Entwicklerdokumentation

Interface:

Zunächst enthält die Klasse Interface fast alle **Konstanten** (wie z.B. Richtungskonstanten, Spielfeldelemente und Konstanten der Zufallsberechnung) sowie Graphischen Informationen, die das Spiel generell benötigt, um ein Spielfeld anzeigen zu können.

Das **Menü** wird mit seinen Button geöffnet sowie erstellt, die **Button** werden deklariert und die Button-Reaktionen wird abgefragt.

Nach dem Menü, werden auch die **Level** bzw. die **Räume** hier erstellt, in ihrer Größe bestimmt und im Anschluss daran erzeugt. Eine Methode fragt dabei ab, um welches Level bzw. welchen Raum es sich handelt. Folgende Methode kümmert sich darum, wenn ein Level geschafft wurde oder man evtl. einen Raum zurück gehen möchte:

```
public static void nextRoom(){
		if (raum<2){
			raum++;
		}
		else {
			level++;
			raum=0;
			aktiveCheckpoint=level*2;
		}
	}
```

Durchläuft der Spieler einen **Checkpoint**, so zählt eine der in dieser Klassen enthaltenen Methoden um eins weiter.

Auch die Methoden „KeyPressed“ und „KeyEvent“ sind enthalten und die „**Key- Aktion**“ wird ebenfalls hier abgefangen.

Die **Spieler-** bzw. **Gegneranzahl**, sowie die Art der Spielfiguren werden erzeugt und eine Methode regelt die Aktionen der Gegner. Wird man getroffen oder trifft man selber einen der Gegner, gibt es auch hierfür Methoden in der Klasse Interface.

Während des Spiels gibt es einen sogenannten „**Storryteller**“, der den Spieler durch die Geschichte leitet. Diese Geschichte bzw. die Methode ist ebenfalls hier zu finden.

Ist zum Schluss ein Spiel gewonnen oder ggf. verloren, so erscheint entweder ein „**Gameover**“ oder eben ein „**Gewonnen**“ **Frame**. Auch die dafür notwendigen Bilder werden hier bereit gestellt.

Aktion:

In der Klasse Aktion ist die **Hauptspiellogik** enthalten. Dort wird festgelegt was zu tun ist, wenn etwas auf dem Spielfeld passiert. Das heißt, es wird zunächst **Grundsätzliches** zum Raum, Storryteller, Sound oder Checkpoint **initialisiert** und im Anschluss daran erzeugt hier eine Methode, ob sich die Spielfiguren anhand der Koordinaten **bewegt** oder **angreift**. Auch die Bewegung der Gegner ist in dieser Klasse enthalten. Zusätzlich finden die Methoden rund um unser Co-Op-Rätsel hier seine Anwendung.

Boss:

Diese Klasse regelt beinahe alles, was mit dem **Endgegner** zu tun hat. Die dort enthaltenen Methoden regeln, wo er steht und wie viele Lebenspunkte er besitzt. Ist er getroffen, so wird die Höhe des Schadens den man angerichtet hat registriert und die Lebenspunkte des Bosses werden reduziert. Zudem wird festgelegt, wie Hoch der Schaden generell überhaupt sein kann, den man ihm zufügen kann. Sinken seine Lebenspunkte auf Null, lässt eine Methode ihn sterben. Dabei blendet eine Methode das Bild Gewonnen“ ein und der Spieler erhält Erfahrungspunkte.

FigurDisplay:

Die dort enthaltenen Methoden stellen die Spielfigur mit ihren **aktuellen Eigenschaften** dar. Dazu gehört, ob man eine Rüstung trägt oder nicht. **Bewegt** sich der Spieler oder ein Gegner, so zeichnet die Methoden in der Klasse FigurDisplay die Darstellung des Bodens sowie die Darstellung der Figur an neuer Stelle.

