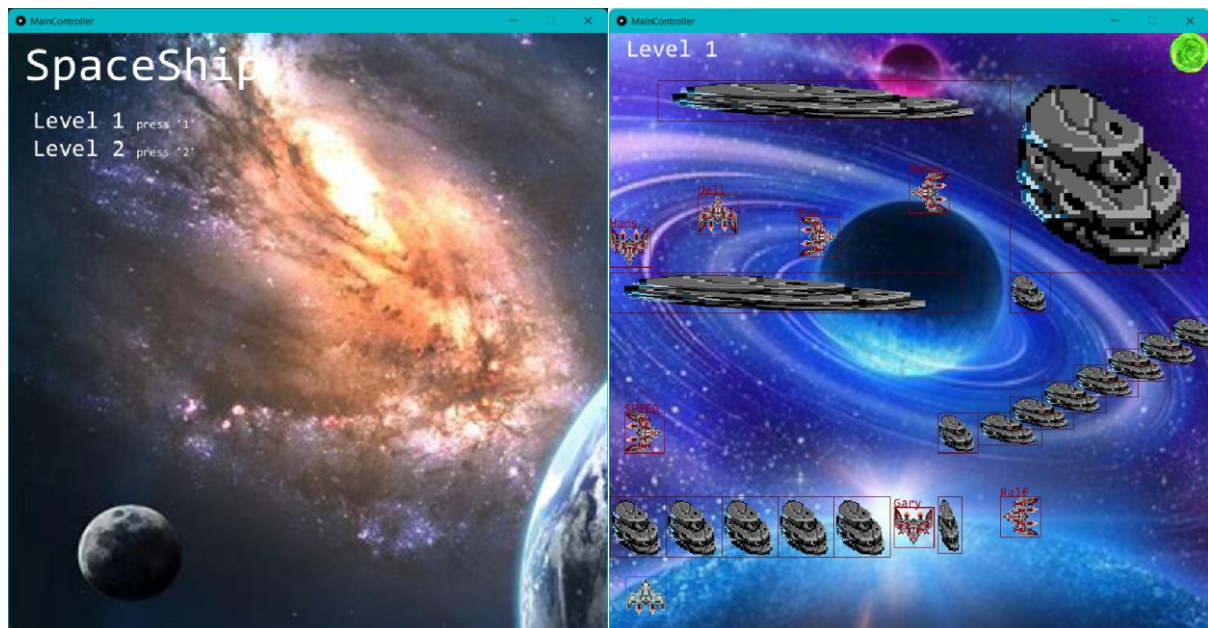


# M226B Projekt Dokumentation



Wendelin Haller

## Inhalt

Was ist mein Projekt? .....	3
Informieren .....	4
Planen .....	5
Zeitpläne .....	5
Entscheiden .....	7
Klassen-Diagramm .....	7
Realisieren .....	8
Code-Snippet und Erklärung .....	8
Vererbung .....	8
Abstrakte-Klasse .....	9
Interface .....	10
Kontrollieren .....	12
Unit und Mockup-Test .....	12
Auswerten und Fazit .....	13
Anhang und Abbildungsverzeichnis .....	14
Abbildungsverzeichnis .....	14

## Was ist mein Projekt?

Ich habe ein Spiel mit zwei Levels programmiert. In diesem Spiel kann man mit den Tasten 'W', 'A', 'S' und 'D' ein Raumschiff in alle vier Himmelsrichtungen bewegen. Das Ziel ist es das Raumschiff sicher und unversehrt zum grünen End Portal zu manövrieren. Auf dem Weg sind aber noch Gegner, die eine stricke Route abfliegen und dich bei Berührung ausser Gefecht setzen. Die Gegner können auch ausser Gefecht gesetzt werden, indem man mit dem Raumschiff auf sie zufliegt und mit der Maus klickt, weil dann schiesst man ein kleines grünes Projektil und wenn man den Gegner damit Trifft ist er ausser Gefecht gesetzt. Dabei werden alle anderen Gegner alarmiert und rasten aus, indem sie schneller fliegen und anfangen zu schiessen. Aber das geht nicht lange und sie beruhigen sich wieder. Nicht nur Gegner sind dir im Weg sondern auch sich nicht bewegende Asteroiden-Wende. Diese setzen dich auch ausser Gefecht, wenn du sie berührst.

