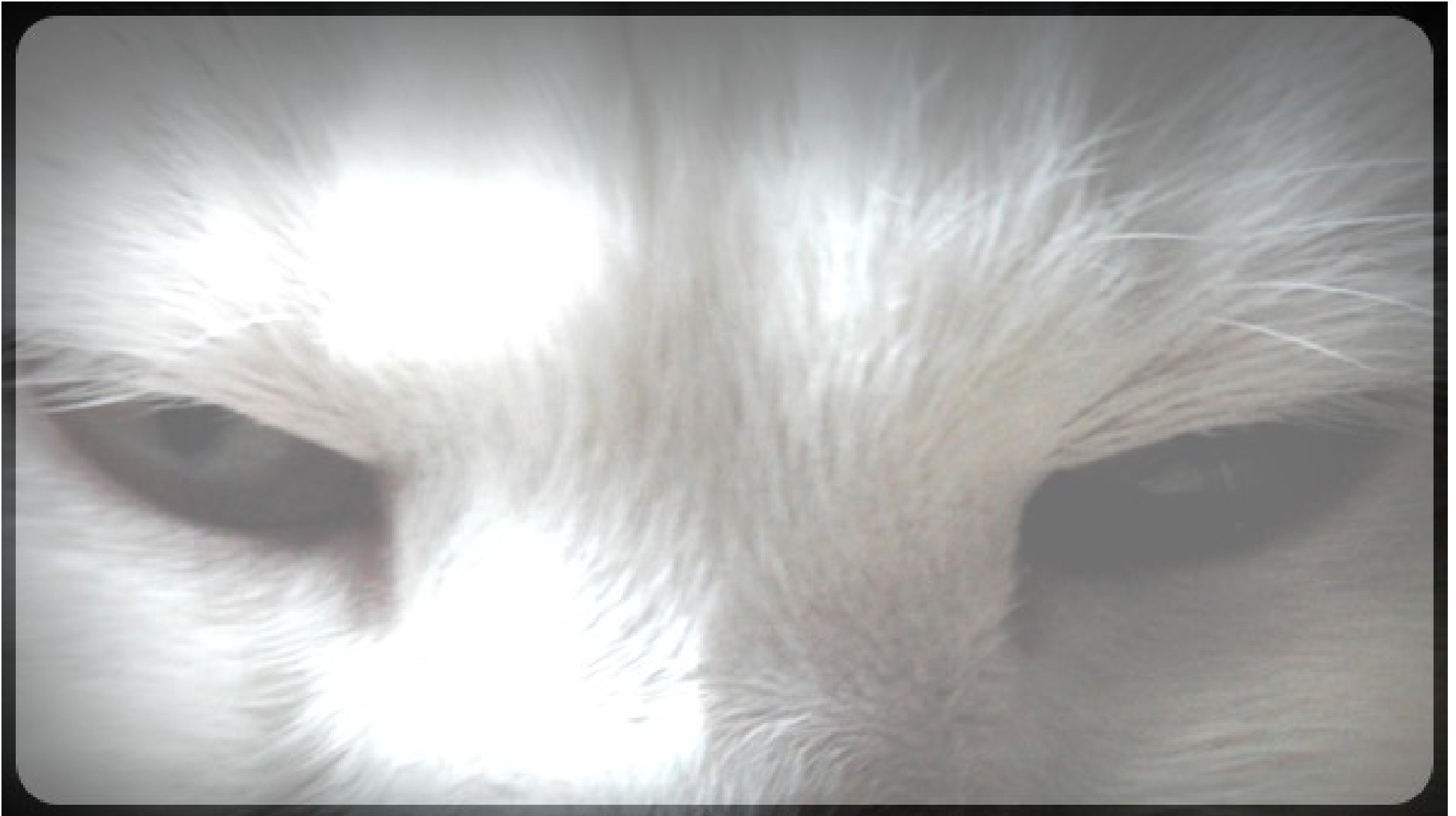


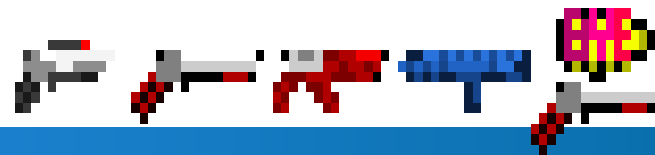


Source Code von Miez Attack

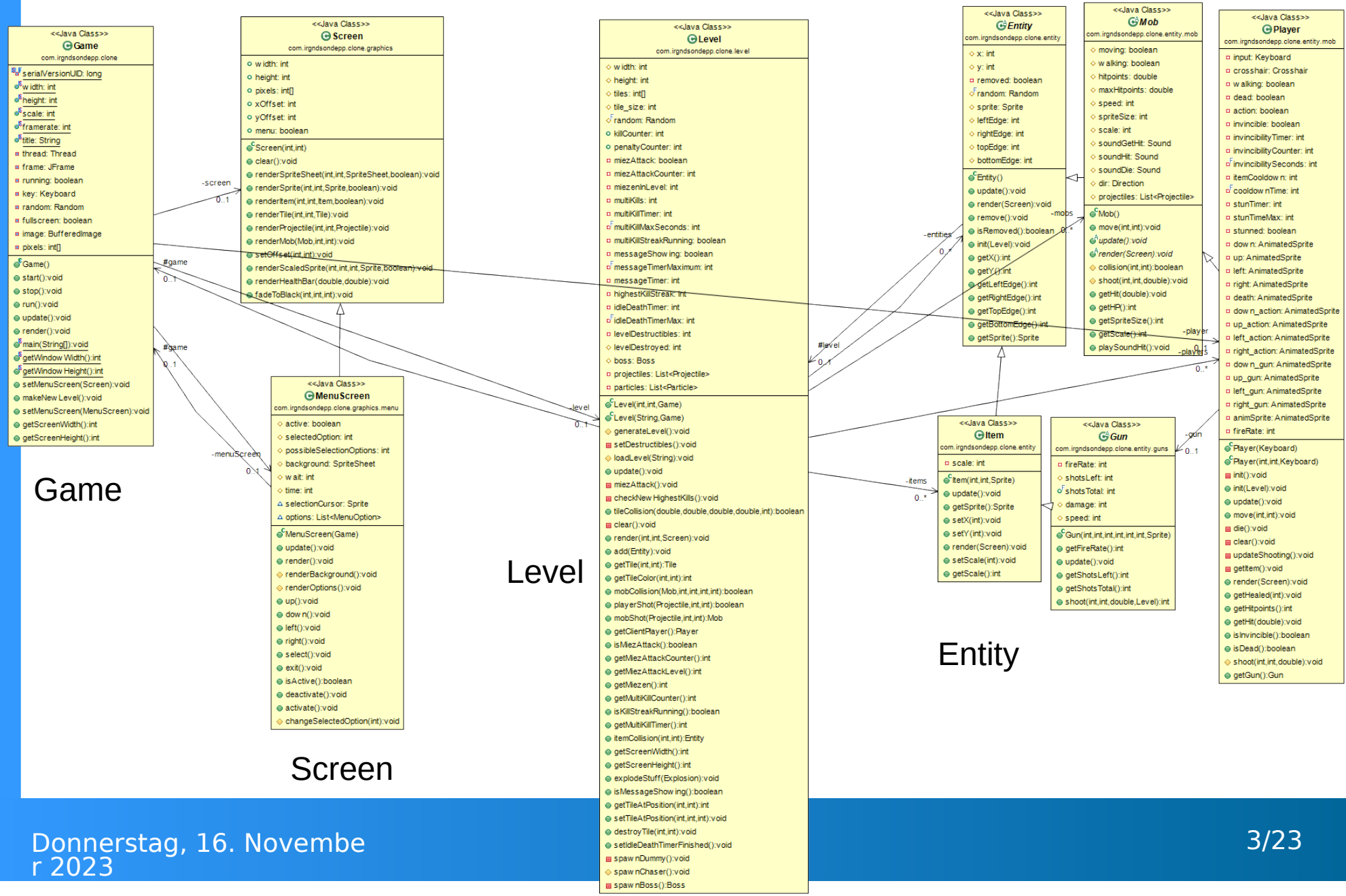


Überblick

- **58 Klassen** mit insgesamt **6600 Zeilen Code**
 - 1 Game Klasse → main()
 - 10 Grafik Klassen:
 - Screen, Menu, Sprite,  
 - 2 Input Klassen:
 - Maus und Tastatur
 - 2 Level Klassen:
 - Level, Spawnlevel
 - 42 Entity Klassen:
 - Spieler, Gegner, Items, Partikel, Projektile, Tiles
 - 1 Sound Klasse



Die wichtigsten Klassen



Wie funktioniert ein Spiel?

START

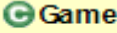
```
boolean running=true;
while(running)
{
    if (time > (1/60)s)
    {
        update();           // kümmere dich um die Spiellogik!
    }
    render();
}
```

STOP

<pre>public void update(){ level.update(); player.update(); menu.update(); ... }</pre>	<pre>public void render(){ level.render(); player.render(); menu.render(); ... }</pre>
--------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------

Game.run()

<<Java Class>>

 **Game**

com.irgndsondepp.clone

S F serialVersionUID: long

S width: int

S height: int

S scale: int

S framerate: int

S title: String

thread: Thread

frame: JFrame

running: boolean

screen: Screen

menuScreen: MenuScreen

key: Keyboard

level: Level

player: Player

random: Random

fullscreen: boolean

image: BufferedImage

pixels: int[]