Gegner:

Die Klasse „Gegner“ enthält zunächst alle aktuellen Eigenschaften eines Gegners und gibt diese auch in dementsprechenden Methoden zurück. Die Koordinaten werden gesetzt und es wird angegeben, in welche Richtung sich der Gegner bewegt. Er bewegt sich entweder vom Spieler abhängig oder zufällig in eine Richtung.
Greift ein Spieler an, so verliert der Gegner (abhängig von der Rüstung des Spielers) an Lebenspunkten und kann schließlich auch sterben, wenn die Anzahl seine Lebenspunkte null erreicht.
Wehrt der Gegner sich , so zeigt eine weitere Methode den Schaden des Spielers an.
Ist der Gegner besiegt, so erscheint an dessen Stelle per Zufallsrechnung ein Manatrank, HP-Trank oder Münzen bzw. Schlüssel, den der Spieler im späteren Verlauf aufsammeln kann. Zusätzlich erhält Spieler Erfahrungspunkte dabei.

Sound:

In der Klasse Sound, sorgt die angegebene Methode dafür, dass die in anderen Klassen enthaltene Musik abgespielt wird.

Spieler:

In der Klasse „Spieler“ findet man zunächst einige Konstanten und generelle Angaben werden abgefragt. Eigenschaften die im Spiel veränderbar sind, werden abgefragt und dargestellt. Bewegungen des Spielers werden registriert und im Falle eines Angriffs o.ä. wird mittels einer Methode der aktuelle Mana- und/ oder Hp-wert um einen bestimmten Schadenswert reduziert. Sind die Lebenspunkte aufgebraucht, wird die Methode „Sterben“ aufgerufen und der Spieler gelangt zum zuletzt erreichten Checkpoint.

Eine weitere Methode ist für die Erhöhung der aktuellen HP-und Manawerte im Bereich des Shops zuständig.

Bei der Methode Levelup wird der aktueller Erfahrungswert aufgeladen, wenn man z.B. einen Gegner tötet. Erreicht man ein neues Level, so erlangt man den sogenannten Schildzauber und zusätzlich erhält man ein Leben.

Beim zweiten Levelup wird zusätzlich der Manawert aufgeladen und ein weiteres Leben wird angerechnet.

Zwei weitere Methoden sind dafür verantwortlich, dass die Münzen und Schlüssel sowohl eingesammelt als auch verloren gehen können.

Zusätzlich wird der momentane HP-, Münz- sowie Schlüssel- und Manawert ausgegeben und die momentanen Werte von Farbe, Rüstung, Schadensfaktor oder Schildaufladung werden zurückgegeben.

Ebenfalls in der Klasse Spieler ist eine Methode enthalten, die überprüft, ob der Schildzauber aktiv ist und entsprechend Mana verbraucht.

Spielfelder:

In der Klasse wird das Spielfeld auf einer logischen Ebene generiert und die Textdateien werden eingelesen. Dabei sortiert das Hauptarray das Spielfeld nach Level, Raum, Spalte und Reihe.
Desweiteren existieren Arrays für Checkpoint, Startpunkt und Zielpunkt. Und neben dem Ziel und dem Checkpoint werden weitere Konstanten und Fehlertypen, wie das Fehlen von Leerzeichen, Leerzeilen oder Zeilenumbrüche, deklariert.

Ein Filechooser sorgt dafür, dass eine Spielfelddatei ausgewählt werden kann.

Ebenfalls in der Klasse enthalten sind Angaben zum Checkpoint. Hier wird dieser gesetzt und abgefragt. Auch Start- und Zielpunkt werden hier für die Level gesetzt.

Auch die Zuweisung bestimmter Spielfeldarray Werte für bestimmte Objekte, wie Mauer oder Boden, werden in dieser Klasse festgelegt.

Weitere Methoden fragen daraufhin die Spielfeldeigenschaften ab, also den Wert der Felder und setzen einen Wert auf die x und y Koordinaten.

StatDisplay:

In dieser Klasse sorgen die Methoden dafür, dass der Spieler weiss, auf welchem Stand er aktuell ist. Das geschieht, indem die entsprechenden Werte wie Mana, HP, Rüstung, Münzen oder Schlüssel am Displayrand angezeigt werden.
Auch die Eigenschaften des Gegners werden in dieser Klasse teilweise deklariert.