## Informieren

Probleme	Wie ich das Ganze auf GitHub packen soll.
Hilfestellung	<a href="https://www.w3schools.io/file/markdown-list/">https://www.w3schools.io/file/markdown-list/</a>
Probleme	Wie soll ich das Status Contrills machen?
und habe dan alle keyPressed Methoden in di MainController gepackt und den View in die einzelnen ViewKlassen.	
Hilfestellung	Ich und Meine Gedanken.
Probleme	Keine
Hilfestellung	<a href="https://processing.org/examples/backgroundimage.html">https://processing.org/examples/backgroundimage.html</a> <a href="https://processing.org/reference/PFont.html">https://processing.org/reference/PFont.html</a>
Probleme	Image für das Raumschiff einbinden.
Hilfestellung	<a href="https://discourse.processing.org/t/applying-a-background-image-to-a-rect/13711/3">https://discourse.processing.org/t/applying-a-background-image-to-a-rect/13711/3</a>
Probleme	Keine
Hilfestellung	Projekt und Moduljournal
Probleme	Nicht gewust wie anfangen und was ich genau die nechsten mahle machen soll.
Hilfestellung	<a href="https://templates.office.com/en-us/Simple-Gantt-Chart-TM16400962">https://templates.office.com/en-us/Simple-Gantt-Chart-TM16400962</a>
Probleme	Darstellung der Asteroiden.
Hilfestellung	- <a href="https://www.ecosia.org/images?q=bixle%20art%20asteoride">https://www.ecosia.org/images?q=bixle%20art%20asteoride</a> - <a href="https://stackoverflow.com/questions/36511675/i-cant-use-image-and-rect-in-the-draw-method-in-javascriptmode">https://stackoverflow.com/questions/36511675/i-cant-use-image-and-rect-in-the-draw-method-in-javascriptmode</a> - <a href="https://py.processing.org/reference/noFill.html">https://py.processing.org/reference/noFill.html</a>
Probleme	java.util.ConcurrentModificationException
Hilfestellung	Herr Schirmer
Probleme	keine
Hilfestellung	vorherig geschribener Code
Probleme	Langsames Game wegen Bilder die immer im Draw geladen werden
Hilfestellung	Sven Schirmer
Probleme	mockup Test erstellen
Hilfestellung	Modul Presentatonen und beispiele wo ich schon MockUp-Tests erstellt habe.
Probleme	Auswal der Klassen und was getestet werden soll
Hilfestellung	Herr Schirmer

## Planen

## Zeitpläne

## Zeitplan vom 13.12.2021

## SpaceShip

Wendelin Haller

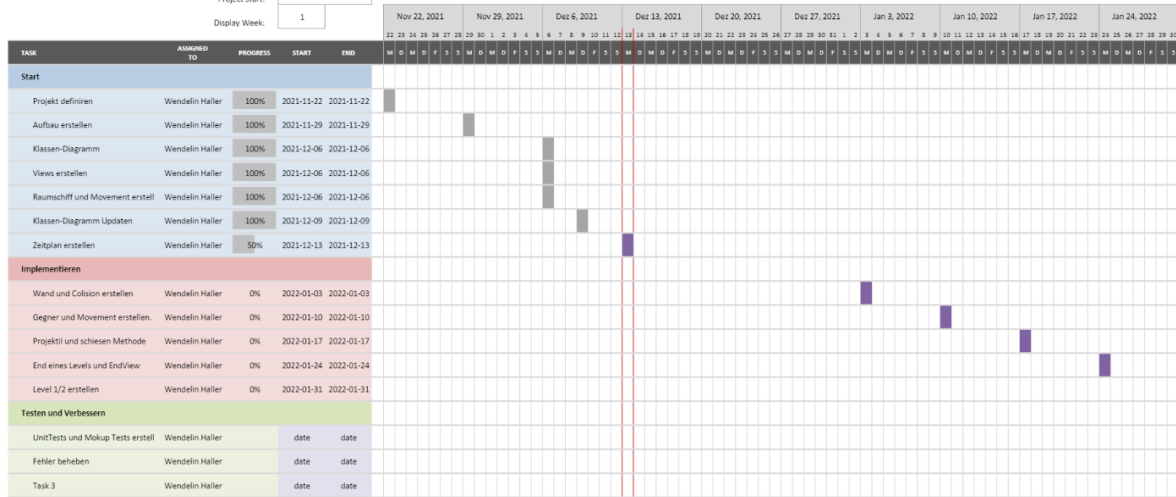
Project Start: Mo, 11.12.2021  
Display Week: 1