Game()

start():void

stop():void

run():void

update():void

render():void

S main(String[]):void

S getWindowWidth():int

S getWindowHeight():int

setMenuScreen(Screen):void

makeNewLevel():void

setMenuScreen(MenuScreen):void

getScreenWidth():int

getScreenHeight():int

```
public void run() {
    long lastTime = System.nanoTime();
    long timer = System.currentTimeMillis();
    final double ns = 1.0e9 / ((double) framerate);
    double delta = 0;
    int frames = 0;
    int updates = 0;

    while (running) {
        long now = System.nanoTime();
        delta += (now - lastTime) / ns;
        lastTime = now;

        while (delta >= 1) {
            update();
            updates++;
            delta--;
        }

        render();
        frames++;

        if (System.currentTimeMillis() - timer > 1000) {
            timer += 1000;
            frame.setTitle(title + "    fps: " + frames + "    updates: "
                           + updates);
            frames = 0;
            updates = 0;
        }
        if (key.exit)
            running=false;
    }
    stop();
}
```

mbe

5/23

Grafik

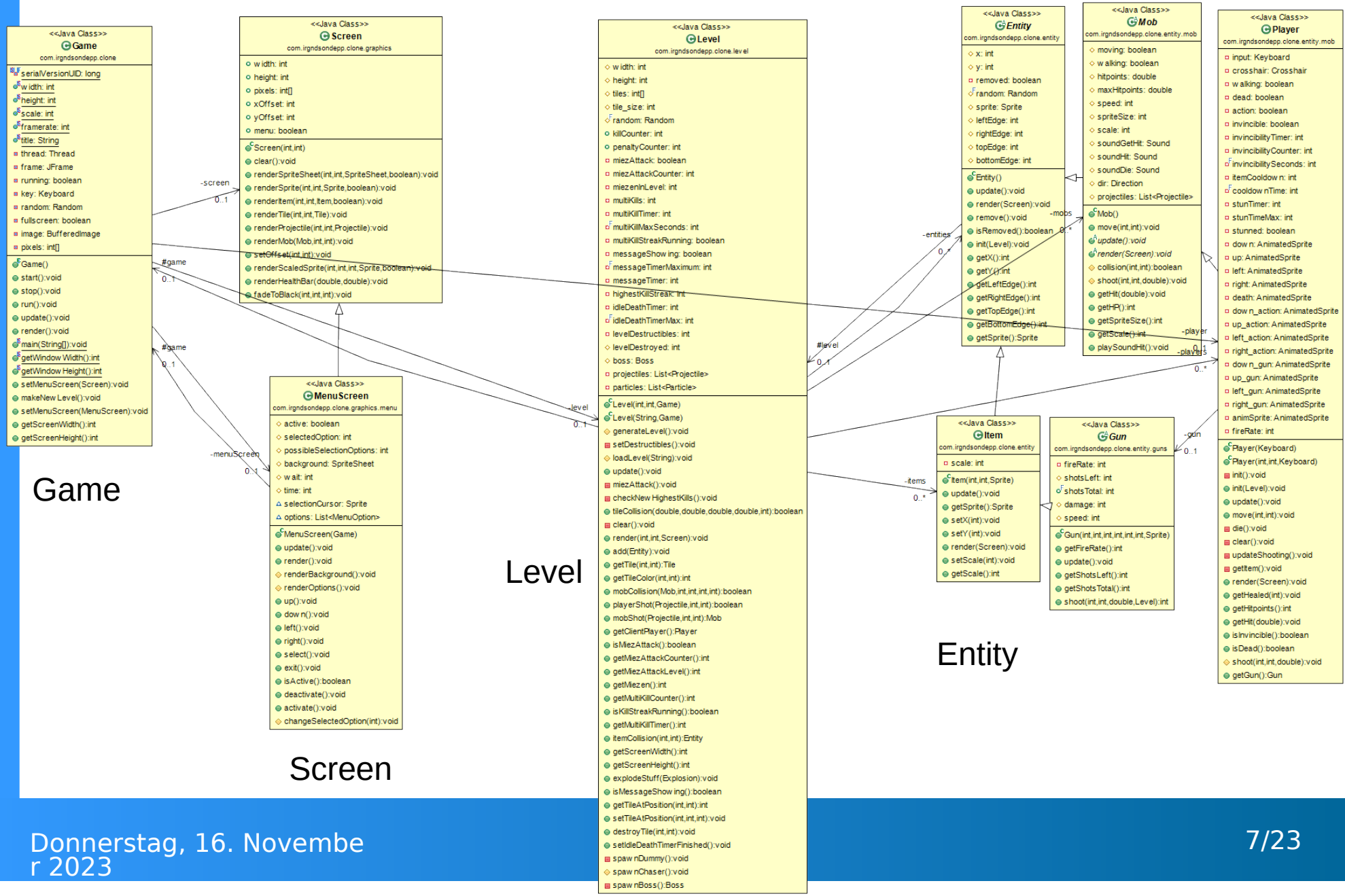
START

```
boolean running=true;
while(running)
{
    render();
}
```

