquipmentWeapon() {  
 return weapon.getValue();  
 }  
  
 public Armor getEquipmentArmor() {  
 return armor.getValue();  
 }  
  
}

**class Mob**

public class Mob extends Creatures implements Lootable, Attacker, Damageable{  
 private EquipmentCell<Armor> armor;  
 private EquipmentCell<Weapon> weapon;  
 private boolean extraPower = false;  
 private CreatureController creatureController;  
  
 public Mob(String name, int health , int powerAttack, int defence) {  
 super(name);   
 setHealth(health);  
 setPowerAttack(powerAttack);  
 setDefence(defence);  
 }  
  
 public Mob() {  
 super("Enemy");  
 }  
  
 public void show() {  
 System.*out*.println(  
 "\nEnemy: " + getName() + ", ID: " + id + ", health : " + getHealth() +  
 ", powerAttack: " + getPowerAttack() + ", defence: " + getDefence()  
 );  
 weapon.getValue().show();  
 armor.getValue().show();  
 }  
  
 public CreatureController getCreatureController() {  
 return creatureController;  
 }  
  
 public void setCreatureController(CreatureController creatureController) {  
 this.creatureController = creatureController;  
 }  
  
 public void setWeapon(Weapon we) {  
 weapon = new EquipmentCell<>(we);  
 }  
  
 public void setArmor(Armor ar) {  
 armor = new EquipmentCell<>(ar);  
 }  
  
 @Override  
 public List<Item> dropLoot() {  
 List<Item> items = new ArrayList<>();  
 items.add(weapon.getValue());  
 items.add(armor.getValue());  
 return items;  
 }  
  
 @Override  
 public void attack(Damageable target) {  
 int DPS = GameLogic.*getDPS*(this, weapon.getValue());  
 GameLogic.*printMassageOfBattle*(this, (Creatures) target, DPS);  
 target.getHit(DPS);  
 }  
  
 @Override  
 public void getHit(int DPS) {  
 int defence = getDefence();  
 if(defence - DPS > 0)  
 setDefence(defence - DPS);  
 else {  
 setHealth(getHealth() - DPS + defence);  
 setDefence(0);  
 }  
 }  
  
 public void OnOffExtraPower() {  
 extraPower = !extraPower;  
 int p = getPowerAttack();  
 p += extraPower ? 1 : -1;  
 setPowerAttack(p);  
 }  
  
 public boolean getExtraPower() {  
 return extraPower;  
 }  
  
 public Weapon getEquipmentWeapon() {  
 return weapon.getValue();  
 }  
  
 public Armor getEquipmentArmor() {  
 return armor.getValue();  
 }  
  
}