StdDraw:

Auf diese Klasse greifen verschiedene, beinahe alle Klassen zu. Folgende Klassen greifen auf StdDraw zu:

- Interface
- Aktion
- Boss
- FigurDisplay
- Gegner
- Spieler
- Spielfeld
- StatDisplay

Will man bestimmte Dinge innerhalb des Programms ändern, so muss man sich in folgenden Klassen orientieren:

Netzwerkmodus

– Noch nicht implementiert

Quests und Rätsel

– **Interface** (Konstante und IMG)
– Storryteller: Erzählt den Spielern indirekt, wie sie die Quests lösen können.

```
//Storyteller für 1. Quest "Versteckter Gang" (Level 1, Raum 3) (Mauerstück als
   versteckter Durchgang)
                    else if ((level==0)&(raum==2)){
                          StdDraw.picture(400,560,Aktion.WEISSIMG);
                          StdDraw.text(HOEHETEXT, BREITETEXT4,
                                       "Auch wenn es alle dementieren: In manchen
   Ausnahmesituationen hilft es,");
                          StdDraw.text(HOEHETEXT, BREITETEXT2,
                                       "auch mal mit dem Kopf durch die Wand zu
   gehen!");



   //Storyteller für 2. Quest "Schlüssel und Tor" (Level 2, Raum 2) (Sammeln von zwei
   Schlüsseln, sodass Tor aufgeht)
                    else if ((level==1)&(raum==1)){
                          StdDraw.picture(400,560,Aktion.WEISSIMG);
                          StdDraw.text(HOEHETEXT, BREITETEXT5,
                                       "Ja ich weiß, da ist ein verschlossenes Tor...
   Mich stört es da ja auch!");
                          StdDraw.text(HOEHETEXT, BREITETEXT3,
                                       "Die Trolle haben es errichtet, wer sonst! Aber
   sie haben auch die Schlüssel dafür...");
                          StdDraw.text(HOEHETEXT, BREITETEXT1,
                                       "Man braucht 2 Schlüssel, um das Tor zu öffen,
   denn es ist ein doppelt gesichertes Tor!");

                    }
```

– IMG:

```
public static final String VERSTECKTERGANGIMG = "Images\\mauer.jpg";
```

– Konstanten werden festgelegt

```
public static final int VERSTECKTERGANG = 26;
```

– Schleife: Stellt den versteckten Eingang graphisch dar.

```
      else if (Spielfeld.wertLesenBeiXY(level,raum,spalte,reihe)==VERSTECKTERGANG){
                                    StdDraw.picture(PIC1+PIC2*spalte,PIC1+PIC2*reihe,
VERSTECKTERGANGIMG);
                              }
```

– **Spielfeld** (Schleife)
– Die Schleife liest die Textdatei ein. Trifft sie dabei auf auf ein “V“ bzw. in Ascii ausgedrückt auf eine “86“, dann weiß das Programm das dort an dieser Stelle ein „VERSTECKTEREINGANG“ ist

```
else if (testChar == Interface.SECHSUNDACHZIG){

      spielfeld[level][raum][spalte][reihe]=Interface.VERSTECKTERGANG;
                                          }
```

– **Aktion**
– Schleife: Läuft der Spieler gegen die Wand (den versteckten Eingang) verschwindet die Wand und ein Weg erscheint.

```
else if
(Spielfeld.wertLesenBeiXY(aktuellesLevel,aktuellerRaum,newFigurX,newFigurY)==Interface.
VERSTECKTERGANG){
                    if (player){
                          returnArray[0]=Interface.VERSTECKTERGANG;
                          Spielfeld.wertSetzenBeiXY(aktuellesLevel, aktuellerRaum,
newFigurX, newFigurY, Interface.VERSTECKTERGANG);
                          display.figurBewegen(figurX,figurY,newFigurX,newFigurY,
farbe, schild);
                          figurX=newFigurX;
                          figurY=newFigurY;
                    }
```

Co-Op-Rätsel

– **Interface** (Konstanten und IMG)
– Storryteller: Erzählt an zwei Stellen im Spiel indirekt was die Spieler zu tun haben

```
//Storyteller für 1. Co-Op-Quest "Doppelportal" (Level 3, Raum 1)
//(beide Spieler muessen vor einem stand-here-Pfeil stehen, sodass das Portal wie auf
Knopfdruck geoeffnet werden kann)
                    else if ((level==2)&(raum==0)){
                          StdDraw.picture(400,560,Aktion.WEISSIMG);
                          StdDraw.text(HOEHETEXT, BREITETEXT5,
                                      "Manchmal macht man hier unten ganz merkwürdige
Entdeckungen...");
                          StdDraw.text(HOEHETEXT, BREITETEXT3,
```

```
                                        "Die Trolle haben anscheinend einen Faible für
Hindernisse");
                        StdDraw.text(HOEHETEXT, BREITETEXT1,
                                        " auf den Wegen... und für Knopfmechanismen!");

                }
                //Storyteller für 2. Co-Op-Quest "Portal" (Level 3, Raum 2) (nur
wenn beide Spieler das Doppelportal ansteuern, laesst es sich oeffnen)
                else if ((level==2)&(raum==1)){
                        StdDraw.picture(400,560,Aktion.WEISSIMG);
                        StdDraw.text(HOEHETEXT, BREITETEXT3, "Gemeinsam seid ihr
stark!!!");

                }
```