Abbildung 1: Zeitplan 13.12.2021

## Zeitplan vom 21.01.2021

## SpaceShip

Wendelin Haller

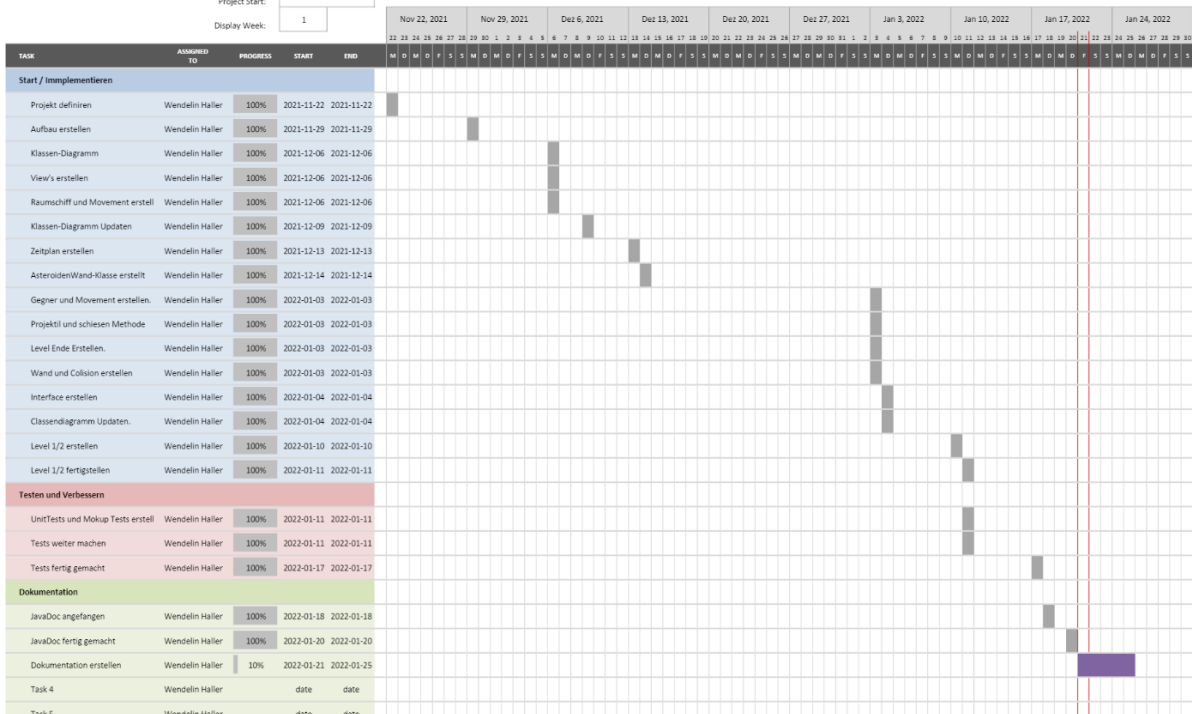
Project Start: Mo, 11.12.2021  
Display Week: 1

Abbildung 2: Zeitplan 21.01.2021

Ich habe das Projekt in drei Themen unterteilt, diese sind Start/Implementieren, Testen und Verbessern und Dokumentation. Jedem Thema habe ich eine Farbe zugeteilt und diese in Zeitplan verwendet. Als ich angefangen habe mit dem ersten Zeitplan 13.12.2021 da habe ich noch ein par Schritte übersehen und den Zeitaufwand unterschätzt.