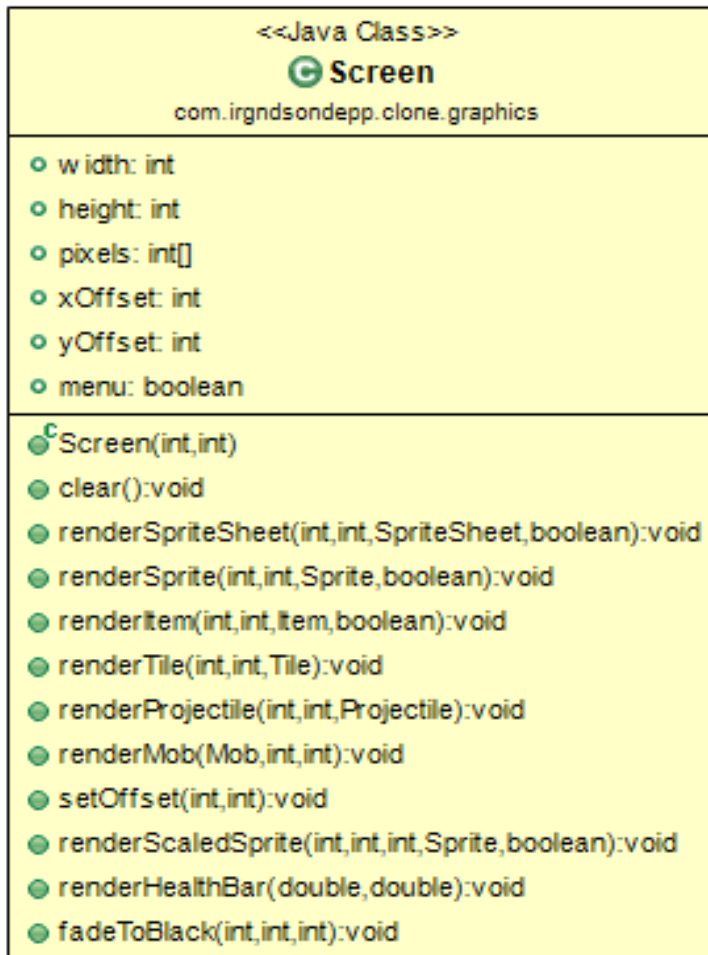
STOP

```
public void render(){
    level.render();
    player.render();
    menu.render();
    ...
}
```

Grafik



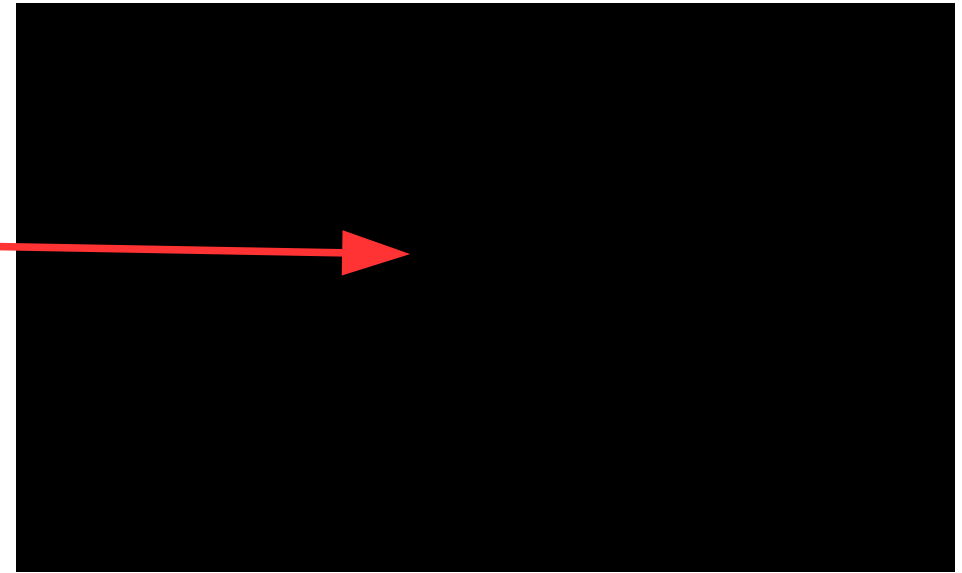
Klasse Screen: Was sieht man?



Relevante Methoden:

- clear()
- Verschiedene renderX(...) Methoden
- fadeToBlack(...)
- setOffset(...)