– Konstanten:

```
public static final int PORTAL= 29;
public static final int STANDHERE= 30;
public static final int FELDVORSTANDHEREEINS= 31;
public static final int FELDVORSTANDHEREZWEI= 32;

public static final int DOPPELPORTALLINKS= 33;
public static final int DOPPELPORTALRECHTS= 34;
```

– IMG:

```
public static final String PORTALIMG = "Images\\portal.jpg";
public static final String STANDHEREIMG = "Images\\standhere.jpg";
public static final String FELDVORSTANDHEREEINSIMG = "Images\\boden.jpg";
public static final String FELDVORSTANDHEREZWEIIMG = "Images\\boden.jpg";
public static final String DOPPELPORTALLINKSIMG = "Images\\portal.jpg";
public static final String DOPPELPORTALRECHTSIMG = "Images\\portal.jpg";
```

– **Aktion** (Methoden)
  – Methode: (Zur Zeit noch ausgeklammert, da unser Netzwerk noch nicht funktioniert)
    – Zur Lösung der Quest müssen beide Spieler zeitgleich vor verschiedenen „Stand here“ -Pfeilen stehen, um ++das Portal öffnen zu können.

RPS- Prinzip für Schadenssystem

– **Interface** (Methode)
  – Methoden:

```
                        /* Angriff oben*/
                        else if (k.getKeyCode() == KeyEvent.VK_W){
                                playerAttack(OBEN);
                                alleGegnerBewegen();
                        }
                        /*Angriff links*/
                        else if (k.getKeyCode() == KeyEvent.VK_A){
                                playerAttack(LINKS);
                                alleGegnerBewegen();
                        }
```

```
                        /*Angriff unten*/
                        else if (k.getKeyCode() == KeyEvent.VK_S){
                                playerAttack(UNTEN);
                                alleGegnerBewegen();
                        }
                        /*Angriff rechts*/
                        else if (k.getKeyCode() == KeyEvent.VK_D){
                                playerAttack(RECHTS);
                                alleGegnerBewegen();
                        }


                       /*Falle*/
                        else if
(Spielfeld.wertLesenBeiXY(level,raum,spalte,reihe)==FALLE){
                                StdDraw.picture(PIC1+PIC2*spalte,PIC1+PIC2*reihe,
FALLEIMG);
                        }
                        /*Mob*/
                        else if
(Spielfeld.wertLesenBeiXY(level,raum,spalte,reihe)==MOB){
                                StdDraw.picture(PIC1+PIC2*spalte,PIC1+PIC2*reihe,
MOBIMG);
                                gegner[gegnerZahl] = new Gegner(ZEHN, spalte, reihe,
OBEN, 0, 2, gegnerZahl,MOB);
                                gegnerZahl++;
                        }
```

– **Aktion** (Methoden)
– Methode: Zeigt die Richtung an, in die der Spieler angreift.

```
public int[] playerAttack(int richtung){
            setNewFigurXY(richtung);
            returnArray[0]=0;
            if
((Spielfeld.wertLesenBeiXY(aktuellesLevel,aktuellerRaum,newFigurX,newFigurY)==Interface
.MOB)

      |(Spielfeld.wertLesenBeiXY(aktuellesLevel,aktuellerRaum,newFigurX,newFigurY)==Int
erface.BOSS1)

      |(Spielfeld.wertLesenBeiXY(aktuellesLevel,aktuellerRaum,newFigurX,newFigurY)==Int
erface.BOSS2)){
                  returnArray[0]=Interface.MOB;
                  returnArray[1]=newFigurX;
                  returnArray[2]=newFigurY;
                  ANGRIFFSOUND.play();
            }
            else if
(Spielfeld.wertLesenBeiXY(aktuellesLevel,aktuellerRaum,newFigurX,newFigurY)==Interface.
BOSS3){
                  returnArray[0]=Interface.BOSS3;
                  ANGRIFFSOUND.play();
            }
            return returnArray;
      }
```