Der grobe Zeitplan meines Projektes war, dass ich zuerst ein Gerüst programmiere mit Start, Level und End-View. Um danach das eigentliche Spiel und die Funktionalität in den Level-View einzubauen, damit ich nicht, wenn ich das Spiel fertig habe, noch einen Start und End-View drumherum bauen muss. Nachdem dann die Views und das Spiel und dessen Funktionalitäten ausprogrammiert habe, setzte ich mich daran drei Klassen zu testen mit Unit und Mockup-Tests. Zu guter Letzt habe ich dann noch diese Dokumentation eingeplant.

Das Einschätze, wie lang ich für einen Task brauche, habe ich immer schwierig gefunden und habe meist länger gebraucht als gedacht.



Insgesamt sind es 13 Klassen und 2 Interfaces. Es gibt 8 Klassen, die von einer anderen Klasse in diesem Diagramm erbt.

Von der abstrakten View-Klasse erben die StartViewController-Klasse, die EndViewController-Klasse und die LevelViewController-Klasse, weil alle diese drei Klassen haben, eins gemeinsahm nämlich:

- Von der LevelViewController-Klasse erben die Level1-Klasse und die Level2-Klasse, weil die einzelnen Levels alle die gleichen Attribute und Methoden brauchen wie die LevelViewController-Klasse. Das Einzige, dass sich zwischen den Levels ändert sind die Implementierung der draw und reset-Methoden.

7 - 16

## Realisieren

### Code-Snippet und Erklärung

#### Vererbung

#### Ein Ausschnitt aus der Raumschiff-Klasse

```
public class Raumschiff extends Figur{
    public Raumschiff(int speed, int x, int y, Direction direction, PApplet window) {
        super(speed, 50, 50, x, y, direction);
        setImg_N(window.loadImage("/img/spaceship_N.png"));
        setImg_E(window.loadImage("/img/spaceship_E.png"));
        setImg_S(window.loadImage("/img/spaceship_S.png"));
        setImg_W(window.loadImage("/img/spaceship_W.png"));
    }
}
```

#### Ein Ausschnitt aus der Figur-Klasse

```
public abstract class Figur {

    private int speed;
    private int width;
    private int height;
    private int x;
    private int y;
    private PImage img_N;
    private PImage img_E;
    private PImage img_S;
    private PImage img_W;

    private ArrayList<Projektil> projektilListe = new ArrayList<>();

    Direction direction = Direction.N;

    public enum Direction {
        N,E,S,W;
    }

    public Figur(int speed, int width, int height, int x, int y, Direction direction) {
        this.speed = speed;
        this.width = width;
        this.height = height;
        this.x = x;
        this.y = y;
        this.direction = direction;
    }
}
```

Bei diesen zwei Code-Snippets sieht man gut wie die Raumschiff-Klasse von der Figur-Klasse erbt.



## Abstrakte-Klasse

## Die abstrakte View-Klasse

```

package controller;

import processing.core.PApplet;
import processing.core.PImage;

/**
 * Dieses abstrakte Klasse ist für alle Views zuständig(Parent-Class).
 * @author Wendelin
 */
public abstract class View {

    private PImage bg;

    /**
     * Diese Methode soll nur einmal vor der draw-Methode
     * ausgeführt werden um alles zu reseten.
     * @param window : PApplet
     */
    public abstract void restart(PApplet window);

    /**
     * Diese Methode zeichnet den View.
     * @param window : PApplet
     */
    public abstract void draw(PApplet window);

    /**
     * @return the bg
     */
    public PImage getBg() {
        return bg;
    }

    /**
     * @param bg the bg to set
     */
    public void setBg(PImage bg) {
        this.bg = bg;
    }
}

```

An diesem Beispiel der View-Klasse sieht man gut, wofür ich diese Klasse abstrakt gemacht habe.