Grafik



`int spritePixels[]`  `int[] pixels = new int[width * height]`

```
if (col != 0xffFF00FF){  
    pixels[i]=col;  
}
```

Screen.pixel[]

Auszug aus Level:

```
// render all the visible tiles
for (int y = y0; y < y1; y++) {
    for (int x = x0; x < x1; x++) {
        getTile(x, y).render(x, y,
screen);
    }
}

// render all items
for (int i = 0; i < items.size(); i++) {
    items.get(i).render(screen);
}

// render all projectiles
for (int i = 0; i < projectiles.size(); i++)
{
    projectiles.get(i).render(screen);
}
```



Logik

START

```
boolean running=true;
while(running)
{
    if (time > (1/60)s)
    {
        update();
    }
    render();
}
```

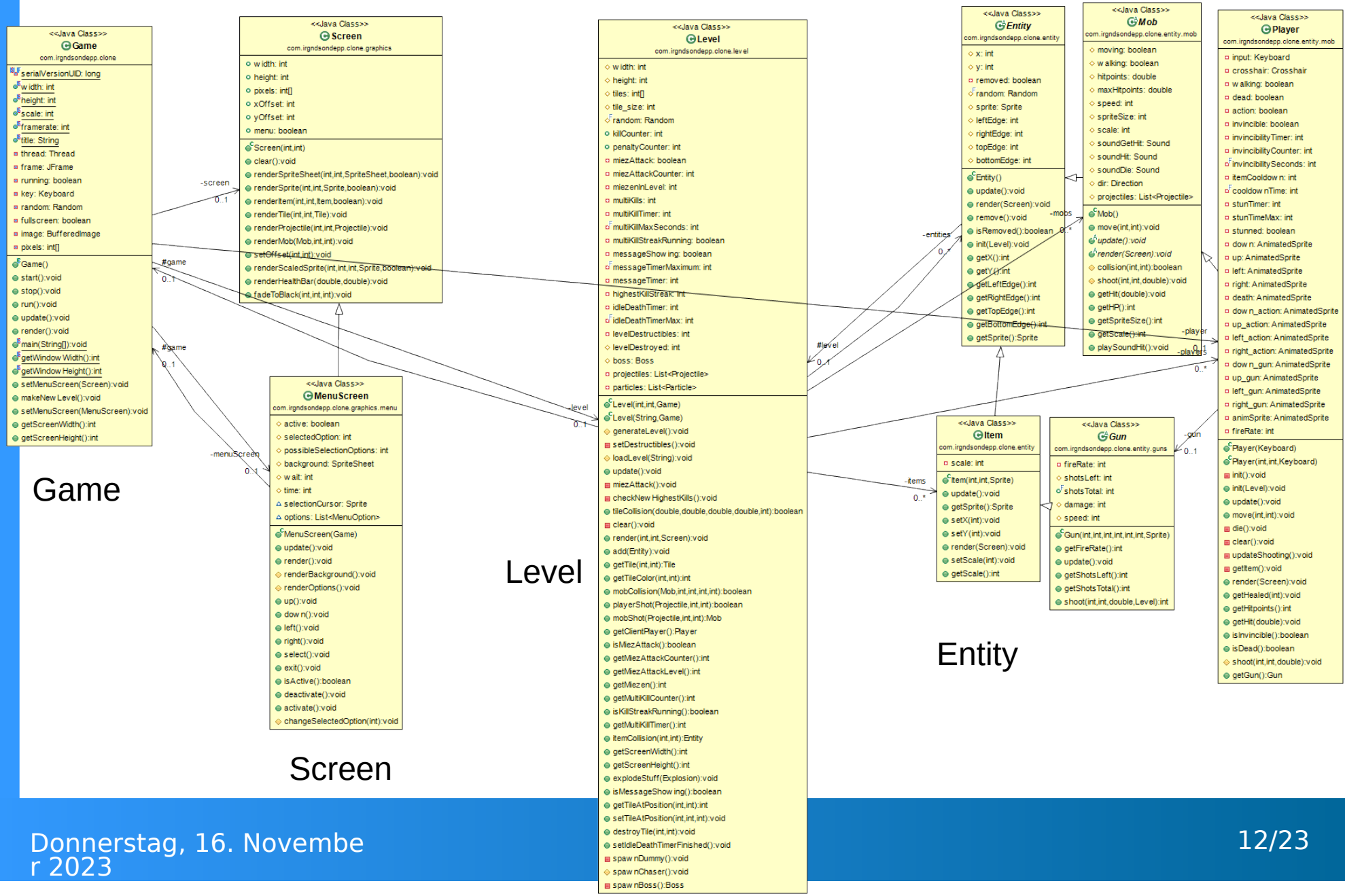
// kümmere dich um die Spiellogik!