– **Boss** (Methoden)

– Werte für HP, Schaden und Rüstung
– Methode: Reduziert Lebenspunkte des Bosses und lässt Gegner sterben

```
public void schadenBekommen(double schaden){
		hp = Math.round((hp-(schaden*((Interface.EINHUNDERT-
RUESTUNG)/Interface.EINHUNDERT)))
				*(double)Interface.EINHUNDERT)/(double)Interface.EINHUNDERT;
		stats.displayGegnerHP(hp, DEFAULTHP, x, y);
		if (hp<=0){
			sterben();
		}
	}
```

– **Gegner** (Methoden)
– Methoden:

```
/**
	 * setzt Lebenspunkte der Gegner herunter, abhaengig von der aktuellen Ruestung
des Spielers
	 * falls Lebenspunkte aufgebraucht sind, wird Methode sterben aufgerufen
	 * Gegner wehrt sich: Spieler erleidet zufallsbasiert Schaden, wenn Gegner
getroffen wird
	 * @param schaden Schaden den der Gegner erleidet
	 */
	public void schadenBekommen(double schaden){
		gegnerSchadenAusteilen=Math.random();
		if (gegnerSchadenAusteilen>Interface.ZUFALL7){
			Interface.player[0].schadenBekommen(1);
		}
		hp = Math.round((hp-(schaden*((Interface.EINHUNDERT-
ruestung)/Interface.EINHUNDERT)))
				*(double)Interface.EINHUNDERT)/(double)Interface.EINHUNDERT;
		stats.displayGegnerHP(hp, defaultHP, x, y);
		if (hp<=0){
			sterben();
		}
	}
```

– Methode: Gegner stirb und an seiner Stelle erscheint eine kleine „Belohnung“

```
	/**
	 * Gegner stirbt, anstelle des Gegners erscheint ein Manatrank, HPTrank oder eine
Muenze
	 * Spieler erhaelt Erfahungspunkte
	 */
	public void sterben(){
		lebendig=false;

	Interface.player[0].erfahrungspunkteSammeln(Spieler.ERHALTERFAHRUNGSPUNKTE);
		loot=Math.random();
		if ((Interface.level==1)&(Interface.raum==0)&(loot<Interface.ZUFALL3)){
			Spielfeld.wertSetzenBeiXY(Interface.getLevel(), Interface.getRaum(),
x, y, Interface.SCHLUESSEL);
```

```
	StdDraw.picture(Interface.PIC1+Interface.PIC2*x,Interface.PIC1+Interface.PIC2*y,
Interface.SCHLUESSELIMG);
			}
			else{
				if (loot<Interface.ZUFALL1){
					Spielfeld.wertSetzenBeiXY(Interface.getLevel(),
Interface.getRaum(), x, y, Interface.HPTRANK);

	StdDraw.picture(Interface.PIC1+Interface.PIC2*x,Interface.PIC1+Interface.PIC2*y,
Interface.HPTRANKIMG);
				}
				else if (loot<Interface.ZUFALL2){
					Spielfeld.wertSetzenBeiXY(Interface.getLevel(),
Interface.getRaum(), x, y, Interface.MANATRANK);

	StdDraw.picture(Interface.PIC1+Interface.PIC2*x,Interface.PIC1+Interface.PIC2*y,
Interface.MANATRANKIMG);
				}
				else{
					Spielfeld.wertSetzenBeiXY(Interface.getLevel(),
Interface.getRaum(), x, y, Interface.MUENZEN);

	StdDraw.picture(Interface.PIC1+Interface.PIC2*x,Interface.PIC1+Interface.PIC2*y,
Interface.MUENZENIMG);
				}
			}
	}
```

– **Spieler** (Konstanten und Methoden)
– Konstanten: Beispiele für Konstanten. Werden generell für den Spieler in dieser Klasse gesetzt (Ausgangs und Maximalwerte)

```
public static final double MAXMANA = 10.0;
public static final double MAXHP = 10.0;

	private int leben;
	private double manaFaktor;
	private double schaden;
	private double aktuellesMana;
	private double aktuelleHP;
	private double ruestung;
```