## Interface

## Das Subjekt-Interface

```

package controller;

/**
 * Das ist ein Interface Subject.<br>
 * Dieses Interface arbeitet mit dem Observer-Interface zusammen.
 * @author Wendelin
 */
public interface Subject {

    /**
     * Diese Methode fügt dem Subject ein Observer-Object hinzu
     * mit dem es später kommunizieren kann.
     * @param observer : Observer
     */
    public void attach(Observer observer);

    /**
     * Diese Methode entfernt dem Subject ein Observer-Object.
     * @param observer : Observer
     */
    public void detach(Observer observer);

    /**
     * Diese Methode benachrichtigt alle Observer-Objecte
     * mit denen die hinzugefügt wurden.
     * @param nachricht : String zu übermittelnde Nachricht
     */
    public void notifyAllObserver(String nachricht);
}

```

## Das Observer-Interface

```

package controller;

/**
 * Das ist ein Interface Observer.<br>
 * Dieses Interface arbeitet mit dem Subject-Interface zusammen.
 * @author Wendelin
 */
public interface Observer {

    /**
     * Diese Methode updated den Observer je nach Nachricht die er bekommt.
     * @param nachricht die übermittelt wird.
     */
    abstract void update(String nachricht);
}

```

Diese zwei Interfaces kommunizieren miteinander indem sich der Observer dem Subject anschliesst mit der attach-Methode und dann kann das Subject alle Observer kontaktieren mit der notifyAllObserver-Methode falls es irgend einen grund giebt allen Observern eine Nachricht zu «senden».

## Collision

```

/**
 * Diese Methode überprüft ob etwas colidiert und
 * wenn es dazu kommt werden entsprechende Attribute verändert oder Methoden aufgerufen.
 */
public void checkCollisions() {

    Rectangle rBounds = raumschiff.getBounds();
    Rectangle eBounds = endPortal.getBounds();
    int eX = (int) eBounds.getX() + 25;
    int eY = (int) eBounds.getY() + 25;

    if (rBounds.contains(eX, eY)) {
        setLevelCompleat(true);
    }

    for (AsteroidenWand a : asteroidenListe) {

        Rectangle aBounds = a.getBounds();

        if (rBounds.intersects(aBounds)) {
            setGameOver(true);
        }
    }

    for (Gegner g : gegnerListe) {

        Rectangle gBounds = g.getBounds();

        if(g.isBroken()) {
            gBounds = new Rectangle(-100, -100, 50, 50);
        }

        for (Projektil p : raumschiff.getProjektilListe()) {

            Rectangle pBounds = p.getBounds();

            if (gBounds.intersects(pBounds)) {
                g.setBroken(true);
                notifyAllObserver("Jemand ist KO gegangen.");
            }
        }

        for (Projektil p : g.getProjektilListe()) {

            Rectangle pBounds = p.getBounds();

            if (pBounds.intersects(rBounds)) {
                setGameOver(true);
            }
        }

        if (rBounds.intersects(gBounds)) {
            setGameOver(true);
        }
    }
}

```

Dies ist die checkCollision-Methode in der LevelViewController-Klasse und diese Methode wird in den einzelnen Levels in der draw-Methode aufgerufen. In der Methode wird mit der intersect-Methode der Rectangle-Klasse überprüft ob sich zwei Rectangle-Objekte aufeinander liegen. Je nach dem ob dann ein «true» oder «false» zurück kommt werden gewisse Methoden aufgerufen um entsprechende Funktionalitäten ins Rollen zu bringen.