STOP

```
public void update(){
    level.update();
    player.update();
    menu.update();
    ...
}
```

```
public void render(){
    level.render();
    player.render();
    menu.render();
    ...
}
```

Logik



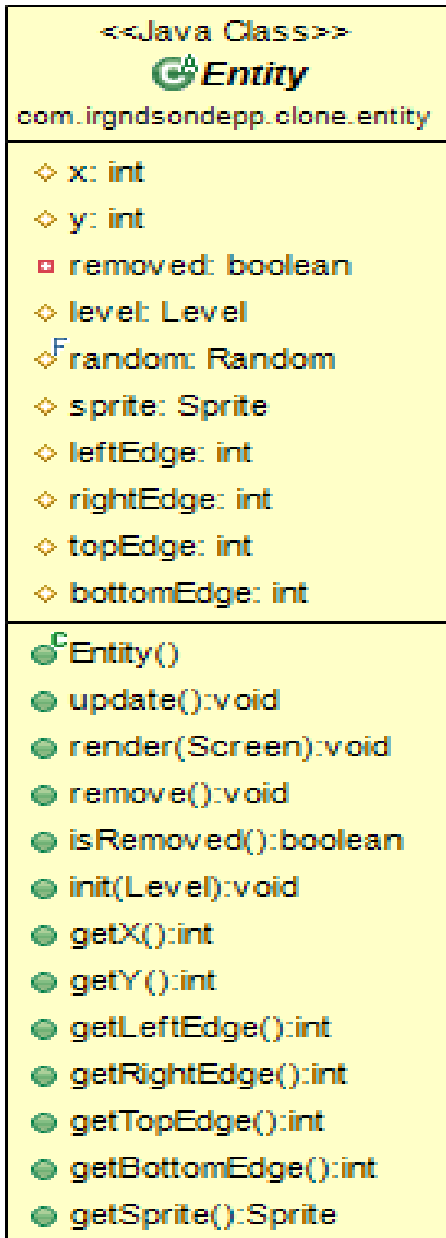
Spiellogik und Grafik



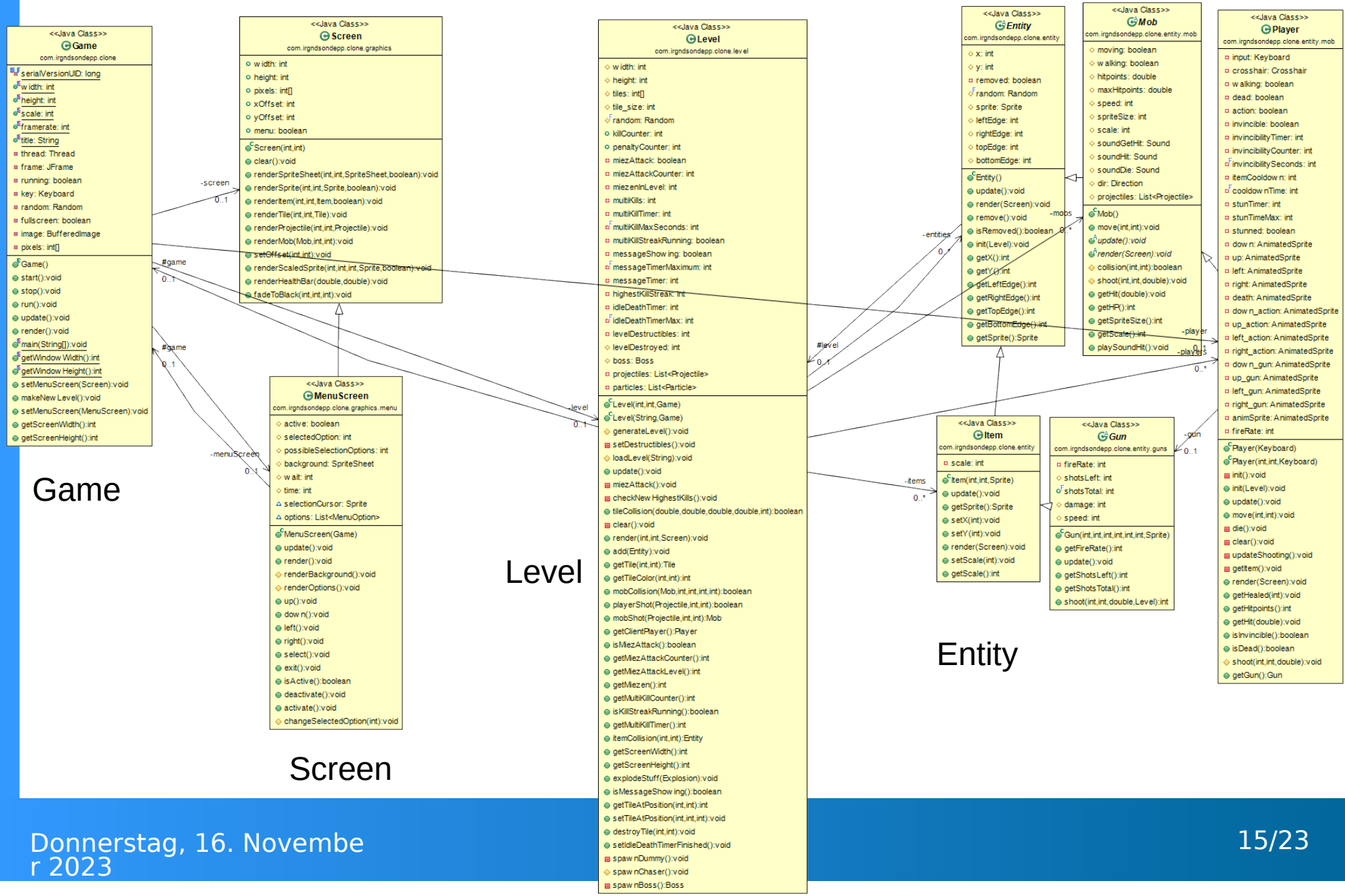
Klasse Entity



- Entity verwaltet die Position in der Spielwelt
 - remove() bestimmt die „Lebenszeit“ eines Spielobjekts
 - Unterklassen besitzen unterschiedliche Logikmethoden
 - z.B. shoot(...), walk(...), ...
 - Nutzen Methoden aus **Level**



Logik



Klasse Level: Was soll passieren?

Enthält alle Objekte eines Levels in Listen

Relevante Methoden:

- update()
- miezAttack()
- spawnDummy()
- tileCollision(...)
- render(...)
- explodeStuff(...)

```
• Level(int,int,Game)
• Level(String,Game)
• generateLevel():void
• setDestructibles():void
• loadLevel(String):void
• update():void
• miezAttack():void