– Methoden:

```
		else if (checkArray[0]==Interface.FALLE){
			if (schadenBekommen(Interface.CHARMOB)){
				Interface.toCheckpoint=true;
			}
		}
		else if (checkArray[0]==Interface.MOB){
			sterben();
		}
		else if (checkArray[0]==Interface.BOSS3){
			sterben();
		}
```

– Methode, die den aktuellen HP-Wert reduziert wenn man getroffen wurde
– Methode, die einen „sterben“ bzw. ein Leben verlieren lässt, wenn man keine HP mehr besitzt.
– Methode, die die Anzeige im Display an der Seite aktualisiert
– Methode, Erhöhung der Mana- und HP Werte wenn man etwas einsammelt
– Methoden, die verschiedene Werte zurückgeben (Schaden-, Mana-,Schild-, Rüstung-, etc.)
– Methode, die Mana, Schaden und Ruestung auf den entsprechenden Faktor der Farbe umstellt
– Methode, die den Manawert verbraucht

StatDisplay (Methoden)

– Methode: stellt Schadens, Rüstungs und Lebenswert graphisch im Display dar:

```
StdDraw.textLeft(Interface.ACHTHUNDERTZWANZIG, Interface.FUENFHUNDERTZEHN,
"Leben:"+leben);
            StdDraw.textLeft(Interface.ACHTHUNDERTZWANZIG,
Interface.VIERHUNDERTNEUNZIG, "Schaden:"+schaden);
            StdDraw.textLeft(Interface.ACHTHUNDERTZWANZIG,
Interface.VIERHUNDERTSIEBZIG, "Ruestung:"+ruestung);
```

Chat

– Noch nicht implementiert

Netzwerklobby

– Noch nicht implementiert

EP & Skill-Tree

– **Aktion**
– Methoden die abfragen, ob die Rüstung bzw. der Schildzauber bereits existieren

– **Boss**
– Methode: Spieler erhält Erfahrungspunkte, wenn der Gegner besiegt wurde

```
      public void sterben(){

      Interface.player[0].erfahrungspunkteSammeln(Spieler.ERHALTERFAHRUNGSPUNKTE);
            Spielfeld.wertSetzenBeiXY(Interface.getLevel(), Interface.getRaum(), x, y,
Interface.SIEG);

      StdDraw.picture(Interface.PIC1+Interface.PIC2*x,Interface.PIC1+Interface.PIC2*y,
Interface.SIEGIMG);
```

```
	}
```

– **Gegner**
– Klasse erstellt Objekte vom Typ Gegner (setzt Schaden etc. fest)
– Methode: setzt (abhängig von der Rüstung des Spielers) Lebenspunkte der Gegner herunter

```
	public void schadenBekommen(double schaden){
		gegnerSchadenAusteilen=Math.random();
		if (gegnerSchadenAusteilen>Interface.ZUFALL7){
			Interface.player[0].schadenBekommen(1);
		}
		hp = Math.round((hp-(schaden*((Interface.EINHUNDERT-
ruestung)/Interface.EINHUNDERT)))
				*(double)Interface.EINHUNDERT)/(double)Interface.EINHUNDERT;
		stats.displayGegnerHP(hp, defaultHP, x, y);
		if (hp<=0){
			sterben();
		}
	}
```

– Methode: Spieler erhält Erfahrungspunkte, wenn ein Gegner stirbt

– **Spieler**
– Konstanten:

```
public static int ERHALTERFAHRUNGSPUNKTE = 100;
public static int aktuelleErfahrungspunkte;
```

– Methoden: Man erhält Erfahrenspunkte

```
else if (checkArray[0]==Interface.SCHLUESSEL){
			aktuelleSchluessel++;
			stats.displayPlayerStats(leben, schaden, ruestung,
manaFaktor,aktuelleHP,

	aktuellesMana,aktuelleErfahrungspunkte,aktuelleMuenzen,aktuelleSchluessel);
		}
		else if (checkArray[0]==Interface.TOR){
			aktuelleSchluessel = 0;
			stats.displayPlayerStats(leben, schaden, ruestung,
manaFaktor,aktuelleHP,

	aktuellesMana,aktuelleErfahrungspunkte,aktuelleMuenzen,aktuelleSchluessel);

		}
```