## Kontrollieren

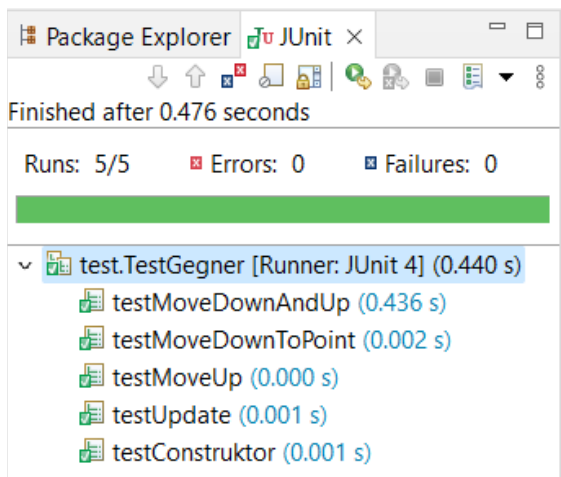
### Unit und Mockup-Test

Ich bei drei Klassen Tests durchgeführt dabei habe ich bei allen den Konstruktor und die move-Methode getestet.

Die Tests habe ich bei den Klassen Gegner, Raumschiff und Projektil durchgeführt und auch Mockup verwendet, um die PApplet-Objekte zu simulieren.

Die Test sind alle Positiv herausgekommen und es sind keine Fehler aufgetreten die ich beheben müsste.

#### TestGegner



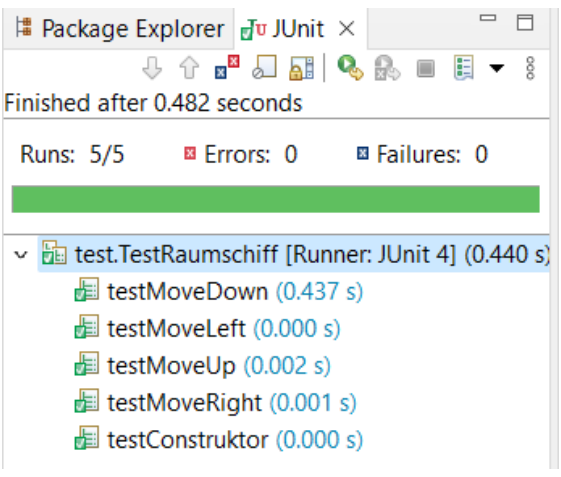
Package Explorer JUnit x

Finished after 0.476 seconds

Runs: 5/5 Errors: 0 Failures: 0

- test.TestGegner [Runner: JUnit 4] (0.440 s)
  - testMoveDownAndUp (0.436 s)
  - testMoveDownToPoint (0.002 s)
  - testMoveUp (0.000 s)
  - testUpdate (0.001 s)
  - testKonstruktor (0.001 s)

#### TestRaumschiff



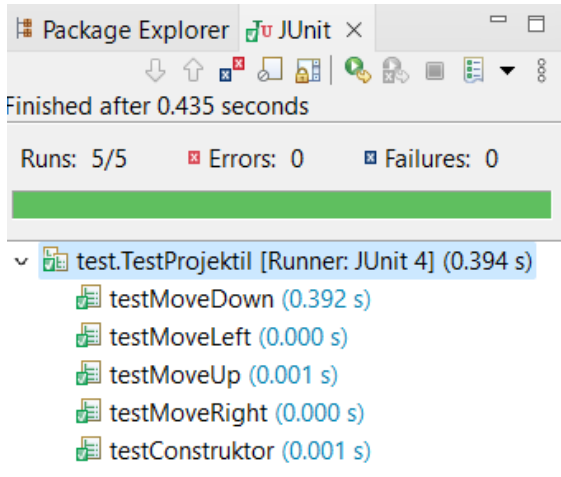
Package Explorer JUnit x

Finished after 0.482 seconds

Runs: 5/5 Errors: 0 Failures: 0

- test.TestRaumschiff [Runner: JUnit 4] (0.440 s)
  - testMoveDown (0.437 s)
  - testMoveLeft (0.000 s)
  - testMoveUp (0.002 s)
  - testMoveRight (0.001 s)
  - testKonstruktor (0.000 s)

#### TestProjektil



Package Explorer JUnit x

Finished after 0.435 seconds

Runs: 5/5 Errors: 0 Failures: 0

- test.TestProjektil [Runner: JUnit 4] (0.394 s)
  - testMoveDown (0.392 s)
  - testMoveLeft (0.000 s)
  - testMoveUp (0.001 s)
  - testMoveRight (0.000 s)
  - testKonstruktor (0.001 s)

## Auswerten und Fazit

Was habe ich gelernt?