• checkNew HighestKills():void
• tileCollision(double,double,double,double,int):boolean
• clear():void
• render(int,int,Screen):void
• add(Entity):void
• getTile(int,int):Tile
• getTileColor(int,int):int
• mobCollision(Mob,int,int,int,int):boolean
• playerShot(Projectile,int,int):boolean
• mobShot(Projectile,int,int):Mob
• getClientPlayer():Player
• isMiezAttack():boolean
• getMiezAttackCounter():int
• getMiezAttackLevel():int
• getMiezen():int
• getMultiKillCounter():int
• isKillStreakRunning():boolean
• getMultiKillTimer():int
• itemCollision(int,int):Entity
• getScreenWidth():int
• getScreenHeight():int
• explodeStuff(Explosion):void
• isMessageShowing():boolean
• getTileAtPosition(int,int):int
• setTileAtPosition(int,int,int):void
• destroyTile(int,int):void
• setIdleDeathTimerFinished():void
• spawnDummy():void
• spawnChaser():void
• spawnBoss():Boss
```

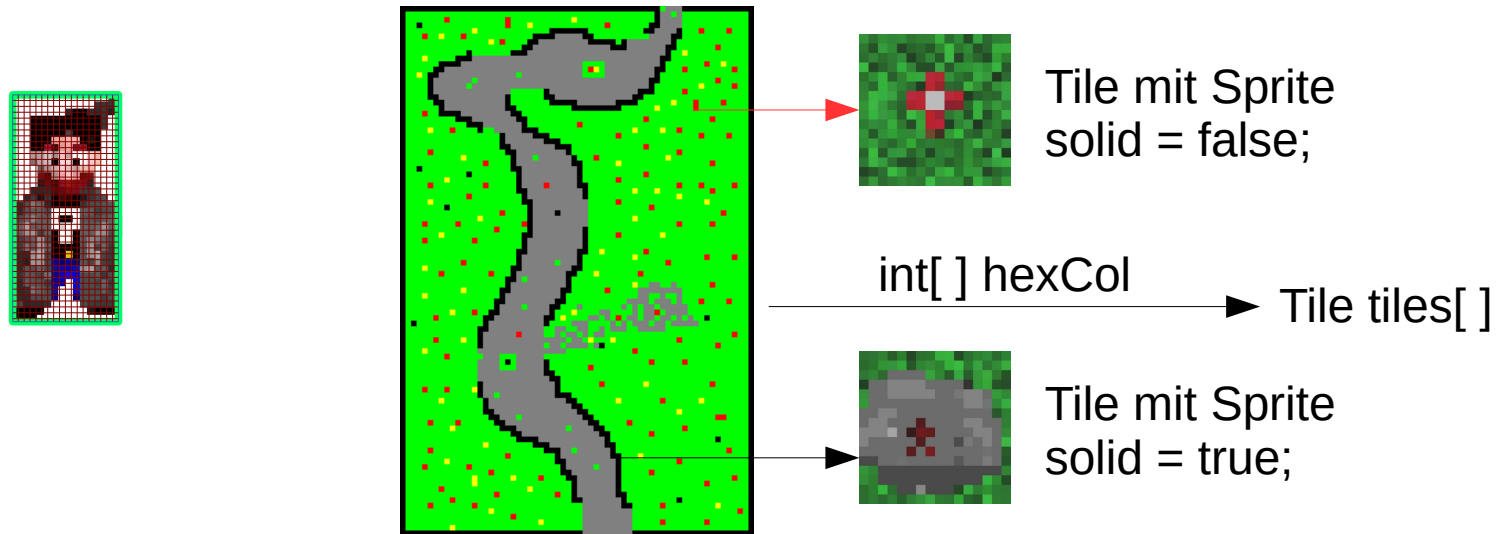

Level.update()