– Methode:Steigt man ein Erfahrungslevel auf, so erhält man als kleinen Bonus, ein Leben und die Fähigkeit den Schildzauber anwenden zu können.
Beim zweiten „Levelup“ erhält man ebenfalls ein Leben und der Manawert wird komplett aufgeladen.

```
	public void erfahrungspunkteSammeln(int erfahrungspunkte){
		aktuelleErfahrungspunkte = aktuelleErfahrungspunkte+erfahrungspunkte;
		if (aktuelleErfahrungspunkte == FIRSTLEVEL | aktuelleErfahrungspunkte ==
SECONDLEVEL){
			Aktion.LEVELUPSOUND.play();
			aktuelleHP = MAXHP;
			leben = leben + 1;
			kannSchildZaubern = true;

			if (aktuelleErfahrungspunkte != SECONDLEVEL){
				StdDraw.picture(400,560,Aktion.WEISSIMG);
				StdDraw.text(Interface.HOEHETEXT, Interface.BREITETEXT4,
						"Glückwunsch, du hast Level 2 erreicht!!! Ab nun
kannst du mit der Leertaste");
				StdDraw.text(Interface.HOEHETEXT, Interface.BREITETEXT2,
						"den Schildzauber einsetzen! Setze ihn weise
ein...");
			}

			if (aktuelleErfahrungspunkte == SECONDLEVEL){
				aktuellesMana = MAXMANA;
				leben = leben + 1;

				if (aktuelleErfahrungspunkte != FIRSTLEVEL){
					StdDraw.picture(400,560,Aktion.WEISSIMG);
					StdDraw.text(Interface.HOEHETEXT,
Interface.BREITETEXT5,
							"Glückwunsch, du hast Level 3
erreicht!!!");
				}
			}
		}
		stats.displayPlayerStats(leben, schaden, ruestung, manaFaktor,aktuelleHP,

	aktuellesMana,aktuelleErfahrungspunkte,aktuelleMuenzen,aktuelleSchluessel);
	}
```

– Methode: sammelt man Münzen ein, steigt der Erfahrungswert.

```
/**
	 * Muenzen einsammeln
	 * @param muenzen Muenzen
	 */
	public void muenzenSammeln(int muenzen){
		aktuelleMuenzen = aktuelleMuenzen+muenzen;
		stats.displayPlayerStats(leben, schaden, ruestung, manaFaktor,aktuelleHP,

	aktuellesMana,aktuelleErfahrungspunkte,aktuelleMuenzen,aktuelleSchluessel);

	}
```

– Methode: Auch beim Einsammeln von Schlüsseln erhält man Erfahrungspunkte
– Methode: fragt an einigen stellen, den aktuellen Wert der Erfahrungswerte ab.

StatDisplay

– Methode: Stellt die Erfahrungswerte im Display graphisch dar.

```
            StdDraw.textLeft(Interface.ACHTHUNDERTZWANZIG,
Interface.DREIHUNDERTSIEBZIG, "Erfahrung:"+erfahrungspunkte);
```

Sound

– **Aktion** (Konstanten setzen und Schleife)
- Konstantenbeispiel:

```
private static final AudioClip MUENZESOUND = Sound.loadSound("src/Sounds/gold.wav");
private static final AudioClip TORSOUND = Sound.loadSound("src/Sounds/tor.wav");
private static final AudioClip BOMBESOUND = Sound.loadSound("src/Sounds/bombe.wav");
```

- Schleifenbeispiel:

```
if (player){
                        returnArray[0]=Interface.FALLE;
                        BOMBESOUND.play();
                }
                else{
                        returnArray[0]=Interface.MAUER;
                }
```

– **Sound** (Methode)
- Diese Methode spielt den Sound ab, der in der vorher beschiebenen Klasse deklariert wurde:

```
public class Sound{

   public static AudioClip loadSound(String string) {
            File f = new File(string);
            try {
                    AudioClip sound = Applet.newAudioClip(f.toURI().toURL());
                    return sound;
            }
            catch (MalformedURLException e) {
                    System.out.println(e);
                    return null;
            }
      }
}
```