Was geht nächstes mahl besser?

Meine Meinung wies gelaufen ist und was am schluss raus kam

# Anhang und Abbildungsverzeichnis

## Abbildungsverzeichnis

Abbildung 1: Zeitplan 13.12.2021 ..... 5

### SpaceShip

Wendelin Haller

Project Start: Mo, 11.22.2021  
Display Week: 1

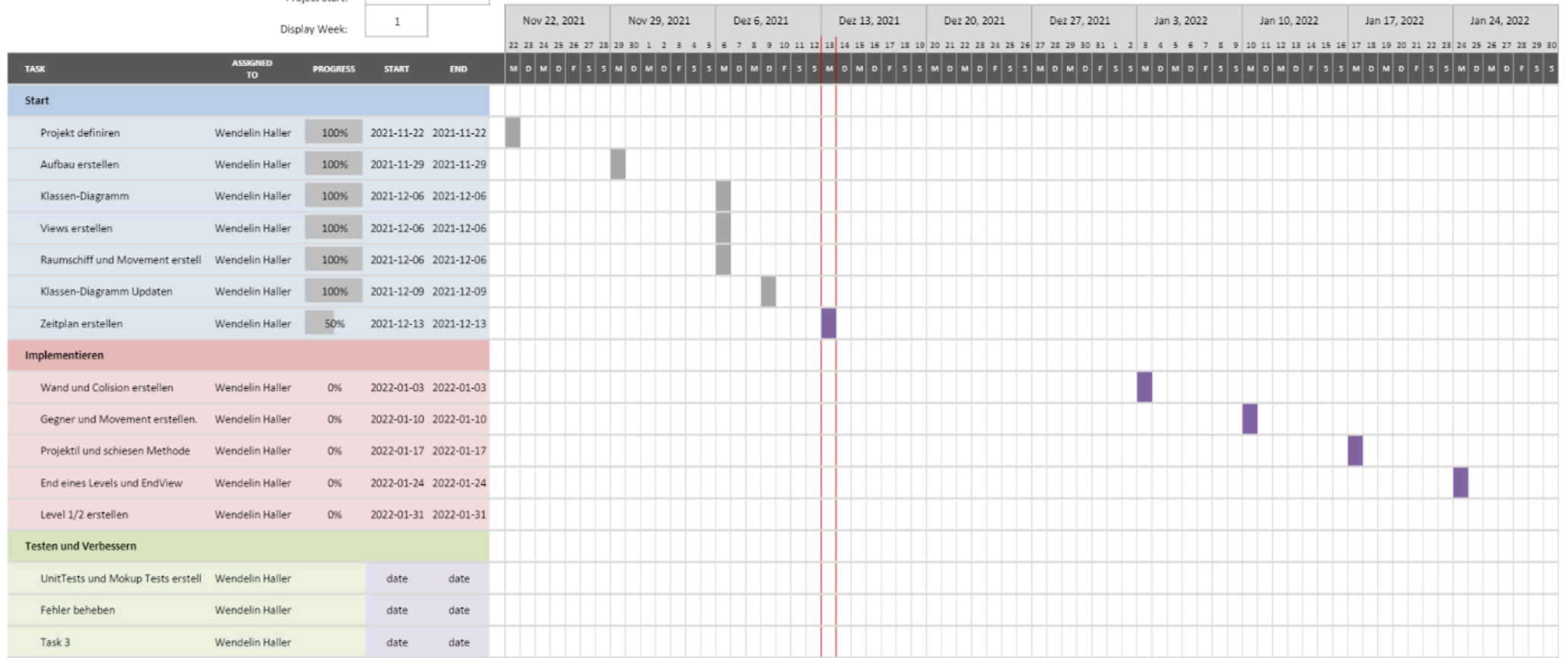


Abbildung 2: Zeitplan 21.01.2021 ..... 5

## SpaceShip

Wendelin Haller

Project Start:

Mo, 11.22.2021

Display Week:

1

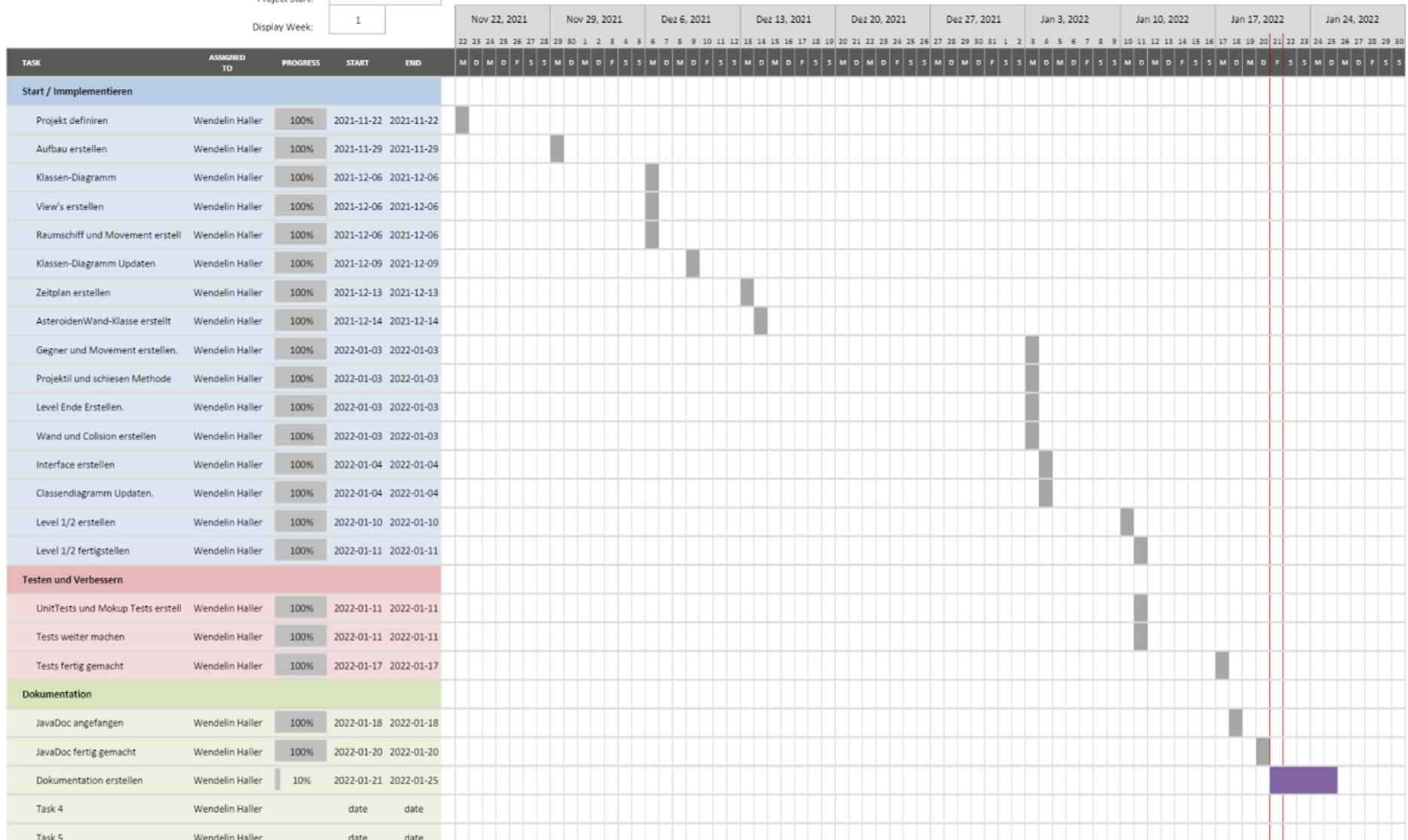


Abbildung 3: Klassen-Diagramm ..... 7