- Entferne alle nicht vorhandenen Spielobjekte
- Mit Schleifen wird für alle Elemente der Entity Listen update() aufgerufen
 - Entities verwenden zur Kollisionsabfrage wieder Level
- Level Variablen werden aktualisiert
 - Anzahl der Gegner
 - Schwierigkeitsgrad
 - Timer

Level.miezAttack()

- Wurde eine gewisse Anzahl von Gegnern besiegt?
 - Vorhandene harmlose Katzen werden durch die aggressive Variante ersetzt.
- Sind zu wenig Katzen im Level vorhanden?
 - Füge neue Katzen hinzu.
- Wurden 100 Gegner besiegt?
 - Füge einen Boss hinzu.

Level.tileCollision(...)



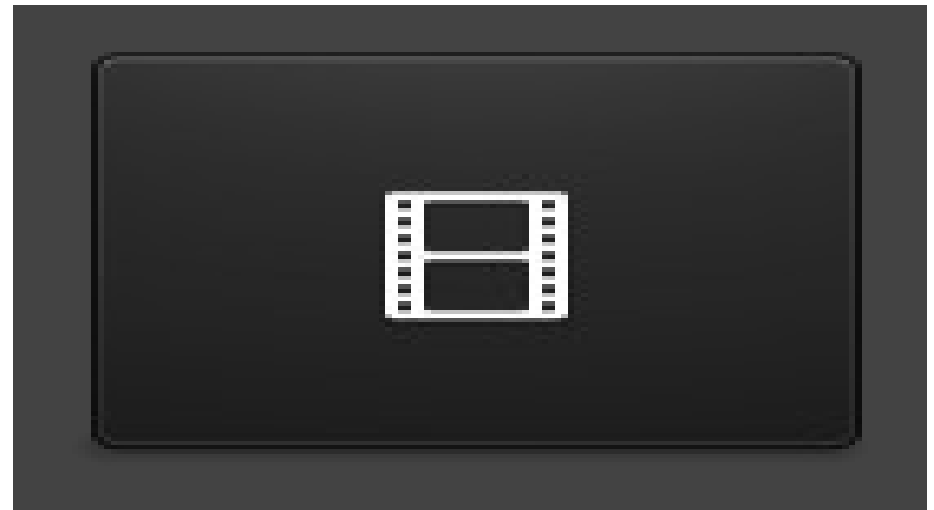
```
public boolean tileCollision(double x, double y, double xchange, double ychange, int size) {  
    boolean solid = false;  
    for (int c = 0; c < 4; c++) {  
        double xt = ((x + xchange) + (c % 2 * (size / 2)) - 4) / tile_size;  
        double yt = ((y + ychange) + (c / 2 * (size / 10)) + 6) / tile_size;  
        if (getTile((int) xt, (int) yt).solid()) solid = true;  
    }  
    return solid;  
}
```

Level.explodeStuff(...)

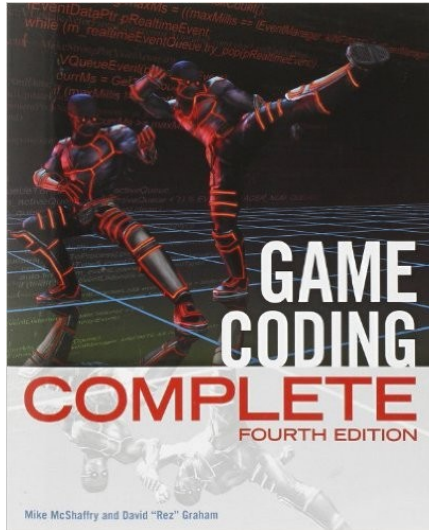
- Befinden sich Gegner im Explosionsradius?
 - Füge den jeweiligen Gegnern Schaden zu.
- Befindet sich der Spieler im Explosionsradius?
 - Füge dem Spieler Schaden zu.
- Befinden sich Steine oder Bäume im Weg?
 - Verwandle Baum Tiles in Wiesen und Stein Tiles in Weg.
- destroyTile(...)

Zusammenfassung

- Game Loop
- Grafische Ausgabe über `int[]` Array in **Screen**
- Simple Logik in **Entity**
- Komplexe Spiellogik in **Level**



Literatur und Tutorials



„Game Coding
Complete“

McShaffry & Graham

Course Technology Cengage
Learning



TheChernoProject

thecherno.com

[www.youtube.com/user/
TheChernoProject](http://www.youtube.com/user/TheChernoProject)

Fragen?

Vielen Dank für
Ihre
Aufmerksamkeit.

